

**School of Engineering
Department of Mechanical Engineering**

**Error Minimising Gradients for Improving
Cerebellar Model Articulation Controller Performance**

Peter Craig Scarfe

**This thesis is presented for the Degree of
Doctor of Philosophy
of
Curtin University of Technology**

April 2009

ABSTRACT

In motion control applications where the desired trajectory velocity exceeds an actuator's maximum velocity limitations, large position errors will occur between the desired and actual trajectory responses. In these situations standard control approaches cannot predict the output saturation of the actuator and thus the associated error summation cannot be minimised.

An adaptive feedforward control solution such as the Cerebellar Model Articulation Controller (CMAC) is able to provide an inherent level of prediction for these situations, moving the system output in the direction of the excessive desired velocity before actuator saturation occurs. However the pre-empting level of a CMAC is not adaptive, and thus the optimal point in time to start moving the system output in the direction of the excessive desired velocity remains unsolved. While the CMAC can adaptively minimise an actuator's position error, the minimisation of the summation of error over time created by the divergence of the desired and actual trajectory responses requires an additional adaptive level of control.

This thesis presents an improved method of training CMACs to minimise the summation of error over time created when the desired trajectory velocity exceeds the actuator's maximum velocity limitations. This improved method called the Error Minimising Gradient Controller (EMGC) is able to adaptively modify a CMAC's training signal so that the CMAC will start to move the output of the system in the direction of the excessive desired velocity with an optimised pre-empting level.

The EMGC was originally created to minimise the loss of linguistic information conveyed through an actuated series of concatenated hand sign gestures reproducing deafblind sign language. The EMGC concept however is able to be implemented on any system where the error summation associated with excessive desired velocities needs to be minimised, with the EMGC producing an improved output approximation over using a CMAC alone.

In this thesis, the EMGC was tested and benchmarked against a feedforward / feedback combined controller using a CMAC and PID controller. The EMGC was

tested on an air-muscle actuator for a variety of situations comprising of a position discontinuity in a continuous desired trajectory. Tested situations included various discontinuity magnitudes together with varying approach and departure gradient profiles.

Testing demonstrated that the addition of an EMGC can reduce a situation's error summation magnitude if the base CMAC controller has not already provided a prior enough pre-empting output in the direction of the situation. The addition of an EMGC to a CMAC produces an improved approximation of reproduced motion trajectories, not only minimising position error for a single sampling instance, but also over time for periodic signals.

Declaration

To the best of my knowledge and belief this thesis contains no material previously published by any other person except where due acknowledgment has been made.

This thesis contains no material which has been accepted for the award of any other degree or diploma in any university.

Signature:

Date:

Acknowledgements

To my Father and Mother, Craig and Marilena, to my Brother Steven and Sister Silvana, and to my Nonna, Rosa, for their support in numerous ways and faith in me over the long journey.

To Reine, who spent several years with me, supporting me along the journey, your support was always appreciated deeply. Thanks so much!

To Ksenia, who over the most difficult part of my journey, motivated me in so many ways to keep going until the very end, and provided an unequalled level of editorial support!

To my PhD co-workers, Josh, Matt and Dan, especially to Matt who always seems to know a lot about everything I need to know... you saved me a lot of time in several areas! Thanks mate! And Dan for keeping my wits up over many late nights working down in the lab...

To the various students that I tutored along the way, many of which became friends once they saw that I wasn't as smart and superior as I pretended to be!

To Jeff, who had to put up with me borrowing this and that at times that were most convenient to me, and always offering his technical expertise and sense of humour over the many years.

To my co-supervisor Jon, who helped me add a final coat of polish to the thesis!

To my supervisor, Euan. To you I owe an enormous debt, your vigorous support spanning over 5 years, with your dedication to me and my fellow co-workers, always lending a hand when most needed, helped shape the final product into the proud piece of work it ended up to be. Thanks again Euan!!

And to everyone else that lent a hand, an ear, or an idea, no matter how big or small, your contribution was much appreciated.

TABLE OF CONTENTS

ABSTRACT	III
TABLE OF CONTENTS.....	VII
LIST OF FIGURES.....	XI
GLOSSARY	XIV
1.0 INTRODUCTION.....	1
1.1 CONTRIBUTIONS OF THE THESIS	1
1.2 THESIS CONTENT	2
2.0 LITERATURE REVIEW	5
2.1 DEAFBLIND COMMUNICATION	5
2.2 DEAFBLIND COMMUNICATION METHODS	6
2.2.1 <i>Partially Deaf, Partially Blind</i>	7
2.2.2 <i>Partially Deaf, Completely Blind</i>	8
2.2.3 <i>Completely Deaf, Partially Blind</i>	8
2.2.4 <i>Completely Deaf, Completely Blind (in later life)</i>	8
2.2.5 <i>Completely Deaf, Completely Blind (in early life)</i>	9
2.2.6 <i>Artificial Hand Purpose</i>	11
2.3 THE HUMAN HAND MODEL	12
2.4 DEAFBLIND FINGERSPELLING HANDS.....	14
2.5 TRAJECTORY CAPTURING – DATA GLOVES	16
2.6 CONCATENATING CAPTURED TRAJECTORIES.....	21
2.7 BIOLOGICALLY INSPIRED CONTROLLERS.....	23
2.7.1 <i>Artificial Neural Networks (ANNs)</i>	24
2.7.1.1 Common Artificial Neural Networks.....	24
2.7.1.2 Artificial Neural Network Training.....	26
2.7.2 <i>Which Artificial Neural Network?</i>	27
2.7.3 <i>Uses and Adaptations of the CMAC</i>	28
2.8 THE CMAC NEURAL NETWORK	29
2.8.1 <i>The Model</i>	30
2.8.2 <i>The Input Vector – Section L1</i>	31
2.8.3 <i>The $S \rightarrow M \rightarrow A$ Mapping – Section L2</i>	32
2.8.4 <i>The Weight Table – Section L3</i>	35
2.8.5 <i>Weight Summation and Adjustment – Section L4</i>	36
2.8.6 <i>CMAC Compensation of System Time Delay</i>	37
2.8.7 <i>Controller-Specific CMAC Additions/Modifications</i>	40
2.8.7.1 Memory Handling	40
2.8.7.2 Uniform Receptive Field Mapping.....	42
2.8.7.3 Receptive Field Sensitivity Functions.....	43
2.8.7.4 Weight Smoothing.....	45
2.8.7.5 Eligibility.....	46
2.8.7.6 CMACs arranged in Control Hierarchies.....	48
2.8.7.7 CMAC Systems Utilising Multiple Control Resolutions.....	51
2.8.7.8 Combined CMAC Enhancement Capability.....	53
2.9 DESIGN OF A SUITABLE CONTROLLER	53
2.10 THESIS CHALLENGE	57
3.0 DYNAMIC AND STATIC CMAC MEMORY ALLOCATION	60
3.1 HASHING	61
3.2 BINARY SEARCH TREES.....	66
3.3 IMPLEMENTING BINARY SEARCH TREES INTO THE CMAC	70
3.3.1 <i>Node Structure</i>	70
3.3.2 <i>Node Memory Assignment</i>	71

3.3.3	<i>Node Key Generation</i>	72
3.3.4	<i>Binary Tree Searching</i>	77
3.3.5	<i>Binary Tree Balancing</i>	80
4.0	COMPARISONS BETWEEN BST AND HASHING WITH THE CMAC	82
4.1	PRE-TEST CONSIDERATIONS.....	83
4.1.1	<i>Hashing With and Without Hash Collisions</i>	83
4.1.2	<i>Binary Search Tree Balancing</i>	84
4.1.3	<i>Test 1: Weight Table Utilisation</i>	86
4.1.4	<i>Test 1.1: Build Times</i>	87
4.1.5	<i>Test 1.2: Search Times</i>	89
4.2	TEST 2: NUMBER OF RECEPTIVE FIELD LAYERS.....	90
4.2.1	<i>Receptive Field Considerations</i>	91
4.2.2	<i>Test 2.1: Build Times</i>	94
4.2.3	<i>Test 2.2: Search Times</i>	95
4.3	TEST 3: CMAC ONLINE TRAINING.....	96
4.3.1	<i>Test 3.1: Additional Training (Variable)</i>	97
4.3.2	<i>Test 3.2: Additional Training (Fixed)</i>	98
4.4	TEST 4: OPTIMAL BST VS. HASHING PERFORMANCE VARIABLES.....	99
4.5	SUMMARY.....	103
5.0	CMAC+PID CONTROLLER	105
5.1	PID CONTROLLER.....	106
5.2	CMAC CONTROL.....	108
5.2.1	<i>Input Vector Parameters</i>	109
5.2.1.1	Desired Position.....	109
5.2.1.2	Desired Velocity.....	110
5.2.1.3	Actual Position.....	111
5.2.1.4	Time (when required).....	112
5.2.2	<i>Generalisation Area</i>	114
5.2.3	<i>Actuator Response Time Delay</i>	117
5.2.4	<i>Weight Smoothing</i>	118
5.2.5	<i>CMAC with PID Issues</i>	120
5.2.5.1	Harmonious CMAC and PID Interaction.....	120
5.2.5.2	Resetting the PID Output.....	122
6.0	THEORETICAL EMGC MODEL	124
6.1	PURPOSE OF THE CONTROLLER.....	124
6.1.1	<i>Situation Definition</i>	127
6.1.2	<i>Advanced Knowledge of a Situation</i>	129
6.2	EMGC THEORETICAL MODEL.....	132
6.2.1	<i>Situation Processing Phase</i>	133
6.2.1.1	Delay – FIFO Buffer.....	134
6.2.1.2	Situation Detection.....	134
6.2.1.3	Situation Memory Variables.....	135
6.2.1.4	Situation Classification.....	136
6.2.1.5	Situation Assignment.....	138
6.2.1.6	Situation Completion.....	139
6.2.2	<i>EMG Processing Phase</i>	140
6.2.2.1	The EMG.....	142
6.2.2.2	Recall Previous EMG Origin.....	142
6.2.2.3	A Practical Consideration: The EMG Origin Offset.....	143
6.2.2.4	Find the EMG Slope with EMG Offset Percentage.....	145
6.2.2.5	The Error Associated with the Situation.....	146
6.2.2.6	EMG Origin Time-Shift Factor.....	148
6.2.2.7	EMG Shift Rate Value.....	148
6.2.2.8	New EMG Generation.....	150
6.2.2.9	EMG Limits.....	150
6.2.3	<i>EMGC CMAC Training</i>	151
6.3	EMGC INTEGRATION WITH CMAC+PID CONTROLLER.....	153
7.0	TESTING SETUP	156

7.1	ACTUATION TEST RIG	156
7.2	REAL-TIME CONTROL OPTIONS AVAILABLE.....	157
7.3	XENOMAI CAPABILITIES	158
7.4	HARDWARE/SOFTWARE USED	159
7.5	SOFTWARE FRAMEWORK DEVELOPED.....	159
7.5.1	<i>Xenomai Threads</i>	160
7.5.2	<i>Linux Threads</i>	162
7.6	DATA LOGGING	163
7.7	GUI INTERFACE	164
7.7.1	<i>Control Buttons</i>	164
7.7.2	<i>CMAC Input Spaces</i>	167
7.7.3	<i>Online Variables</i>	167
7.7.4	<i>Graphing Applet</i>	169
7.7.5	<i>Status Bar</i>	170
8.0	EMGC INITIAL SQUARE WAVE TESTS.....	172
8.1	EMG OFFSET PERCENTAGE DETERMINATION.....	172
8.2	EMGC SITUATION ERROR PERFORMANCE	177
8.2.1	<i>Square Wave Situation Testing (First Spring: $k=608N/m$)</i>	179
8.2.2	<i>Square Wave Situation Testing (Second Spring: $k=912N/m$)</i>	184
8.2.3	<i>Comparison of First and Second Return-Spring Test Data</i>	188
8.2.4	<i>Explanation of Results</i>	190
8.3	EMGC DIRECT EFFECT ON ACTUAL TRAJECTORY RESPONSE	194
8.4	CHAPTER CONCLUSIONS.....	196
9.0	APPROACH AND DEPARTURE GRADIENT TESTS.....	198
9.1	APPROACH AND DEPARTURE GRADIENTS METHODOLOGY	199
9.2	APPROACH AND DEPARTURE GRADIENT TESTS	200
9.2.1	<i>Initial Error Ratio vs Approach and Departure Gradient Relationships</i>	201
9.2.2	<i>Error Gain vs Approach and Departure Gradient Relationships</i>	208
9.2.3	<i>Error Ratio vs Error Gain Relationship</i>	210
9.3	CONCLUSIONS.....	214
10.0	CONCLUSIONS	216
10.1	THESIS CONCLUSIONS	216
10.2	FUTURE WORK.....	219
10.3	CONTRIBUTIONS OF THE THESIS.....	220
REFERENCES		221
A.0	HUMANOID HANDS.....	230
A.1	HUMANOID HAND USES	230
A.2	HUMANOID HAND ACTUATION	232
A.3	HUMANOID HAND SENSORY FEEDBACK	234
B.0	PID GAINS	235
B.1	PROPORTIONAL GAIN:	235
B.2	DERIVATIVE GAIN:.....	235
B.3	INTEGRAL GAIN:	236
C.0	VELOCITY INPUT VECTOR PARAMETER	237
D.0	INTERFACE HARDWARE.....	240
D.1	ELECTRONICS HARDWARE – PWM GENERATION	240
D.2	AIR-MUSCLE UPGRADES.....	242
D.3	PBV BOARD UPGRADES	242
D.3.1	<i>PBV Board Manifold Regulator/Filter</i>	242
D.3.2	<i>PBV Airflow Taps</i>	243
D.3.3	<i>Higher Usable Pressure</i>	243
D.3.4	<i>Accurate PBV Calibration Method</i>	243

D.3.5	<i>Possible Improvements</i>	244
D.4	PBV BOARD VARIABLES	244
D.4.1	<i>Pneumatic Supply Pressure</i>	244
D.4.2	<i>PBV Board Manifold Pressure</i>	244
D.4.3	<i>Air-muscle Volume</i>	245
D.4.4	<i>Maximum Control Voltages</i>	245
D.4.5	<i>PBV Specific Variables</i>	245
D.4.6	<i>Solenoid Air-Gap Height</i>	245
D.4.7	<i>Airflow Tap Adjustment</i>	248
D.4.8	<i>Outlet Pipe Air-Gap</i>	248
D.5	PBV CALIBRATION PROCEDURE	251
E.0	TOTAL ERROR REDUCTION METHOD	254
F.0	EMGC TEST LOG	258
G.0	RESULTS FROM TEST 73 TO TEST 122	280
H.0	ADDITIONAL APPROACH AND DEPARTURE GRADIENT TESTS	281
I.0	PRELIMINARY PVB CALIBRATION TESTS	282

LIST OF FIGURES

FIGURE 1.2 – ASPECTS OF THE HUMAN HAND CONTROLLER FOR LEARNING AND PERFORMING DEAFBLIND HAND SIGNS	3
FIGURE 1.1 – THE DEAFBLIND MANUAL ALPHABET FINGERSPELLING SIGNS (DBNZ, 2006)	10
FIGURE 2.1 – THE BONES AND JOINTS IN A HUMAN HAND (SHIM, 2004)	13
FIGURE 2.2 – RALPH FINGER-SPELLING HAND (JAFFE, 1994)	15
FIGURE 2.3 – RESISTIVE BEND SENSOR FUNCTIONALITY (TELEO, 2004)	17
FIGURE 2.4 – RESISTIVE BEND SENSORS EMBEDDED ON FINGERS OF A GLOVE (KRIETE, 2004).....	17
FIGURE 2.5 – MODERN SINGLE-AXIS GYROSCOPE SITTING ON USA QUARTER DOLLAR COIN (SPARKFUN, 2005).....	18
FIGURE 2.6 – NASA/DARPA ROBONAUT TELEOPERATED SYSTEM (DIFTLER ET AL., 2003).....	20
FIGURE 2.7 – THE JOINING OF HAND SIGN TRAJECTORIES, SEPARATED BY ‘SITUATIONS’	22
FIGURE 2.8 – BLOCK DIAGRAM OF ALBUS’S CMAC FOR A SINGLE JOINT ACTUATOR (ALBUS, 1981)..	30
FIGURE 2.9 – A TWO-DIMENSIONAL CMAC MAPPING WITH FOUR QUANTISING FUNCTIONS.....	32
FIGURE 2.10 – $S \rightarrow M \rightarrow A$ MAPPING OF TWO-DIMENSIONAL INPUT VECTOR $S = (5, 9)$	34
FIGURE 2.11 – SECTION L4: CMAC WEIGHT SUMMATION AND ADJUSTMENT.....	36
FIGURE 2.13 – CMAC WEIGHT TRAINING WITH NON-COMPENSATED TIME DELAY	38
FIGURE 5.9 – CMAC TIME DELAY HANDLING USING OVERLAPPING RECEPTIVE FIELDS.....	38
FIGURE 5.10 – INCORRECT TRAINING OF NON-OVERLAPPING WEIGHTS DUE TO TIME DELAY	40
FIGURE 2.12 – CMAC WITH HASH-CODING IMPLEMENTED (HSU ET AL., 2002)	41
FIGURE 2.13 – UNIFORM RECEPTIVE FIELD MAPPING EXAMPLE	42
FIGURE 2.14 – RECEPTIVE FIELD SENSITIVITY FUNCTION EXAMPLE	43
FIGURE 2.15 – (A) CMAC APPROXIMATION OF THE FUNCTION FSIN WITH BINARY SENSITIVITY FUNCTIONS AND (B) REMAINING ABSOLUTE ERROR. (ELDRACHER ET AL., 1994).....	44
FIGURE 2.16 – (A) CMAC APPROXIMATION OF THE FUNCTION FSIN WITH GAUSSIAN CONTINUOUS SENSITIVITY FUNCTIONS AND (B) REMAINING ABSOLUTE ERROR. (ELDRACHER ET AL., 1994)	45
FIGURE 2.17 – ELIGIBILITY IMPLEMENTED IN A TWO-DIMENSIONAL CMAC WEIGHT TABLE (SMITH, 1998)	46
FIGURE 2.18 – TRAJECTORY OUTPUTS WITH AND WITHOUT ELIGIBILITY (SMITH, 1998)	47
FIGURE 2.19 – A HIERARCHICAL STRUCTURE OF CMAC CONTROLLERS. (ALBUS, 1975A)	49
FIGURE 2.20 – DETAILED ARCHITECTURE OF A 4-INPUT-VARIABLE HCMAC (LEE ET AL., 2003)	50
FIGURE 2.21 – MULTIPLE DIFFERENT-RESOLUTION CMACS IN PARALLEL (HUANG ET AL., 1997).....	52
FIGURE 2.22 – POSSIBLE CMAC CONTROL OF ACTUATOR FOR DESIRED CONCATENATED TRAJECTORIES.	54
FIGURE 2.23 – ERROR MINIMISATION OVER TIME	55
FIGURE 2.24 – ERROR MINIMISED CONTROL OF ACTUATOR FOR DESIRED CONCATENATED TRAJECTORIES.	56
FIGURE 3.1 – HASHING PROCESS OVERVIEW	61
FIGURE 3.2 – COLLISION RESISTANT HASHING PROCESS	63
FIGURE 3.3 – STATIC HASHING VS. DYNAMIC BINARY SEARCH TREE MEMORY ALLOCATION	65
FIGURE 3.4 – A SINGLE BINARY TREE NODE	66
FIGURE 3.5 – SIMPLE BINARY TREE CONTAINING 4 NODES	67
FIGURE 3.6 – (A) A STICK, (B) A BALANCED BINARY TREE	68
FIGURE 3.7 – STRUCTURE OF DATA CONTAINED IN EACH NODE WITHIN PHYSICAL MEMORY	71
FIGURE 3.8 – UNH CMAC CODE RECEPTIVE FIELD KEY ASSIGNMENT	73
FIGURE 3.9 – INPUT VECTOR TO UNIQUE RF VECTORS IN RECEPTIVE FIELD SPACE.....	74
FIGURE 3.10 – FOUR RECEPTIVE FIELD VECTORS REPRESENTED AS A SINGLE 32BIT INTEGER	76
FIGURE 3.11 – RANDOM NODE KEY GENERATOR ARRAY	78
FIGURE 3.12 – BINARY TREE NODE SEARCHING AND INSERTION	79
FIGURE 3.13 – BINARY TREE BALANCING – FROM UNBALANCED TO BALANCED TREE.....	81
FIGURE 4.1 – HASH COLLISIONS VS. WEIGHT TABLE UTILISATION (FOR ~65000 WEIGHTS).....	84
FIGURE 4.2 – RANDOMLY GENERATED BST VS. BALANCED BST SEARCH TIMES.....	85
FIGURE 4.3 – CMAC TRAINING FUNCTION	87
FIGURE 4.4 – BUILD TIMES VS. WEIGHT TABLE UTILISATION (FOR ~65000 ALLOCATED WEIGHTS).....	88
FIGURE 4.5 – SEARCH TIMES VS. WEIGHT TABLE UTILISATION (FOR ~65000 ALLOCATED WEIGHTS) .	89
FIGURE 4.6 – FEWER LARGE BSTS VS. NUMEROUS SMALL BSTS	92

FIGURE 4.7 – WEIGHTS USED VS. NUMBER OF RF LAYERS	93
FIGURE 4.8 – BUILD TIMES FOR DIFFERENT NUMBERS OF RF LAYERS	94
FIGURE 4.9 – SEARCH TIMES FOR DIFFERENT NUMBERS OF RF LAYERS.....	95
FIGURE 4.10 – CMAC TRAINING WITH ADDITIONAL DATA TO ALREADY ESTABLISHED CMAC (RF=16)	97
FIGURE 4.11 – CMAC TRAINING TIMES WITH 5% ADDITIONAL NEW DATA.....	99
FIGURE 4.12 – TRAINING TIME VS. INPUT SPACE VECTOR USAGE.....	100
FIGURE 4.13 – TRAINING TIME VS. INPUT SPACE VECTOR USAGE (3-D).....	101
FIGURE 4.14 – PREFERRED BST AND HASHING PARAMETERS.....	102
FIGURE 5.1 – CMAC+PID CONTROLLER IN THE OVERALL CONTROL SYSTEM	105
FIGURE 5.2 – POSITION AND ERROR RESPONSE COMPARING PID ONLY CONTROL TO CMAC+PID CONTROL	108
FIGURE 5.3 – NEAR INFINITE DESIRED VELOCITIES ON RISING AND FALLING EDGES	110
FIGURE 5.4 – DESIRED SQUARE WAVE INPUT MAPS TO ONLY TWO CMAC INPUT SPACE VECTORS .	112
FIGURE 5.5 – NON-QUANTISED CMAC INPUT SPACE TO OUTPUT SPACE RELATIONSHIP (#RF = 8)...	115
FIGURE 5.6 – QUANTISED CMAC INPUT SPACE TO OUTPUT SPACE RELATIONSHIP (#RF = 8)	115
FIGURE 5.7 – NON-QUANTISED CMAC INPUT SPACE TO OUTPUT SPACE RELATIONSHIP (#RF = 16) .	116
FIGURE 5.11 – FIFO INPUT VECTOR BUFFER FOR TIME DELAY COMPENSATION	118
FIGURE 5.12 – CMAC WEIGHT SMOOTHING ON AND OFF	119
FIGURE 6.1 – EMGC IN THE OVERALL CONTROL SYSTEM.....	124
FIGURE 6.2 – ERROR MINIMISATION OVER TIME (IDENTICAL TO FIGURE 2.23)	125
FIGURE 6.3 – POSSIBLE CMAC CONTROL OF ACTUATOR FOR DESIRED CONCATENATED TRAJECTORIES (IDENTICAL TO FIGURE 2.22).....	126
FIGURE 6.4 – ERROR MINIMISED CONTROL OF ACTUATOR FOR DESIRED CONCATENATED TRAJECTORIES (IDENTICAL TO FIGURE 2.24).....	126
FIGURE 6.5 – SQUARE WAVE TRAJECTORY SITUATION	127
FIGURE 6.6 – NATURAL TRAJECTORY SITUATION.....	128
FIGURE 6.7 – (A) SQUARE WAVE WITHOUT ERROR MINIMISATION, (B) SQUARE WAVE WITH ERROR MINIMISATION.....	131
FIGURE 6.8 – EMGC INTERNAL THEORETICAL MODEL	133
FIGURE 6.9 – FIFO BUFFER.....	134
FIGURE 6.10 – (A) THRESHOLD GRADIENT DETERMINATION, (B) SITUATION DETECTION USING GRADIENT COMPARISON	135
FIGURE 6.11 – SITUATION CLASSIFICATION VECTORS FOR TWO DIFFERENT SITUATIONS.....	137
FIGURE 6.12 – MULTIPLE SITUATIONS WITHIN HORIZON TIME	138
FIGURE 6.13 – STANDARD TRAJECTORY PLOT POINTS.....	141
FIGURE 6.14 – ORIGINAL SITUATION WITH THE ADDITION OF EMG.....	142
FIGURE 6.15 – INITIAL PHYSICAL RESPONSE OF THE AIR-MUSCLE BEFORE THE EMG SUPERPOSITION OCCURS.....	144
FIGURE 6.16 – CREATION OF AN EMG USING A 50% EMG OFFSET PERCENTAGE	146
FIGURE 6.17 – SITUATION ERROR REGIONS	147
FIGURE 6.18 – (A) EMG SHIFT RATE SMALL (B) EMG SHIFT RATE LARGE	149
FIGURE 6.19 – GENERATION OF THE NEW EMG ORIGIN AT POINT (D).....	150
FIGURE 6.20 – EMGC CMAC TRAINING DATA	152
FIGURE 6.21 – EMGC CMAC OUTPUT SPACE FOR INPUT VECTOR DIMENSIONS M & Cv.	153
FIGURE 6.22 – EMGC OUTPUT PROVIDING NEW DESIRED TRAINING SIGNAL TO CMAC	154
FIGURE 7.1 – PICTURE OF AIR-MUSCLE RIG USED FOR TESTING THE EMGC	156
FIGURE 7.2 – LINUX/XENOMAI SOFTWARE FRAMEWORK	160
FIGURE 7.3 – LINUX GRAPHICAL USER INTERFACE CREATED WITH WxWIDGETS FOR XENOMAI CMAC CONTROLLER.....	165
FIGURE 8.1 – EMG OPD TEST. GRAPHING APPLLET RESPONSE. EMGC OFF.	173
FIGURE 8.2 – EMG OPD TEST. TOTAL ERROR FOR EMG PERCENTAGE OFFSET COMPARISON TEST.	174
FIGURE 8.3 – EMG OPD TEST. ERROR RATIO FOR EMG PERCENTAGE OFFSET COMPARISON TEST. .	175
FIGURE 8.4 – EMG OPD TEST. MAXIMUM AV FOR EMG PERCENTAGE OFFSET COMPARISON TEST.	175
FIGURE 8.5 – EMG OPD TEST. EMGS FOR EMG PERCENTAGE OFFSET COMPARISON TEST.	176
FIGURE 8.6 – EMG OPD TEST. EMG OFFSET PERCENTAGE AT 100%.....	177
FIGURE 8.7 – TEST TRAINING FUNCTION. THREE SECOND PERIOD, 60 BIT MAGNITUDE, SQUARE WAVE.	178
FIGURE 8.8 – FIRST SPRING: ERROR RATIO VS. TOTAL ERROR VS. ITERATION 3-D (TEST 08 TO TEST 15)	179

FIGURE 8.9 – FIRST SPRING: TOTAL ERROR VS. ITERATION (TEST 08 TO TEST 15).....	180
FIGURE 8.10 – FIRST SPRING: ERROR GAIN VS. ITERATION (TEST 08 TO TEST 15).....	182
FIGURE 8.11 – FIRST SPRING: ERROR RATIO VS. TOTAL ERROR (TEST 08 TO TEST 15).....	183
FIGURE 8.12 – SECOND SPRING: ERROR RATIO VS. TOTAL ERROR VS. ITERATION 3-D (TEST 21 TO TEST 28).....	184
FIGURE 8.13 – SECOND SPRING: TOTAL ERROR VS. ITERATION (TEST 21 TO TEST 28).....	185
FIGURE 8.14 – SECOND SPRING: ERROR GAIN VS. ITERATION (TEST 21 TO TEST 28).....	186
FIGURE 8.15 – SECOND SPRING: ERROR RATIO VS. TOTAL ERROR (TEST 21 TO TEST 28).....	187
FIGURE 8.16 – FIRST AND SECOND SPRING TESTS: ERROR RATIO VS. TOTAL ERROR (TEST 08 TO TEST 28).....	188
FIGURE 8.17 – FIRST AND SECOND SPRING TESTS: ERROR GAIN VS. TOTAL ERROR (TEST 08 TO TEST 28).....	189
FIGURE 8.18 – EMGC OUTPUT PROVIDING NEW DESIRED TRAINING SIGNAL TO CMAC (IDENTICAL TO FIGURE 6.22).....	191
FIGURE 8.19 – INITIAL PHYSICAL RESPONSE OF THE AIR-MUSCLE BEFORE THE EMG SUPERPOSITION OCCURS.....	192
FIGURE 8.20 – (A) EMGC NOT BENEFICIAL FOR SMALL INITIAL ERROR RATIOS (B) EMGC BENEFICIAL FOR LARGE INITIAL ERROR RATIOS.....	193
FIGURE 8.21 – POSITION AND ERROR RESPONSE COMPARING EMGC TO CMAC+PID CONTROL.....	195
FIGURE 9.1 – NATURAL HAND GESTURE TRAJECTORY APPROACH AND DEPARTURE PROFILES FOR A SITUATION.....	198
FIGURE 9.2 – DESIRED TRAINING TRAJECTORY FOR SIMULATED SITUATION APPROACH AND DEPARTURE GRADIENTS.....	199
FIGURE 9.3 – POSITIVE AND NEGATIVE APPROACH AND DEPARTURE GRADIENTS.....	200
FIGURE 9.4 – DEPARTURE AND ARRIVAL GRADIENT COMBINATIONS USED IN TEST 73 TO TEST 122.....	201
FIGURE 9.5 – ERROR RATIO VS. DEPARTURE GRADIENT FOR VARIOUS APPROACH GRADIENTS (TEST 73 TO TEST 122).....	202
FIGURE 9.6 – ERROR RATIO VS. APPROACH GRADIENT FOR VARIOUS DEPARTURE GRADIENTS (TEST 73 TO TEST 122).....	203
FIGURE 9.7 – FIRST ERROR VS. APPROACH GRADIENT FOR VARIOUS DEPARTURE GRADIENTS (TEST 73 TO TEST 122).....	204
FIGURE 9.8 – SECOND ERROR VS. APPROACH GRADIENT FOR VARIOUS DEPARTURE GRADIENTS (TEST 73 TO TEST 122).....	205
FIGURE 9.9 – THEORETICAL ACTUAL SYSTEM RESPONSE DUE TO INHERENT SITUATION PRE-EMPTING FOR VARYING APPROACH GRADIENTS.....	206
FIGURE 9.10 – PRACTICAL ACTUAL SYSTEM RESPONSE FOR POSITIVE APPROACH GRADIENTS.....	208
FIGURE 9.11 – ERROR GAIN VS. DEPARTURE GRADIENT FOR VARIOUS APPROACH GRADIENTS (TEST 73 TO TEST 122).....	209
FIGURE 9.12 – ERROR GAIN VS. APPROACH GRADIENT FOR VARIOUS DEPARTURE GRADIENTS (TEST 73 TO TEST 122).....	210
FIGURE 9.13 – ERROR GAIN VS. ERROR RATIO FOR VARIOUS APPROACH GRADIENTS (TEST 73 TO 122).....	211
FIGURE 9.14 – ERROR GAIN VS. ERROR RATIO FOR VARIOUS DEPARTURE GRADIENTS (TEST 73 TO 122).....	211
FIGURE C.1 – A VS. B FROM EQUATION C.2.....	238
FIGURE C.2 – DESIRED VELOCITY TO CMAC INPUT SPACE MAPPING RELATIONSHIP.....	239
FIGURE D.1 – 20Hz ENHANCED RESOLUTION TIMING DIAGRAM.....	241
FIGURE D.2 – SOLENOID AIR-GAP / FORCE RELATIONSHIP (JAYCAR, 2003).....	246
FIGURE D.3 – RELATIONSHIP BETWEEN SOLENOID PBV VOLTAGE AND AIR-MUSCLE POSITION.....	247
FIGURE D.4 – MODIFIED SOLENOID FOR PBV USAGE (SCARFE, 2004).....	248
FIGURE E.1 – MINIMUM TOTAL ERROR AND ERROR RATIO RESPONSE FOR SYMMETRICAL SITUATION.....	254
FIGURE E.2 – MINIMUM TOTAL ERROR AND ERROR RATIO RESPONSE FOR NON-SYMMETRICAL SITUATION.....	257

GLOSSARY

Ap	‘Actual Position’ response of the actuator.
Approach Gradient	The gradient of the trajectory immediately preceding a situation occurrence.
Binary Search Tree	A dynamic structure for storing and searching objects in memory, utilising a parent node with two child node system, where one child node is of lesser value than the parent, and the other child node being of great value than the parent.
CMAC	‘Cerebellar Model Articulation Controller’. An adaptive controller.
Departure Gradient	The gradient of the trajectory immediately following a situation occurrence.
Dp	‘Desired Position’ response for the actuator.
Eligibility	The concept by which the current output of a controller is the result of the contribution of both current and previous controller parameters learnt from previous training iterations.
EMG	‘Error Minimising Gradient’. Outputted from the EMGC, and used to replace a ‘situation’ in the desired trajectory.
EMG Offset Percentage	A percentage of the maximum attained gradient (of the desired trajectory over the time span of an active situation) used to offset the maximum attained gradient, to saturate the controller output.

EMGC	‘Error Minimising Gradient Controller’. Used to modify the CMAC training signal by minimising the error over time created by ‘situations’.
Error Gain	The ratio of Total Error between when the EMGC is On and when is it Off, defined by Equation 8.1. Used to compare EMGC effectiveness against CMAC+PID control alone.
Error Ratio	The ratio of errors between the first and second error regions, as defined by Equation 6.2.
Generalisation Area	The magnitude of the space covering an x number of input vectors, defined by the Generalisation Parameter and the quantisation level of the input space.
Generalisation Parameter	The number of receptive fields used in a CMAC.
Hashing	A memory storage and retrieval algorithm which maps a larger multidimensional data space into a smaller space of pre-determined size.
Horizon Distance	The length of the delay time used in the EMGC, to allow the EMGC to compensate for a situation before the situation actual occurs.
Initial Error Ratio	The final converged Error Ratio of a situation by use of the CMAC+PID controller only (without the EMGC).
Inherent Situation Pre-empting	The level of Situation Pre-empting that the CMAC+PID controller produces (without use of the EMGC).
Online	Referring to ‘Online Training’, meaning the training is performed in real-time.

PBV	Pressure Balancing Valve: a solenoid based air valve used for controlling the pressure in the air-muscles.
Quantising Function	A function that defines the quantity and magnitude (in input space resolution elements) of the receptive fields in the CMAC input space, defining the area of local generalisation produced by the excitation of a single input vector.
Receptive Field	The size of an n-dimensional space associated with the excitation of a single CMAC input vector, corresponding directly to a weight address in the CMAC weight table.
Situation	Part of a desired trajectory which exceeds the physical response limitations of the output device, such as an actuator. Can also be used to refer to the period of time of error caused by a situation.
Situation Pre-empting	The concept by which the system starts to move in the direction of a situation before the situation occurs, producing a predictive effect.
Total Error	The combined sum of errors that exist before and after a situation, bounded by the actual position response and the desired position response.
Weight Smoothing	The concept that attempts to reduce spikes in the CMAC output by penalising weights that deviate excessively from the average of the weights associated with a single excited input vector.
Xenomai	An open source real-time kernel which runs along side a Linux kernel, providing real-time functionality to the Linux environment.

1.0 INTRODUCTION

This thesis is intended to be the initial step in a project involving the creation of an adaptive robotic humanoid hand to learn and perform deafblind sign language. This project involves several key design areas, such as the design of the physical hand itself, a means of capturing physical hand signs performed by a human teacher, and a means of training and controlling the robotic hand.

Without understanding a project in its entirety, it is easy to develop part of it without considering the integration factors required to integrate it to the rest of the project. For this reason, being the initial step in a project of such a large scope, a wide variety of literature has been studied, ranging from deafblind hand signing languages to human hand anatomy to biologically inspired adaptive controllers.

This wide variety of literature has been studied to define the overall project scope, to understand the individual goals of the project and how each aspect of the problem fits together with the next, spanning a wide spectrum of scientific fields and disciplines. This was required to narrow down the final specific problem for this thesis and to provide a wider knowledge base to build the entire project on.

The overall goal of the entire project is to allow a user to type a sentence into a computer and have the robotic hand sign out the equivalent hand gestures. The focus of this thesis, and thus the new work presented, is in the adaptive control side of the project.

1.1 CONTRIBUTIONS OF THE THESIS

The main contributions of the thesis include the following.

1. An improved method of training Cerebellar Model Articulation Controllers (CMACs) was developed, reducing the error in terms of both position and time for situations created when the desired trajectory velocity requirements exceeded that of the driven actuator's capabilities. The Error Minimising Gradient Controller (EMGC) is able to replace desired trajectory situations with EMGs in the CMAC's

training signal, and thus adaptively minimise the sum of the magnitudes of the position error over time.

2. The EMGC model was tested in real-time on hardware connected to a non-linear actuator for a variety of situations. The test results illustrated that depending on the inherent level of Situation Pre-empting achieved by the base CMAC controller, the EMGC can either reduce or increase a situation's Total Error. From the tests, metrics were obtained to allow the EMGC to decide whether a situation's Total Error was able to be reduced by use of the EMGC, allowing the EMGC to guarantee no degradation in CMAC controller performance.

3. A dynamic and clean method was devised for storing data in a CMAC, using binary search trees. This method was compared to the commonly used CMAC memory storage method of hashing, with test results illustrating that BSTs are a suitably fast alternative to hashing for small to medium sized CMAC weight tables.

1.2 THESIS CONTENT

While the depth of this thesis will not be involved in the complete creation of a humanoid deafblind signing hand, the overall scope in such a project is still needed to be understood to understand the basis and constraints on which to develop the controller. The problem to be tackled here specifically involves the creation of a suitable controller which will link to each part of the overall humanoid hand concept. Figure 1.1 illustrates the links between the controller and the various aspects of the overall humanoid hand creation concept.

As Figure 1.1 illustrates, it is the controller which lies at the heart of the overall system, and especially as the subject of this thesis. It links the various aspects of the overall project together, controlling the physical outputs to match the physical inputs, with a level of control suitable to learn and perform deafblind sign language on a humanoid robotic hand.

Thus initially each of these aspects must be researched, providing the basis and constraints for the developed controller. To present the achieved work, the thesis will be presented over the following chapters:

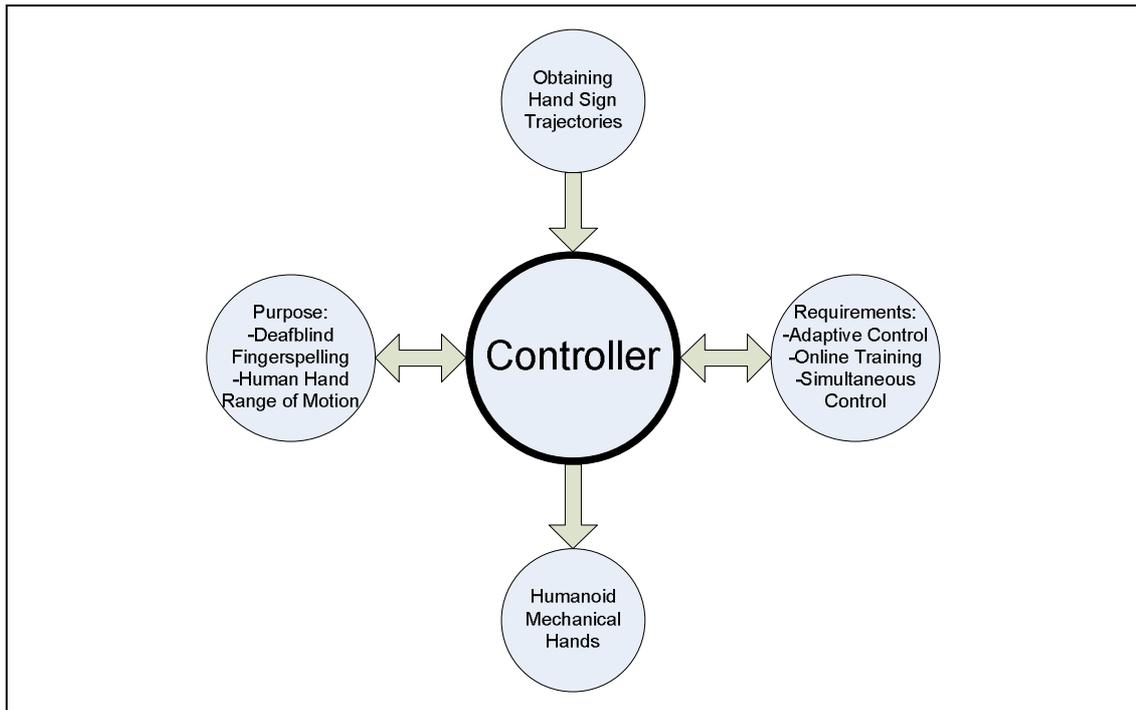


Figure 1.1 – Aspects of the Human Hand Controller for Learning and Performing Deafblind Hand Signs

Chapter 2.0 Literature Review. Provides a wide spectrum literature review for the entire scope of the project, with emphasis on the Cerebellar Model Articulation Controller (CMAC). This chapter provides the previous ground work from which the work in this thesis builds on. It combines several fields of research into a single document, illustrating the variety of disciplines a project like this brings together.

Chapter 3.0 Dynamic and Static CMAC Memory Allocation. Provides a description of how static memory allocation in the CMAC has been achieved previously, together with a custom dynamic memory algorithm structure using binary search trees for the CMAC.

Chapter 4.0 Comparisons between BST and Hashing with the CMAC. Provides a graphically presented comparison and discussion of the static memory allocation method of hashing against the new dynamic binary search tree implementation for CMAC.

Chapter 5.0 CMAC+PID Controller. Discusses the feedback/feedforward control algorithm using a CMAC and PID controller. This chapter specifically deals with the

use of the CMAC in this project, with all important CMAC parameters discussed, and how both the feedback and the adaptive feedforward parts work in synergy.

Chapter 6.0 Theoretical EMGC Model. Discusses the new model to train the CMAC+PID controller, and the purpose behind it with relevance to the use of joining deafblind hand signs. The workings of the internal model are presented in detail.

Chapter 7.0 Testing Setup. Discusses the software used in making the theoretical controller possible for testing on the electronic and pneumatic hardware. This involves the real-time operating system framework developed with the CMAC using Xenomai, together with the software architecture used to interface the real-time code to the custom made Linux GUI.

Chapter 8.0 EMGC Initial Square Wave Tests. Presents the initial physical tests conducted to better understand important design parameters used in the EMGC. These parameters are then used in further physical testing of the controller in a variety of dynamic situations to demonstrate the error minimising capabilities of the controller compared to when the EMGC is not used.

Chapter 9.0 Approach and Departure Gradient Tests. Presents a series of tests which simulate the use of the EMGC with situations that would occur in the joining of deafblind hand signs. Performance of the EMGC is discussed and compared against results obtained without the EMGC.

Chapter 10.0 Conclusions. Presents the conclusions and contributions of the thesis and discusses further work that can stem from the work conducted here.

2.0 LITERATURE REVIEW

The first step in the literature review is to present the underlying issue behind the motivation for the thesis which is that of Deafblind Communication. This topic is followed by a brief investigation of the complexities of the human hand, to provide an understanding as to what range of motion a controller will have to control. This will then lead into previously built deafblind fingerspelling hands, followed by a review of how hand motion trajectories are captured. This will be followed by a look at the type of trajectories which need to be reproduced on an artificial hand, leading into a study of biologically inspired controllers, with specific depth given to the Cerebellar Model Articulation Controller (CMAC). Finally a look at what the controller is required to achieve in terms of trajectory control will be presented, followed by a plan of how this is to be accomplished.

2.1 DEAFBLIND COMMUNICATION

95% of what we learn about ourselves and the world is obtained through our sight and hearing (SENSE, 2005). Vision and hearing are complementary senses which enhance each other. Often when it is difficult to hear, visual cues supplement our understanding. One example of this is body language cues in association to what is being verbally spoken, where only 7% of what is communicated in a face to face conversation is through words (Mehrabian, 1972). 55% is communicated through body language and facial expression, and the remaining 38% is communicated through voice quality. When vision is poor, hearing plays a major role in the localisation of sounds.

Touch, taste and smell all require the individual to be in direct contact with what they are sensing. Vision and hearing are the only two out of the five senses humans possess that perceive at distance. Thus deprivation of both of these senses can leave an individual feeling increasingly withdrawn, depressed and isolated (Southern & Drescher, 2005).

For some individuals in our community, the deprivation of both sight and hearing is part of their reality. An individual who suffers from the loss of both vision and hearing, either through trauma, diseases or inherited syndromes, is termed

‘deafblind’ (Moller, 2003). People who are deafblind face enormous challenges, especially in learning to communicate, accessing information and finding out about the world around them.

Deafblindness can occur at any time during an individual’s life, and for any number of different reasons. The loss of the senses due to birth defects, disease or old age are all reasons that can cause deafblindness to occur. Depending on when an individual becomes deafblind, their chosen form of communication will be vastly a product of what they previously used to communicate, and their level of deafblindness.

A national survey conducted in the USA in 2003 indicated that there are approximately 10,000 deafblind individuals between the ages from birth to 21 (TRI, 2003). In total however, the estimated number of deafblind individuals is about 70,000, but note that many more deafblind people exist than have been officially recorded (DBI.org, 2005). A survey conducted in the UK in 2004 by Southern and Dreshcher (2005) indicate that there are about 23,000 people who are deafblind living in the UK, with the majority over the age of 80. In addition, another 250,000 individuals are gradually losing their hearing and sight. The majority of those are also over the age of 80. Not all the deafblind individuals surveyed have complete hearing and visual loss. Many have some slight degree of either or both senses, but little enough to be pronounced deafblind. Depending on the degree of visual and auditory acuity and the period in life when an individual losses their vision and hearing, a number of different communication methods have been developed to help these individuals communicate with the outside world.

2.2 DEAFBLIND COMMUNICATION METHODS

Communication is vital for survival in the world today. For a deafblind person, communication is an issue that can be very difficult and frustrating, leaving only three fully functioning senses out of the five available, none of which act at distance. However, a number of different ways exist that deafblind individuals use to communicate with each other and with those that are not deafblind. Communication levels also depend on the degree of functional visual and auditory sensory capability the individual possesses in addition to when the individual became deafblind. For example, a person who developed deafblindness later in their life will most likely

retain their communication methods rather than learn new methods, such as those taught to individuals born deafblind (SENSE, 2005).

Southern and Drescher's 2004 survey (2005) undertaken in the UK provides a summary of the forms of communication deafblind individuals use together with their popularity. An adapted version of the results is included in Table 2.1. Note that some of the 326 respondents use more than one form of main communication, and their preferences appear multiple times.

The communication methods of Spoken English and Standard and Large Print in Table 2.1 are the same as that used by non-deafblind individuals. British Sign Language (BSL) is a two-handed form of visual sign language used for communication with deaf people.

Chosen Method of Communication	No. of Respondents
Spoken English	116
Large print	185
Standard print	47
British Sign Language (BSL)	34
Braille	26
Deafblind Manual	29
Hands-on-Signing	26

Table 2.1 – Deafblind Individual's Preferred Methods of Communication [adapted from (Southern & Drescher, 2005)]

Braille is used for blind individuals whereby the individual will read text with their fingers, represented as a series of embossed dots (IBRS, 1997). Deafblind Manual and Hands-on-Signing are both tactile forms of conversational communication whereby the deafblind individual will feel the shape or hand placement made by the hands of another individual. This will be explained in more depth in later sections. The communication methods used and their reasons will now be discussed.

2.2.1 PARTIALLY DEAF, PARTIALLY BLIND

Those individuals who have retained some vision and hearing often continue to use their previous forms of communication with the addition of an aid. For those who still possess both visual and hearing abilities, speech and print are often the preferred forms of communication as these methods have been previously learnt and thus are

the easiest to use. To aid hearing, some kind of hearing aid device is usually worn by the deafblind individual, and speech will be used in reply. To aid vision, a visual aid such as a strong magnifying glass together with prescription spectacles may be used to assist in reading standard or large print and to assist with writing.

2.2.2 PARTIALLY DEAF, COMPLETELY BLIND

Those individuals who have retained some degree of hearing but have completely lost all sense of vision often use a combination of spoken English together with Braille. With the assistance of a hearing aid, spoken English can be used for verbal communication. An individual who is completely blind may use Braille to read text, and often if the individual was blind before they started to lose their hearing, then they would already be accustomed to reading Braille. A combination of these two are the most likely forms of communication for a person who is partially deaf and completely blind.

2.2.3 COMPLETELY DEAF, PARTIALLY BLIND

Those individuals who have retained some degree of sight but have completely lost all sense of hearing often use a combination of sign language together with standard or large print. With the assistance of a visual aid such as a strong magnifying glass, print may still be able to be read. In addition, an individual who was previously deaf and lost their sight later in life would commonly know sign language for interpersonal communication. Thus a sign language such as BSL could still be used, and often is performed up close in front of the deafblind individual's face where he/she can see the signs. The deafblind individual can then perform their own signs to communicate back. A combination of print and sign language are the most likely forms of communication for a person who is partially deaf and completely blind.

2.2.4 COMPLETELY DEAF, COMPLETELY BLIND (IN LATER LIFE)

Those that lose their hearing and vision later in life often find it easier to continue to use one of their previous forms of communication. In most cases, either the loss of vision or hearing starts to gradually deteriorate. For those who lose their sight first, hearing is primarily used however Braille may be learnt so that text can still be read. For those that lose their hearing first, sign language may be learnt for interpersonal communication. Depending on which sensory capability was lost first, the chosen

form of communication, or that which comes easiest, will be used primarily as their major form of communication. Braille will often continue to be used by those that have learnt it. However, those individuals who learnt sign language as a result of losing their hearing, and then later lost their sight, will be more likely not to have learned Braille. However their knowledge of sign language can still be used for communication.

The communication form called 'Hands-on-Signing' is where a person performs a hand sign language such as BSL and the deafblind individual puts his/her hands over the signer's hands to feel the signs being made. Many adapt to hands-on-signing when they lose vision if for example BSL is their first language (O'Malley, 2005). Hands-on-Signing is basically a standard sign language communicated through the sense of touch instead of vision.

Hands-on-Signing can also be used with fingerspelling. Fingerspelling is much the same as sign language, however instead of making hand signs to represent words, the individual letters of the alphabet are signed out.

Different forms of both sign languages and fingerspelling exist for different spoken languages. For the English language there is British Sign Language (BSL), American Sign Language (ASL) and Australian Sign Language (AUSLAN). BSL, ASL and AUSLAN each have their own form of alphabet hand signs or fingerspellings. BSL and AUSLAN produce two-handed alphabet fingerspellings and ASL produces single-hand fingerspellings.

Due to the fact that deafblindness occurs in the majority of people over the age of 80, usually either Braille or Hands-on-Signing is used as the main form of communication for individuals who have completely lost vision and hearing later in life.

2.2.5 COMPLETELY DEAF, COMPLETELY BLIND (IN EARLY LIFE)

This category includes those that are born completely deafblind and those that have not had the chance to learn another language or form of communication in life, most often babies and young children. For these individuals specifically, a different form

of communication exists called 'Deafblind Manual'. Figure 2.1 shows the different hand gestures used to make up the Deafblind Manual.

Much like fingerspelling, Deafblind Manual involves the individual spelling out of individual letters of the alphabet (Scott, 1998). It is an adapted form of the two-handed fingerspelling alphabet taken from BSL. In the Deafblind Manual, the majority of time the signer's right hand will touch the receiver's left hand fingers and palm in certain places to represent the different letters of the alphabet (DeafBlind_UK, 2005). This way the deafblind individual feels where the 'speaker' is touching their hand, and each different touch represents a different letter.



Figure 2.1 – The Deafblind Manual Alphabet Fingerspelling Signs (DBNZ, 2006)

The number of deafblind individuals who use spoken English and read print represent the majority of the deafblind community, with the majority of deafblind individuals over the age of 80. Many still have some degree of visual and/or auditory acuity remaining. BSL and Braille are both forms of communication that can be represented visually as in the case of BSL, or via a series of embossed dots as in the case of Braille. However, Deafblind Manual and Hands-on-Signing can only be performed via the direct touching of a human hand between the ‘talker’ and the ‘listener’. These two forms of communication cannot be performed any other way. From Southern and Dreshcher’s survey (2005), at least 8% of the deafblind community depend on this form of communication for interpersonal communication. Assuming that these figures are representative values for the whole international deafblind community, then 8% of the UK and USA deafblind population account for 1840 and 5600 individuals respectively, all of which rely solely on a direct hands-on form of interpersonal communication. And in every case, since a physical hand is required, a human must be present for communication with these individuals.

2.2.6 ARTIFICIAL HAND PURPOSE

The requirement of having another individual around at all times places a burden on the deafblind individual and their assistant, their family and friends. If there is no interpreter to communicate with a deafblind individual who only knows Deafblind Manual or hands-on-signing, it can cause a serious problem.

This requirement, relying on a human to provide a means of communication between the individual and the outside world, could be relaxed if technology could in some way help. Communication technology available these days helps individuals access information in a vast number of ways. For a deafblind person however, many of the communication devices available are not useable. A mobile phone, the television or a home computer do not provide the communication means required by a deafblind individual.

For those deafblind individuals who require a human to be present as their mediator with the outside world, their reliant communication sense is that of touch via deafblind sign language. It is understandable why a human is required, as both the ‘listening’ and ‘talking’ are communicated via the sense of touch through the hands.

Thus it would be highly desirable for a deafblind individual if there was a device which could mediate between themselves and the outside world, without the reliance on a human mediator.

An artificial hand, with the sensory requirements to be able to perform deafblind sign language, would be such a device that would allow a deafblind individual to communicate without the use of a human translator. Such a hand however would need several similarities to a human hand, being highly dexterous and tactile, with a high degree of delicate motion control.

Such a device could act in place of a human translator and could both 'hear' and 'talk' to the deafblind individual, translating typed text from a computer keyboard to deafblind signings. Interfacing an artificial hand to a device such as a computer would allow any individual that can type text into a keyboard to communicate to a deafblind individual without the need to learn a signing language. It would also allow remote communication to be possible through the use of media such as the internet.

The controller for a hand that could perform deafblind sign language would have to learn deafblind sign language initially, and be able to perform accurately the signs on a robotic humanoid hand. It is this controller which this thesis will develop and test, to produce a suitable controller capable for use with a robotic hand for the purpose of learning and performing deafblind sign language.

2.3 THE HUMAN HAND MODEL

The human hand is a remarkable piece of mechanical and electrical engineering. Consisting of 25 degrees of freedom when the hand is allowed to move freely (Tubiana, 1981), the level of dexterity, high strength to weight ratio, combined with the complex sensory feedback, make the human hand far more advanced than any man made humanoid hand design to date. Figure 2.2 illustrates the medical notation for the bones and joints in the skeletal structure of the human hand.

The arrangement of bones and joints in the human hand produce the many degrees of freedom and thus the high dexterity property that the hand possesses. The degrees of

freedom that the hand and forearm possess are combined together to produce the 25 degrees of freedom of the human hand, as outlined in Table 2.2.

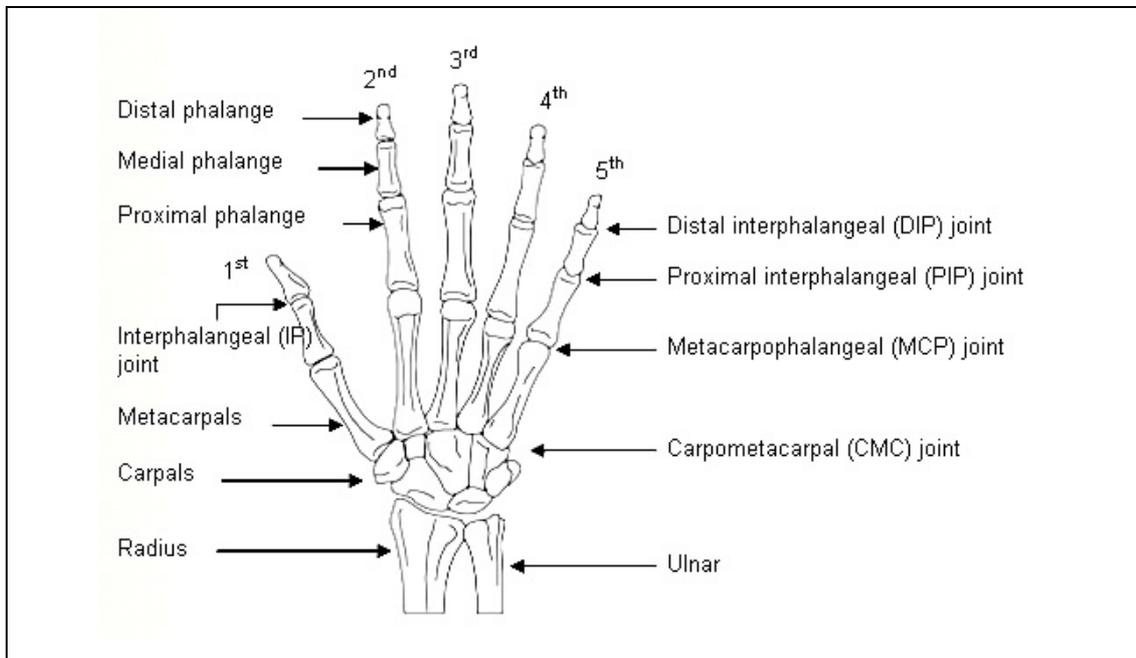


Figure 2.2 – The Bones and Joints in a Human Hand (Shim, 2004)

JOINT	Flexion/ Extension	Hyper- Extension	Abduction/ Adduction	Other	DOF Per Joint	Total DOF
Finger DIP	Yes ROM:90°	No	No	No	1 x 4 Fingers	4
Finger PIP	Yes ROM:90°	No	No	No	1 x 4 Fingers	4
Finger MCP	Yes ROM:90°	Yes ROM:20°	Yes ROM:30°	No	2 x 4 Fingers	8
Thumb IP	Yes ROM:90°	Yes ROM:15°	No	No	1	1
Thumb MCP	Yes ROM:55°	Yes ROM:10°	No	No	1	1
Thumb CMC	Yes ROM:50°	Yes ROM:10°	Yes ROM:45°	Yes Slight Rotation of Thumb	3	3
Wrist	Yes ROM:75°	Yes ROM:70°	No	Yes Radia:20°/Ulnar:35° (deviation)	2	2
Forearm	No	No	No	Yes Supination:85°/ Pronation:70°	1	1
Elbow	Yes ROM:145°	No	No	No	1	1
Total DOF	17 (Same DOF)		5	3		25

Table 2.2 – Degrees of Freedom of the Human Hand [adapted from (Li, 2003; Shim, 2004)]

Note: The Range of Motions (ROM) are approximates and vary depending on the flexibility of the subject.

The complexity of the human hand has provided a substantial challenge for those that have tried to replicate it. Several humanoid hands exist to date, built for their own specific purpose with a variety of actuation methods. Appendix A provides an overview of various hands built to date. The focus here however involves hands built for deafblind sign language, and therefore previous mechanical hands built for this specific purpose will now be discussed.

2.4 DEAFBLIND FINGERSPELLING HANDS

Over the past three decades, there have been attempts made at creating fingerspelling hands to communicate with deafblind people. However the research that has been undertaken in this area remains narrow and the majority of work found to date is presented below. The programming of such devices has been restricted to only ASL fingerspelling, and the hands' degrees of freedom limited.

As early as 1978, the South-West Research Institute (SWRI) designed and built a mechanical fingerspelling hand (Laenger & Peel, 1978). Though primitive in design, the hand enabled a deafblind person to receive communications from anyone who could use a keyboard to type in a word. The word was then converted into a series of fingerspelling hand gestures by the mechanical hand.

In 1985, the Rehabilitation Engineering Centre of the Smith-Kettlewell Eye Research Foundation designed and fabricated an improved fingerspelling hand. It addressed major issues the Laenger & Peel hand had, by improving timing and allowing finger positions to be modified easily. These qualities were realised in a new robotic fingerspelling hand named "Dexter" (Jaffe, 1994).

In 1989 a new and improved version of Dexter, named Dexter-II was designed and built by three Stanford graduate mechanical engineering students (Jaffe, 1989). The hand could produce four sign letters per second, twice the speed of the original Dexter. The hand was also substantially smaller, the mechanical drive system being approximately 1/10 the size of the original Dexter.

The latest design of fingerspelling hands developed by Jaffe was the fingerspelling hand, RALPH (Robotic ALPHabet), built in 1994. RALPH uses 8 servo motors to

actuate the entire hand and reproduce all the letters of the American One-Hand Manual Alphabet (Jaffe, 1994). RALPH is faster, more accurate and more compact than all previous designs, including Dexter II, and is about the size of a female adult hand (excluding the mechanical base containing the servo motors). RALPH is shown in Figure 2.3.

The requirement to know fingerspelling is not always possible when faced with the task of communicating with a deafblind individual who only can communicate via the use of fingerspelling.



Figure 2.3 – RALPH Finger-Spelling Hand (Jaffe, 1994)

Furthermore, the fingerspelling language that a deafblind individual knows may be different from that known by another individual wanting to communicate. For example, Deafblind Manual and ASL Hands-on-Signing. Previous fingerspelling hands have been restricted by only knowing ASL. A universal fingerspelling translator that knows multiple forms of fingerspelling would therefore be very useful to those deafblind individuals that rely on Deafblind Manual and/or Hands-on-Signing.

The large range of motions required to perform fingerspelling and sign language is best performed on a fully dexterous hand. This is a language communicated via motions of the hand, not simply an end-effector tool for manipulating objects. Thus

every single degree of freedom possessed by the human hand should be incorporated into the design of such a project. A fully dexterous design allows the hand to be used not only with the various forms of fingerspelling languages available, but also allows the possibility of performing the various languages of deaf signing if a pair of hands is built without worrying about dexterity limitations.

One way of reproducing humanlike motion on a humanoid hand is by capturing and mimicking human hand gestures. Recreating dexterous human hand motion requires an apparatus which is capable of capturing each degree of freedom in the human hand.

This therefore leads into the area of hand motion data capture. Capturing the motion of a human hand, recording it, and then presenting it to an adaptive controller to learn from, provides a natural based approach to training and replicating human motion on mechatronic systems. The following section will discuss such motion capture methods.

2.5 TRAJECTORY CAPTURING – DATA GLOVES

Biological systems, both neural and kinesiological, not only in advanced creatures such as humans, but also in lower forms of animal life such as insects, have yet to be replicated to even a fraction of their biological capabilities (Albus & Meystel, 2001). The capability of biological mechanical systems and their motion generation involves many complex and advanced processes. Research into the neural processes behind the efficient generation of natural motions (Lee et al., 2005; Lim et al., 2005) is one area that is currently being studied. However for particular systems, the direct mimicking of recorded biological motions proves a simple and feasible option. Capturing human body motion, facial expressions, and hand motion with the use of the various motion capture technologies available, provides a natural motion basis which can then be replicated mechatronically. The capturing and recording of natural motion data provides a suitable trajectory map for a controller to mimic.

Several technologies exist to capture human motion. Optical tracking methods include using marker dots on certain parts of the body, silhouette analysis, and 2-D and 3-D visual analysis using multiple cameras (Sturman et al., 1994). Such methods however require direct line of sight and substantial amounts of data cleansing,

resulting in additional editing time. Sensors placed directly on the body such as resistive or optical bend sensors and gyroscopes, are more intrusive to wear, though are not effected by line-of-sight occlusions and provide clean accurate data which requires minimal data cleansing (Animazoo, 2004a). Resistive bend sensors are thin resistive strips, composed of tiny patches of carbon that change resistance when bent from convex to concave shapes. Figure 2.4 illustrates this concept, and Figure 2.5 illustrates how bend sensors can be applied to capture finger motion when embedded on a glove.

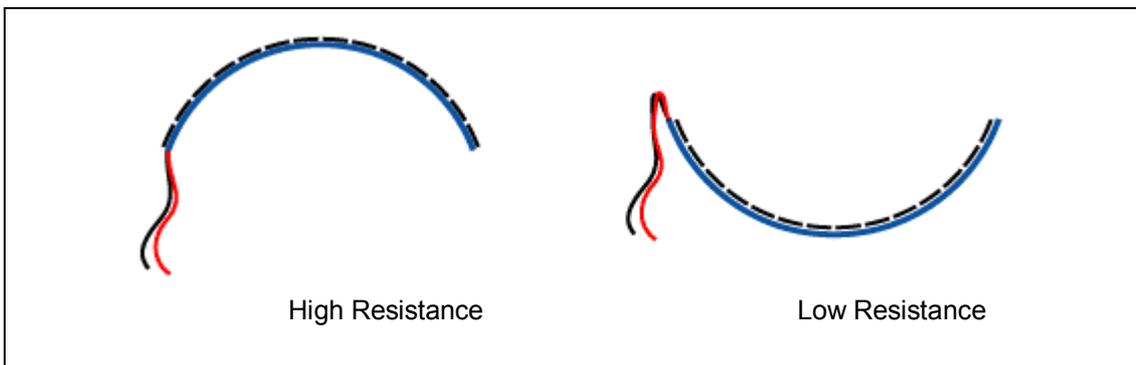


Figure 2.4 – Resistive Bend Sensor Functionality (TELEO, 2004)



Figure 2.5 – Resistive Bend Sensors Embedded on Fingers of a Glove (Kriete, 2004)

Modern gyroscopes also provide a direct motion measurement form of data capture, and are more robust over bend sensors as they will not wear or tear. Figure 2.6

demonstrates the size advantage of modern gyroscopes. This single-axis gyroscope measures only 7x7mm. The recent miniaturisation of gyroscopic technology has provided the versatility of suit based systems with very smooth, accurate motion data, providing very life-like captures. The GypsyGyro-18 is one such miniature-gyroscope-based full body suit providing less than 0.1degree resolution per sensor at up to 120 frames per second (Animazoo, 2004b).

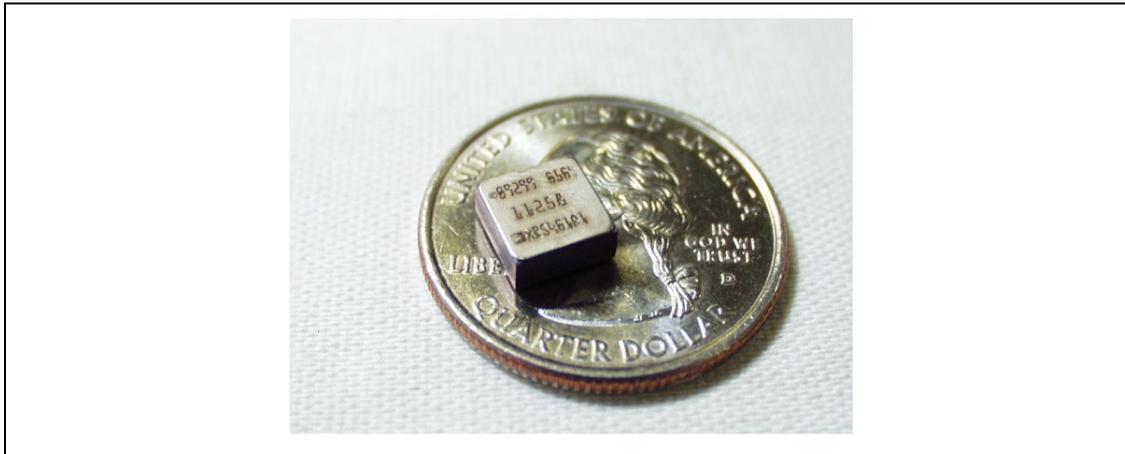


Figure 2.6 – Modern Single-Axis Gyroscope Sitting on USA Quarter Dollar Coin (SparkFun, 2005)

For hand-specific motion capture however, technologies such as direct visual analysis and/or bend sensors are often used, incorporated into a device worn on the hand – most commonly referred to as a ‘data glove’. Several commercial data gloves exist that are used by virtual reality researchers, those involved in humanoid hand teleoperation and those involved in motion capture analysis. Commercial data gloves include the CyberGlove (Immersion, 2005), the 5DT data-glove (5DT, 2005) and the P5 glove (Virtual_Realities, 2006). Most popular in various teleoperation humanoid hand applications appears to be the CyberGlove, providing data capture totally from resistive bend sensors, and capturing 22 degrees of freedom (Immersion, 2005). The price of most commercial gloves ranges from several thousands of dollars (AUS) for the 5DT data-glove to tens of thousands for the CyberGlove. A comprehensive review of the CyberGlove was performed by Kessler and Hodges et al. (1995). Note that an upgraded glove (CyberGlove II) has since been released.

At the lower cost end of the spectrum exists the P5 Data Glove, mainly used in the video game industry to increase the level of computer game interactivity. The

advantage of this glove is that its position in 3-D space and the pitch, roll and yaw of the hand are all measured, a feature which does not come standard with the gloves mentioned above. For computer games, the data capture resolution is adequate, but modifications would have to be made to use this glove for precision motion capture. The price tag for the P5 glove is below US\$100 (Virtual_Realities, 2006). For a more detailed overview of the different types of technologies data gloves use for joint and hand position tracking and the different gloves available, Sturman and Zeltzer's paper (1994) though somewhat dated, provides a useful reference.

While the argument not to use physical means like data gloves to capture hand motion/gestures and instead use vision based methods was made strong by Pavlovic et al. (1997), the ability to track and provide precise and especially consistent results without the means of editing captured data still remains inferior to physical data glove devices (Turner et al., 2000). Background noise, lighting, shadows, distance from vision sensors (cameras), occlusions, camera resolution, multiple capture angles, etc., all have to be accounted for when using a vision based system. This is made even more difficult if a natural hand is used without any reference markings. For this reason, popularity amongst researchers has resulted in the CyberGlove being a prominent choice as the hand motion capture device for many research projects.

Teleoperation is one common use for data gloves. Where a system has to be controlled via the use of hand and/or finger gestures, a data glove can capture the required gestures and either via the internet or another means of data transportation, the hand gestures can be recreated. Often combined with a vision system for visual feedback, the teleoperator can see and control a device in another location. The NASA/DARPA Robonaut project are one such team that have created a dexterous humanoid hand that is controlled by an individual with telepresence gear, consisting of a pair of CyberGloves and a virtual reality head piece (for vision) (Diftler et al., 2003; Goza et al., 2004). Figure 2.7(a) shows the Robonaut being controlled via the human operator in Figure 2.7(b) via the telepresence gear, mimicking in real-time the gestures the operator produces.

In addition to the NASA/DARPA Robonaut humanoid hands, the HIT/DLR hand project has also used a CyberGlove to provide teleoperation hand control (Hu et al.,

2004). Remote internet operated hands have also been investigated with custom designed data gloves and a dexterous robotic arm by You and Wang et al. (2001). In industry, telemanipulation of an industrial robot consisting of a 4 DOF hand-like end-effector attached to the end of an industrial robot arm was controlled using a CyberGlove to stack objects, amongst other applications (Turner et al., 2000).

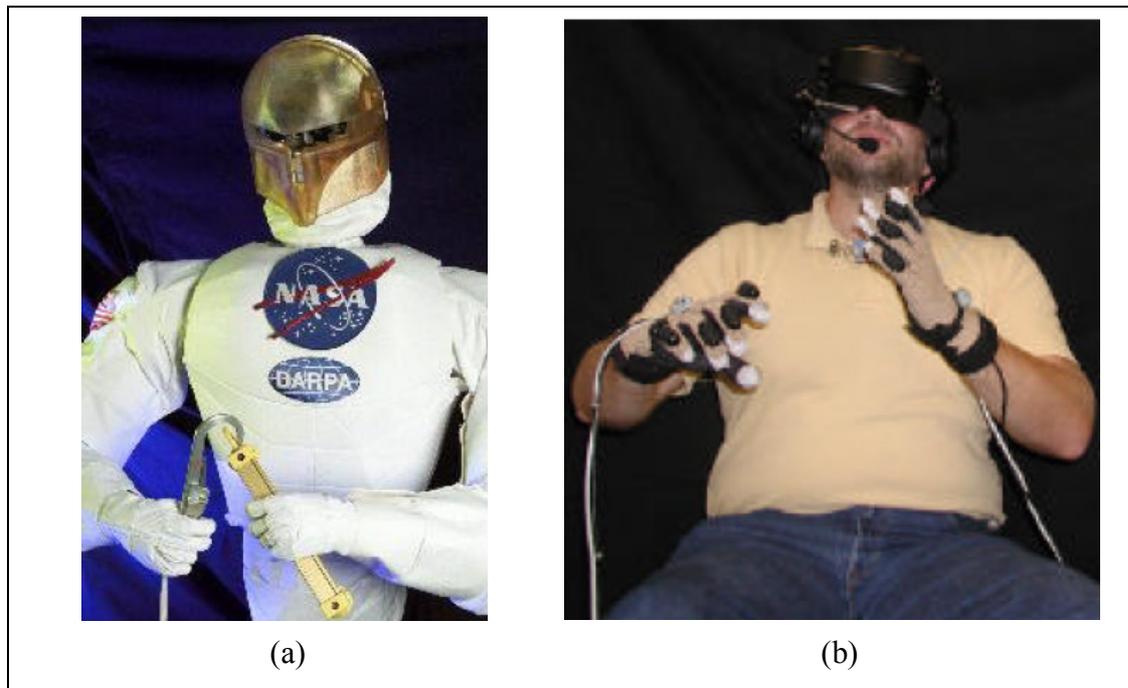


Figure 2.7 – NASA/DARPA Robonaut Teleoperated System (Diftler et al., 2003)

Gesture recognition is another field of research where data capture gloves are used extensively, in uses such as hand-grasp detection and sign language fingerspelling. Programming by demonstration using a Hidden Markov Model (HMM)/trajectory-based hybrid method was used by Ekvall and Kragic (2005) to learn 15 different types of hand grasps. Bernardin and Ogawara et al. (2003) have also used HMM with a CyberGlove fused with a tactile sensor array to learn 14 different grasps, with 92.2% accuracy for a single user system.

Sign language specific recognition using data gloves has been attempted for the various forms of international sign languages and fingerspellings. The CyberGlove was used by Allen and Asselin et al. (2003) along with the Matlab neural network toolbox to learn the American Sign Language Fingerspellings with 90% recognition accuracy for a single user system (i.e. can recognise 90% of the time gestures

performed by the same person who trained the artificial neural network). Similar attempts using data gloves for ASL recognition have been made by Liang and Ouhyoung (1995). A gesture recognition system for the Korean Sign Language using a pair of VPL Data-Gloves (VPL_Research_Inc., 1992) cited in (Kim et al., 1996) was developed using a fuzzy min-max neural network. Similarly, a gesture recognition system for the Japanese Sign Language was developed, with the ability to recognise 42 symbols with 98% recognition rate for registered people, using a 13 DOF data glove (Murakami & Taguchi, 1991). Another system was also designed by Kadous (1995) for the recognition of Australian Sign Language (AUSLAN) using 95 different signs with an 80% recognition accuracy.

Other systems involve using data gloves for finger positioning together with a stereo vision system for hand depth and space positioning (Ogawara et al., 2001; Ogawara et al., 2000) and in the assessment of rehabilitative treatments in assessing patients' functional impairments (Dipietro et al., 2003). The virtual conduction of a synthetic orchestra (Morita et al., 1991), and dynamic signature verification (Tolba, 1999) are some other novel ideas utilising data gloves.

Data gloves provide the direct mapping of finger and hand joint trajectories to be recorded in real-time, allowing research to be conducted in many fields requiring detailed hand motion data. Such a device provides a very useful learning tool to train a controller to mimic finger joint trajectories on a dexterous humanoid hand. The use of such captured trajectories for deafblind fingerspelling on a humanoid hand will now be discussed.

2.6 CONCATENATING CAPTURED TRAJECTORIES

In the context of deafblind fingerspelling, for each letter in the deafblind manual alphabet, there exists an array of trajectories which can be captured by the use of a data glove. Each letter motion would be captured independently, producing a series of captured trajectory data to be mimicked by a controller on a humanoid hand.

For each degree of freedom captured, the position trajectory over time will need to be reproduced on the humanoid hand, replicating the trajectory as accurately and smoothly as possible to mimic the original human motion.

In addition, each individual hand sign needs to be concatenated to the next to create complete words, which finally produce entire sentences. Figure 2.8 illustrates this concept, whereby three individual hand sign trajectories are concatenated to produce the word ‘eat’.

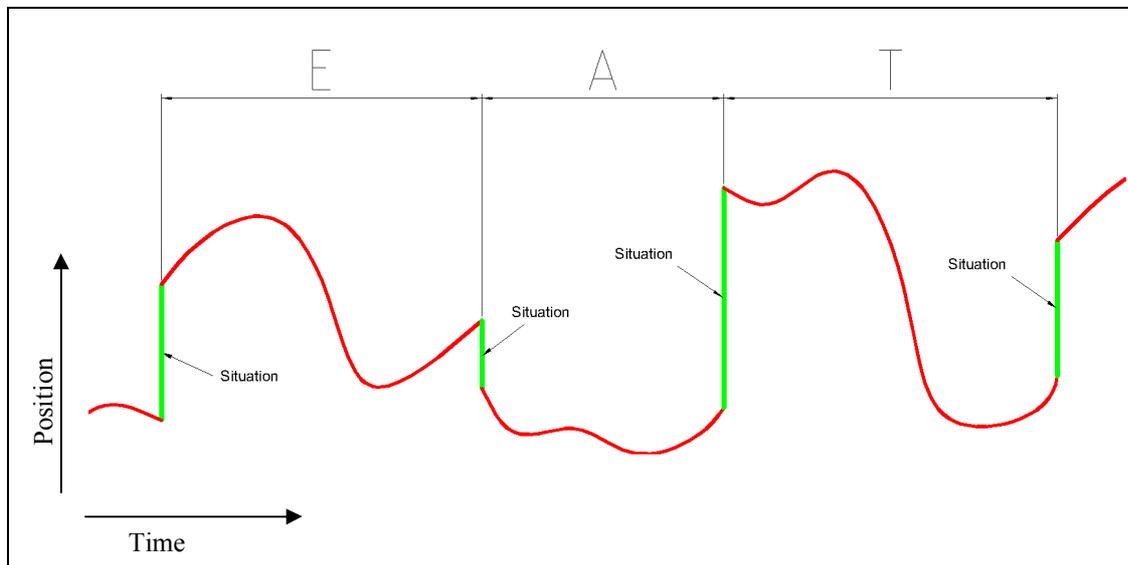


Figure 2.8 – The Joining of Hand Sign Trajectories, Separated by ‘Situations’.

Figure 2.8 illustrates a sequence of trajectories for a single degree of freedom that the controller will be presented with. The controller will have to reproduce this sequence as accurately and as smoothly as possible on the humanoid hand. Each smooth desired trajectory will naturally be separated by a ‘situation’, as illustrated in Figure 2.8 as a discontinuity from the end of one trajectory to the start of the next. Situations are explained in detail in Section 6.1.1.

As there is a very large number of possible ways to join individual letter trajectories to each other, ‘situations’ are used, and are simply a discontinuity from the end of one trajectory to the start of the next. Situations here have an infinite desired velocity, and as such cannot be reproduced exactly on a physical system. In addition they would not want to be reproduced, as the instantaneous required movement would be sharply unnatural. On a system that is trying to replicate smooth human motion, this is highly undesirable.

The output system, namely the humanoid hand's degrees of freedom, wants to move from one trajectory to the next as quickly as possible, to minimise the risk in the loss of information transfer. For example, if the controlled actuator moves a finger too slowly from one trajectory to the next, loss of sign lingual information will occur, and the receiver of the hand sign may be confused as to what the hand is trying to communicate. A controller which can replicate the captured human motion as accurately as possible is thus required to minimise the loss of language information communicated through an artificial humanoid hand.

In addition to the dexterity that such a hand must possess, the control method driving the actuators is vitally important. Classically, robotic behaviour has been somewhat rigid, performing quite the opposite of 'natural-like' behaviour. This is not only a result of the lack of mechanical dexterity capability of the system, but also a lack in natural-control performance by which the system is controlled. A system that can grow, learn and adapt in much the same way as our brain controls our muscles is necessary to control a dexterous hand to perform natural-like motion. Also, a way of teaching the controller hand-gestures based on the replication of human hand-gestures would also be beneficial to provide a natural-like replication of human fingerspelling gestures. The following section deals with such methods available to produce just that: an adaptive controller that can learn and mimic human hand gestures, including deafblind fingerspelling.

2.7 BIOLOGICALLY INSPIRED CONTROLLERS

The PID (Proportional, Integral, Derivative) controller is the most common form of feedback controller (Astrom, 2002). However for non-linear control systems, often an adaptive feedforward controller is also needed to adequately solve the control problem. For systems with a large degree of complex and/or non-linear behaviour, it is often not feasible to mathematically model the inverse behaviour of the system. The impracticality lies in the time required and the theoretical models available, where at best a non-linear system requires a constantly changing and dynamic model. Systems like this are therefore more easily solved with a controller that can adapt, evolve and learn its own optimal control output. The Artificial Neural Network (ANN) is one such adaptive evolution controller that is often used to control difficult and non-linear control systems.

2.7.1 ARTIFICIAL NEURAL NETWORKS (ANNs)

Artificial Neural Networks are an artificial intelligence technique that mimic a simplified neural process map of biological brains. The brain consists of billions of neurons which link together to form neural networks. One of the first attempts at modelling a single neuron was conducted by McCulloch and Pitts (1943). Negnevitsky's book 'Artificial Intelligence: A Guide to Intelligent Systems' (2002) provides an excellent overview.

Since 1943, however, several other artificial neural network models have been developed for a range of applications, including those to now be presented.

2.7.1.1 *Common Artificial Neural Networks*

Firstly, one of the most widely used and best understood of all the different neural networks is the concept of the Perceptron (Picton, 2000). The term was first used by Rosenblatt (1958) who implemented a neuron model in software as an algorithm. The model consists of a linear combiner followed by a hard limiter. The weighted sum of the inputs is applied to the hard limiter, which produces either +1 if the input is positive or -1 if the input is negative (Negnevitsky, 2002). A single perceptron however is most often combined together with other perceptrons in several layers to form the Multi-Layer Perceptron (MLP). The MLP was introduced by Rumelhart and McClelland (1986) to improve the capabilities of the single layer perceptron model, and is undoubtedly the most widely used neural network structure implemented in a vast number of applications and covering various scientific fields (Arena et al., 1998). MLPs are used for many function approximation and data modelling tasks because of their ability to generalise well with unseen data and learn non-linear functions (Tham, 1994). MLPs have been used successfully in a wide variety of applications, including handwriting classification (Liu & Gader, 2002), heart disease diagnosis (Yan et al., 2006), solar radiation estimation (Dorvlo et al., 2002; Hontoria et al., 2005) and a hysteresis compensation controller (Islam & Saha, 2005). However, while the MLP is a widely used and implemented neural network over a range of applications, other specific artificial neural networks have also been developed.

The Hopfield Network developed by Hopfield (1982; 1985), was designed as both a classifier and an optimiser (Lisboa, 1992). When trained with a set of patterns and then tested with noisy or incomplete similar patterns, the Hopfield network is able to classify each noisy pattern into its most likely class. When used in reverse, the Hopfield network becomes an optimiser and can provide optimal solutions to large and complex data sets (Lisboa, 1992). Applications where Hopfield networks have been used include pressurised water reactor management optimisation in nuclear reactors (Sadighi et al., 2002), optimising routing algorithms for communication networks (Venkataram et al., 2002), medical imaging shape matching (Banerjee & Majumdar, 2000) and incomplete survey data classification (Wang, 2005).

Kohonen Self-Organising-Maps (SOM) are an unsupervised artificial neural network designed by Kohonen (1990). No correct answer is presented to the network for the network to learn from. An SOM inspects the data as it is presented, and organises the data in such a way as to form an ordered description of the data onto a two-dimensional map. Most often the input data is not two-dimensional, but is of arbitrary dimensions (Tarassenko, 1998). This way input vectors that are close together in the original n-dimensional space will be neighbours in the 2-D array. Applications where Kohonen SOMs are being used include geographical classification of crude oils (Fonseca et al., 2006), classification of finite elements (Beltzer & Sato, 2003), non-linear process dynamic modelling (Alexandridis et al., 2002), vibration signal monitoring (Wong et al., 2006), and cancerous DNA gene analysis (Hsu et al., 2003).

Another form of neural network that was designed specifically to be used as a manipulator controller is the Cerebellar Model Articulation Controller (CMAC – pronounced ‘see-mac’). Designed in 1975 by Albus (1975a; 1975b), the CMAC was based on a simple model of the cerebellum, the part of the brain found in ‘higher-animals’ that enables high-level ‘fine-movement’ control of the muscles to be learnt. The similarity is rather superficial though, as many biological properties of the cerebellum are not modelled (Smith, 1998), yet enough are to produce a practical biologically inspired controller. The CMAC was originally designed by Albus to perform rote learning of movements of an artificial arm and has since been used extensively in a wide variety of systems from controllers to classifiers (see Section

2.7.3). The CMAC is able to learn the dynamic properties of a manipulator or actuator, and then respond to a desired input trajectory by producing the desired control response on the connected actuator. The more diverse the dynamic training data presented, the more skilled the CMAC will become. After several training iterations the CMAC can control the actuator in a similar way as humans control the muscles in their own body, with the level of control developing from a baby to an adult. The more it trains over a wide range of dynamic motion, the more coordinated it becomes at controlling the dynamic properties of the actuator to which it is connected.

2.7.1.2 Artificial Neural Network Training

The training methods for neural networks fall into two categories: supervised and unsupervised learning. The adaptive quality of neural networks is the result of the training or adjusting of weights which produce the required output. For each different input state, a different set of weights is trained to produce the corresponding output.

Supervised learning requires that the target output is already known. For a classifier ANN, this will simply be the pattern to be learnt. For example, if an MLP used as a classifier is learning to classify handwriting into its individual letters, then several versions of each individual letter of the alphabet will be presented to the MLP. If the output from the network differs from the desired output, for example the MLP classifies the shape of the letter 'a' as the letter 'e', then the network weights are modified in the direction to achieve the correct output. After several iterations the MLP will have learnt what the shape of the letter 'a' represents as opposed to the shape of the letter 'e'. This way the training of the MLP is supervised, as the MLP is presented with enough iterations of the individual letters until it can classify correctly by itself. Supervised learning is used with the MLP, Hopfield networks, and the CMAC.

Unsupervised learning is a form of learning that does not require the output to be previously known. During learning, an ANN receives a number of different input patterns and then automatically sorts these patterns into appropriate classes (Negnevitsky, 2002). Depending on the internal design of the ANN, the output will

be a product of the input patterns and the design of the network. Unsupervised learning is used in Kohonen SOMs.

The various types of artificial neural networks discussed have been used in applications ranging from classifiers, optimisers, and controllers. The range of ANNs developed to date covers a vast extent of techniques and designs. Miller and Glanz (1996) and Sarle (2002) both provide comprehensive lists that explain the various neural network models and their applications. The key models are now major classes within their own right, with newer models and modifications being designed based on the models mentioned previously.

The goal of this thesis however is to design a controller suitable to control a humanoid hand to mimic deafblind sign language. As presented, not every artificial neural network is suitable for control purposes, and only the CMAC and MLP neural networks have been used in control applications. A comparison between the CMAC and MLP for control specific applications will now be discussed.

2.7.2 WHICH ARTIFICIAL NEURAL NETWORK?

The CMAC and MLP are both used in control applications. One of the most important features of the CMAC over the MLP is its exceptional learning characteristic: the speed of learning may be much higher than that of a corresponding MLP (Szabo & Horvath, 1999). MLP training requires computation of all the weights in the network for each set of training data, whereas a CMAC only trains the weights that are directly associated with each training data sample (Tham, 1994). More precisely, MLP networks tend to be slow to train but are relatively quick to recall (Dwinnel, 1998). In addition an MLP requires many more iterations to converge when compared to a CMAC, and as a result is inappropriate for online real-time learning unless implemented on custom designed hardware (Miller et al., 1990a). CMACs require fewer iterations to converge and each individual iteration takes less time, which makes them suitable for fast online real-time learning. Also, the problem of 'temporal crosstalk' (Jacobs et al., 1990) found in MLP networks, where weight updates for input patterns far apart in input space affect the output values of other input patterns, can usually be avoided in a CMAC (Tham, 1994).

The CMAC's overall modelling capability however is inferior to that of an MLP (Brown et al., 1993). This problem has a lot to do with how the CMAC is trained (Horvath, 2004). The memory storage requirement of a CMAC with many input dimensions is also significantly greater than for an MLP (Tham, 1994), though there are effective ways of reducing the memory requirements (Albus, 1981; Hsu et al., 2002; Miller & Glanz, 1996).

The CMAC's many advantages as an adaptive actuator controller make it suitable to perform fast online real-time training for real-time actuator control. In addition, several others have been successfully using the CMAC in a variety of applications, some of which will now be presented.

2.7.3 USES AND ADAPTATIONS OF THE CMAC

Since Albus proposed the CMAC in 1975 (Albus, 1975a; Albus, 1975b), several research groups have been using the CMAC with considerable success. Many researchers have also modified the original purpose of the CMAC which was intended by Albus to be used as a robotic controller, to applications such as valve fault detection and diagnosis (Wang & Jiang, 2004) and classifiers (Cornforth, 2001). As such, the basic CMAC model in itself is an extremely diverse and useful concept which has also been modified in numerous ways (as will be shown) to perform a variety of tasks. Additionally, a CMAC requires minimal prior knowledge of the system and therefore can be applied as a generic adaptive controller for many different types of systems, including non-linear systems which are inherently difficult to control via conventional and classical methods (Abdelhameed et al., 2002).

CMAC uses and modifications include the Parametric-CMAC (P-CMAC) designed by Almeida (Almeida & Simoes, 2003) which adds parametric fuzzy components to Albus's original CMAC, used in the control of the output voltage of a proton exchange membrane fuel cell (Almeida & Simoes, 2005). Incorporating fuzzy logic into CMAC has also been used for ship steering applications (Shen et al., 2004; Shen et al., 2005) and for medical image diagnostics (Jian et al., 2005). Another modification includes Cornforth's and Horvath's modified Kernel CMACs (Cornforth, 2001; Horvath, 2003; Horvath, 2006), which both have expanded the use

of Albus's CMAC from robotic trajectory controller to data-mining classifiers. Cornforth has also modified the CMAC and proposed and tested a new training algorithm which results in a fast, effective classifier for large databases requiring only a single pass of the training data (as opposed to traditional classifiers using MLPs) (Cornforth, 2001). The University of New Hampshire have used the CMAC in real-time control of an industrial robot and in other applications, usually employing CMACs with thousands of adjustable weights that can be trained to approximate non-linearities which are most often not explicitly known (Miller et al., 1990b). Wang has successfully used a modified CMAC called a Recurrent-CMAC (RCMAC) in monitoring and diagnosing the degradation in the performance of heating/cooling coil valves by comparing degradation signals with the usual characteristics of healthy valves (Wang & Jiang, 2004). Control of a parallel hybrid-electric propulsion system for a small unmanned aerial vehicle is yet another example where a standard CMAC has been used (Harmon et al., 2005). A rotor position estimator for switched reluctance motors using a CMAC (Mese, 2003), a trajectory tracking piezoelectric actuated tool post CMAC controller (Abdelhameed et al., 2002), walking biped CMAC control both in simulation (Smith, 1998) and in hardware (Sabourin & Bruneau, 2005) and the dynamic CMAC control of a robot arm (Cembrano et al., 1997), are just a few of the many applications that CMACs have been used in. For a comprehensive list of CMAC uses and applications, Miller and Glanz's implementation of the CMAC paper provides a good list of CMAC uses prior to 1996 (Miller & Glanz, 1996) and themselves utilised the CMAC in a number of real-time applications (Miller et al., 1990a).

A detailed description of the basic CMAC neural network together with several useful additions to the basic architecture will now be presented.

2.8 THE CMAC NEURAL NETWORK

This section will explain how the CMAC operates, starting from basic principles to various modifications of the system. For a more comprehensive description, the book "Brains, Behavior and Robotics" (Albus, 1981) is recommended. The notation used here is consistent with that used by Albus.

The CMAC was designed based on the biological inspiration of the cerebellum, the part of the brain found in ‘higher animals’ that can learn and control fine and precise muscle movements. Playing the piano or even speaking are both muscle actuations in humans that require precise control over muscles, either individually or together, as a function of time. A human’s brain first plans the goal of a purpose, and then automatically plans a trajectory based on reaching this goal. The muscles then guide the limbs and joints to obtain the goal automatically, while the underlying control for each muscle occurs without us even thinking about it. It is the cerebellum that learns and produces the final output drive signals to each individual muscle. A solid biological explanation of the brain’s cerebellar and motor control complexity and behaviour is provided in (Albus, 1981; Houk et al., 1996; Smith, 1998), with special note given to Albus, as in his book he actually links the biological workings of the cerebellum into his mathematical model very clearly. This section however will discuss the workings of the CMAC from an engineering/mathematical perspective only.

2.8.1 THE MODEL

A basic CMAC controller for a single joint actuator is illustrated in Figure 2.9 as a block diagram. The diagram is broken up into four levels, L1 to L4.

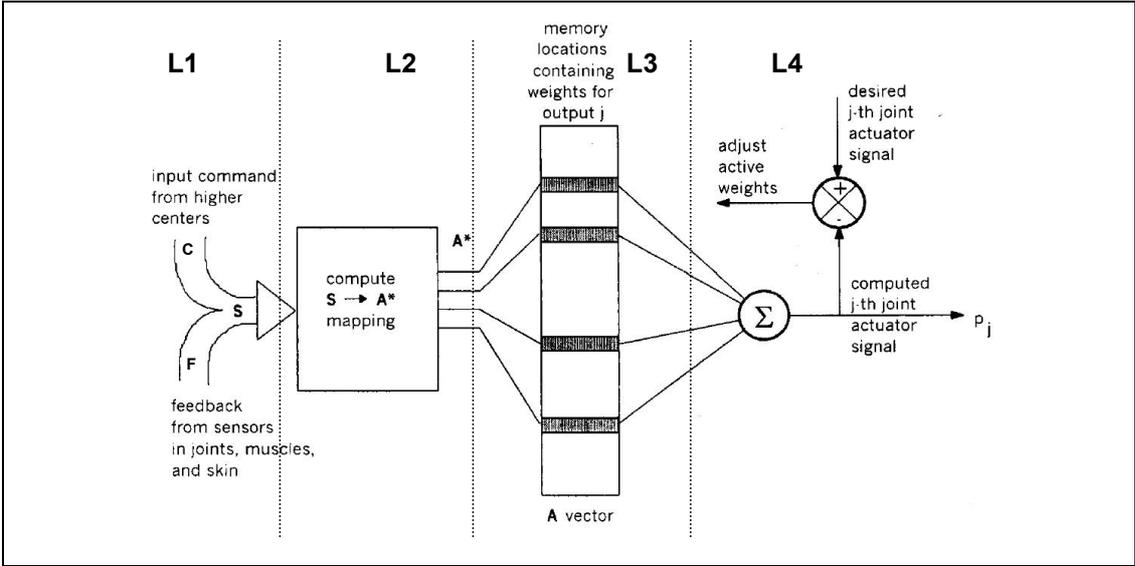


Figure 2.9 – Block Diagram of Albus's CMAC for a Single Joint Actuator (Albus, 1981)

Each level represented in Figure 2.9 represents a layer of mappings, starting from an input vector \mathbf{S} to an output vector or some scalar output value p , with various mappings in-between.

$$\mathbf{S} \rightarrow \mathbf{M} \rightarrow \mathbf{A} \rightarrow p \quad \text{Equation 2.1}$$

‘ \mathbf{S} ’ represents an input vector of multiple dimensions. ‘ \mathbf{M} ’ represents the single dimensional components in CMAC input space. ‘ \mathbf{A} ’ represents the multidimensional representation receptive fields of ‘ \mathbf{S} ’. ‘ \mathbf{A} ’ directly corresponds to the weights associated with ‘ \mathbf{S} ’. The summation of weights in ‘ \mathbf{A} ’ then produces the output, ‘ p ’.

2.8.2 THE INPUT VECTOR – SECTION L1

The input into a CMAC is a vector \mathbf{S} , consisting of the summation of two separate vectors: a command vector \mathbf{C} , and a feedback vector \mathbf{F} .

$$\mathbf{S} = \mathbf{C} + \mathbf{F} \quad \text{Equation 2.2}$$

where \mathbf{S} is the vector combination of vectors \mathbf{C} and \mathbf{F} , defined by combining two vectors or list of variables into a single vector of variables. For example, take $\mathbf{C} = (c_1, c_2, c_3)$ and take $\mathbf{F} = (f_1, f_2, f_3)$. Then the combination of the two vectors into a single vector \mathbf{S} becomes:

$$\mathbf{S} = (c_1, c_2, c_3, f_1, f_2, f_3) \quad \text{Equation 2.3}$$

Command vectors can be a position in space, a velocity or a force, etc. Additionally, a command vector may be a vector from the output of a higher level controller, such as another CMAC being used in series (see Section 2.8.7.6). The feedback vector can be the result of the position of a joint, the force being applied, the velocity of the actuator being controlled, etc., or a combination of all of the above. The combination of the two vectors, \mathbf{C} and \mathbf{F} to form vector \mathbf{S} is then mapped onto or ‘encoded’ into a unique function.

2.8.3 THE $S \rightarrow M \rightarrow A$ MAPPING – SECTION L2

A CMAC input vector space may be multidimensional, also known as a hyperspace (i.e. N^{th} dimensional: 4, 5, ..., n dimensions). However for simplicity, the CMAC mapping process here will be illustrated using a two-dimensional input vector space. From Figure 2.9, the input to this section L2 is the vector S . The two-dimensional input vector $S = (s_1, s_2)$ will be mapped onto a set of 4 quantising functions. To explain the CMAC $S \rightarrow M \rightarrow A$ mapping, Figure 2.10 illustrates two sections, L2 and L3 in detail, which correspond to the same sections in Figure 2.9. Section L2 contains an $N=2$ (i.e. two-dimensional) input space, the first dimension being S_1 , and the second dimension being S_2 , corresponding to the input vector space S . Dimension S_1 is given a resolution of 13, and S_2 is also given a resolution of 13. Represented as $R_{S1} = 13$ and $R_{S2} = 13$ respectively, this produces a total number of 169 possible input vectors over the two-dimensional input space. A single excited input vector $S = (5, 9)$, is illustrated by the shaded square on the two-dimensional input space in Figure 2.10.

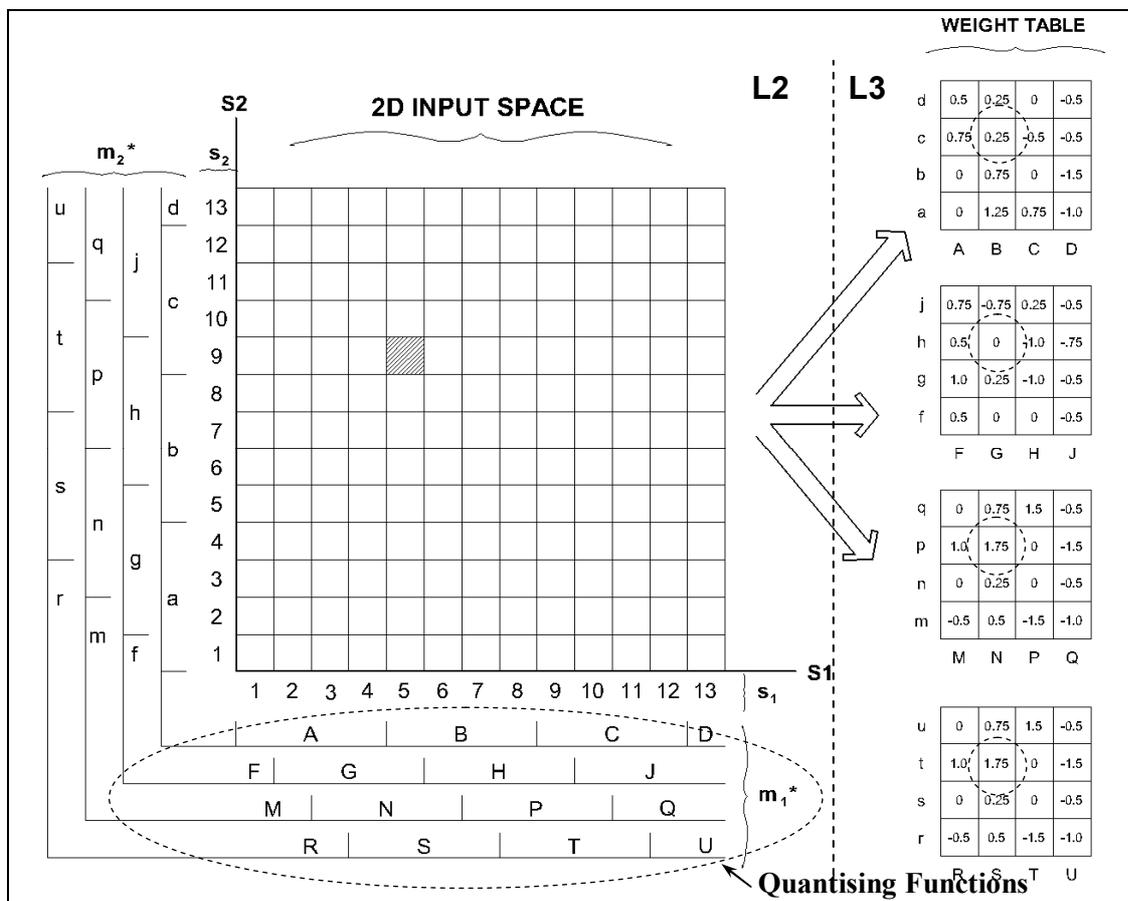


Figure 2.10 – A Two-Dimensional CMAC Mapping with Four Quantising Functions

Consider we want to map the input vector \mathbf{S} to the map M . This is achieved by first defining a set of K quantising functions, each with a resolution Q , where Q is defined as the number of resolution elements on each quantising function. Figure 2.10 shows four Quantising Functions ($K=4$) in the oval dashed-line segment, with each Quantising Function having a resolution also equal to four ($Q=4$). Here we can see that along the S1 dimension, the dimension has been broken up into 4 discrete elements, four times, from A to D, from F to J, from M to Q and from R to U, with each element offset from the next by one input space resolution unit. Each quantising function can be defined as iC_j where i corresponds to dimension S_i and j corresponds to the quantising function row. For example, take the first dimension S1, and the second quantising row from F to J. This can be represented as:

$${}^1C_2 = \{F, G, H, J\} \quad \text{Equation 2.4}$$

The same convention applies to dimension S2 and any other dimension included in other multidimensional input spaces.

Taking the input vector $\mathbf{S} = (5, 9)$ and tracing the shaded square in Figure 2.10 down to the dimension S1 quantising functions, it can be seen that input vector $\mathbf{S} = (5, 9)$ crosses the $\{B, G, N, S\}$ quantising function elements. Similarly, tracing the shaded square across to the S2 quantising functions, it can be seen that input vector $\mathbf{S} = (5, 9)$ crosses quantising function elements $\{c, h, p, t\}$.

For every possible input space coordinate for each dimension S_i , there exists a unique set m_i^* consisting of the set of variables defined by the K quantising functions, where i corresponds to the dimension S_i . In the example above, both the sets:

$$m_1^* = \{B, G, N, S\} \quad \text{Equation 2.5}$$

$$m_2^* = \{c, h, p, t\} \quad \text{Equation 2.6}$$

create a new map M , superimposed on the input space, corresponding to the input vector $\mathbf{S} = (5, 9)$.

The intersections of each n^{th} variable in each one-dimensional mapping m_i^* combine as is illustrated in Figure 2.11, to form the complete multidimensional mapping A , where:

$$A^* = \{Bc, Gh, Np, St\} \tag{Equation 2.7}$$

where A^* is the address of the vector A , found in the weight table in section L3. Here A^* contains K number of ‘receptive fields’, represented each as N -dimensional spaces superimposed over the input space (where N is the number of dimensions in the input space). In this example, we have four receptive fields, Bc, Gh, Np and St, each of two-dimensional space. These are called receptive fields because they are directly associated with the input vector, as the input vector lies within each receptive field.

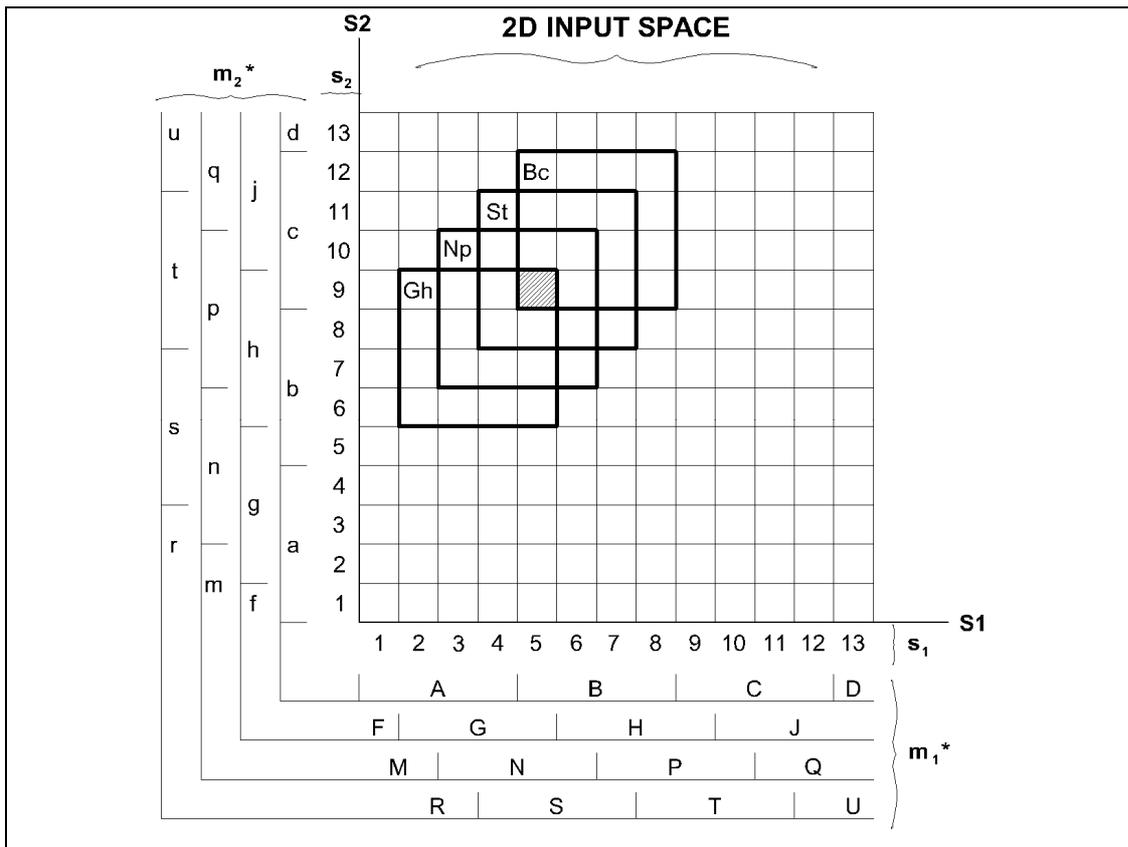


Figure 2.11 – $S \rightarrow M \rightarrow A$ Mapping of Two-Dimensional Input Vector $S = (5, 9)$

With the input vector \mathbf{S} mapped to a series of activated receptive fields, the next section, section L3, can be discussed, continuing with the current input vector $\mathbf{S} = (5, 9)$ example.

2.8.4 THE WEIGHT TABLE – SECTION L3

Section L3 of Figure 2.9 and Figure 2.10 contains the weight table, which contains K number of N^{th} dimensional arrays, each array containing Q^N number of weights. Each weight W_{I_j} corresponds to each receptive field, I_j . In this example, the receptive fields, Bc, Gh, Np and St correspond to the weights (the weight values) in the weight table (represented as the circled weights with a dashed line in Figure 2.10):

$$W_{Bc} = 0.25$$

$$W_{Gh} = 0$$

$$W_{Np} = 1.75$$

$$W_{St} = 1.75$$

This way, the number of weights (n) required by a CMAC is not R^N , but instead:

$$n = KQ^N \quad \text{Equation 2.8}$$

In this example, it means that the $R_S = 13 \times 13 = 169$ resolution input space can be mapped into a substantially smaller index that consists of only $4 \times 4^2 = 64$ weights. This allows the memory requirements of a CMAC to be substantially less than the number of possible input space addresses.

It also means that input vectors which are close together in input space (i.e. are in close proximity of each other), will map to several of the same weights. This property allows local generalisation to occur, which is a very useful property of the CMAC. For example, take the input space vector $\mathbf{S} = (5, 9)$, and another in close proximity such as $\mathbf{S} = (5, 7)$. The vector $\mathbf{S} = (5, 7)$ maps the receptive fields Bb, Gh, Np, Ss. Two of these receptive fields, Gh and Np, are the same as the receptive fields of the previous input vector $\mathbf{S} = (5, 9)$. In this way, input vectors that are close together in input space will have similar receptive fields, and thus map to several of the same weight locations. It also means that input vectors that are far apart in input space will map to completely different receptive fields. The usefulness of this is that

the CMAC will produce similar outputs for similar inputs, in effect generalising when a new yet close input vector (to a previously presented input vector) is presented to the CMAC.

2.8.5 WEIGHT SUMMATION AND ADJUSTMENT – SECTION L4

The final result of the mappings, $S \rightarrow M \rightarrow A \rightarrow p$, results in the scalar variable p . This variable however may instead be a vector \mathbf{P} if the CMAC contains multiple outputs. For our example however, the output is a single scalar variable. From Figure 2.9, section L4 sums the selected weights corresponding to the input vector \mathbf{S} . In this example, this results in:

$$\begin{aligned}
 p &= W_{Bc} + W_{Gh} + W_{Np} + W_{St} \\
 p &= 0.25 + 0 + 1.75 + 1.75 \\
 p &= 3.75
 \end{aligned}$$

Thus the input $\mathbf{S} = (5, 9)$ produces the output $H(\mathbf{S}) = 3.75$, where H is the function generated by the mappings of the CMAC. Thus for every input space vector, $H(\mathbf{S})$ will produce a unique output scalar p .

The final stage of section L4 is the adjustment of weights based on the error between the desired output \hat{p} and the output scalar p . Figure 2.12 illustrates this process.

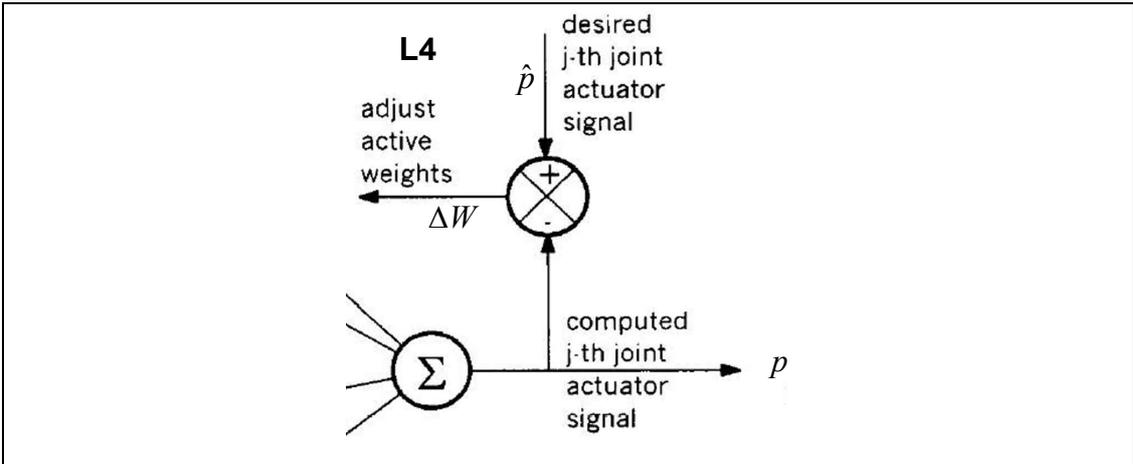


Figure 2.12 – Section L4: CMAC Weight Summation and Adjustment

In Figure 2.12, the weight adjuster ΔW is calculated by the following function:

$$\Delta W = g \left(\frac{\hat{p} - p}{|A^*|} \right) \quad \text{Equation 2.9}$$

where g is a user-defined gain factor between 0 and 1, and $|A^*|$ is the number of weights in the set A^* for any particular output p . In this example, $|A^*| = 4$.

The weight adjuster ΔW is then added to each active weight (i.e. the weights contributing to the output p), to produce a new updated weight which overwrites the old weight. For any set of active weights, the weight adjustment calculation will increment or decrement only the active weights in the direction of the desired value \hat{p} . This way after several training iterations, the CMAC can learn to produce the correct output p until a suitably low error rate is achieved. Note that the weight adjustment can be continuous, since the CMAC is capable of fast online real-time training. However if the user does not wish the weights to be continually updated, then running the CMAC without the weight adjustment equation will provide a ‘testing’ mode only controller.

2.8.6 CMAC COMPENSATION OF SYSTEM TIME DELAY

The local generalisation property of the CMAC allows the CMAC to automatically deal with slight time delays in a control system. The amount of time delay a CMAC can withstand before becoming unstable is dependent on the size of the generalisation area. A greater amount of time delay can be handled with larger generalisation areas, though the finer levels of control will be lost.

When training a CMAC, it is optimal that the adjusted weights should be those weights which produced the current output state. Time delay does not naturally allow for this though, and thus weights are adjusted for an output state that does not correspond to the weights that produced the current output state. Building on from Figure 2.12, if we incorporate time as a variable for a CMAC without any time compensation, we obtain Figure 2.13.

Without any time compensation included in the control process, the standard CMAC will be presented with time delay $t+n$, adjusting weights at time $t+n$ for an output produced at time t .

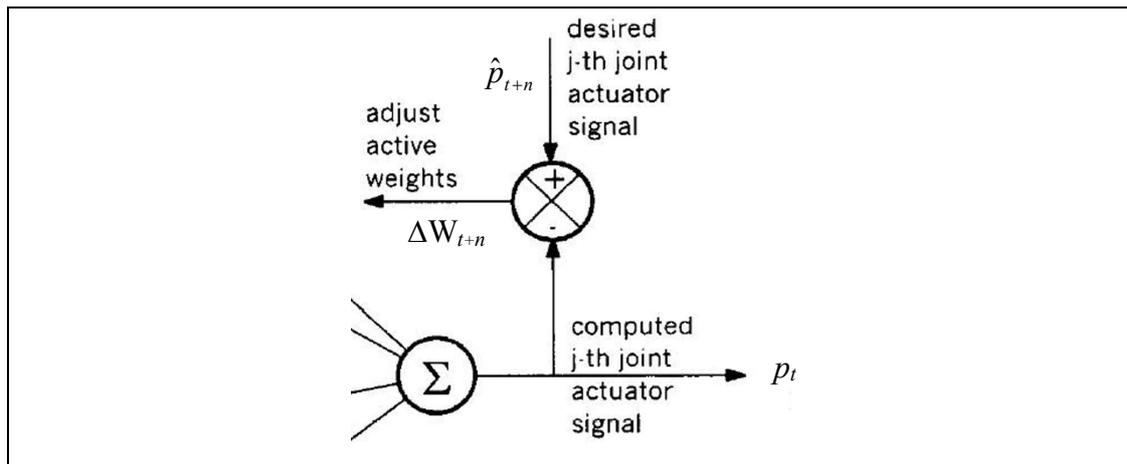


Figure 2.13 – CMAC Weight Training With Non-Compensated Time Delay

Initially it would be expected that if weights at $t+n$ are adjusted based on an output produced at time t , then the system would quickly become unstable. However, the local generalisation property of the CMAC automatically keeps a system stable if the amount of time delay is within the CMAC's manageable limits.

This concept is best explained via an illustration. Figure 2.14 illustrates how for two CMAC input space vectors within close proximity of each other, there exist overlapping receptive fields which map to both of the input vectors.

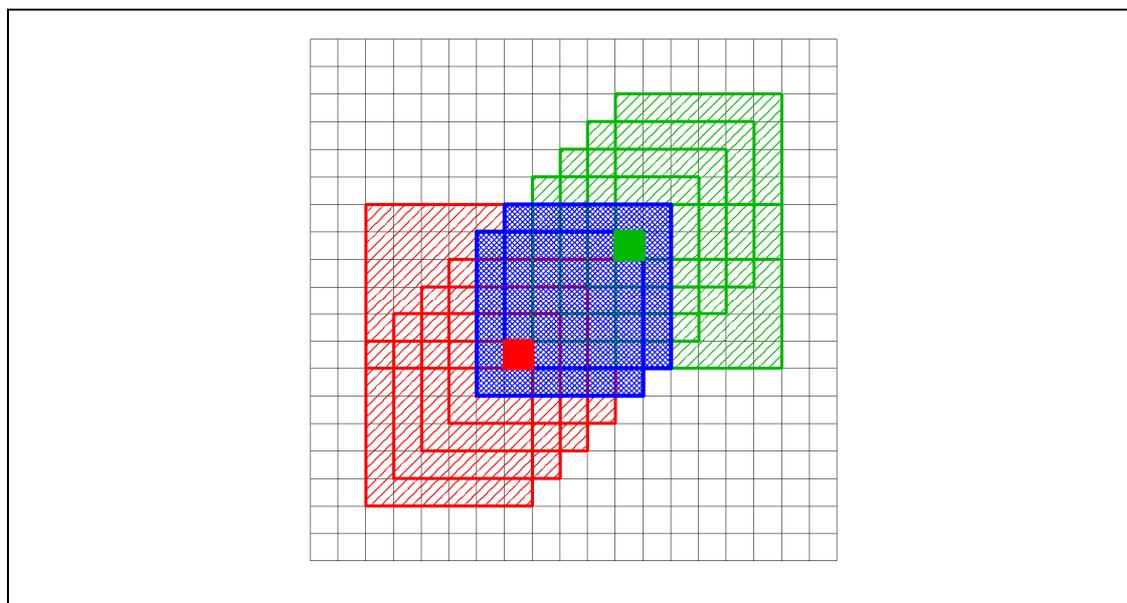


Figure 2.14 – CMAC Time Delay Handling Using Overlapping Receptive Fields

For two input vectors which have overlapping receptive fields, changing the weights associated with one input vector will also modify those weights corresponding to the overlapping receptive fields of the other input vector. This means that a change in weight adjustment for a particular input vector will also influence the output of adjacent input vectors in the same direction of the weight adjustment.

As explained previously, this is the standard local generalisation property of the CMAC. It also means that for a control system with a time delay greater than one sampling period, adjusting the weights used at time $t+n$ will also adjust some of the weights used at time t . This means that the weights adjusted at time $t+n$ are not completely independent of the weights adjusted at time t . This allows the output value at time t to move in the required direction, even though the error used in the training of these weights corresponds to the error of another input vector.

For larger generalisation areas, a CMAC can handle larger time delays as each input vector maps to weights spanning a larger range of adjacent input vectors. If time delayed input vectors lie far apart however, mapping to totally independent receptive fields with no generalisation area overlap, then the CMAC will not be able to automatically compensate for the time delay. This is particularly true for small generalisation areas. If the time delay of the system is not compensated for, then all of the weights at time $t+n$ are independent of the weights at time t , and no overlapping receptive fields exist.

All of the weights at time $t+n$ will be trained using an incorrect error value based on the error associated with the CMAC output at time t , potentially causing the system to become unstable. Figure 2.15 illustrates this scenario, where an input vector at time t maps to a generalisation area of the CMAC input space independent from the area that the input vectors at time $t+n$ map to. The error at time $t+n$ is the result of the output from the weight at time t , and since the weights associated with the generalisation area at time $t+n$ do not overlap with the generalisation area at time t , convergence will not occur and the CMAC will never learn to minimise the errors.

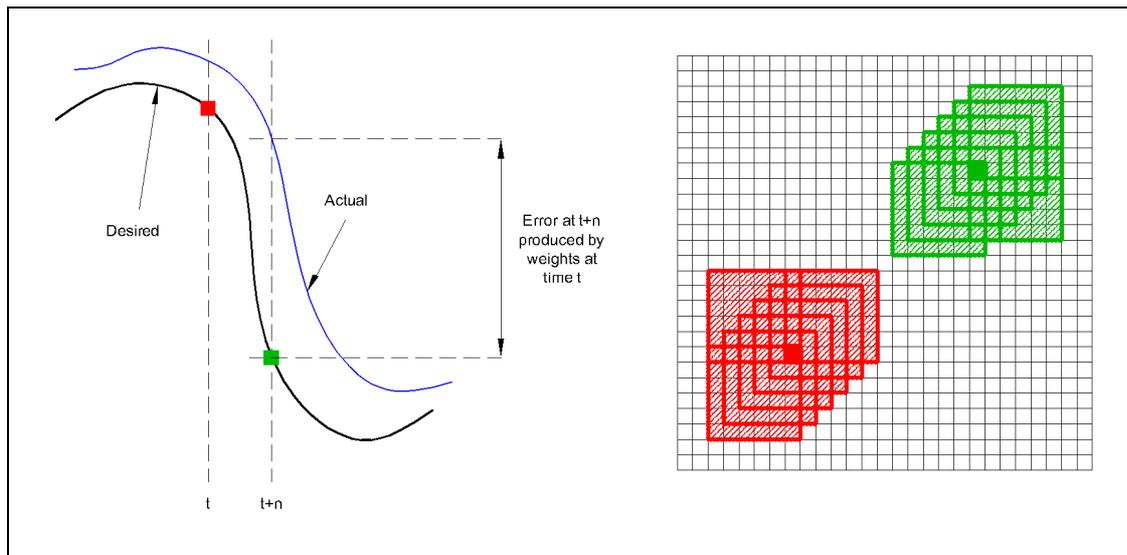


Figure 2.15 –Incorrect Training of Non-Overlapping Weights Due to Time Delay

2.8.7 CONTROLLER-SPECIFIC CMAC ADDITIONS/MODIFICATIONS

The CMAC has been used in a number of different applications and modified in numerous ways. However there are a few important key modifications to the original Albus CMAC that are specific to enhancing CMACs used specifically for actuator control. It is often possible to extend the majority of these additions to CMACs used for other purposes such as classification, to improve the modelling capability of the CMAC. While several additions/modifications will be presented here to illustrate the diversity of CMAC development to date, those most useful in enhancing the CMAC functionality for the purpose of this thesis will only be implemented.

2.8.7.1 *Memory Handling*

CMAC requires a unique memory address to store each receptive field weight in to produce a clean output. There are two major obstacles in reality that determine how this memory allocation is performed. If there were no memory limitations, then each memory address could be accessed directly for every possible weight. If there were no time limitations, then it would be possible to use minimal amounts of memory using sequential search methods. However, this is not the case.

Without appropriate memory management methods, multidimensional CMACs require more memory than is usually physically available, as with each additional input dimension, the memory size grows exponentially. It is important to note

however that the amount of memory utilisation in multidimensional systems is usually far less than the memory size allocated to the CMAC. In other words, the majority of weight memory locations contain no data. For this reason, traditionally the CMAC memory requirement has been compressed by a method called ‘Hash-coding’ or ‘Hashing’ to reduce the memory requirements, making the CMAC practical to implement (Wang et al., 1996). Efficient use of the available memory and fast access to the memory are the prime concerns of any hashing method (Sedgewick, 1990). This method was suggested to be used with a CMAC by Albus (1975a; 1975b) as a way to make the CMAC implementable on the hardware available at the time. Hashing works by compressing a large but sparsely populated address space into a smaller, denser space. However hash collisions may result. This occurs when two addresses are compressed into a single address, such as in Figure 2.16, where the contents of cells a_2 and c_3 are written to the same address in memory, P1 (Hsu et al., 2002b). Hash collisions provide unfavourable effects on the convergence of the CMAC (Szabo & Horvath, 1999) and result as noise in the output. While methods have been designed to deal with hash collisions (Miller & Glanz, 1996; Sedgewick, 1990), the static memory addressing and limited capability to utilise 100% of only the previously addressed memory weights provides only a partial solution to efficient memory handling.

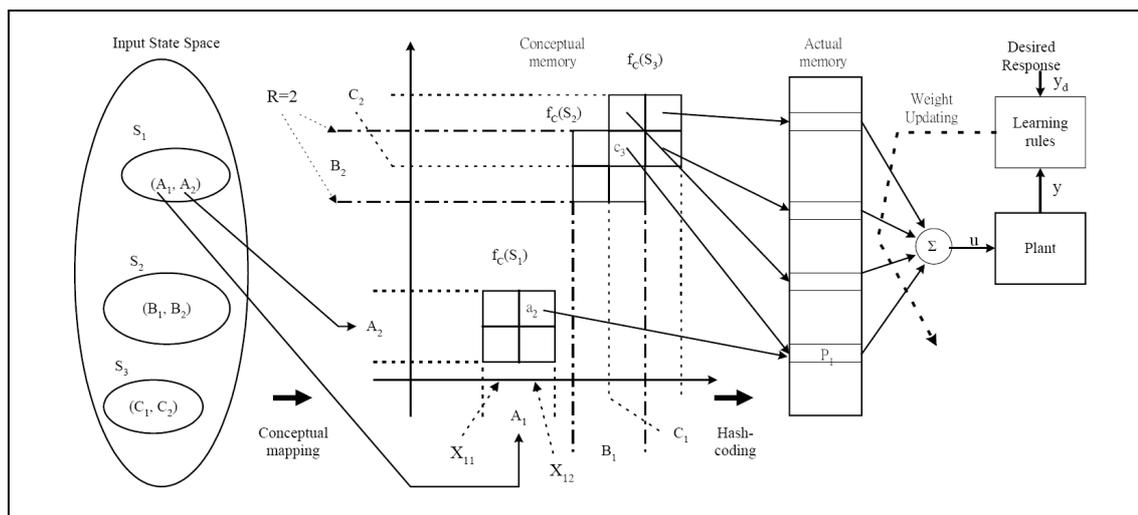


Figure 2.16 – CMAC with Hash-Coding Implemented (Hsu et al., 2002)

In addition to hash-coding, Hsu and Hwang, et al. (2002) proposed a CMAC with a Content Addressable Memory (CAM) which is used in place of the hash-coding

method. This method uses a memory content search method to search for identical data in memory which has been previously assigned to a particular input vector. If no match is found, then the next available memory location is assigned to the new input vector. This way 100% of the memory is utilised, and grows unsupervised whenever the CMAC is presented with new input vectors. Hsu and Hwang’s CAM CMAC, called a CCMAC, was designed for implementation on direct very large scale integration (VLSI) hardware (Hsu et al., 2000). Ker and Kuo et al. (1997) also designed a VLSI implementation to reduce the CMAC memory allocation requirements with a memory utilisation rate near 100%.

2.8.7.2 Uniform Receptive Field Mapping

An improvement to the CMAC’s generalisation ability involves the reorganising of the receptive field distribution pattern for any given input vector. This is shown in Figure 2.17.

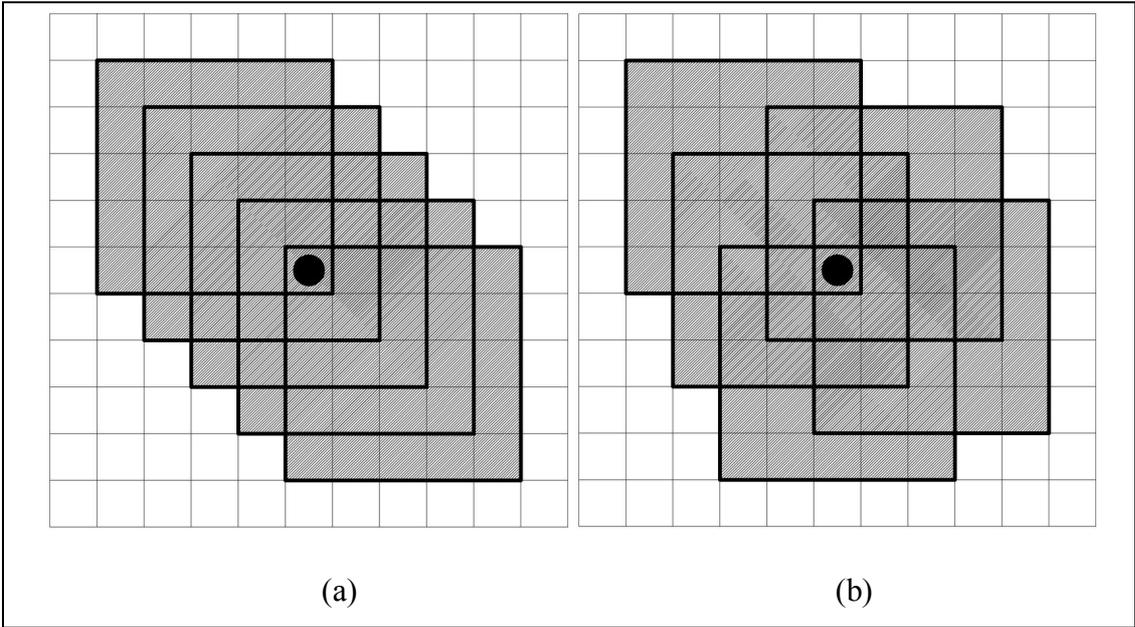


Figure 2.17 – Uniform Receptive Field Mapping Example

Figure 2.17(a) shows the traditional Albus receptive field distribution pattern associated with the corresponding input vector, which involves offsetting the receptive fields along hyper-diagonals in the input space. This however provides an inhomogeneous generalisation. Using a uniform lattice arrangement as in Figure 2.17(b) provides improved generalisation properties than using the original hyper-

diagonal approach, especially providing greater uniform local generalisation in higher dimensional input spaces (Miller & Glanz, 1996; Parks & Militzer, 1991).

2.8.7.3 Receptive Field Sensitivity Functions

A CMAC will provide the same value to the output vector regardless of where the input vector lies within each associated receptive field by using a simple summation of the weights associated with the input vector to produce the CMAC output. This ‘black/white’ method for associating weights with an output vector is known as a binary sensitivity function (also known as a binary basis function). When using a receptive field sensitivity function, or continuous sensitivity function, in place of the standard receptive field weight summation, greater preference is provided to the output from the weights with receptive fields having the input vector closest to their centre. This is illustrated in Figure 2.18. Figure 2.18(a) represents a two-dimensional input vector with five corresponding receptive fields, where each receptive field contributes equal weightings to the output. In Figure 2.18(b) the darker shaded receptive fields contribute more to the CMAC output than the light grey shaded receptive fields.

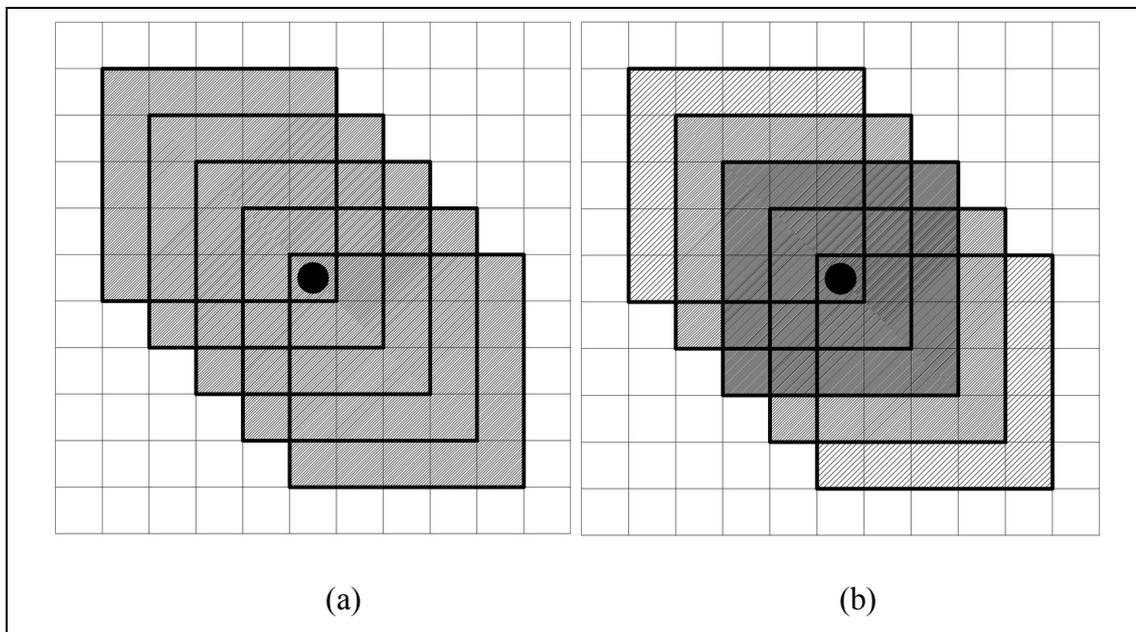


Figure 2.18 – Receptive Field Sensitivity Function Example

The receptive field sensitivity function can be any function that gives greater preference to weights with the input vector closest to their centre, and less preference

to weights with the input vector lying close to their outer edge. For example, a gaussian shaped sensitivity function has been used by Eldracher and Staller (1994), linear shaped and gaussian sensitivity functions by Miller and Glanz (1996) and B-spline sensitivity functions by Lane and Handelman (1992).

Eldracher and Staller (1994) demonstrate the superior modelling capabilities of using a gaussian sensitivity function over the original binary sensitivity function, for a CMAC with the same input space resolution and number of memory weights. Figure 2.19 and Figure 2.20 demonstrate how the modelling capability of the CMAC can be improved when using continuous sensitivity functions as opposed to binary sensitivity functions.

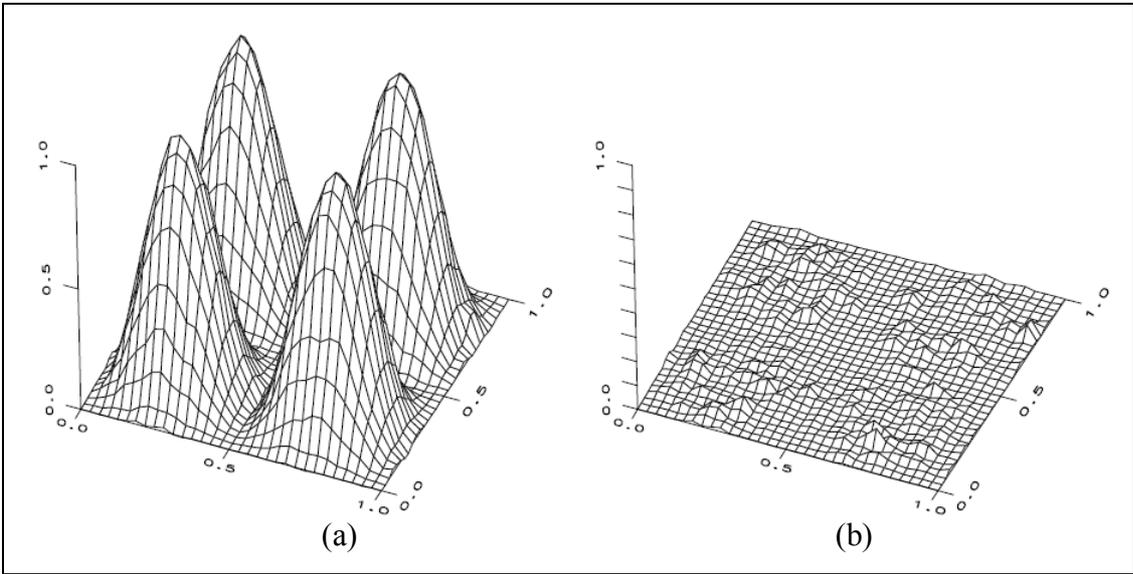


Figure 2.19 – (a) CMAC approximation of the function FSIN with binary sensitivity functions and (b) remaining absolute error. (Eldracher et al., 1994)

The CMAC used in Figure 2.19 has been trained using binary sensitivity functions. The CMAC in Figure 2.20 has been trained using Gaussian continuous sensitivity functions.

Figure 2.19(a) and Figure 2.20(a) both illustrate the CMAC output approximation of the FSIN function. Figure 2.19(b) and Figure 2.20(b) both illustrate the difference between the absolute CMAC approximation error for the learnt function.

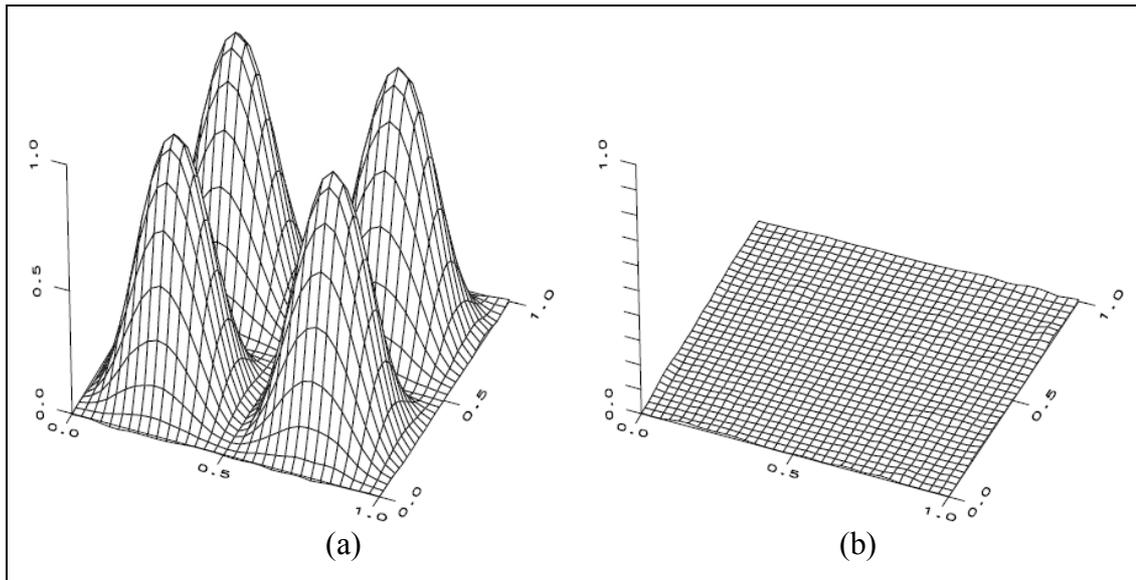


Figure 2.20 – (a) CMAC approximation of the function FSIN with Gaussian continuous sensitivity functions and (b) remaining absolute error. (Eldracher et al., 1994)

Notice the reduced error resulting solely from changing the binary sensitivity functions to gaussian sensitivity functions, improving the modelling capability of the CMAC.

2.8.7.4 Weight Smoothing

Weight smoothing is a concept that attempts to reduce spikes in the CMAC output. Due to residual errors and sensor noise, there are combinations of weights that can occur in any CMAC that will cause continual small adjustments to the weights, creating the occasional excessively large weight value causing a spike in the output. This is due to the fact that a single input vector maps to multiple receptive fields, causing other input vectors in close proximity to also map to some of the same receptive fields. While this property provides the ability of the CMAC to locally generalise, it can also allow residual errors to build up over time. Smith proposed a weight smoothing algorithm which adjusts the weights when training occurs so that each weight is moved towards the average of all the weights by an amount proportional to the training rate (Smith, 1998). However no test results were offered to prove the usefulness of his algorithm. Miller also proposed a weight smoothing algorithm which again adjusts the weights when training occurs, whereby weights that deviate too far from the average of all the weights contributing to a particular output vector are penalised and reduced in size (Miller & Glanz, 1996). Horvath has

also implemented weight smoothing into his kernel modified CMAC controller, with vast improvements over a kernel CMAC without weight smoothing (Horvath, 2004).

2.8.7.5 Eligibility

Eligibility is a concept whereby the current output of a controller is the result of the contribution of both current and previous controller parameters. By previous controller parameters it is meant that the output is the result of learning and weight adjustments from previous training iterations. It also means that the particular current output is the result of a string of previous weights that have contributed to it in the current iteration. While this is obvious, for no desired output could be achieved without the previous weight adjustments over time to reach that output, the previous weights can be adjusted so that the future desired output is achieved in the best possible way with the lowest average error. This concept is illustrated in Figure 2.21.

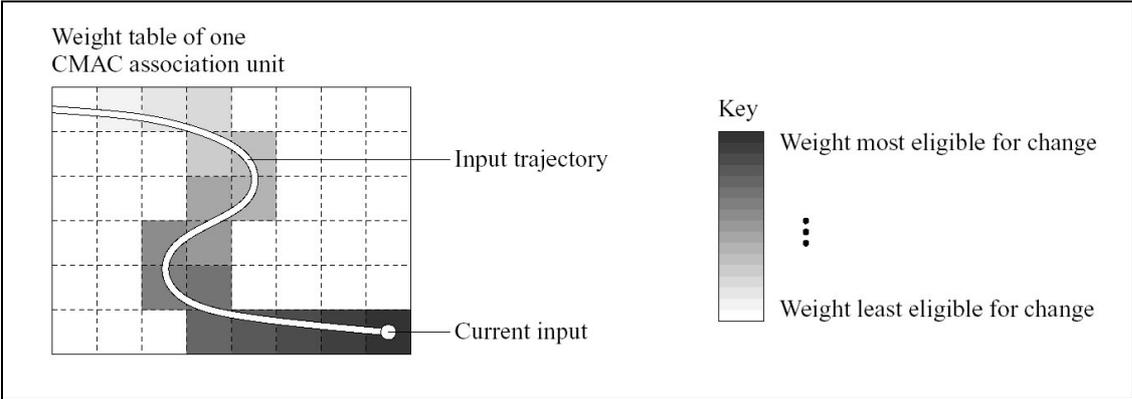


Figure 2.21 – Eligibility Implemented in a Two-Dimensional CMAC Weight Table (Smith, 1998)

Figure 2.21 shows a two-dimensional weight table representation of a CMAC, where each input vector maps to a single receptive field of equal size to that of a single input space resolution unit. Thus the number of weights in the weight table for this CMAC is equal to the total number of resolution units of the CMAC input space. It also shows how the current input is preceded by the inputs that have been previously used to arrive at the current input, which is represented by the lighter shaded resolution units in Figure 2.21. In this example as explained above, each weight corresponds to each input space resolution unit. Thus by changing the weights that were previously accessed by a small amount in favour of the current desired output, the current output can be achieved both faster and with a lower error rate.

The key in Figure 2.21 shows how the darkest shaded regions of the 2-D CMAC input space are most eligible for change, and the lightest shaded regions are the least eligible. The current output is affected by weights that have been recently activated, and not by weights activated in the distant past. A demonstration on the usefulness of including eligibility in a controller is illustrated in Figure 2.22.

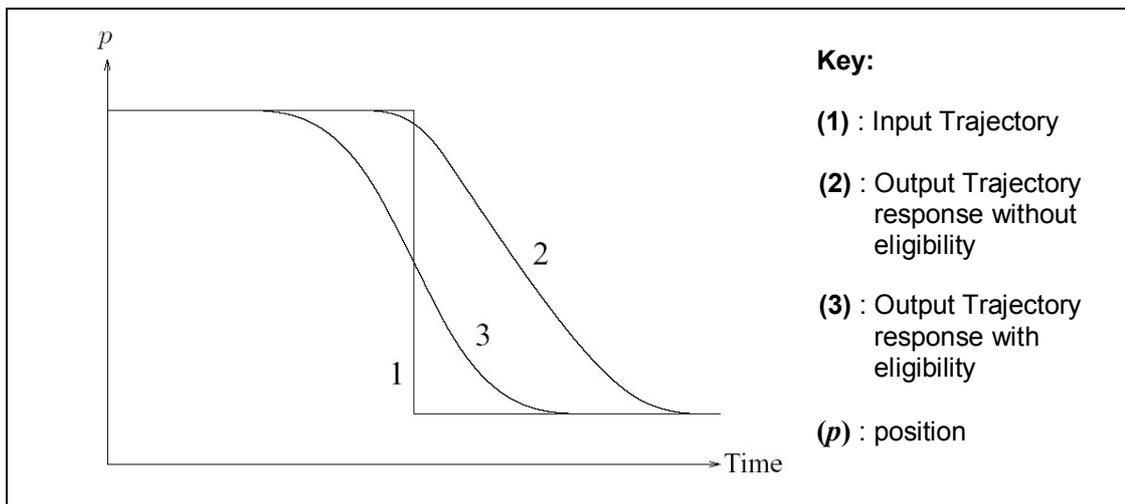


Figure 2.22 – Trajectory Outputs with and without Eligibility (Smith, 1998)

Every actuator has a bounded ability to accelerate and decelerate according to its limitations. Thus no actuator can follow a step-change in input response exactly. Therefore a controller without the use of eligibility will produce a response to the desired input step function **(1)** in Figure 2.22 similar to the output trajectory **(2)**. Due to the local generalisation property of a CMAC, the trajectory will start adjusting to the required output just before the step in desired trajectory occurs. However this approach produces a large error, as can be seen from the large area bounded by the input and output trajectories, **(1)** and **(2)** respectively.

When eligibility is introduced into the controller, the present information after the step change occurs indicates that a large change in the input trajectory has occurred, and therefore the previously used weights are adjusted in the direction of the step change. In other words, the controller adjusts the previously active weights based on the concept: ‘The next time I do this, I am going to react earlier’ and thus the controller pre-empt the step change. After several iterations, this results in an output trajectory similar to that of **(3)**, whereby the output starts to move in the direction of

the step change far before the actual step change occurs at the input. When comparing the error bounded by the input trajectory **(1)** and the output with and without eligibility, **(2)** and **(3)** respectively, the overall error of the trajectory with the use of eligibility is approximately 50% smaller. Thus the sum of the error magnitudes over time has been reduced.

This is the concept of ‘eligibility’ and has been implemented into the CMAC by Smith (1998) and Collins (1999). Smith implements his CMAC with eligibility controller named FOX (Fairly Obvious eXtension (to the CMAC)) to trajectory learning applications with noticeable success and used it in the simulation of a walking biped and the real-time control of an inverted pendulum system. Collins used a slightly different yet similar eligibility model into his CMAC whereby if a weight was not used in a subsequent iteration, then the weight change effect for it was decreased to 60%, then 20%, then 0%, using his CMAC to control a line-following robot car. In addition Shen and Guo (2005) used Smith’s exact eligibility model with their Fuzzified CMAC (FCE) model for ship steering applications.

Both Smith and Collin’s approaches used a fixed eligibility profile. The generalisation property of neighbouring CMAC receptive fields together with weight smoothing (if applied) automatically implements a degree of inherent eligibility to a CMAC’s memory space, as adjacent weights cannot deviate too far from each other. The work in this thesis however goes further into producing a controller with an adaptive ‘eligibility like’ pre-empting effect, applied when only required and to the level required to produce minimal error summation over time.

2.8.7.6 CMACs arranged in Control Hierarchies

A concept that was explained by Albus in the 1970s from both a biological and mathematical perspective is the principle of multiple-level hierarchy controllers (Albus, 1981). Albus explained this concept in detail from the biological principle of ‘hierarchical goal-directed behaviour’ whereby all tasks undertaken by an animal are achieved via a string of controllers, with each controller operating at a different level in the hierarchy. To explain this, take a human with the task of assembling a jigsaw puzzle. The top-most level controller, or highest level, instigates the task to find a jigsaw piece that will fit together with another piece. This controller issues a

command to ‘find matching piece’ to a middle level controller. This middle level controller then issues commands to ‘use eyes to search for piece’, which talks to a lower level controller to ‘move individual eye muscles’. Such a system is very basic but explains the key concept of hierarchical goal driven behaviour.

To make use of this biological control design, Albus proposed how his CMAC can be used in much the same way as a multi-level hierarchical controller explained above. This can be seen in Figure 2.23 by which each control level is a dedicated CMAC, whose outputs are the command vector inputs for each subsequent lower level CMAC. The main advantage of using a hierarchical structure is a means of partitioning the control problem into manageable sub-problems (Albus, 1975a).

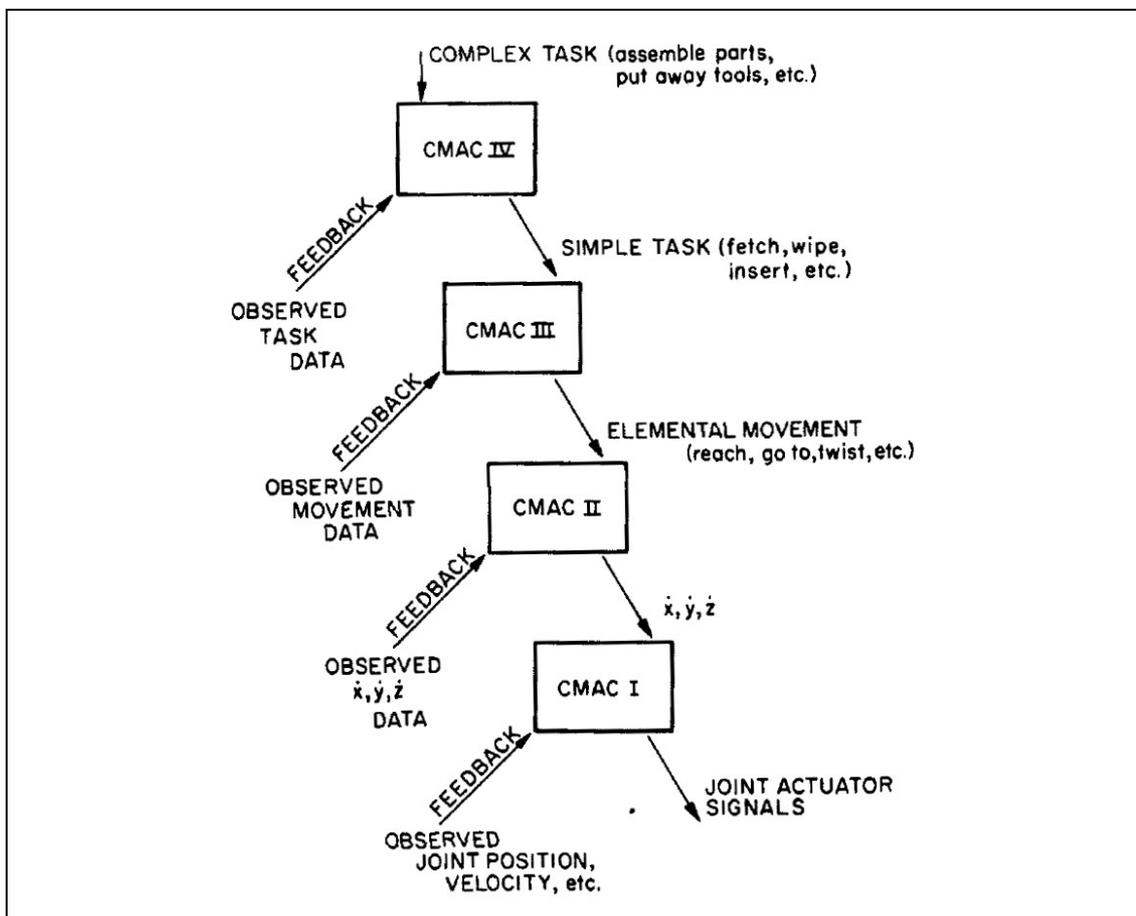


Figure 2.23 – A Hierarchical Structure of CMAC Controllers. (Albus, 1975a)

Hierarchical CMAC structures have been implemented by various researchers such as Tham who incorporated a CMAC into the Hierarchical Mixtures of Experts (HME) architecture creating a HME-CMAC used in non-linear dynamic manipulator

control (Tham, 1995). Lin proposed a walking gait hierarchical algorithm to control the motion of a four-legged machine, however only used a single CMAC in the hierarchy, with the other hierarchical control levels being of classical non-adaptive approaches (Lin & Song, 1992).

Another concept of arranging CMACs in hierarchies stems not from breaking the control problem down into sub-groups, but for providing a solution to solve multiple dimensional problems with the use of multiple low-dimensional CMACs.

The Hierarchical CMAC (HCMAC) (Chen et al., 2004; Lee et al., 2003) provides a solution to resolve high-dimensional classification problems using a hierarchical topology of individual two-dimensional CMACs with Gaussian Basis Functions (GCMACs). Figure 2.24 shows the architecture of a 4 input (4-dimensions) HCMAC, of input dimensions s_1, s_2, s_3 and s_4 .

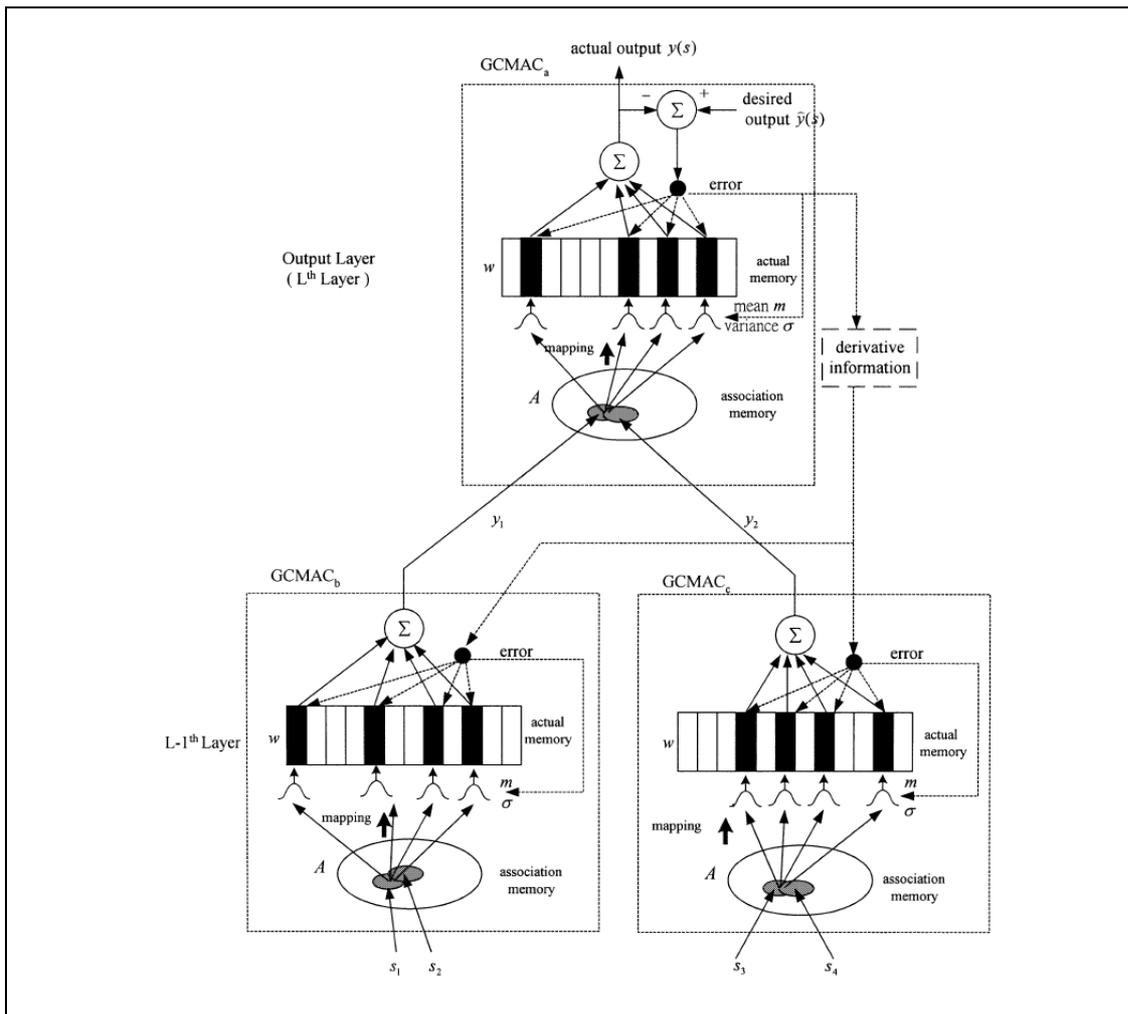


Figure 2.24 – Detailed Architecture of a 4-Input-Variable HCMAC (Lee et al., 2003)

The HCMAC architecture can be extended to highly multiple dimensional systems by using a larger hierarchy of two-dimensional GCMACs.

A similar approach was also devised by Lin and Li (1996) using again a hierarchy of two-dimensional CMACs, using the product of several two-dimensional CMACs in a sub-module. The addition of multiple sub-modules are added until the output error is satisfactorily small. In addition, Jan and Hung (2001) presented a learning structure called the Macro Structure CMAC (MS-CMAC). This model was developed by connecting several one-dimensional CMACs as a tree-structure, which decomposes a multidimensional problem into a set of 1-D sub-problems.

The motivation behind the decomposition of a multidimensional CMAC into a hierarchy of low-dimensional CMACs is due to the fact that a high-dimensional CMAC requires huge amounts of memory, even though most portions of the memory space are usually unused. Thus these hierarchical low-dimensional CMACs are used in place, providing satisfactory results in most cases with reduced memory requirements. However the loss of a real multidimensional input space means that the coupling of variables and cross-talk may result, though should provide adequate results for sparsely populated input spaces.

2.8.7.7 CMAC Systems Utilising Multiple Control Resolutions

Another approach of breaking down the control problem into subcategories involves using CMAC receptive field layers of differing resolutions. Moody and Darken (1989) proposed such a multi-resolution approach, whereby the receptive field layer with the largest sized receptive fields (i.e. the layer with the coarsest resolution) was trained first. The subsequent progressively finer resolution layers were then trained one at a time. Training of the previously trained layers is halted once the training of a finer resolution is started, making the overall system unable to adapt to new changes in system dynamics. Menozzi and Chow (1997) also make use of this concept, however only the coarsest layer covers the whole input space. Subsequent finer resolution layers only cover the portion of the input space where higher levels of training accuracy are required.

In addition both Pak-Cheung (1991) and Huang et al. (1997) have proposed using separate CMACs in parallel, each with a different resolution. This concept is illustrated in Figure 2.25, where CMACs of increasing resolution, 1 to L , are summed together to provide the output value. This concept extends on from Moody's work of using single layers of different size receptive fields (Moody & Darken, 1989), and performs with better results (Miller & Glanz, 1996). The drawback of this approach however is that each CMAC is trained with the same function, and thus memory requirements grow with the addition of each new CMAC, even if the higher resolution CMAC is not required to contribute significantly to the output for certain input states. In addition the training of previously trained CMACs is stopped while subsequent layers are trained.

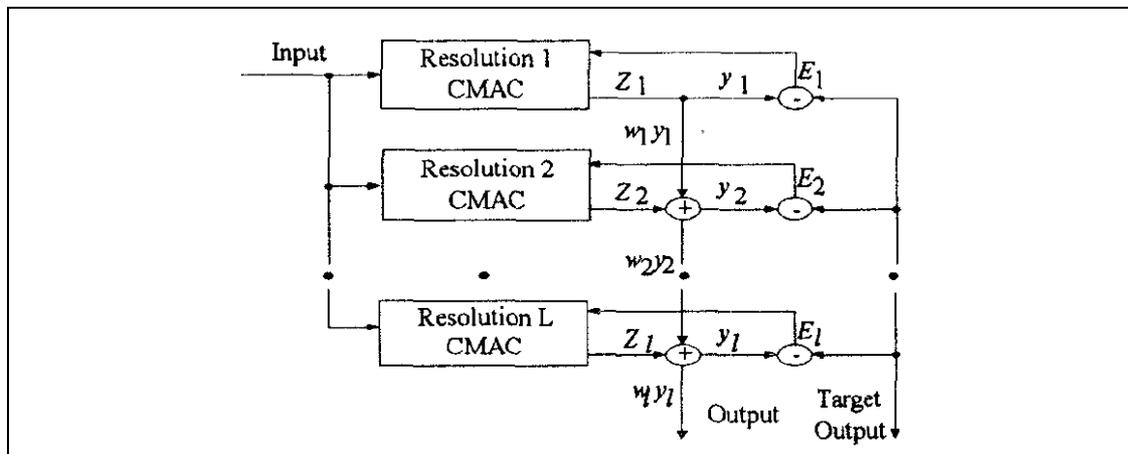


Figure 2.25 – Multiple Different-Resolution CMACs in Parallel (Huang et al., 1997)

Shewchuk and Dean (1990) also devised an Online Multi-resolution CMAC (OMRC) which overcame the future adaptive training problems that Moody (1989) encountered. Instead of training each CMAC sequentially, the CMACs are trained in parallel, using a different fixed learning rate for each CMAC, such that the higher the CMAC resolution, the slower the training (Dean et al., 1992). While this parallel approach learns slower than Moody's sequentially trained approach, the OMRC can adapt to systems with changing dynamics. However the primary disadvantage with the algorithm is the decrease in accuracy (Shewchuk & Dean, 1990). Zheng and Luo et al. (2006) also designed an adaptive system to control a double inverted pendulum in simulation using two CMACs, one with a low generalisation parameter and one

with a high generalisation parameter together with reinforcement learning with successful results.

The main advantage of using hierarchical and multidimensional CMACs is that the control problem can be broken down into separate levels, simplifying the amount of work each different level controller has to perform. It also allows the control problem to be broken up into a system of controllers that each performs a few functions extremely well, instead of leaving a single controller to perform every task. This can include the separation of individual tasks or breaking up tasks into different resolutions. This makes the design of individual level controllers simpler and the whole control process more efficient as each control level is only being used for the simple processes that it was intended for.

2.8.7.8 Combined CMAC Enhancement Capability

The CMAC concept has proven very useful as both a controller and classifier in several areas of research to date. In addition to being used successfully in its original form, the basic CMAC model proposed by Albus has been enhanced with several applied modifications and additions by a variety of different research groups. However, to date the literature does not show a CMAC incorporating each of the above enhancements together. Miller and Glanz (1996) incorporated weight smoothing, uniform receptive field mapping and receptive field sensitivity functions. Smith (1998) incorporated eligibility into the original CMAC model. Hsu, Hwang et al. (2002) incorporated CMACs with dynamic memory handling. The hierarchy model Albus proposed (1981) together with the above mentioned CMAC enhancements is yet to be developed and tested. Each enhancement enables the CMAC model to become either more implementable and/or provide better modelling capabilities each in their own way. While the majority of researchers follow their path of individual enhancements, the combination of several key enhancements would prove by far superior in the modelling and control capabilities of the CMAC.

2.9 DESIGN OF A SUITABLE CONTROLLER

Dealing with the controller as the focus point of this thesis, specifically using the CMAC as the adaptive neural network foundation, the trajectory control problem needs to be understood.

As the heart of the controller will be the adaptive CMAC controller, the adaptive qualities of the controller will automatically drive the actuator to its potential, reducing the position error at a single sampling instance in time over the course of the trajectory. Using a standalone CMAC, the following type of control can be obtained, as illustrated in Figure 2.26.

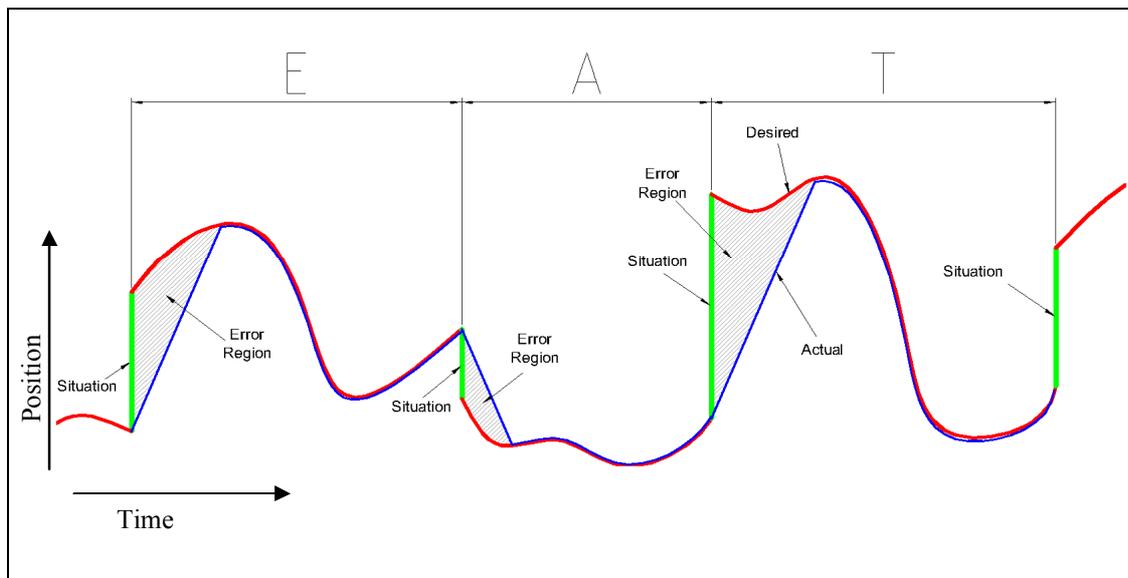


Figure 2.26 – Possible CMAC control of actuator for desired concatenated trajectories.

For input trajectories that require greater acceleration and/or velocity than an actuator can provide, such as in Figure 2.26, significant errors will exist which cannot be corrected via traditional or adaptive approaches that do not use future states as control variables. The output from the system is smooth and accurate as desired, but on a sluggish actuator, the performance after a situation is rather poor, with large error regions created. Over the course of a desired input trajectory, the output response will produce significant response lags. Even when using adaptive approaches, periodic signals will continue to produce these response lags at the output.

In terms of learning deafblind sign language trajectories, these errors mean that there will be a loss of conveyed information at the start of each concatenated trajectory. As mentioned, situations are infinite velocity discontinuities in the overall desired trajectory, and no physical system is capable of responding at infinite speed, nor

would it want to. Thus for any physical system, in this example there will exist some degree of error at the beginning of each concatenated trajectory.

Using a CMAC alone however means that language information will be lost in the large error regions illustrated in Figure 2.26. Without altering the desired trajectory, there should exist some method of minimising the error created by a situation, over time. What this means is that not only is the position error minimised for a single instance in time, but also over time. This concept is similar to the eligibility concept explained in Section 2.8.7.5, and is illustrated here in Figure 2.27.

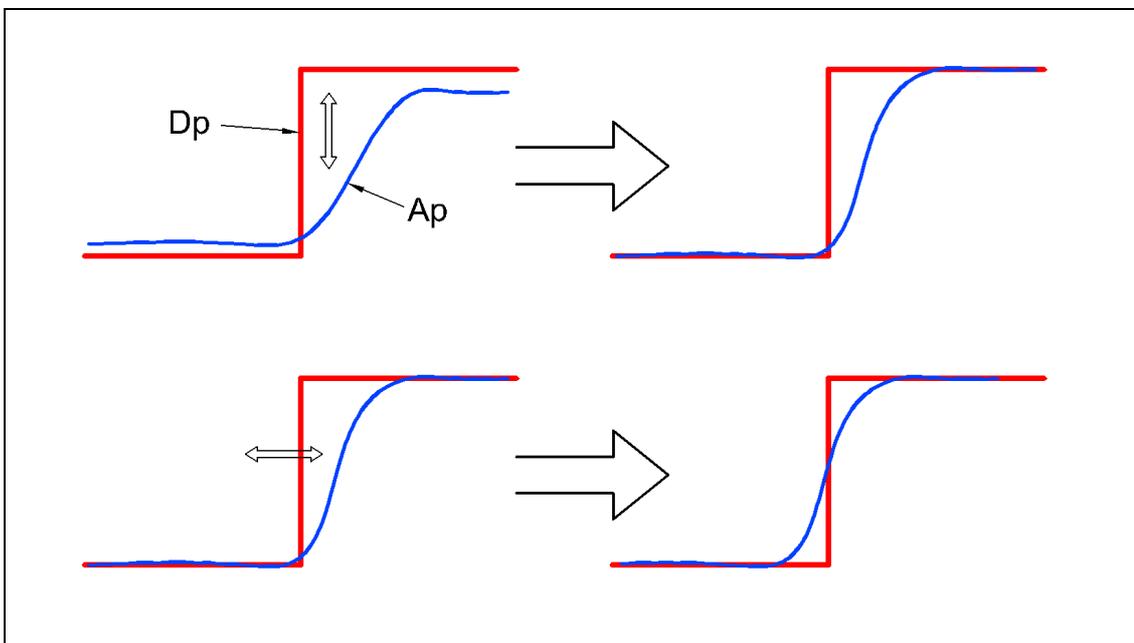


Figure 2.27 – Error Minimisation over Time

Similar to the eligibility concept, the overall error summation over time is reduced by starting to move the output in the desired direction before the desired output is required. In this thesis the pre-empting effect of moving the output in a situation's direction before the situation occurs, will be referred to as 'Situation Pre-empting'. What this does in effect is to reduce the overall error over time, producing a better approximation of the desired function by reducing the loss of conveyed information. This is particularly important when dealing with the communication of a language through motion control, as increased error means loss of information. Thus minimising the error not only in a single instance in time, but also over time, in turn

minimises the loss of language information induced by the physical limitations of the driven actuator.

For periodic signals that can be planned in advance (i.e. a string of hand gestures to spell out a word or sentence), there should exist an adaptive component which minimises response lags, modifying the control signal in some way so that the actuator starts to move in the direction of the desired response before a situation occurs. After several training iterations, the controller should be able to find a suitable point in time to start the actuation which will produce minimal error. Regardless of the dynamics of the actuator, the controller should be able to adapt to provide a reduced error response between the desired and actual trajectories for a particular actuator, with no prior knowledge of the actuator's dynamics. The purpose of using adaptive control approaches is so the control dynamics of an actuator do not need to be explicitly known.

Using the error over time minimisation technique illustrated in Figure 2.27, and applying this to the error regions of Figure 2.26, produces an error minimised trajectory for both position and time errors, as illustrated in Figure 2.28.

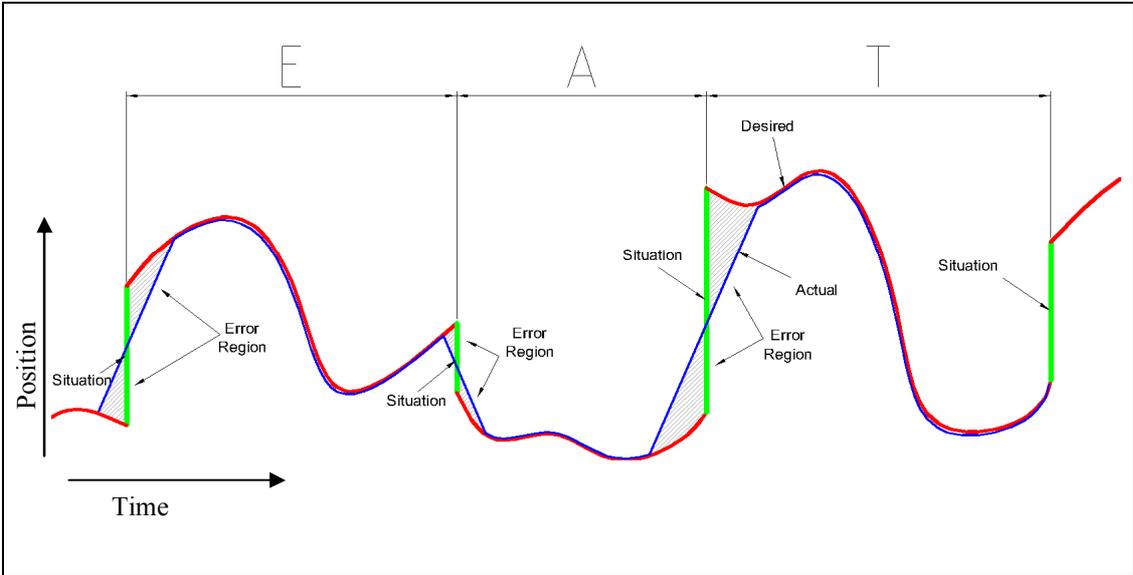


Figure 2.28 – Error Minimised Control of Actuator for Desired Concatenated Trajectories.

Figure 2.28 illustrates how the error summation over time is reduced if the output is driven in the desired direction before the concatenated desired trajectory is

encountered in present time. This means that language information is preserved as much as possible, with a slight amount of the ending of the previous letter trajectory and a slight amount of the beginning of the subsequent letter trajectory, being lost to the actuator's dynamic limitations. However, instead of losing a greater amount of information from one letter, both letters lose a little, producing a smoother and clearer depiction of the word being communicated. Thus an improved approximation of the original desired trajectory is produced.

Such a concept would work if and only if the controller had adequate knowledge about the desired upcoming trajectory. In the case of this thesis, a fingerspelled word is made up of a sequence of trajectories, with each subsequent trajectory known before it is physically acted out. Thus for this system, prior knowledge of upcoming desired trajectories will be known, allowing the controller to move the output in the desired direction before the next trajectory is required.

Therefore, for any clean input trajectory to any actuator, after adaptive learning, the system should be able to produce a minimal error output over the entire course of the input trajectory. Each 'situation' encountered throughout the input trajectory should be 'smoothed' so to speak, producing less overall output error.

2.10 THESIS CHALLENGE

Trajectories containing large discontinuities in the form of a situation pose a problem using a regular controller in minimising the error associated with a situation. Building on from the discussed literature and the nature of the problem at hand, the main challenge of this thesis is:

Thesis Challenge

To develop an adaptive method of training CMACs to control situations where the command trajectory exceeds the capabilities of the actuator, such that the combined sum of error magnitudes before and after a situation is minimised.

This challenge will be achieved by completing and integrating a range of sub-challenges. These include:

1. Training and using a CMAC online and in real-time to control a non-linear actuator.
2. To produce an effective, fast and dynamic memory handling system for CMAC, with every weight mapping to a unique address in the CMAC memory space, producing a clean CMAC output.
3. Developing a controller which will be able to provide both feedback and feedforward types of control, and integrating the two types of control seamlessly together.
4. Adding a higher level of training to the base control system, to enable the system output to minimise the error in terms of both position and over time. In the case of reproducing deafblind sign language, this will preserve the linguistic information as accurately as possible.
5. To observe and understand when and why the newly developed training method performs well and when it does not, allowing the system to know when and when not to use the new training method for best overall performance.
6. Solving the control problem at a realistic, practical and physical level, to be capable of controlling an external non-linear actuator in real-time.
7. Developing a physical control system that is capable of testing the developed controller on hardware. This system should be flexible and powerful enough to monitor and record controller parameters at will, both graphically and numerically, in real-time.
8. Developing physical actuation hardware to interface and test the controller on.

The controller developed called the Error Minimising Gradient Controller (EMGC) will be tested for a variety of realistic hand gesture trajectories. It is impossible to test the controller over every possible desired trajectory, and thus the controller will be tested over a range of simulated trajectories, to accurately test the performance of the controller. Thus the testing will involve:

1. Testing the developed EMGC against a standard adaptive feedforward/feedback controller using CMAC, for all tests.
2. Testing the EMGC over a range of 'situation' sizes, to test the controller's performance over a wide variety of desired position magnitudes.
3. Testing the EMGC's performance on actuators of different physical dynamics.
4. Testing for which physical and controller variables produce the best error minimising performance from the EMGC, to obtain the best performance from the controller.
5. Testing to see when the EMGC minimises error and when it does not.
6. Testing the EMGC over a range of trajectory gradients approaching a situation.
7. Testing the EMGC over a range of trajectory gradients departing a situation.

The first step in the creation of the desired controller involves implementing a dynamic memory handling algorithm into the CMAC, with comparisons made to the traditionally used static method of hashing.

3.0 DYNAMIC AND STATIC CMAC MEMORY ALLOCATION

The CMAC alone, without an accompanying memory handling system, is unable to learn and adapt from training experience. The weight table used in a CMAC requires weights to be stored in memory of some type. This memory needs to be managed, recalling correctly mapped weights when required to produce an output, and allocating new weights when new input vectors are activated. It is important that each possibly activated receptive field in a CMAC maps to a unique weight stored in memory. This is required to produce a clean CMAC output, directly mapping to a unique set of weights for a given input vector. This chapter deals with designing a memory allocation method for managing a CMAC's weight table memory, with preference given to dynamic and unique memory allocation over previously used methods.

Dynamic allocation of memory when using CMACs on any conventional computational device is a most desirable property, as it is often not known how much memory is required to be allocated for a particular CMAC prior to its usage. As the CMAC's knowledge grows, if more memory is needed than was originally anticipated, unless the memory can be allocated when needed, the CMAC will fail to adapt further while retaining previous learnt knowledge.

Dynamic allocation of memory is particularly useful when a CMAC goes offline and data is required to be stored or transferred from its fast access online memory, usually RAM of some sort, to its slow access offline memory, usually a hard disk. Storing only the weights which have actually been used is preferable over storing large amounts of unused data predefined by the size of the static table allocated.

This section of the thesis deals with the static memory allocation technique hashing, which has widely been used to store data in CMACs as opposed to the proposed dynamic allocation technique. The static memory allocation method presented here is based on the University of New Hampshire's (UNH) code (Miller & Glanz, 1996). This method is used as a testing benchmark to compare the dynamic memory allocation algorithm integrated into the UNH CMAC code. The UNH CMAC code is

written in the programming language C, providing flexibility and power to the implementation. All dynamic memory allocation implementation will therefore continue to be implemented in C. As such many of the following concepts, to help describe the process, deal with the specific C implementation.

3.1 HASHING

Typically, data storage in CMACs has been implemented via the use of hashing. This method was originally used by Albus (1981) as the preferred choice of data storage with CMACs, because it allows large multidimensional input spaces to be mapped to a small memory storage space of pre-determined discrete size. The method works well for sparsely distributed low-density data sets, so long as the number of input vectors maps to fewer weights than the maximum size the hash table will allow.

Hashing provides a very space efficient method of storing data, with only one memory block (usually an integer) used to store the data. A key is presented to the hash algorithm, which in this case is a unique vector in either the CMAC input space or receptive field space. Once the key is processed in the hashing algorithm, the output from the algorithm is an address in memory or the address of an array where the data will be stored. This simple process is illustrated in Figure 3.1.

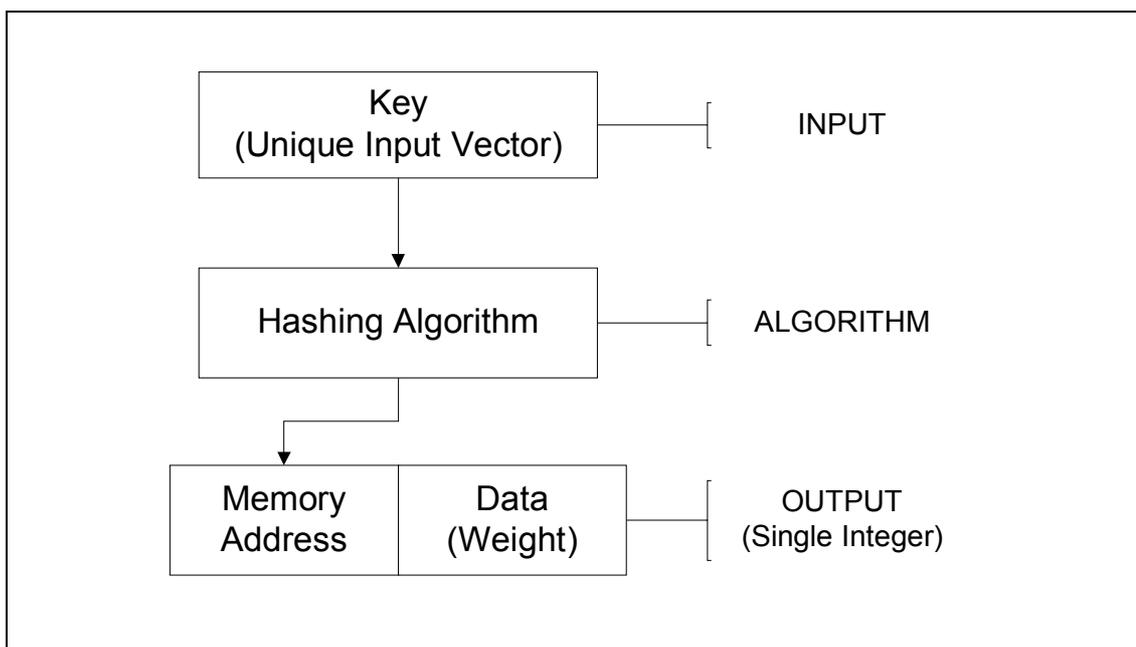


Figure 3.1 – Hashing Process Overview

The purpose of the hashing algorithm is to map evenly and randomly the corresponding generated memory A'_i addresses from key a_{ij} for receptive field layer i over the memory space of magnitude M . The UNH code uses the hashing function:

$$A'_i = \left(\sum_{j=1}^N T_j [a_{ij} \% R_j] \right) \% M \quad \text{Equation 3.1}$$

where T_j represents a single dimension array containing previously random generated values of magnitudes no less than the value of M , with R_j total entries in the array and N being the total number of dimensions in the CMACs input space. $\%$ denotes the modulus (remainder) operator.

A perfect hashing algorithm when used with a CMAC would produce a unique set of weights in the hash table for every input vector. However for most practical cases this does not occur. When more than one unique key produces the same memory address at the output of the hash algorithm, this is known as a hash collision. Thus the data stored at the computed memory address no longer becomes a unique subset of the key.

When different input vectors map to the same weight in the CMAC weight table, in the case of a hash collision, the result is noise at the CMAC output. Therefore without additional algorithms implemented together with the hash function to deal with hash collisions, noise will always occur for the majority of hashing implementations. The UNH CMAC code takes hash collisions into consideration, and allows the user to either use the CMAC with hash collisions, or without them. Their method of hash collision avoidance is called Collision-Resistant Hashing.

Collision-resistant hashing assigns an extra piece of data to each memory address produced by the hash function. This data is a random number, named the hash tag, and identifies the output to the key. In this way, when a memory address is generated via the hash function, a comparison is made between the hash tag and the key. If a match is made then the hash algorithm has found the correct memory address containing the corresponding data. However if the hash tag does not match the key then the data is stored in the next free subsequent memory address. Using collision-

resistant hashing however increases the memory requirement by a factor of 2 over regular hashing without collision-resistance. For each memory address generated, an integer is required for the data and an additional integer for the hash tag. This is illustrated in Figure 3.2.

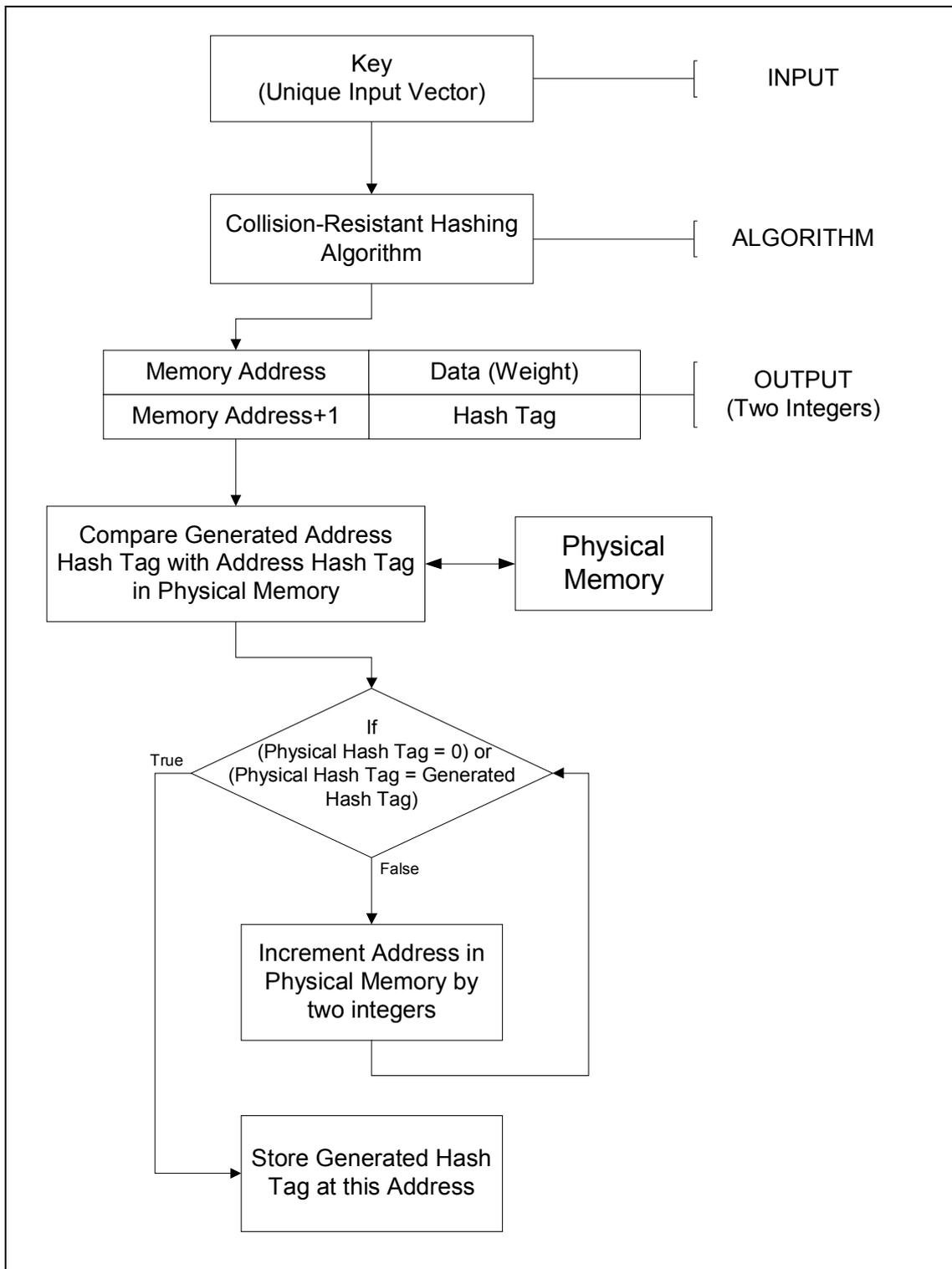


Figure 3.2 – Collision Resistant Hashing Process

For collision-free hashing, the hash-tag would have to be a unique identifier for every key inputted to the hash function. Using the key itself would be the most obvious solution as it is both unique and directly identifies itself. However consideration must be given to the number of CMAC input dimension vectors that the key is made up of and also the maximum size of key integers used on whichever software compiler the code is being implemented on. For highly multiple dimensional input spaces, the key would produce an integer larger than the maximum size permitted by the compiler and thus the integer would overflow and no longer be unique. For this reason, single dimension arrays of randomly generated values are used to generate a corresponding unique identifier from the key, providing a robust collision-resistant solution.

The UNH code authors (Miller & Glanz, 1996) argue that collision-resistant hashing will over time retain no longer useful memories compared to raw hashing (hashing without collision avoidance) which will overwrite past memories with only the most current ones. This is one of the natural properties of hashing, as any function in the input space will be stored in a memory of static size. Hash collisions will simply cause the most current collisions to write over the previous data. This argument has both benefits and disadvantages in regards to CMACs. Rarely will old memories in the general sense of adaptive systems be no longer required to be retained. A system may be trained in stages, and once training is completed all sub-systems will be required to function independently and coherently, no matter what future training the sub-systems are given. For systems where the training of every sub-system is to be totally independent from each other, the argument for hashing over collision-resistant hashing is feasible. However for systems where each subsequent training for every sub-system relies on previously learnt data from preceding sub-systems, hashing by itself will not provide adequate and accurate training results.

The two hashing methods used by the UNH CMAC code to store CMAC weight data provide a basis for benchmarking other memory storage algorithms. Their open source implementation and efficient use of memory make the UNH CMAC code a benchmark not only to test other algorithms against, but to also allow direct modification to the structure of the memory handling code to be exchanged for another. This approach leaves the core CMAC algorithm untouched, which allows

better comparisons to be made when comparing memory handling parameters, as it prevents the CMAC core code from having influences on the memory handling parameters.

Hashing however is only one of several popular data storage approaches. Where dynamic storage allocation of memory is required, a static approach such as hashing is not recommended. A data storage structure which is naturally dynamic is preferable. Such data structures can grow and shrink dynamically, requiring only the amount of memory necessary to store all current data. As opposed to hash tables which should never be completely full, dynamic allocation of memory can utilise up to 100% data storage space at any given time. Figure 3.3 illustrates this concept, comparing the memory allocation difference between a static (hashing) and a dynamic method using Binary Search Trees.

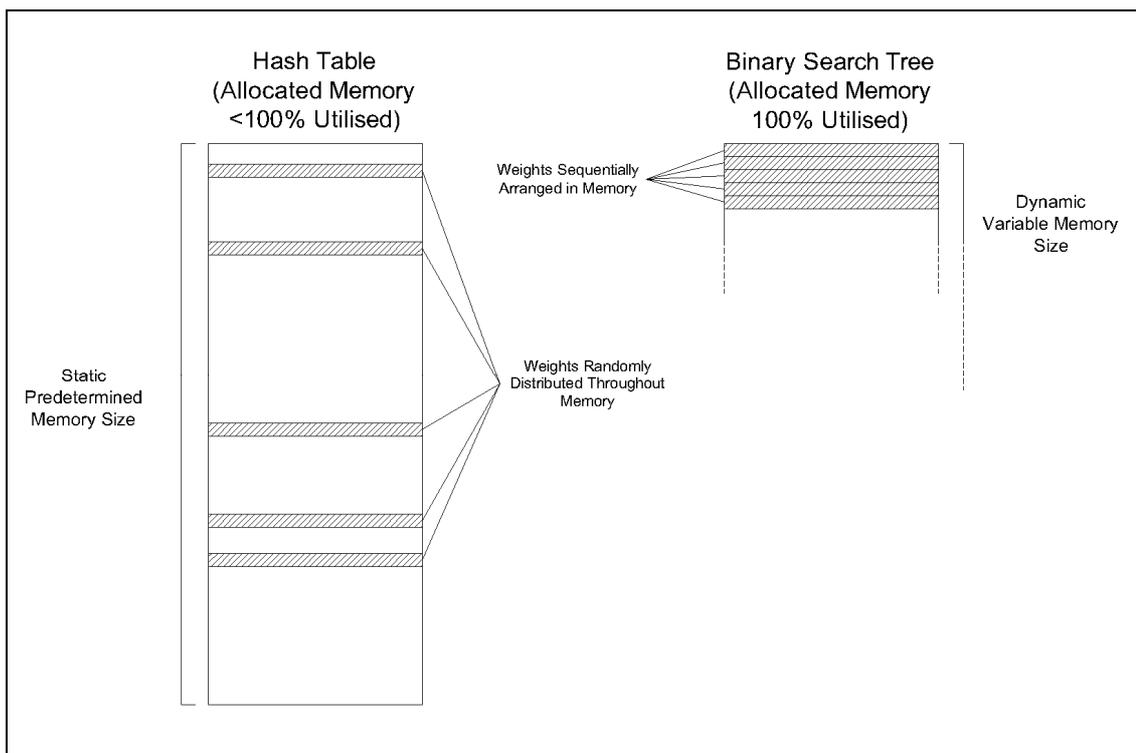


Figure 3.3 – Static Hashing vs. Dynamic Binary Search Tree Memory Allocation

When using a dynamic method for memory allocation, only the memory size required to store the current data is used, thus always utilising 100% of the current allocated memory. Binary search trees are one such method to provide dynamic

allocation of data, and it is specifically binary search trees which will be used here for dynamic memory handling with a CMAC.

3.2 BINARY SEARCH TREES

A Binary Search Tree (BST) is a very common and widely used method for storing data sets. Compared to hashing, binary search trees are dynamic, thus no advance information on the number of insertions is needed. They also provide guaranteed worst case performance, as everything could hash to the same place, even if the best hashing method available is used (Sedgewick, 1990). It is also easier to perform functions such as sorting on binary search trees as opposed to hashing. Hashing in its basic form is generally faster and the algorithm somewhat simpler, together with having consistent search times. However algorithm processing times and memory access times are factors involving such claims.

A binary search tree obtains its name from the structure by which it grows and is searched. Each piece of data is stored in a node. Each node contains a key, some data, and two memory addresses. The two memory addresses point to other nodes in memory, one which will have a key value less than the current node's key and the other which will have a key greater than the current node's key. Figure 3.4 illustrates this concept.

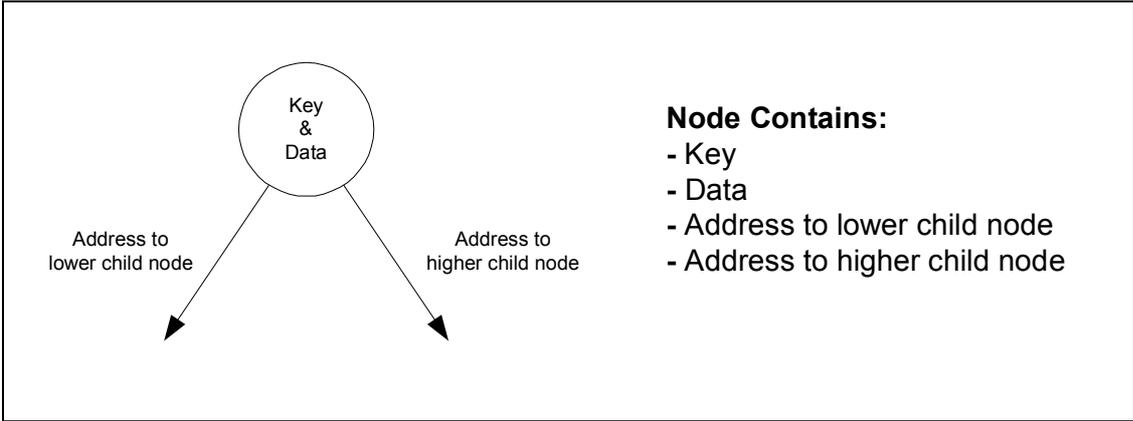


Figure 3.4 – A Single Binary Tree Node

A binary search tree then grows by adding subsequent nodes to the legs of the current node. If a node has subsequent nodes attached to its legs, then it is known as the

parent node to the nodes attached to its legs. Similarly, the nodes attached to its legs are referred to as the child nodes to its parent.

Trees grow by adding a single node at a time. For instance, take a data set of random numbers between 0 to 99, where the first random number to be generated by a random number generator is the number 43. 43 thus becomes the first node, at the top of the tree. Now take a second random number, say 67. 67 is greater than 43 and because the higher leg of node 43 is vacant, node 67 is attached to the higher leg of the parent node, node 43, and becomes node 43's child. Then take a third random number, say 12. 12 is less than 43 so it becomes 43's child on 43's lower leg as this leg is vacant. Finally, take a fourth random number, say 55. We compare 55 to 43 and see that 55 is greater, so we move to the node attached to the higher leg of node 43 since this leg is not vacant. Then we compare it with node 67 and see that 55 is less than this value. Node 67's lower leg is vacant, so we attach node 55 to the lower leg of node 67. Our simple tree of four nodes can be seen in Figure 3.5.

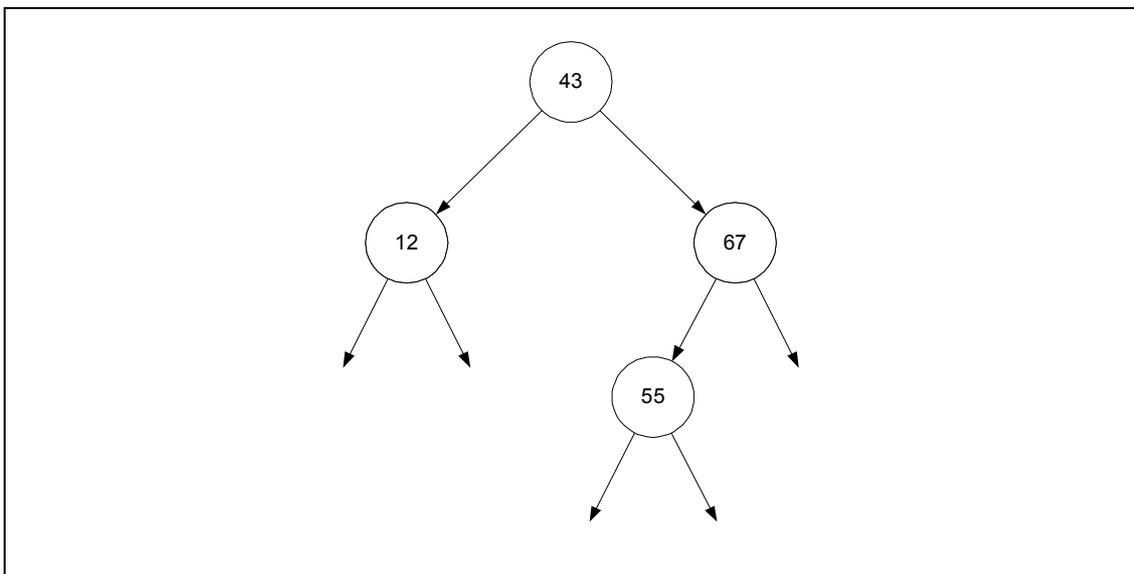


Figure 3.5 – Simple Binary Tree Containing 4 Nodes

The efficiency of a binary search tree is highly dependent on the number of nodes that must be traversed in the tree before the required node is found. Thus building the tree in a suitable order and refining the order by which the data in the tree is stored are both desirable processes to consider. For example, take a data set where the keys are sorted from lowest to highest, between 0 to 99. If a tree is built from this string of

sorted keys, the resulting tree will be one very long, single sided tree. The first key to be added to the tree is key 0, followed by 1, 2... 99. This will add every subsequent node to the higher leg of its parent. The resultant tree is known as a stick, and is illustrated in Figure 3.6(a). This is the worst situation for a binary tree generator to encounter, as the binary search tree loses its efficient search time property, and acts just like a sorted list which is to be searched linearly. This means that the maximum number of nodes required to be searched is equal to N , where N is the total number of nodes in the tree.

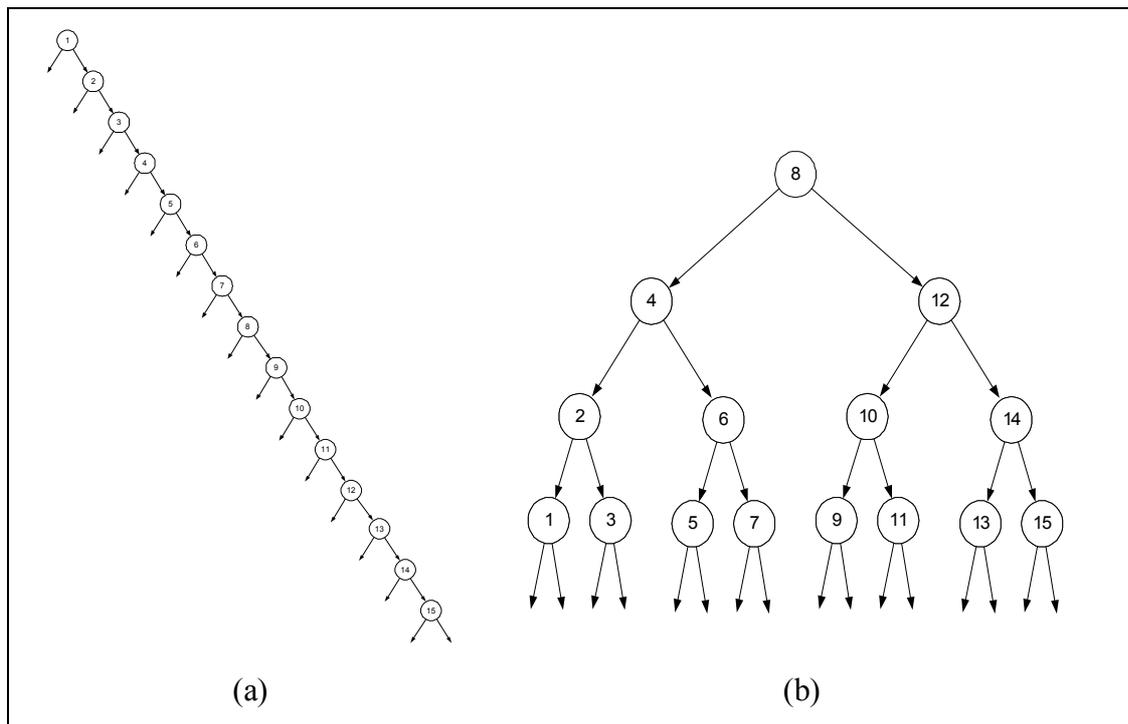


Figure 3.6 – (a) A Stick, (b) A Balanced Binary Tree

The most efficient binary search tree is one that is completely balanced. This means that an equal number of nodes exist on both the higher and lower legs of every parent node which exists in the tree. Figure 3.6(b) illustrates a balanced binary tree. In a balanced tree the maximum number of nodes to be traversed to find any node in the tree is:

$$|\log_2(N)+1| \qquad \text{Equation 3.2}$$

where N is the total number of nodes in the binary tree.

A balanced tree will therefore provide the best case performance when randomly searching for nodes in the tree, and a binary tree built as a stick will provide the worst case performance. For example, take a tree containing 2047 nodes. If the tree is balanced, then the maximum number of nodes to search through to find any node in the tree from Equation 3.2 is 11. If the tree was a stick however, the maximum number of nodes to search through to find any node is 2047. Thus it is clear to see that the structure of the tree is vitally important to make searching efficient.

While balanced trees provide the most efficient search times for the lookup of any random node in the tree, it is rarely possible to build trees which are perfectly balanced if the order of the data is not known prior to building the tree. In these situations, a good practice when building binary search trees is to insert the keys in a randomised order. This way the tree will grow and be fairly balanced automatically without having to know specific knowledge about the insertion order of the nodes (Pfaff, 2004; Sedgewick, 1990). The maximum number of nodes that must be traversed to find any key in a randomly ordered tree is usually unknown as the structure is completely random. However on average, a search in a binary tree requires about $2\log_e(N)$ comparisons in a tree built from N random keys (Sedgewick, 1990).

While online sorting algorithms exist which rearrange a binary search tree into a close to balanced tree every time a new node is inserted, these algorithms require significantly more processing time. Such trees include AVL trees, Red-Black trees and Splay trees (which also re-sort for every search) (Black, 2006). The additional time required by these algorithms however make them a poor choice when comparing them to a time efficient data storage algorithm such as hashing. For this reason these online sorting methods will not be considered here.

In conclusion, the most desirable binary search tree is a balanced one as search times will be minimized and so too will any future insertions. When the insertion order of keys is unknown, however, a randomised order is preferred as this will produce a close to balanced tree automatically. Creating a binary tree from randomised keys and then later sorting the tree to produce a balanced tree is the preferred method that

will be used here to implement binary search trees as the data storage algorithm into the CMAC.

3.3 IMPLEMENTING BINARY SEARCH TREES INTO THE CMAC

3.3.1 NODE STRUCTURE

To implement a binary search tree into the CMAC, the first thing required is to know what data each node will store. As a binary search tree is constructed solely with nodes, these nodes will also occupy space in memory, the assignment of which has to be considered. Following the UNH CMAC code which is written in C allows powerful, direct and efficient memory management to be utilised when assigning memory for the CMAC.

As illustrated in Figure 3.4, each node in the binary search tree is required to store data of two types, values and addresses. The first type contains two separate integers, the first which is the key of the node. Each node requires a key as explained in Section 3.2 to be uniquely identified amongst all the nodes in the tree. This key should represent the receptive field to which it is assigned either directly or via some mapping which preserves the uniqueness of the receptive field. The key is assigned a single 32bit integer to hold this data. The second integer, also of 32bit size, holds the weight associated with the node. This is the primary piece of data required in the node, and while other data may be added for other various CMAC requirements (though not implemented here) the weight directly corresponds in size to the data stored in the raw hashing method used by the UNH CMAC code.

The second data type consists of two addresses. As explained and illustrated in Section 3.2, each leg of a node points to its associated child node. As these nodes are contained in usually random addresses in memory, each node requires two addresses, also known as pointers in C. Each pointer points to its child node contained somewhere within the allocated memory space. One address points to the node's lower child node and the other address points to the node's higher child node. As the code is written to be implemented on 32bit CPUs with 32bit memory addressing, each pointer is therefore a 32bit address.

The structure of data contained in each node as it would in the computer's physical memory is illustrated in Figure 3.7. This is implemented in C via a structure type. Four separate segments of data, each of 32bit size requires 16bytes of memory per node.

Physical Memory Address (in bytes)	Data Stored in Memory Address
0x00000000	32bit integer (key)
0x00000004	32bit integer (weight)
0x00000008	32bit address (pointer to lower child node)
0x00000012	32bit address (pointer to higher child node)

Figure 3.7 – Structure of Data Contained in Each Node within Physical Memory

3.3.2 NODE MEMORY ASSIGNMENT

When a node is required to be inserted into memory, memory has to be assigned prior to insertion, with the knowledge about what type of data the memory is going to contain. As the data structure of each node is known, memory must be allocated to this structure type. The purpose of using binary search trees over hashing is their dynamic property. Only the excited receptive fields in the history of the CMAC's training are required to be allocated weights in memory. However singularly assigning memory every time a node needs to be inserted into a binary search tree is somewhat inefficient.

A common technique when dynamically allocating memory in C is to allocate a small section of memory which can accommodate a number of nodes, the number which is determined by the requirements of the program. This method proves to be more efficient as opposed to assigning memory individually for each node when it is required to be inserted into the tree in memory (Bentley & Sedgwick, 2001). For example, memory can be allocated dynamically in allotments of 1MB instead of allocating memory for every insertion instance. As mentioned, each node requires 16 bytes of memory to store all its data, and therefore 1MB will allot space in memory for 65536 nodes. Depending on the CMAC requirements, this is usually more than enough nodes for any online training session. However the flexibility here allows this value to be adjusted to any allotment size depending on the CMAC training requirements.

In systems where 1GB or more memory is available to a CMAC, dynamically allocating 1MB at a time means that only used memory within 1MB will ever be used and saved, keeping the dynamic property of using a binary search tree. Hashing on the other hand would require a fixed amount of memory prior to any CMAC training, and if up to 1GB is required, then 1GB would have to be assigned and saved every single time the CMAC was being used.

Note that assigning memory to store nodes is only required when inserting new nodes into the binary tree. If while online the 1MB buffer becomes full, then another 1MB of memory is assigned automatically online the instant it is required. The time for this is insignificant compared to assigning memory for each node every time a new node is inserted, and reduces the instances of memory assignments by 65535 when assigning memory in 1MB size allotments.

3.3.3 NODE KEY GENERATION

Generic to the CMAC is the property by which each excited receptive field directly corresponds to a weight in memory. Therefore the key, by which each node in the binary search tree can be identified, can be the vector of the excited receptive field itself.

The UNH CMAC code identifies each receptive field by a corner of the hyperfield in input space resolution. The corner of the receptive field chosen is the smallest input space vector for each dimension. Thus the receptive field input space corner vector is the initial key by which the hash algorithm produces its corresponding memory address. Figure 3.8 illustrates how the excited receptive fields from input vector (3,9) produces corresponding keys (1,7), (2,8), (3,9) to uniquely identify each excited receptive field A_c , G_j , and N_q respectively.

This method produces unique keys directly mapping each excited receptive field to its weight in terms of the receptive field's location in input space. For a four-dimensional CMAC input space with each dimension having a resolution of 2048 segments, unique vectors for all the possible excited receptive fields in the total input space would require an 11 bit number for each dimension.

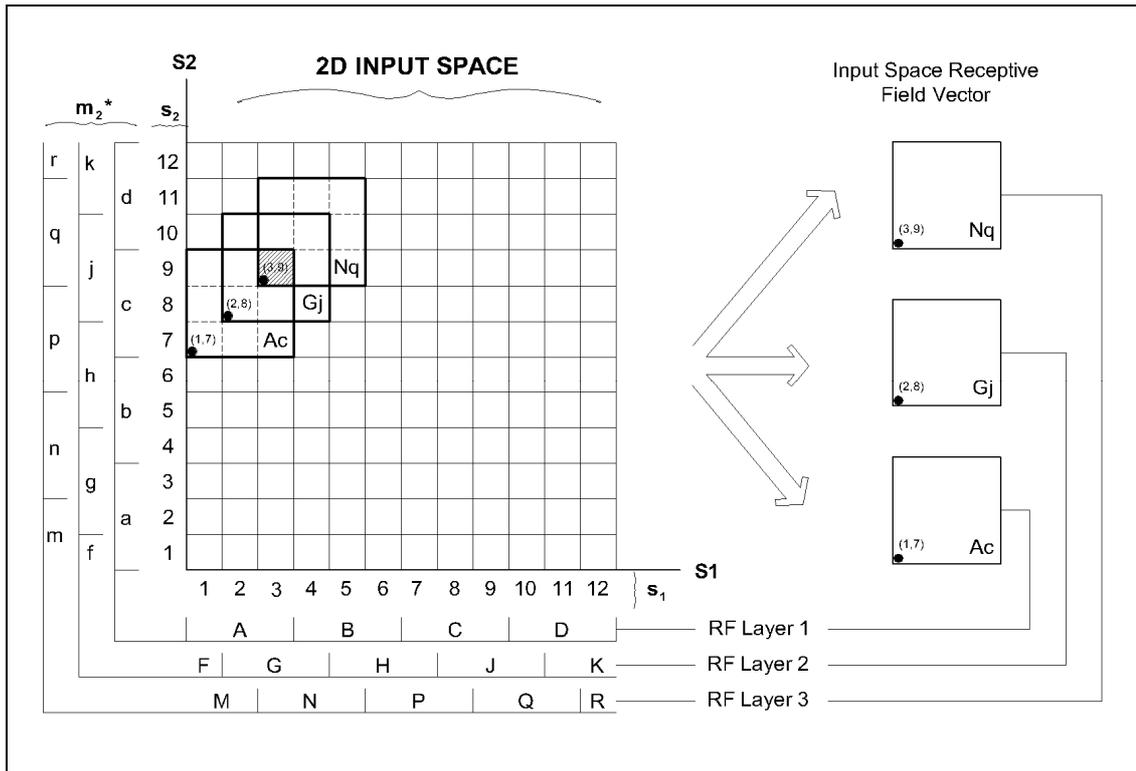


Figure 3.8 – UNH CMAC Code Receptive Field Key Assignment

Therefore 44 bits are required to store any vector uniquely in memory. This means that each key would be required to be made up of four 11bit integers, producing a single 44 bit integer where each 11 bit dimension vector is shifted by 11 bits.

This however is not the most space efficient method of storing unique vectors for each excited receptive field, though this statement is dependant on the number of receptive field layers used in a particular CMAC. If a single receptive field is to be assigned for each input vector, then the CMAC loses its local generalisation property totally, and the memory truncation property of using receptive fields disappears. Thus for the majority of CMACs, local generalisation is a property to be desired and so too is the input space memory truncation. For this reason we will consider CMACs with a receptive field generalisation parameter of 8 or above.

CMAC receptive field layers exist in parallel to each other, with each receptive field having its own vector in receptive field space for each receptive field layer. These layers co-exist and it is the sum of the weights corresponding to each parallel receptive field which produces the CMAC output. Take a two-dimensional CMAC input space with 8 receptive field layers per dimension. Each receptive field will

occupy a space of $8 \times 8 = 64$ input space resolution units. The lowest corner vector of this receptive field is used as the key in the UNH CMAC code hash algorithm. However while this receptive field occupies 64 input space resolution units, it occupies a single 1×1 resolution unit in its own receptive field layer. Thus this receptive field is unique in its own receptive field layer. For an input space resolution of 2048 units per dimension, with 8 receptive field layers, and each receptive field occupying 8 input space resolution elements, $2048 \div 8 = 256$ receptive field resolution elements per receptive field layer. Thus it is possible to uniquely identify each receptive field with only 8bits ($256 = 8\text{bits}$) per dimension. Figure 3.9 illustrates this concept but on a smaller scale. Here there are 3 receptive field layers and the unique receptive field vectors have been extracted from the input space into receptive field space in their separate receptive field layers. For receptive fields Ac, Gj and Nq, the input space vector representation is (1,7), (2,8), (3,9) respectively, and in receptive field layer space, they are represented by the vectors (0,2), (0,2), (0,2).

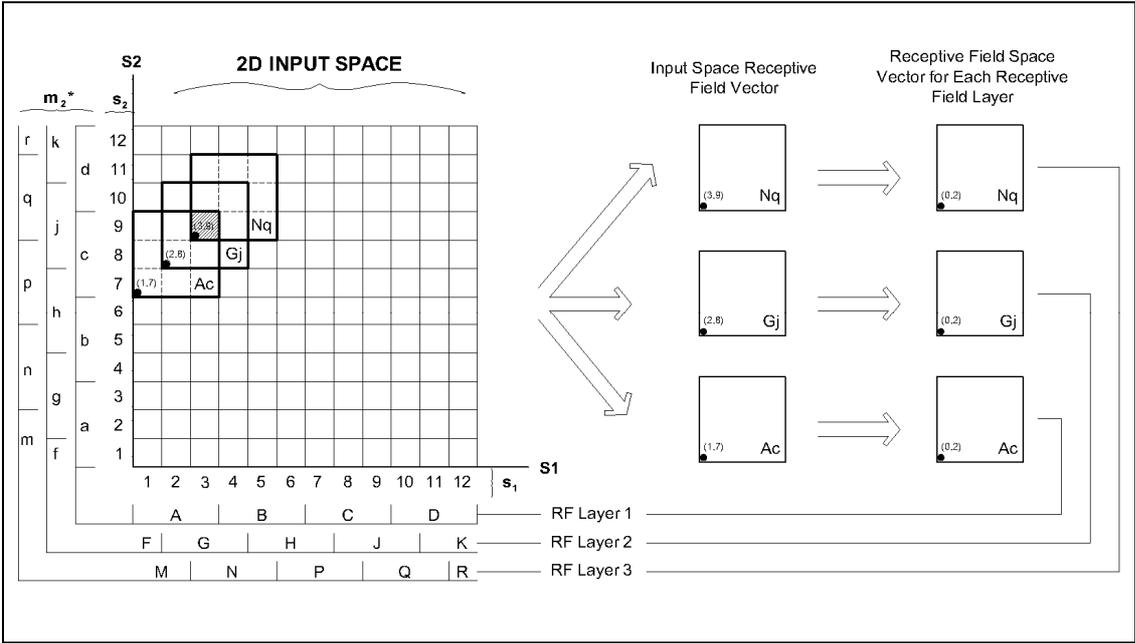


Figure 3.9 – Input Vector to Unique RF Vectors in Receptive Field Space

While it is possible for more than one receptive field to contain the same receptive field vector as another receptive field in another layer such as in this example, because the layers exist independently parallel to each other, data for each excited receptive field can be contained in totally separate memory spaces. This means that a separate binary search tree can be used to store data for each receptive field layer and

also that the 8bit receptive field vectors per dimension can be used as the unique keys for each node in its binary search tree. This makes it possible to store a unique key belonging to a 4-dimensional CMAC input space with each dimension having a resolution of 2048 segments in $8 \times 4 = 32$ bits of data, as opposed to 44bits previously.

The UNH CMAC code had to be modified to produce the parallel receptive field vectors from the input space corner vectors for each excited receptive field. The equation the UNH CMAC code uses to determine the corners of the receptive fields in the input space is as follows:

$$I[i] = \begin{cases} Q[i] - ((Q[i] - B[i]) \% G) & ; Q[i] \geq B[i] \\ (Q[i] + 1) + ((B[i] - (Q[i] + 1)) \% G) - G & ; Q[i] < B[i] \end{cases} \quad \text{Equation 3.3}$$

where I is the lowest corner coordinate for dimension i in the input space of the receptive field, Q is the input vector in dimension i , B is an incremental value which displaces the location of the next receptive field layer by a predetermined amount depending on the distribution pattern of the receptive fields, and G is the number of receptive field layers in the CMAC. $\%$ denotes the modulus (remainder) operator. $I[i]$ is used by the UNH CMAC code as the initial key for a single receptive field in dimension i . This key is then manipulated to provide a suitable hash key which allows a degree of randomness to result for better hash table distribution. This process is then repeated for every G receptive field layers.

For the binary search tree receptive field key however, a mapping from the input space to the receptive field space must occur to produce the corresponding receptive field space vector. This requires the following to be carried out on $I[i]$. First a value must be added to $I[i]$ to map $I[i]$ onto a virtual space which can then be divided into receptive field segments. Depending on the generalisation parameter n , $I[i]$ will produce n receptive fields, each displaced by the product of the displacement factor D and the current receptive field layer j . The product of D and j however may result in a receptive field which does not lie over the input vector, thus the modulus of the product must be taken. This value is then added to $I[i]$ and then divided by the size of

a single receptive field for the given dimension i . Thus the equation developed becomes:

$$R_j[i] = (I[i] + ((D[i] * j) \% n)) \gg M \tag{Equation 3.4}$$

In the equation above, $R_j[i]$ is the receptive field vector for dimension i in receptive field layer j . M is the binary right shift value which corresponds to the generalisation parameter n where $n = 2^M$. Thus this means that only generalisation parameters which are equal to 2^M for positive integer values of M which are greater than 3 (for memory preservation) can be used throughout the code. The value of $R_j[i]$ will therefore be limited between 0 to 255. % denotes the modulus (remainder) operator.

As mentioned earlier, a 32bit number can hold four 8bit integers. Using this dimensional limitation for the current code, the receptive field vectors for each dimension i will be allocated in series to create a unique node key for the binary search tree for up to four dimensions. Figure 3.10 illustrates this process.

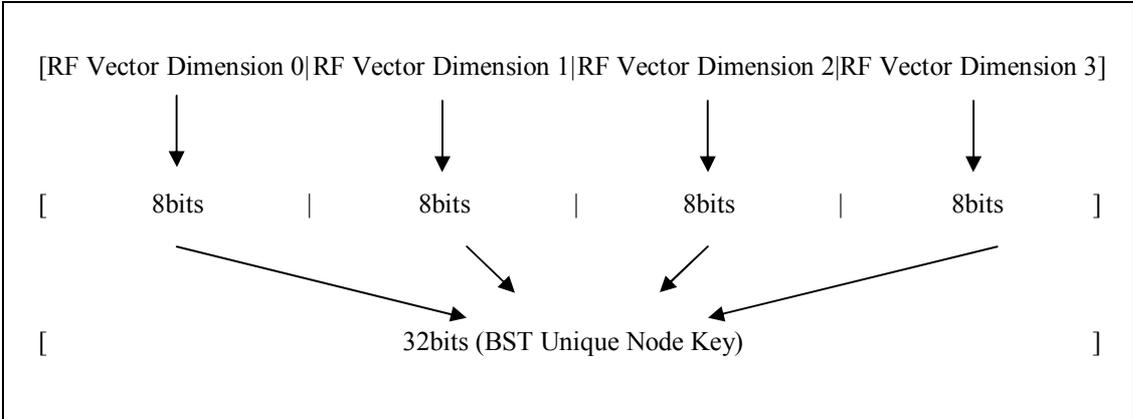


Figure 3.10 – Four Receptive Field Vectors Represented as a Single 32bit Integer

The parallel receptive field layers mean that two or more receptive field vector node keys may produce the exact same key. However since the receptive field layers exist in parallel to each other, this is both expected and of no concern as separate binary trees are assigned to each receptive field layer. Once the key is produced, a randomised process must be used to allow a close to balanced tree to be produced automatically. Then the binary tree can be searched to find the address of the corresponding weight associated with the key.

3.3.4 BINARY TREE SEARCHING

The CMAC input space motion trajectories that train the CMAC always occur in the form of strings. In other words each input vector is succeeded by an input vector directly adjacent to it for all positions except at the end of the string. This causes a problem when adding nodes to a binary search tree as the tree will be the result of a number of sticks, far from an optimal balanced tree.

As mentioned in section 3.2, the binary search trees are to be built online using random keys to produce a close to balanced tree. Thus to solve this problem, the receptive field vectors for each dimension are scrambled to produce a corresponding random vector. This process provides a unique random output vector for any unique input vector, directly traceable to the original vector. This means that the same unique vector will be generated for every single unique receptive field vector.

This receptive field vector scrambling was achieved by producing an array equal to the size of the resolution of a single receptive field layer. This array was then filled with random unique integers from 0 to the maximum receptive field layer vector value.

For each receptive field vector for dimension i , the value of the vector corresponds to an equivalent address in the array that holds a random value. It is this value stored at this address in the array that is then used in place of the original receptive field vector to be used as one of the 8bit segments in the key for a node in the binary search tree. In Figure 3.11 below, for a random array of size 8bits per segment, a receptive field vector for dimension i of value 3 would map to 104. This value of 104 would then be used as the key for dimension i as the unique identifier for the node in the binary search tree.

The keys generated for each dimension i are the keys used as the RF Vectors for their appropriate dimensions in Figure 3.10. The final 32bit key is then used directly as the key for a node in the binary search tree for the key's corresponding receptive field layer.

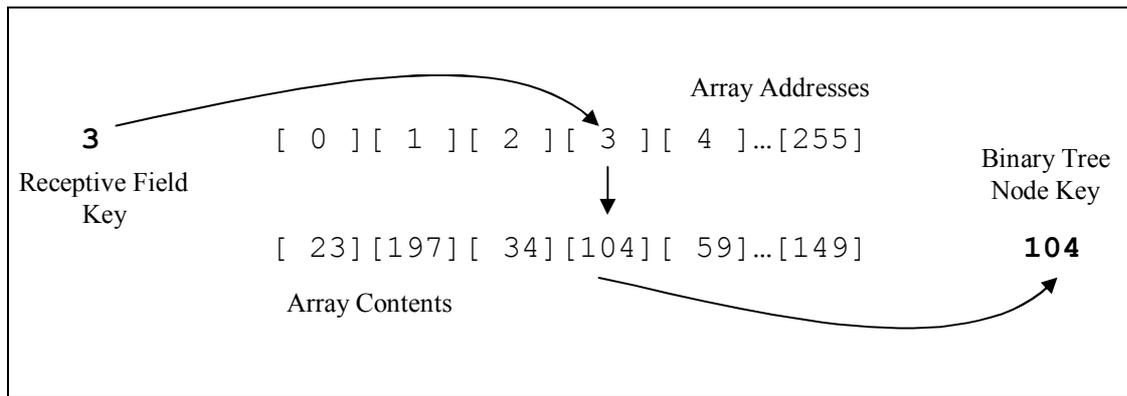


Figure 3.11 – Random Node Key Generator Array

Depending on which receptive field layer node is currently active, the corresponding binary search tree is searched to see if this node already exists. This is completed by starting from the very top node in the tree, which is located at a previously assigned known memory address. The key of this node is compared to the current key being searched for. If the key is less than the key of the top node, then the top node’s lower leg address is used to jump to the child node for the next comparison to the current key. Similarly, if the key is greater than the key of the top node, then the node’s higher leg address is used to jump to the child node for the next comparison. The memory structure of each node is that of Figure 3.7. This process is then repeated for each node traversed until either the leg of a node contains no address, in which case the new node is assigned to the next free available address in memory and the new node is stored here, or until the corresponding node with the same key is found. This tree searching process is illustrated in Figure 3.12.

Once the node has been found or newly inserted in the tree, then the address of this node is returned. This process occurs for all receptive field layer trees until all have been searched. The code then returns a list of pointers (addresses) to the CMAC code that can then directly access the node whenever it is needed to read and write to the weight data in the node. This is how the binary search tree is implemented in place of hashing to provide a dynamic memory management alternative to the CMAC.

Creating the binary tree this way using the random key generator creates a close to balanced tree for whatever sequence of input vectors the CMAC encounters.

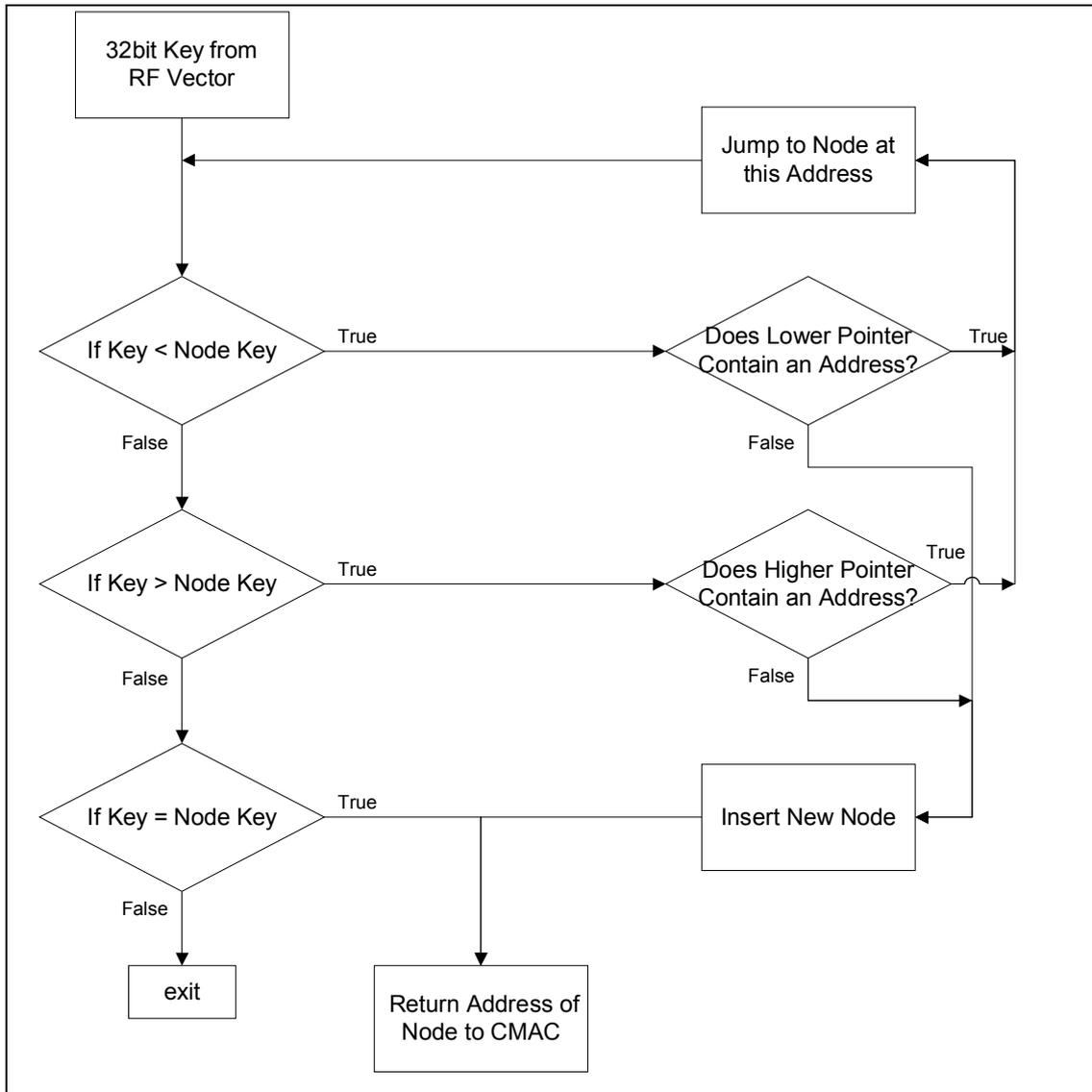


Figure 3.12 – Binary Tree Node Searching and Insertion

However as tests reveal in Section 4.0, an optimally balanced tree can search for and insert nodes more efficiently than when the tree is created from randomly chosen nodes. For online tree creation, the random key generator requires negligible processing overhead and provides insertion and searching efficiency compared to if the nodes were inserted in the direct order before the random key generator. The CMAC however will never be constantly online. When offline, it is possible to re-sort the nodes in the CMAC's binary search trees to create optimally balanced trees. The benefit of this is that when the CMAC returns online, all future search and insertion times will be optimised.

3.3.5 BINARY TREE BALANCING

When the CMAC is offline, the CMAC memory, i.e. the nodes in the binary search trees, can be rearranged in the most efficient order to create balanced binary trees. To do this, an algorithm was created which first sorted all the nodes in a tree. The unbalanced tree is searched throughout from lowest node key to highest and the locations of each node in memory is stored in a sorted string. This produces a string of key addresses from lowest key to highest key. The string is then halved finding the middle node of the total string. This node then becomes the new top node of the tree, and its lower and higher pointers are the new mid-nodes of the two remaining sticks. This process is repeated recursively until no more nodes exist. The end result is a balanced tree.

Figure 3.13 illustrates the above process using a simple example of 7 nodes. First we start with an unbalanced binary tree. The tree is then sorted into a string from lowest to highest. The string is then halved to find the middle node of the string. In this case the middle node is 4. This node is then extracted from the string and becomes the new top node of the new balanced binary tree. The removal of node 4 leaves two smaller strings, of nodes 1-3 and 5-7. The same is continued recursively for each of these strings. The middle node is extracted and becomes the child node of the node above it. This process continues until all the nodes in the sorted string are assigned to their appropriate location in the new balanced binary tree.

It should be noted that the actual location of each node relative to each other is not altered in the process. The only alteration that occurs is the change in addresses of child nodes which each parent node points to. The relationship between node addresses in the tree is purely relative. Every time a CMAC binary tree is loaded into RAM, the addresses pointed to by each parent node will be incorrect as the whole tree will more than likely be shifted by some factor. Therefore for every new loading of CMAC binary trees into RAM a reordering of the addresses in each node need to be altered to point to the correct current address in memory. This process is completed every time a CMAC is loaded into RAM from the hard disk (or other storage device) based on the difference between the previous top tree node address to

its current address. This way each of the addresses in the tree can be altered by this relative amount.

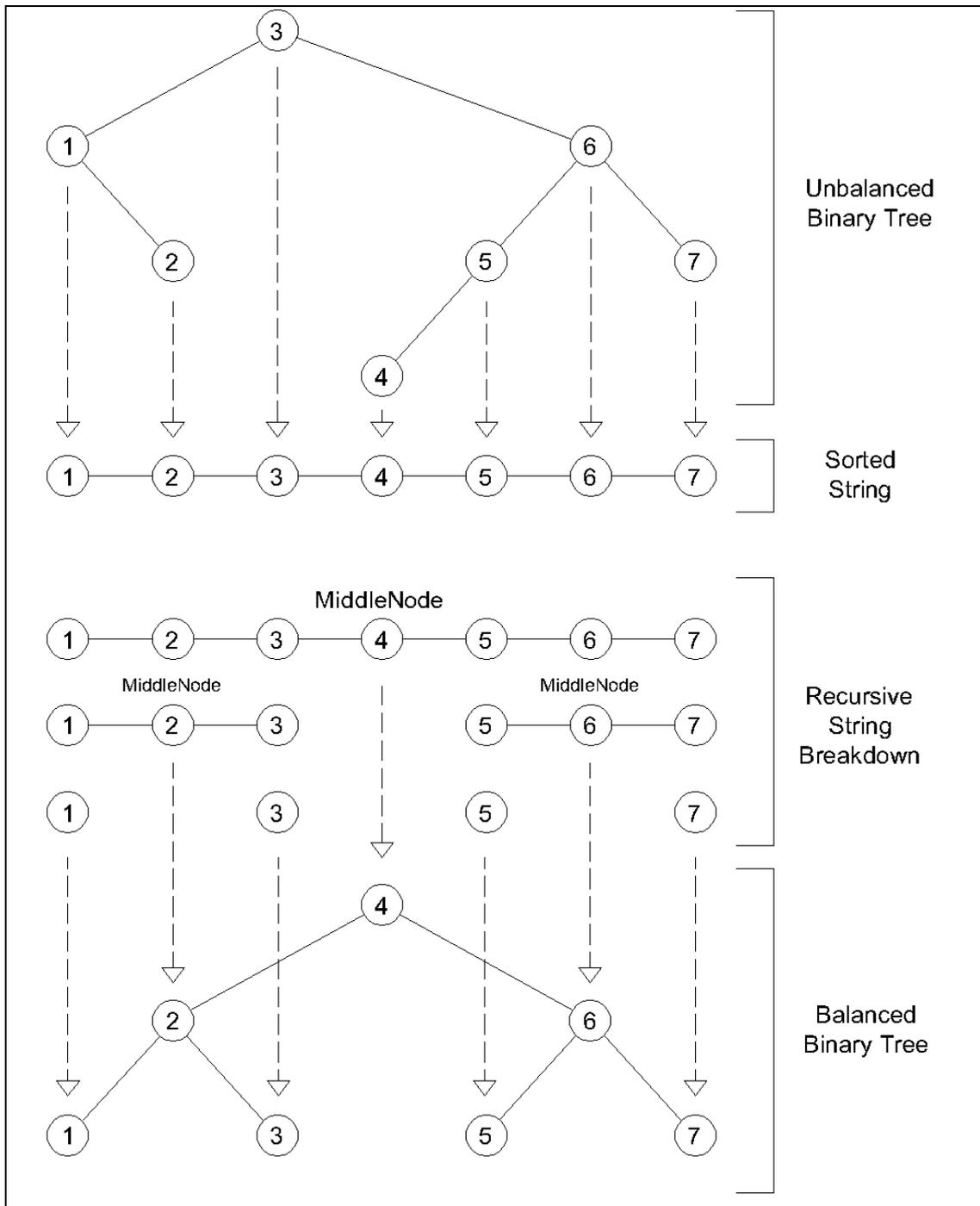


Figure 3.13 – Binary Tree Balancing – From Unbalanced to Balanced Tree

The two methods, hashing and binary search trees, used as the memory storage methods with a CMAC will now be tested and compared in the following chapter.

4.0 COMPARISONS BETWEEN BST AND HASHING WITH THE CMAC

This section compares test results between the binary search tree algorithm implemented with the CMAC against the UNH hashing memory allocation method previously explained in Section 3.3. For the test system setup below, results indicate that binary search trees are a practical alternative to hashing, with each method possessing its own relative strengths and weaknesses for certain CMAC parameters.

The following tests have been implemented on the following system for consistency:

Processor	2.8Ghz Pentium D (w/Intel 945P Express Chipset, FSB 1066Mhz)
RAM	4 x 512MB (667MHz) Dual Channel Configuration
Operating System	Windows XP Pro
Compiler	Intel 9.1 C++ Compiler (Intel processor optimisation enabled)

Table 4.1 – Processing Hardware and Software Used for Testing BST and Hashing CMAC Code

The Intel 9.1 C++ Compiler optimises the code for Intel based CPUs into discrete threads to make use of the dual processors in the Pentium D. However only the parts of the code capable of being executed in parallel are broken up into such threads. For the UNH CMAC hashing code and the new BST CMAC code, writing and rewriting the code to make use of the parallel code executions has not been undertaken and thus actual implementation of the program is only executed using a maximum 50% processor resources within the Pentium D PC.

It is widely known and accepted that hashing methods are superior in speed to binary search trees, with hashing being the preferred choice when high search speed is required and memory space is required to be optimised for a given number of storage keys. The reasons behind such claims lie within this main reason: Hash algorithms can generate a hash tag much faster than a key can be searched on average throughout a BST. However this generalisation is totally dependent on the speed by which the memory in a computer can be accessed. Factors such as bus speed, etc., corresponding to the memory access time in a PC affect the speed by which memory contents can be accessed. In general however, a BST implemented on a system with

very fast memory access times can be searched much faster than a processor with a slow clock speed computing a hash tag from a complex hash algorithm.

Nevertheless, for the practical current test system established above, for moderate sized binary search trees, and considering the dynamic capabilities of the BST memory implementation, the results to be shown appear quite practical as an alternative to hashing.

The following sub-sections deal with several tests which compare the build, search and training times of CMAC weight tables using binary search trees and the UNH hash resistant collision form of hashing. The tests deal with the effect CMAC factors such as the number of receptive fields (generalisation parameter), the input of input vectors, etc. have on the relative performance of BSTs and hashing, and also the effect that tree balancing has upon BST performance. However first a few considerations must be discussed before the various tests can be presented.

4.1 PRE-TEST CONSIDERATIONS

Two considerations must be discussed here before further tests are presented. The first deals with raw hashing which produces hash collisions, and the second deals with binary search tree balancing.

4.1.1 HASHING WITH AND WITHOUT HASH COLLISIONS

The UNH CMAC code incorporates two methods of hashing as discussed in Section 3.1: raw hashing (with collisions) and hash-resistant hashing (without collisions). The following tests comparing BSTs to hashing compares BSTs to the UNH hash-resistant method only.

The reason that raw hashing is omitted from the tests lies in the fact that the hash collisions produced from raw hashing produces noise at the CMAC output. A test was performed to see how many hash collisions actually occur within a hash table for a given set of stored data. The data was gathered from training a CMAC using the UNH CMAC raw hashing algorithm, trained in a single instance with a fixed set of input vectors, producing 65,000 weights. The size of the weight table however varied for each test, producing different weight table utilisation percentages. The hash collision results are illustrated in Figure 4.1.

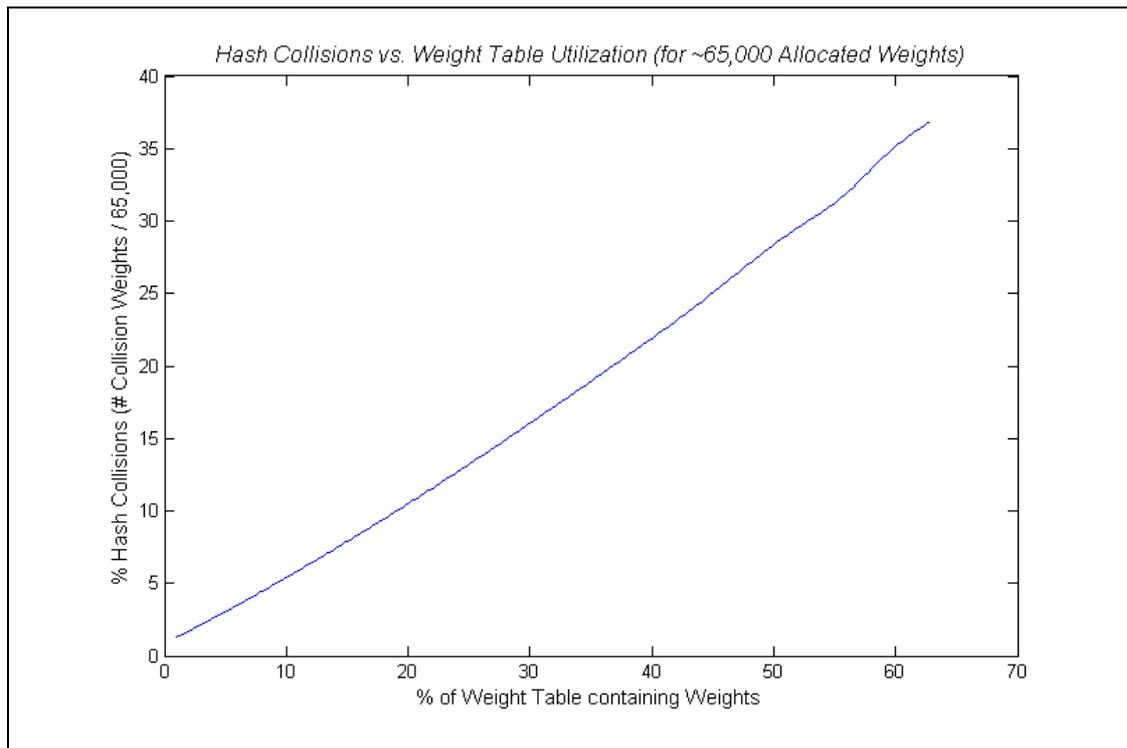


Figure 4.1 – Hash Collisions vs. Weight Table Utilisation (for ~65000 Weights)

Figure 4.1 illustrates that as the weight table percentage utilisation increases, the percentage of hash collisions increases almost proportionally. This means that noise is evidently present at the CMAC output for all but VERY large hash tables. For the noise at the output using this method to be negligible, the size of the hash table needs to be at least an order of magnitude greater than the number of weights in the table. The maximum number of collisions for a hash table with a size of 65,000 available weights was 37%. However the table was only 63% full, not close to 100% as would be initially expected. As the number of hash collisions increase, many of the weights are overwritten and thus many remain empty. It is this reason which produces a 63% full table even when 65,000 weights are allocated. As the CMAC output is a functional average of several weights determined by the sensitivity function, with 37% of the inserted weights resulting in hash collisions, the CMAC output would produce a significant amount of error. This error results as output noise, and as such raw hashing will be omitted from any further testing.

4.1.2 BINARY SEARCH TREE BALANCING

All BST search times and training times which will be presented in the following tests involve searching through BSTs for required CMAC weight data. As discussed

previously in Section 3.3.5, BSTs are searched more efficiently when the BST is balanced.

A test was conducted to see the efficiency gain in searching a balanced BST to a randomly build BST. Several CMAC builds were generated for different numbers of input vectors with a fixed generalisation parameter of 16. The BSTs were then searched using 100% of the already trained input vectors. First a search was performed without tree balancing, using the raw randomly generated binary tree. The BST was then balanced offline, and the test performed again. The results are illustrated in Figure 4.2.

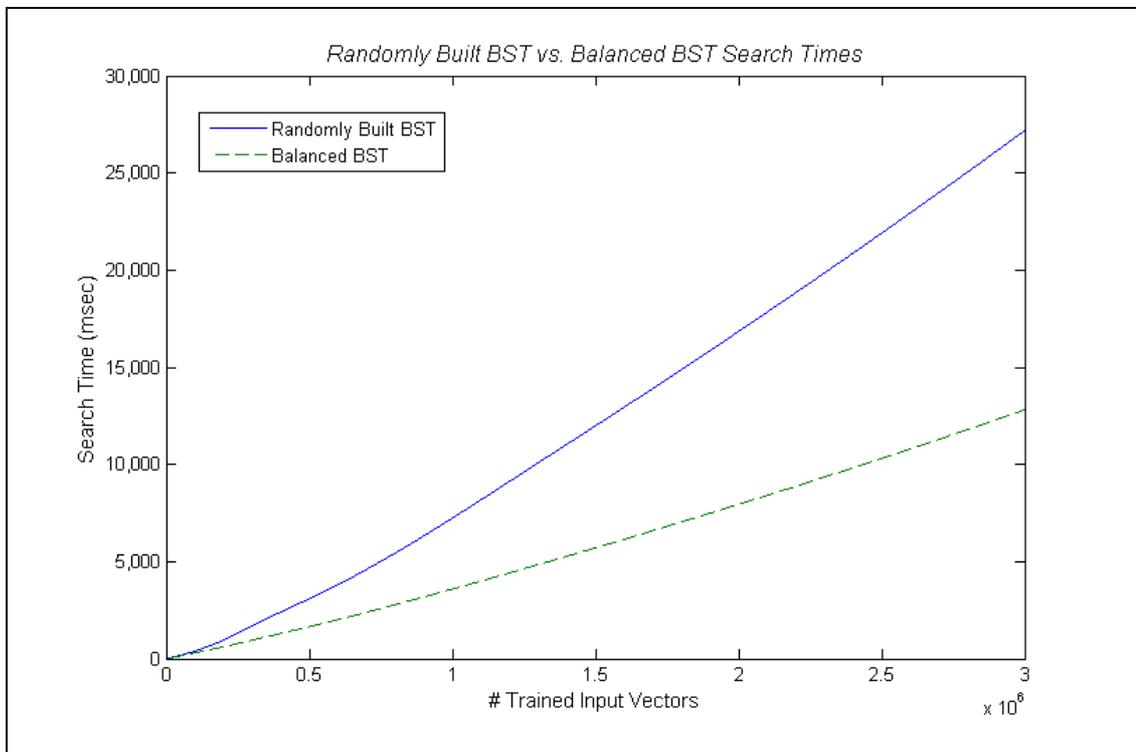


Figure 4.2 – Randomly Generated BST vs. Balanced BST Search Times

Figure 4.2 illustrates the considerable efficiency gain in searching a balanced BST over a BST generated from randomly inserted nodes. Thus offline balancing is an important step when achieving optimal performance using CMACs with BST memory handling routines.

With the reasons for using only collision-resistant hashing and balanced BSTs discussed, all further mentions of ‘hashing’ deals with collision-resistant hashing, and all BSTs searched and trained are balanced, unless stated otherwise.

4.1.3 TEST 1: WEIGHT TABLE UTILISATION

The first test will now be presented in two parts, building and searching, dealing with the effect that weight table utilisation percentage has upon BST and hashing performance.

The size of space in memory allocated to the storage of CMAC data is a factor effecting hashing and BST memory storage methods, specifically the amount utilised. For hashing, the size of the allocated memory must be assigned before any CMAC training can be undertaken, and this memory size will remain constant throughout the lifetime of the CMAC training. For BSTs, the memory is allocated on the fly when needed, in predetermined blocks, every 1MB or as desired. This way the CMAC can use only as much memory as is required, and makes the training phase of a CMAC unrestricted by predetermined memory size allocation.

For the following tests below, the BST memory size is allocated in a single instance of predetermined size, in the same manner by which the hash memory allocation is created. The following CMAC parameters were used:

Sensitive Function	LINEAR
Input Dimensions	2
Output Dimensions	1
Number of Trained Input-Space Vectors	1000 x 1000
Number of Receptive Fields	16

Table 4.2 – Test 1 CMAC Parameters

A two-dimensional input space was trained using 1000 x 1000 unique input space vectors, each trained once using the function:

$$z = 1001 + 1000 * \sin\left(\frac{x}{100}\right) * \sin\left(\frac{y}{100}\right)$$

Equation 4.1

This function produces a CMAC input to output space mapping as illustrated in Figure 4.3, where x and y are the two-dimensional input vectors and z is the output vector.

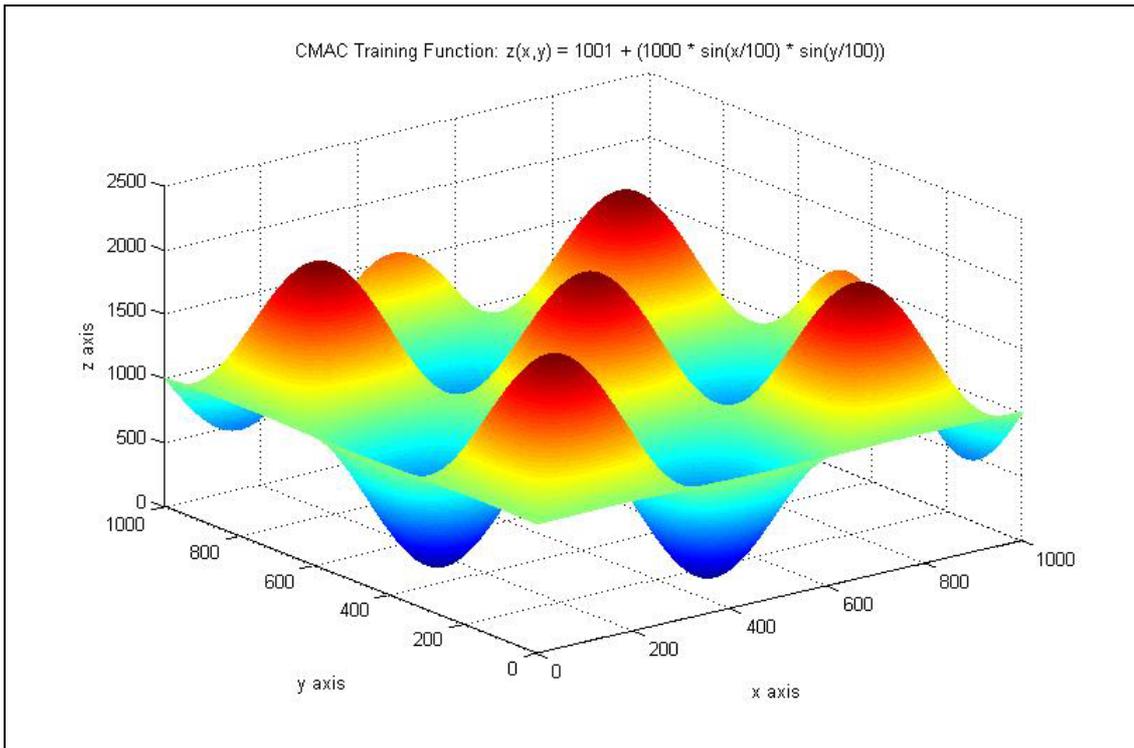


Figure 4.3 – CMAC Training Function

The training procedure of a CMAC in its initial build involves training each point once in the input space area of 1000 x 1000, producing 1,000,000 separate training points. Each training point maps to its equivalent unique set of 16 receptive fields which are then trained using a linear sensitivity function. The weights for each receptive field are stored in their corresponding hash table or BST, in memory. The time taken to insert the CMAC weights into memory, called a build, was recorded for both BST and hashing memory handling methods.

4.1.4 TEST 1.1: BUILD TIMES

The memory allocation size test was used to see the effect that different sized memory allocation sizes had on the percentage of the memory size which actually

contained weights. For hashing, a CMAC’s memory size must be determined and fixed before any training is performed. Take for example a CMAC where in its initial training session, 65,000 weights are trained. If only enough memory is allocated to store a maximum of 70,000 weights in a hash table, then the table is approximately 93% full. Depending on the memory allocation method used, the time used in storing these weights in memory during an initial build will differ. For the memory allocation methods tested, the average time taken for each initial CMAC build (over 8 samples) is illustrated in Figure 4.4.

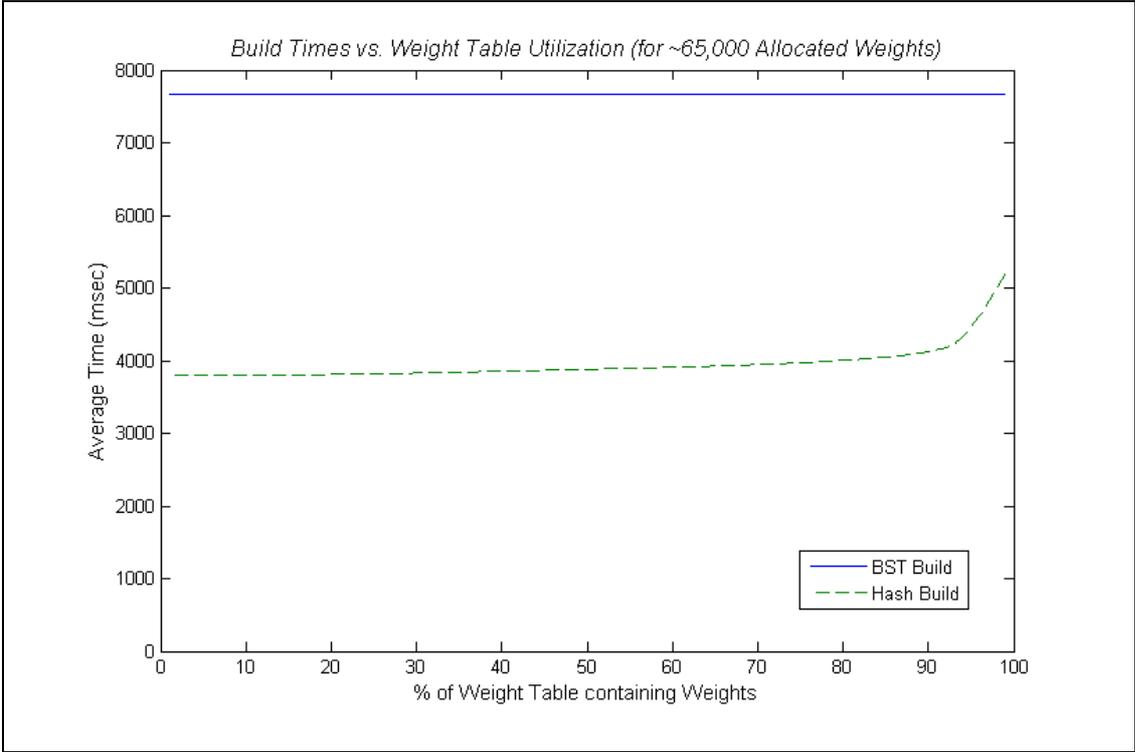


Figure 4.4 – Build Times vs. Weight Table Utilisation (for ~65000 Allocated weights)

Figure 4.4 illustrates that the BST method is not dependent on the percentage weight table utilisation, whereas the hashing method is. Building the CMAC weight table using hashing without collisions resulted in a rather sharp time increase as the weight table utilisation approached 100%. For the majority of the weight table’s utilisation, the build time remains consistent until approximately 80% of the table is full. Thus highly utilised hash tables take longer to build than near empty hash tables, yet are still faster throughout than the BST method.

The BST build using the BST random node insertion method developed requires substantially more time overall compared to hashing. However it should be noted that this time is associated with a complete build of 65,000 weights at once: the whole BST has been created from start to finish randomly, inserting 65,000 nodes into the tree in a random order.

4.1.5 TEST 1.2: SEARCH TIMES

Since the BST has been created by randomly inserting nodes into the tree, the tree should be balanced to produce a BST that is more efficient to search for the already contained nodes. This balancing process is completed offline, and once tree balancing has been completed and the previously allocated 65,000 weights sorted into a balanced binary search tree, searching through a CMAC's memory structure can be used to find any output corresponding to one of the 1,000,000 trained input vectors. Without any further training and pure CMAC input to output vector mapping, for the 65,000 trained weights previously stored in memory, Figure 4.5 illustrates the comparisons between memory allocation methods.

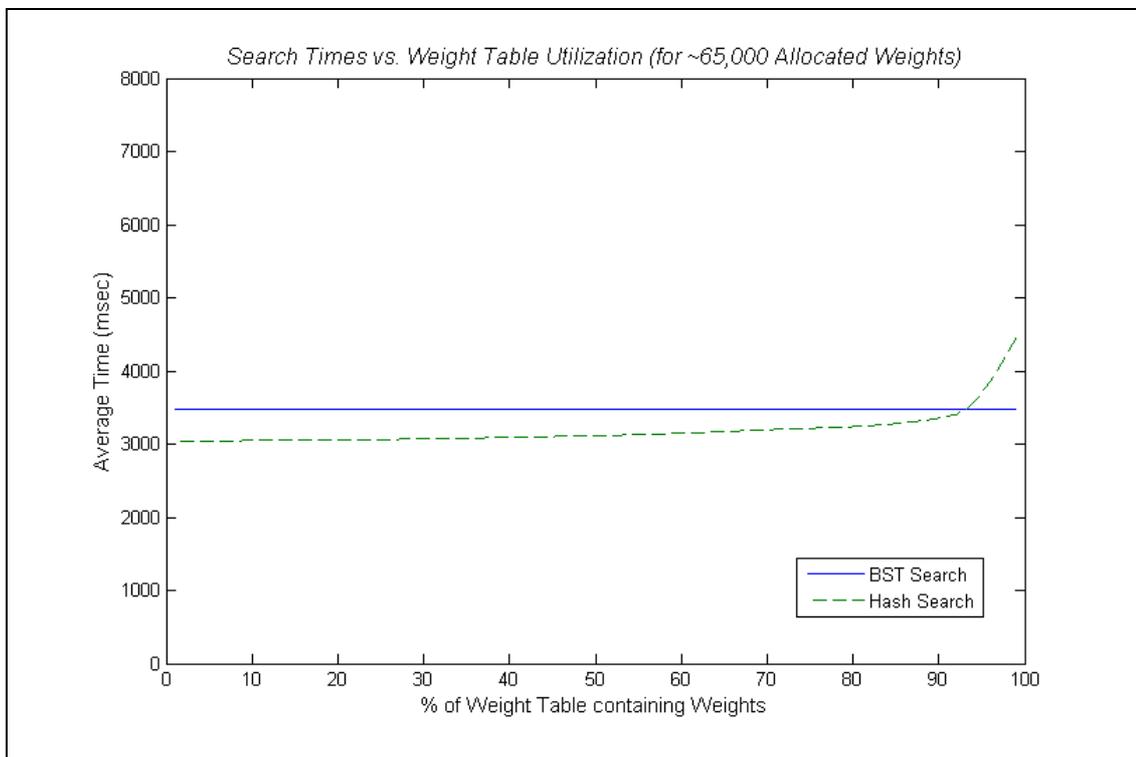


Figure 4.5 – Search Times vs. Weight Table Utilisation (for ~65000 Allocated Weights)

The most notable difference between Figure 4.5 from Figure 4.4 is that the BST search time is considerably less than the BST build time. This change is mainly due to the offline binary search tree balancing algorithm implemented between the build and search stages. The effect of tree balancing was illustrated in Figure 4.2, and is practically emphasised in the difference between build and search times here.

In terms of hashing, from comparing Figure 4.5 to Figure 4.4 it is noticeable that while the behavioural trends of the curves remain the same, the hashing search times are slightly less than the hashing build times. This offset is due to the lack of the training algorithm that updates the old weights to new values, required only for online adaptive use. However if a CMAC is not required to learn, then the training algorithm only adds additional overhead time to the search process. The omission of the training algorithm for this test remains consistent for both BST and hashing algorithms and therefore is not a variable in the search time comparison data between memory allocation methods.

The increase in time is evident again for the hashing method without hash collisions when the weight table approaches 100% utilisation. Once again the times remain consistent for the searching of weights for hashing with collisions and BSTs. All these times however rely on the number of receptive fields corresponding to each input vector and the type of sensitive function used. The choice of sensitive function remains constant throughout the choice of memory allocation method selected and thus is not a parameter to be considered throughout these memory allocation method comparison tests. The number of receptive fields is a factor and Test 2 deals with this variable.

4.2 TEST 2: NUMBER OF RECEPTIVE FIELD LAYERS

As previously mentioned, the number of receptive field layers corresponding to each input vector is a variable in the comparison between the BST and hashing memory allocation methods. The following tests were conducted with the following CMAC parameters.

Sensitive Function	LINEAR
Input Dimensions	2
Output Dimensions	1
Number of Trained Input-Space Vectors	600 x 600
Number of Receptive Fields	Variable
Hash Table Size (max # weight addresses)	1,200,000

Table 4.3 – Test 2 CMAC Parameters

A two-dimensional input space was trained using 600 x 600 unique input space vectors, producing 360,000 unique training points, each trained once using the function in Equation 4.1. Using a hash table of size 1,200,000 produced a maximum hash table utilisation of 3.83% throughout the tests, a small percentage indicating hash performance is close to optimal. From Figure 4.4 and Figure 4.5 it is clear to see that this percentage of utilised hash table space produces fast and efficient build and search times.

4.2.1 RECEPTIVE FIELD CONSIDERATIONS

A few considerations must be first made here before the tests can be explained with regards to the differences between the number of receptive fields mapped to each input vector and the numbers of receptive fields suitable to be used by the code.

Firstly, the UNH CMAC code requires that the number of receptive fields (i.e. generalisation parameter) be an integer power of 2. (E.g. 1, 2, 4, 8, 16... 2^n). For speed and memory space requirements, the BST code also requires that the number of receptive fields be an integer power of 2, not less than 8. Since one of the CMAC's most useful properties is that of local generalisation, using less than 8 receptive fields produces very little generalisation of neighbouring vectors. Thus for the majority of purposes here, the number of receptive fields used in a CMAC will be equal to or greater than 8. For the tests in this sub-section the number of receptive fields will be limited from 8 to 128 inclusive.

Secondly, the CMAC receptive field (RF) mapping structure results in numerous large receptive fields for large generalisation parameters and few small receptive fields for small generalisation parameters. This means that per receptive field layer, for large generalisation parameters there are fewer weights associated with each RF layer. Similarly for small generalisation parameters there are numerous weights associated with each RF layer.

In the BST case, this means that for a generalisation parameter of 8, there will be 8 BSTs, one per RF layer, each large in size. For a generalisation parameter of 128, there will be 128 BSTs, one per RF layer, each small in size. This is illustrated in Figure 4.6.

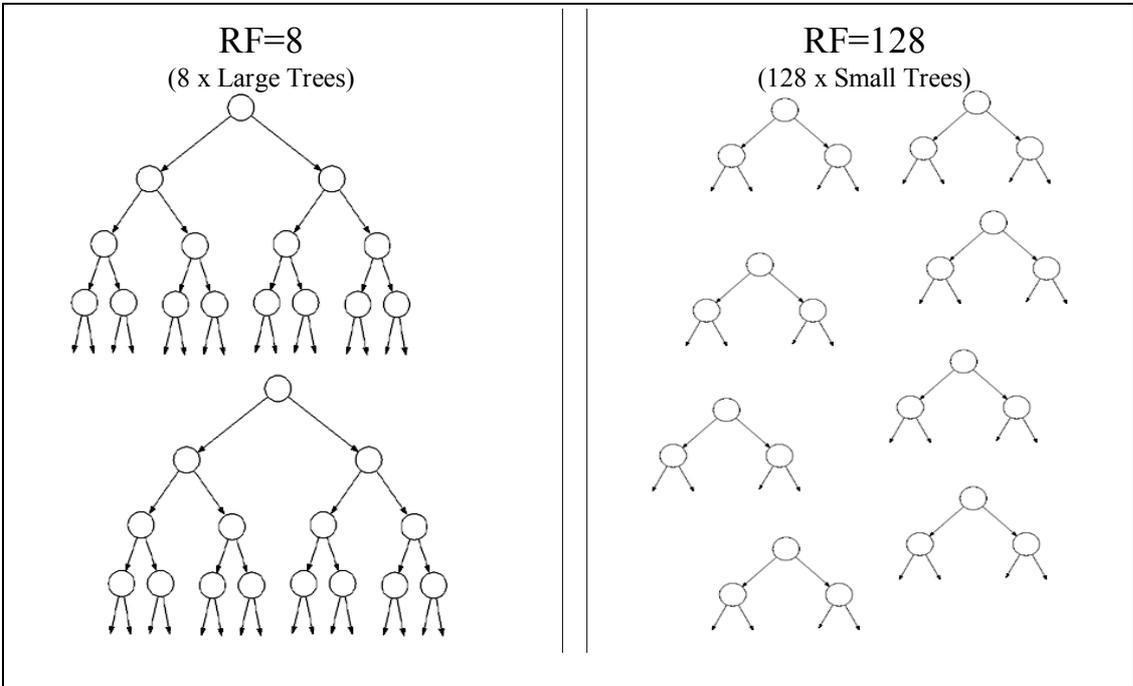


Figure 4.6 – Fewer Large BSTs vs. Numerous Small BSTs

As the generalisation parameter changes from 8 to 128, the number of RF layers increase and therefore so does the number of BSTs. But as the generalisation parameter increases, the overall generalisation property of the CMAC increases, meaning that fewer weights are required to store data for the same number of input vectors. This produces fewer overall weights in the weight table, distributed over numerous BSTs.

The large generalisation property of a CMAC when using a large generalisation parameter can be illustrated by the low numbers of weights required to store the trained data of a given set of input vectors. For 360,000 trained input vectors, this is illustrated in Figure 4.7.

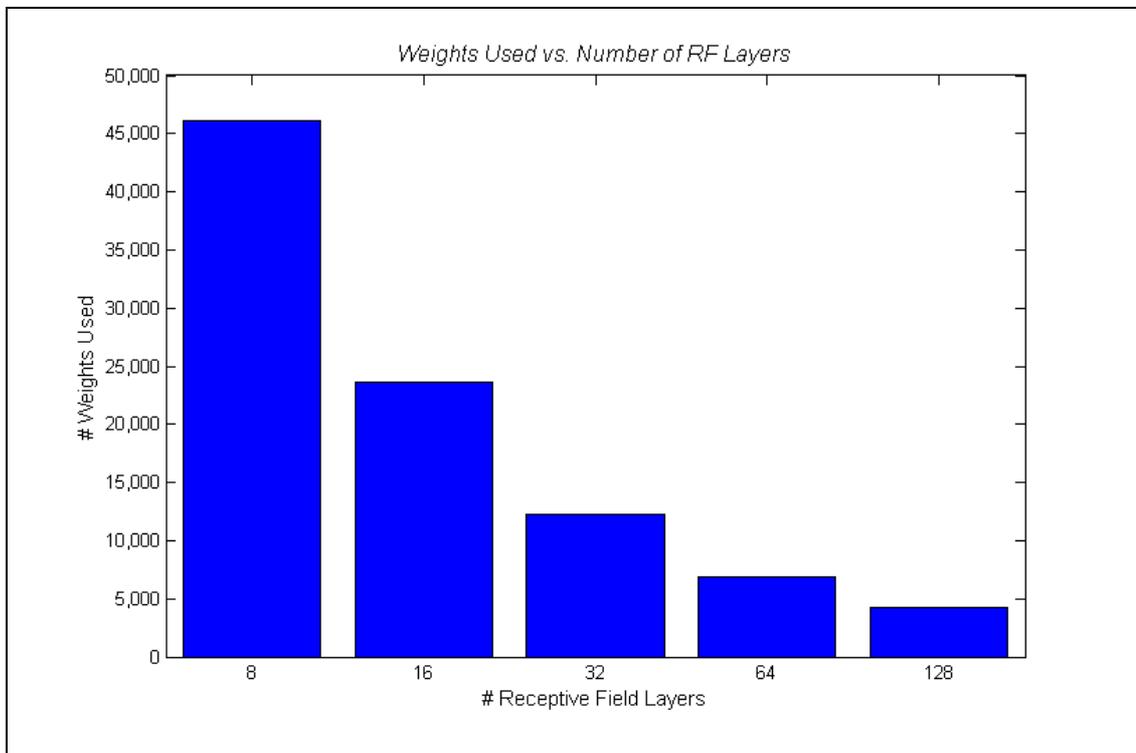


Figure 4.7 – Weights Used vs. Number of RF Layers

Figure 4.7 illustrates the relationship between the number of weights used and the number of receptive fields, for a given set of 360,000 input vectors. The large generalisation property is evident with RF = 128, with fewer weights representing the total number of input vectors, as opposed to when RF = 8. Thus a large amount of generalisation will occur at the output for a given input vector. As these results are CMAC algorithm dependent, BSTs and hashing are not parameters in these results.

The time taken to jump between RF layers when searching for weights is a factor in determining the relative efficiency between BSTs to hashing. Is it quicker to search for a fixed number of nodes within a large BST than to jump between several smaller BSTs? This question will be answered in the following sub-sections.

The next section will continue on with the effect that changing the generalisation parameter has on CMAC build and search times, comparing BSTs and hashing.

4.2.2 TEST 2.1: BUILD TIMES

The build times for training a CMAC with 360,000 unique training points using the two memory handling methods is illustrated in Figure 4.8 for a range of generalisation parameters of integer powers of 2, from 8 to 128 inclusive.

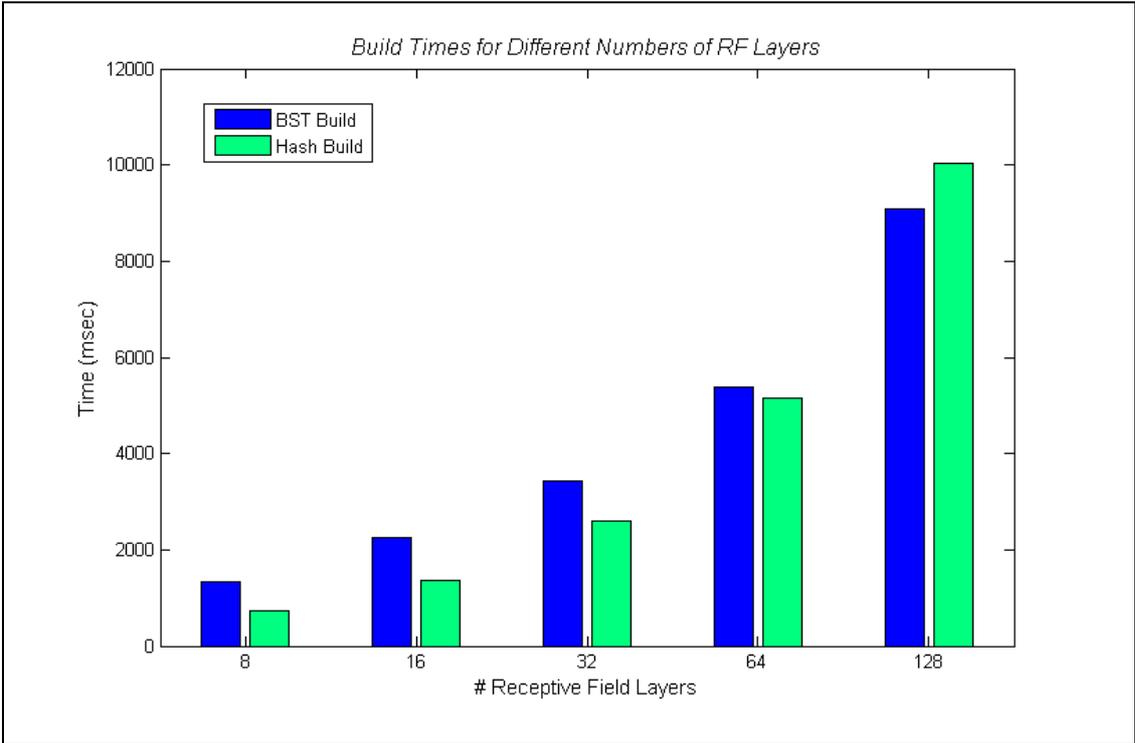


Figure 4.8 – Build Times for Different Numbers of RF Layers

As illustrated in Figure 4.8, using a lower generalisation parameter results in faster builds for both BSTs and hashing. This is only to be expected, as for each output only 8 weights need to be retrieved from memory and subsequently trained. As the generalisation parameter increases, so does the overall build time for both hashing and BSTs. Notice that for lower generalisation parameters, hashing is more efficient than BSTs. However as the number of receptive field layers increase, hashing becomes less efficient in respect, and BSTs become the more efficient approach. Again these BST times are for initial BST builds, built from scratch by randomly inserting nodes into a binary search tree.

4.2.3 TEST 2.2: SEARCH TIMES

In Section 4.1.3 it was stated and shown in Figure 4.4 and Figure 4.5 that the complete build of a CMAC using a large number of weights resulted in an unbalanced binary search tree. This produced greater build times in comparison to when the same tree was balanced and subsequently searched. Thus for this test, after the BST was built in Test 2.1, the BST was balanced offline. Subsequently, the CMAC search times using balanced BSTs compared to hashing for the different generalisation parameters is illustrated in Figure 4.9.

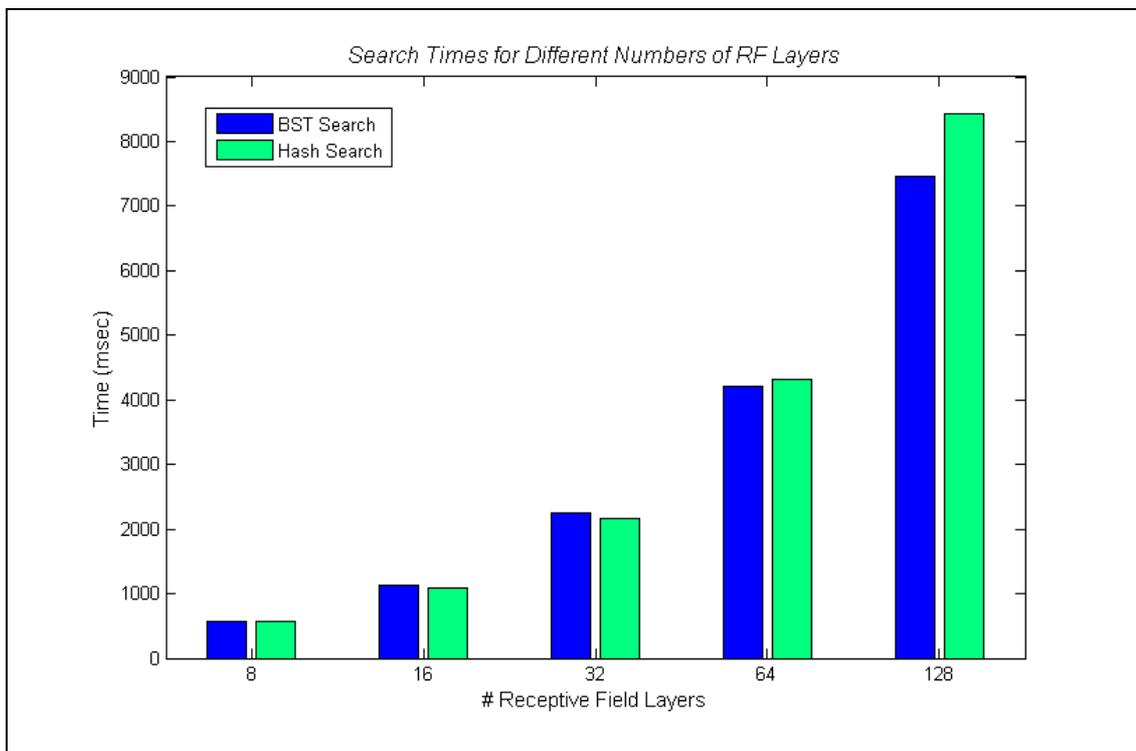


Figure 4.9 – Search Times for Different Numbers of RF Layers

Figure 4.9 shows that once BST balancing has been completed, the BST compares very similarly, if not better to hashing in terms of the time required in retrieving CMAC weights from memory. This is especially evident for larger generalisation parameters. Again for the same time domain reasons as in building the CMAC weight tables illustrated in Figure 4.8, for larger generalisation parameters, BSTs produce more efficient results. However BSTs also provide a comparable and useful dynamic solution when choosing smaller generalisation parameters for searching CMAC weights, as the search time differences in Figure 4.9 are relatively minor.

The majority of CMAC usage will not only involve preliminary builds and searches. Online training, where weights are retrieved to provide the CMAC output and then trained online in real-time, makes up a significant amount of CMAC usage time, and thus this area will now be discussed.

4.3 TEST 3: CMAC ONLINE TRAINING

Established CMACs with previously trained data that are being continually trained after periods of offline binary tree balancing, should make up the majority of CMAC usage cases when using CMACs with BST memory handling. Often the addition of new data is presented for the CMAC to learn, however the majority of the time this additional data should only amount to 0 to 10% of the data already contained in the CMACs memory. Thus for these common cases, it is important to show here how the two memory storage methods compare to each other. The following test was conducted with the following CMAC parameters.

Sensitive Function	LINEAR
Input Dimensions	2
Output Dimensions	1
Number of Trained Input-Space Vectors	600 x 600
Number of Receptive Fields	Variable
Hash Table Size (max # weight addresses)	1,200,000

Table 4.4 – Test 3 CMAC Parameters

A two-dimensional input space was trained using 600 x 600 unique input space vectors, producing 360,000 unique training points, each trained once using the function in Equation 4.1. Using a hash table of size 1,200,000 produced a maximum hash table utilisation of 7.67% for this test, again a suitably small percentage indicating hash performance is close to optimal.

4.3.1 TEST 3.1: ADDITIONAL TRAINING (VARIABLE)

The first test for this section involves a comparison between online training with the addition of newly added data to a CMAC's weight table, using the two memory handling approaches of hashing and BSTs. The results are illustrated in Figure 4.10. For this test the generalisation parameter was fixed to 16.

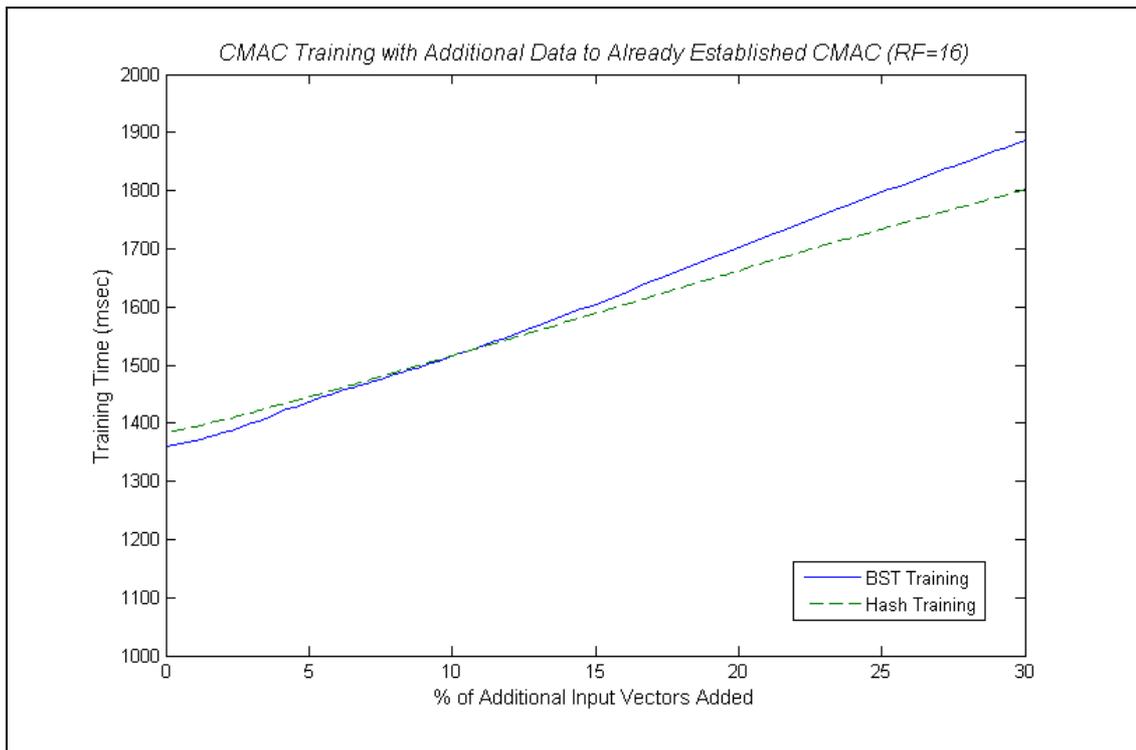


Figure 4.10 – CMAC Training with Additional Data to Already Established CMAC (RF=16)

The test producing the data for Figure 4.10 was conducted as follows. After a single training of the 360,000 input vectors has been completed, equivalent to a build of the same data as in Section 4.2, the binary search tree was then balanced offline for the BST memory storage method. This step is not required naturally after training when using hashing. The CMAC is then retrained using exactly the same 360,000 input vectors which had been previously learnt. For the hashing case, the time taken to retrain the CMAC is equal to training the CMAC with the first pass of the data (the original training with the 360,000 input vectors). With the BST method however, the tree is now balanced, with all the randomly inserted nodes generated from the original build now in a balanced tree. Therefore the search time for each node in the tree is decreased, much the same as the search performed on the trees in Section 4.2, however with the additional time taken to retrain the weight data in the nodes. The

end result is a binary search tree with exactly the same structure as before the CMAC BST was retrained.

From Figure 4.10 it can be seen that the crossover point between the two data sets lies at approximately 10% of the additional vectors added. At this point hashing is more efficient than using a BST to continually add data to a CMAC. Note that these training times with the additional data are inclusive of the retraining of 100% of the original data.

If more than 10% of the data is required to be added to a CMAC, then rebalancing should occur offline, followed by the additional new data trained into the CMAC. This provides the optimal solution in this case, as for every new piece of data which is added to a balanced binary search tree becomes a randomly generated node which is added to the end of the already balanced BST. The greater the number of additional random nodes added to a BST, the more the performance will be potentially degraded, similar to the difference between times of randomly generated BST searches and balanced tree searches as illustrated previously in Figure 4.2.

4.3.2 TEST 3.2: ADDITIONAL TRAINING (FIXED)

The test illustrated by Figure 4.10 was performed to find a suitable percentage of additional data that could be added to a CMAC before offline tree balancing had to be performed. 10% was the limit, however the BST performed better at lower percentages than this. Therefore another test was performed to show the training time performances with a midway 5% additional training data for the different generalisation parameters. The results are illustrated in Figure 4.11.

The performance of using BSTs in comparison to hashing with a CMAC is evident here throughout the tests using different numbers of receptive fields when training. Limiting the amount of new data added to a CMAC produces more efficient results when using BSTs for all generalisation parameters tested. However, this test takes into account a CMAC with 360,000 input vectors only. As this is a variable that must be taken into account, a further test was performed to distinguish which generalisation parameters are most suitable for use with different values of input vectors for both memory handling methods.

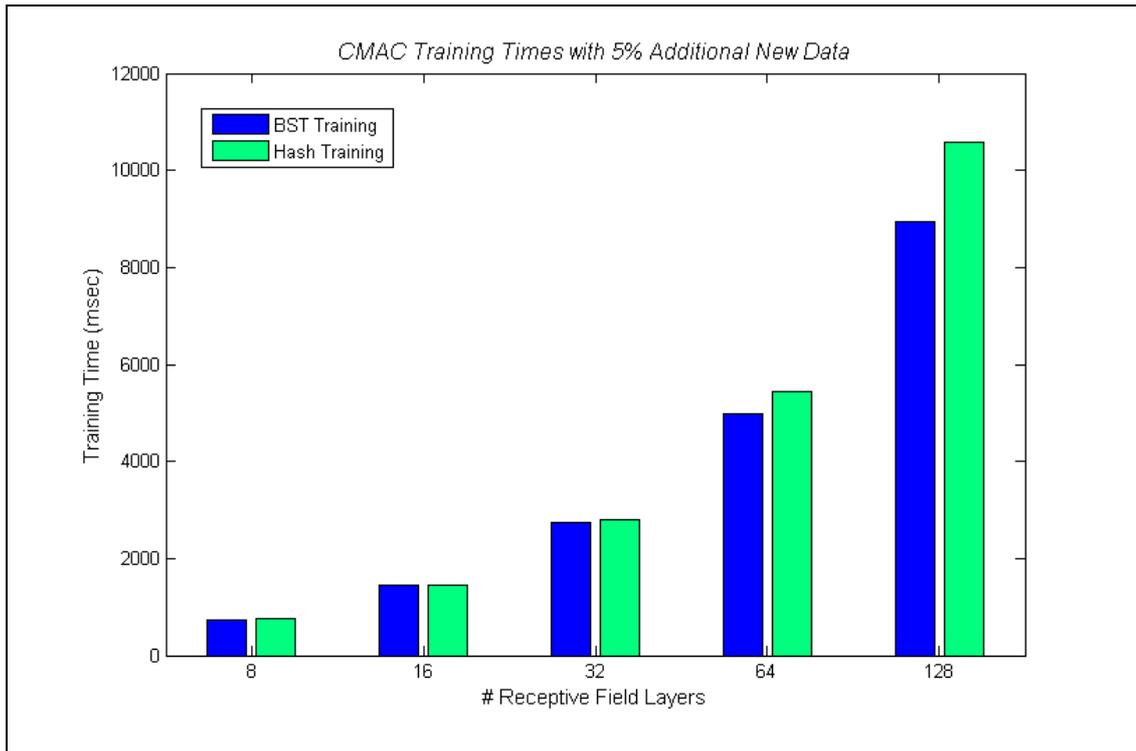


Figure 4.11 – CMAC Training Times with 5% Additional New Data

4.4 TEST 4: OPTIMAL BST VS. HASHING PERFORMANCE VARIABLES

A further test was performed to determine which memory handling method is most suitable for use with the available generalisation parameters for different numbers of input vectors, for the majority usage case when online training is performed. The test was performed with the following CMAC parameters:

Sensitive Function	LINEAR
Input Dimensions	2
Output Dimensions	1
Number of Trained Input-Space Vectors	Variable
Number of Receptive Fields	Variable
Hash Table Size (max # weight addresses)	1,200,000

Table 4.5 – Test 4 CMAC Parameters

The CMAC’s two-dimensional input space was trained with a variable number of input vectors, ranging from 200 x 200 (4×10^4) to 1600 x 1600 (2.56×10^6) input vectors, trained to learn the function in Equation 4.1. At 1600 x 1600 input vectors, the maximum hash table utilisation was 26.67% (at RF=8), still a suitable percentage for efficient hash performance. The CMAC was first trained, then balanced in the BST case, and retrained with an additional 5% of new data in addition to retraining 100% of the already established data for both memory handling methods. The results are illustrated in Figure 4.12.

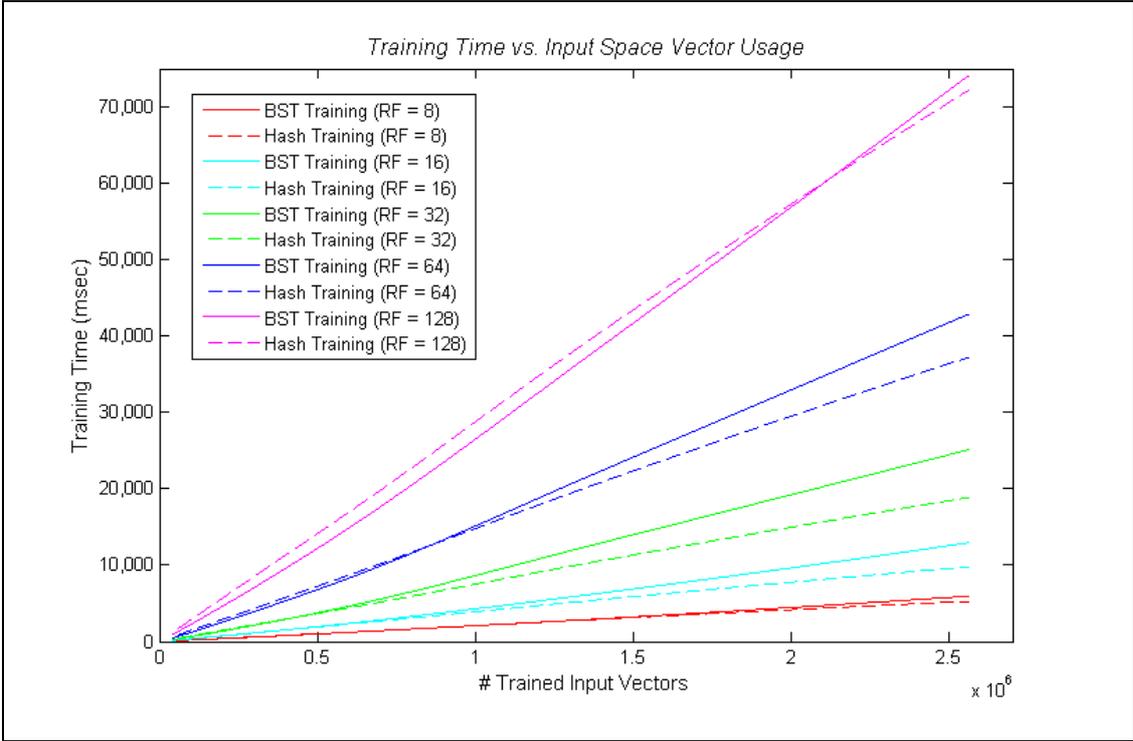


Figure 4.12 – Training Time vs. Input Space Vector Usage

Figure 4.12 illustrates the plots for each generalisation parameter from 8 to 128 inclusive, comparing BST to hash training memory storage methods. The generic trend for all plots is a faster BST training for smaller numbers of trained input vectors and faster hash training for the larger numbers of trained input vectors.

This data however is best represented as a 3-D graph, with the number of receptive fields on a separate axis. Thus Figure 4.13 illustrates the 3-D representation of the same data.

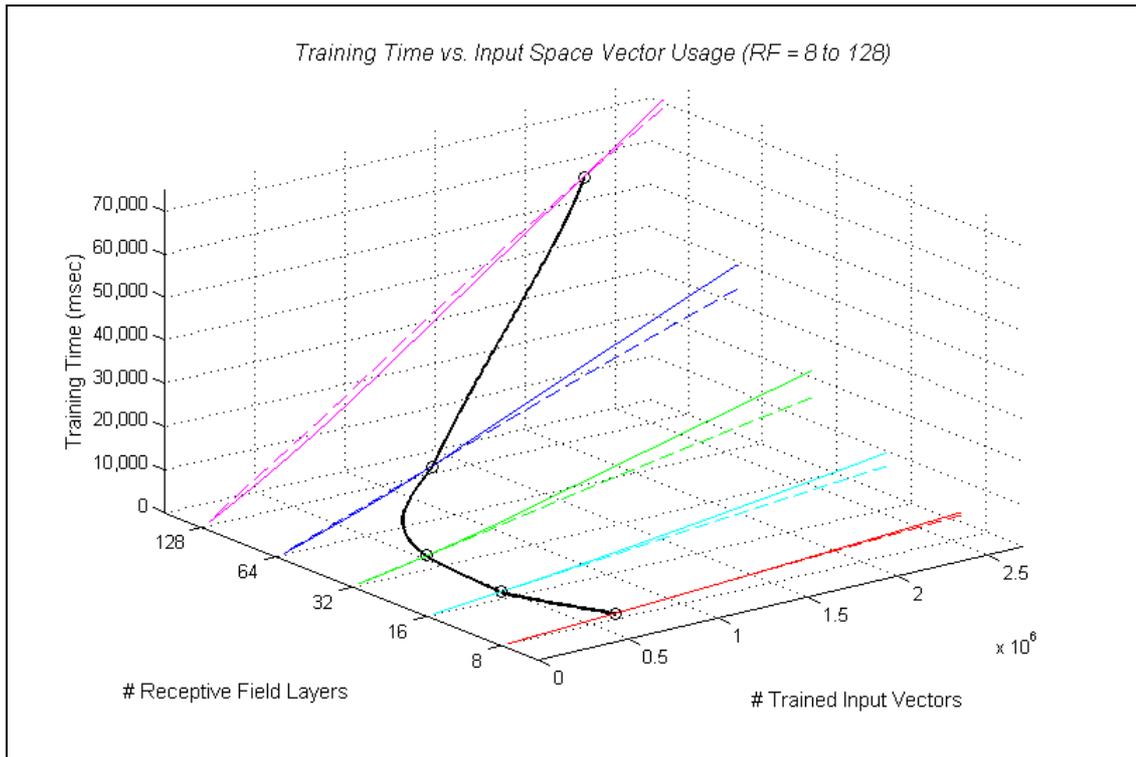


Figure 4.13 – Training Time vs. Input Space Vector Usage (3-D)

Figure 4.13 illustrates the relationship between training time, the number of trained input vectors and the number of receptive field layers. An intersection curve, a cubic spline interpolation, is added through the intersection points where the BST and hashing plots for each receptive field layer cross over. This curve shows the range of numbers of trained input vectors where BSTs are more efficient than hashing and consequently where hashing outperforms BSTs. Removing time as a factor, the 2-D representation of Figure 4.13 can be illustrated in Figure 4.14.

Figure 4.14 shows the final results of the tests, indicating which numbers of training input vectors are most suitable to each type of generalisation parameter. The overall optimal efficiency for BSTs over hashing lies to the left of the intersection curve, and the optimal efficiency for hashing over BSTs lies to the right. Since the generalisation parameters are discrete, this data is better represented in Table 4.6 as a discrete value for each generalisation parameter.

As can be seen in Figure 4.14 and Table 4.6, there is no linear relationship between the generalisation parameter number and the number of trained input vectors.

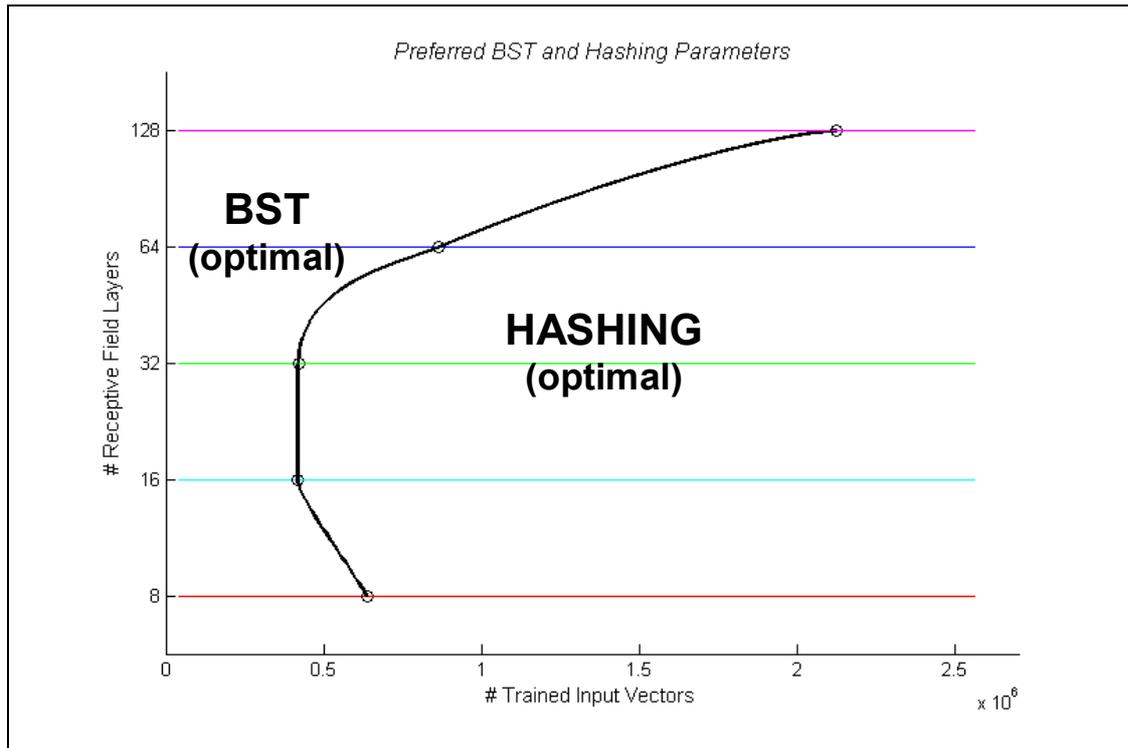


Figure 4.14 – Preferred BST and Hashing Parameters

Generalisation Parameter (#RFs)	Max # Input Vectors Suitable for Optimal BST Efficiency
8	650,000
16	420,000
32	420,000
64	800,000
128	2,150,000

Table 4.6 – Max Number of CMAC Input Vectors for Optimal BST Efficiency

The question put forth in Section 4.2.1, “Is it quicker to search for a fixed number of nodes within a large BST than to jump between several smaller BSTs?” can now be answered. The answer is evident in the shape of the intersection plot in Figure 4.14. The intersection plot curves around for lower generalisation parameters, indicating that it is quicker to search further through BSTs than to jump to the next BST and start searching again in the BST code produced.

4.5 SUMMARY

The choice of using BSTs or hashing as the memory handling method for CMAC is clearly dependent on several factors, with no single superior method standing out.

Several tests were performed illustrating which memory handling method is more efficient for a particular CMAC application. Consideration must be given to BSTs that are not balanced. For initial CMAC builds with large amounts of input data, especially for low generalisation parameters, BSTs struggle to compete with hashing. After offline tree balancing is performed, the access time for each node/weight in a BST is significantly improved. Consideration must also be given to the use of hash tables, as for largely populated hash tables, the weight access time is significantly increased. BSTs however remain unaffected by the weight table utilisation percentage.

The majority of CMAC training is performed online with large amounts of previously stored data retraining occurring, with a small percentage of new additional data added. Thus for the majority of the time, a CMAC's BST will be balanced, with a small percentage of new data added to the BST. Large amounts of new data are not desirable to be added to BSTs all at once, as node/weight access performance starts to decrease. Suitable amounts of incremental training however with offline balancing performed for every 10% of additional data, is required to keep BST performance high. Hashing weight access time is not affected by the increased growth of newly added data, unless the hash table utilisation starts to approach its size limit.

The most important factors in choosing which memory handling method to use lies in the choice of the generalisation parameter and the number of input vectors expected to be trained. Overall, hashing is more efficient than BSTs for large input data sets. BSTs on the other hand are more efficient handling smaller data sets. In addition however, BSTs provide a continuous dynamic solution to CMAC memory storage, whereas the size of a hash table is required to be pre-determined prior to any CMAC training. For further expansion, BSTs provide a suitable solution to the ongoing CMAC learning, with only the current number of weights saved offline at

any time. For hashing, the whole hash table is always stored offline, regardless of how many weights are currently stored.

Overall, no single method is superior in managing CMAC weight tables. The decision to use BSTs or hashing with a CMAC lies primarily in whether the user wants to have dynamic memory handling capabilities or not, and secondly in the CMAC parameters used. In this thesis however, the CMAC model used will incorporate the BST model developed here, allowing as much memory as is needed to be allocated dynamically throughout all CMAC testing.

The following chapter will discuss the CMAC model used together with a PID controller, which will be used as the heart of the overall controller in this thesis.

5.0 CMAC+PID CONTROLLER

With the dynamic CMAC memory handling algorithm implemented and tested utilising binary search trees, the next stage in developing the overall controller involves implementing a foundation controller on which to build the Error Minimising Gradient Controller (EMGC).

The first and most important stage of the overall control system is the feedforward / feedback controller, utilising a CMAC together with a standard PID control feedback loop. The feedforward/feedback controller is a standard approach used heavily in control applications (Huang, 2006). Often a more advanced feedforward section is a neural network of some kind (Albus & Meystel, 2001). Here it is specifically a CMAC, allowing real-time in-loop learning of the desired system to be controlled. The CMAC+PID portion of the overall control system is illustrated in Figure 5.1. This chapter focuses on the CMAC+PID controller section of the overall control system.

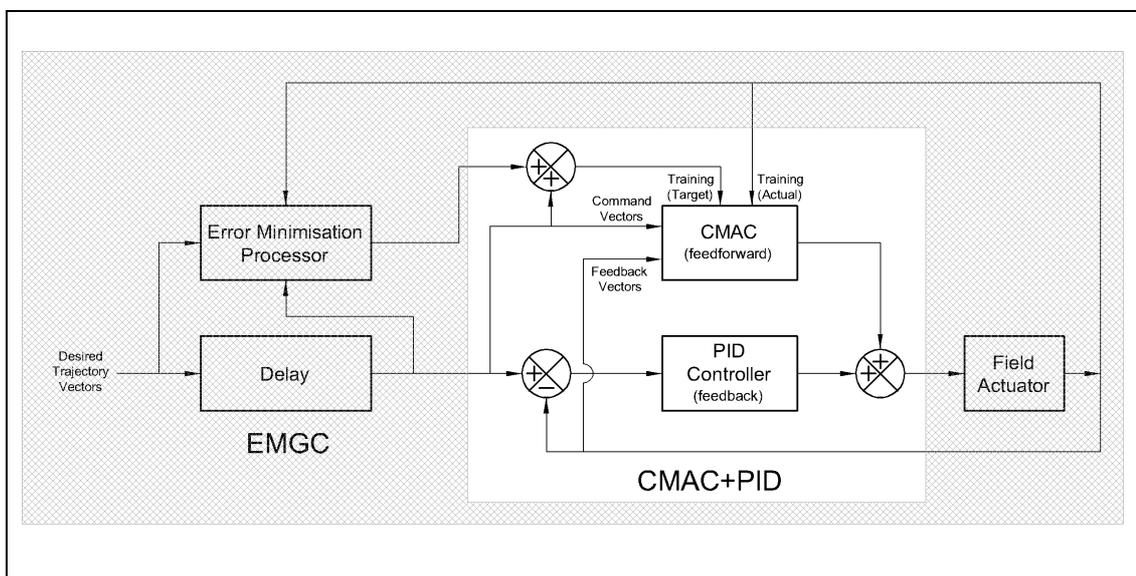


Figure 5.1 – CMAC+PID Controller in the Overall Control System

The CMAC+PID controller in Figure 5.1 accepts three independent inputs and produces a single output. The first input consists of the desired trajectory command input vectors, leading out of the delay from the EMGC. The second input consists of the EMGC's Error Minimising Gradient vectors. The third input consists of the actual field feedback vectors, used for both immediate feedback control through the

PID controller, as an input vector to the CMAC and also for CMAC training. The output from the CMAC+PID controller is the control signal which drives the actuator.

5.1 PID CONTROLLER

A widely used form of feedback control is the PID controller. However for our air-muscle actuator a PID controller on its own performs rather poorly. There are two main reasons for using a PID controller together with a CMAC, bettering a sole CMAC approach.

Firstly, the system will still respond when new and unknown states are encountered. For these newly encountered states, a CMAC alone would produce an output of zero, as the weights associated with the creation of new CMAC Binary Search Tree nodes produce no output for their first iteration. (All previous inactivated CMAC weights produce an output of zero upon their first activation as they do not yet have a trained weight value.) In these circumstances, the PID controller will solely drive the system in the desired direction, until after several iterations, the CMAC will tune itself to minimise the error and thus the PID output will reduce to almost zero. Without the PID controller, there will be no output from the overall controller when new input states are encountered. Even if new CMAC BST nodes are set by default to some constant value, there is no way of knowing which way to drive the system for the first iteration of these new states.

Secondly, training time is reduced when using a PID controller as the system's actual position is driven in the required direction based on the system's current position error. In other words, if using the CMAC alone, with the zero output (or some constant value) from the CMAC occurring when new BST nodes are being created, the system has to move from this resultant constant state to the final desired state. Thus time is wasted as there is no driving force pushing the error to zero, meaning more time is required for the system to stabilise at the desired state. Together with the PID controller, the actual position is already driving towards the desired position, allowing the CMAC to learn only from the initial PID output for times when new CMAC input vector states are encountered.

While the PID controller helps the CMAC achieve convergence, in itself it fares quite poorly for the current system. Not only is hysteresis a factor when controlling air-muscles, but undesirable and often unpredictable changing dynamics of the system cannot be compensated for by a PID controller alone. The form of the PID equation used in this thesis is:

$$u(t_n) = u(t_{n-1}) + K \left(e(t) + \frac{1}{T_i} \int_0^t e(\tau) d\tau + T_d \frac{de(t)}{dt} \right) \quad \text{Equation 5.1}$$

where $u(t)$ is the PID controller output signal, and $e(t)$ is the control error determined from subtracting the current position from the target position. The controller parameters include the proportional gain K , integral gain T_i and derivative gain T_d .

A comparison of a PID controller response to the CMAC+PID controller illustrated in Figure 5.1, for control of a single air-muscle with a spring return, produces a response comparison such as that in Figure 5.2.

Figure 5.2 illustrates the benefits that the CMAC+PID controller has over a single PID controller when controlling a single air-muscle. The actual response from the system with the PID controller is quite poor, in terms of lag, steady state error, response time and hysteresis, in comparison to the CMAC+PID response. It is clear that a satisfactory solution to this control problem will require more than a simple PID control solution. This situation becomes worse as the fixed PID gains try to control the non-linear system dynamics at the edges of the position limits. (Refer to Figure D.3 for non-linear actuator relationship.) In addition, PID gains would have to be both found and adjusted continuously to obtain continuous reasonable performance.

Seeing the numerous response advantages the CMAC+PID controller has over the PID controller on the air-muscle response leads into the presentation of the CMAC used in the CMAC+PID controller. Information on the PID gains selected for use in the CMAC+PID controller can be found in Appendix B.

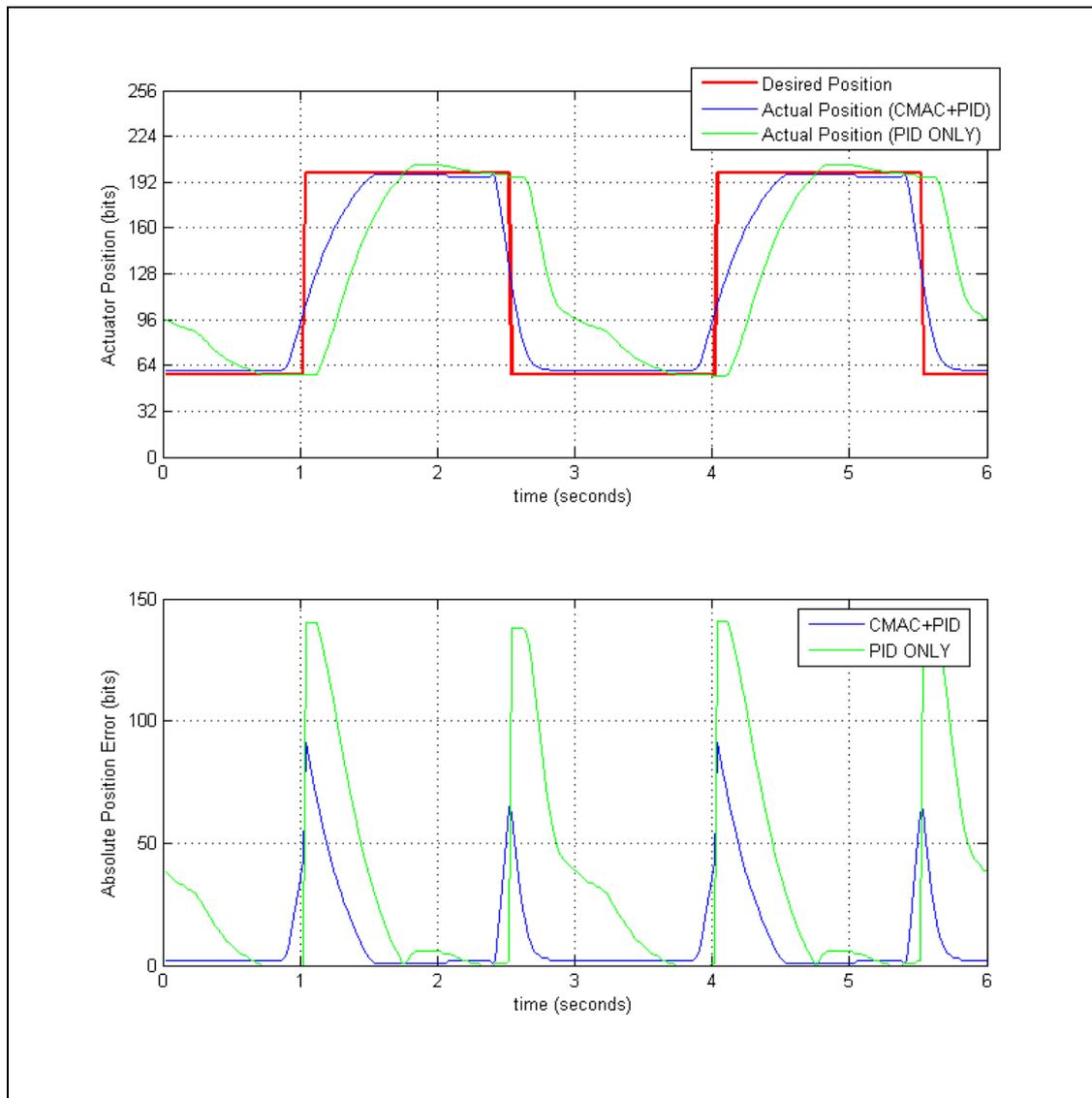


Figure 5.2 – Position and Error Response Comparing PID Only Control to CMAC+PID Control.

5.2 CMAC CONTROL

As explained in the Literature Review in Section 2.8 of this thesis, the CMAC is an adaptive controller that can be trained online while being used in real-time control applications. The CMAC is used as the centre piece of the adaptive control architecture used in this thesis. Much of this section will present the various aspects of the CMAC controller parameters used in conjunction with the PID controller.

The overall controller output as illustrated in Figure 5.1 is a summation of the PID and CMAC outputs, and thus once the error drops to zero for a given set of input states, the PID controller will also stop producing a fluctuating output and the

CMAC will be providing close to full control of the actuator. This reason, together with the CMAC being complex in nature compared to the simplistic PID controller, means that significant attention will now be given to the various aspects of the CMAC controller used in this project.

The CMAC used here requires several parameters to be considered to produce a stable and efficient control solution to the problem. While many papers in the past have only used standard Albus CMACs without any modifications as a solution to their problem, the following sections will provide a detailed explanation of each enhancement and aspect required to produce a practical adaptive feedforward controller for use in this thesis.

5.2.1 INPUT VECTOR PARAMETERS

It is crucial that suitable input vector parameters are chosen to both provide and produce a satisfactory number of different CMAC input space vectors for each different possible state of the system. If too few input vector parameters are used, the CMAC will not be able to differentiate between differing system states. If too many redundant input vector parameters are used, wasted memory usage and possible increases in training times may result. Therefore only enough input vector parameters to differentiate between the various states the system will encounter and those that will directly influence the system output should be used. For example, the ambient temperature of the room is a redundant input vector parameter here as a change of 1 degree will cause the CMAC to map a whole new trajectory curve through the input vector space, which for our application is totally wasteful and unnecessary. Note that the input vectors used here are not derived from the CMAC neural network, and can be applied to other neural network control solutions. However, since the CMAC is the aptive controller of choice in this thesis, the explanation of the chosen input vectors will be used in context with the CMAC neural network for clarity of the solution being applied. The selection of input vector parameters chosen will now be presented together with their relevant importance.

5.2.1.1 *Desired Position*

Using the desired position as a command input vector parameter is an obvious choice, as different CMAC outputs will be required over the range of possible

desired positions. Thus as the desired position changes, so too should the equivalent CMAC input space vector. Note that the desired position is bounded, being well within the input vector boundaries set in Section 3.3.

5.2.1.2 *Desired Velocity*

On its own, the desired position of the system will not suffice in separating the range of desired states the system will find itself in. For example, the direction of motion is extremely important in deciding the equivalent CMAC output value for the current desired state. Hysteresis is directly related to the direction of motion of an air-muscle and thus an output value at a desired position x will need to be differentiated from when x is approached from the opposite direction. This leads to the requirement of a desired velocity command input vector parameter as well.

Based on the work presented in Section 3.3, each CMAC input vector dimension has a maximum resolution of 2048 when using a Generalisation Parameter of 8. Therefore it is a requirement that all CMAC input vectors be bounded, as the CMAC input space is of finite size. For the desired position input vector parameter this is achieved by default, as the actuator position limits are bounded. For velocity however, this is not the case. For example, a square wave input trajectory means that depending on the sampling period of the control system, close to infinite velocities can occur at the rising and falling edge transitions. Figure 5.3 illustrates this.

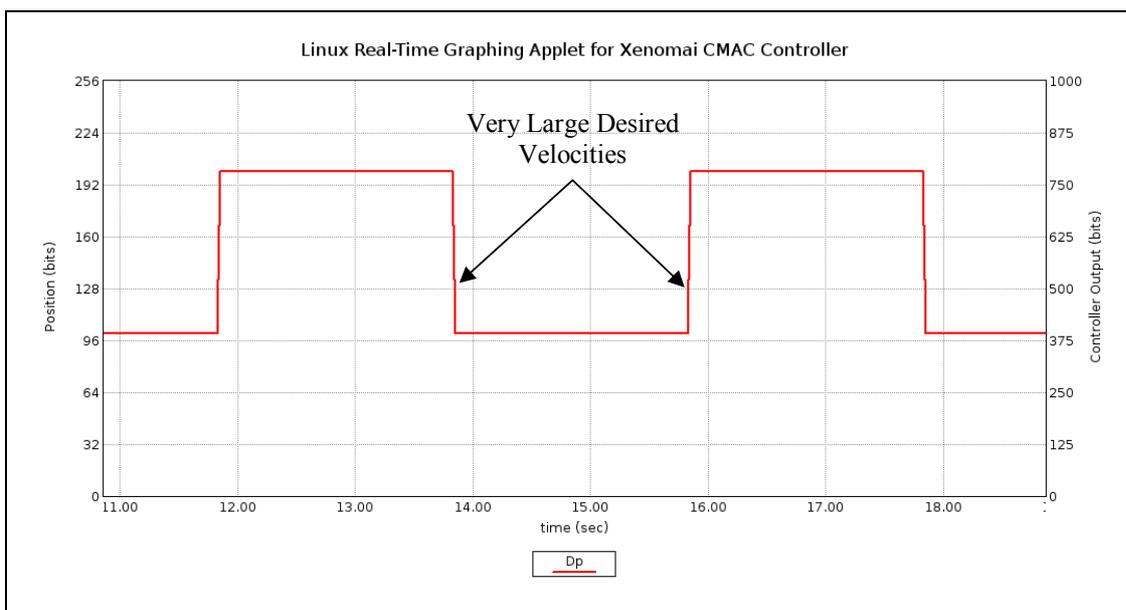


Figure 5.3 – Near Infinite Desired Velocities on Rising and Falling Edges

On the opposite end of the velocity scale, when the position of the input trajectory remains constant such as in the plateaus of Figure 5.3, there is zero velocity. Thus the possibility of the velocity range extends from zero to infinity in a single direction. As a bounded space is required for the desired velocity input vector dimension, the desired velocity must be mapped to a finite address space.

This mapping process requires any input value from zero to infinity to be scaled to a value within the limits of the CMAC input space. Finer resolution should be allocated to smaller velocity values enabling finer control of the actuator at low desired velocities. Coarser resolution should be given to the large desired velocity values, as control here is less critical. It is wasteful having a similar resolution over the whole range of possible velocities because high desired velocities will often exceed the maximum attainable velocity of the actuator. Therefore, the control values (weights) for all desired velocities above the maximum attainable velocity of the actuator will contain the same value, wasting precious CMAC input vector space associated memory weights. For this reason, it makes sense to use very little of the input space for larger desired velocities and use the majority of the CMAC input space for when accurate control is required. The $\tan(x)$ function is one such function that can easily map a zero to infinite value to a finite value while maintaining the CMAC input vector space boundaries. Appendix C discusses the equations and system parameters used to produce a feasible $\tan(x)$ mapping function for the system.

Using both desired position and velocity as CMAC input vectors, hysteresis will no longer be a control problem as velocity allows the various directions of motion to be separated within the CMAC input space.

5.2.1.3 *Actual Position*

Without the use of feedback input vectors, a CMAC cannot adapt to handle reoccurring external disturbances. The CMAC also cannot respond to desired inputs where there are too few states changing over time. Thus this leads to the need of another input vector, that being the actual position of the system.

Using the actual position of the system as an input vector solves the problem when the system has to learn to recover from external disturbances.

The benefits of using the actual position as a feedback input vector allows the CMAC to know what the current state the system is in, and thus allows compensation paths to be created in ‘actual position’ input space dimension between desired command input dimension states, to get the system back on track.

5.2.1.4 Time (when required)

If the CMAC output needs to vary whilst there are no changing command or feedback input vector states, then an external variable that will provide a smooth sequence of outputs is required to hold the system steady. For changing input vector states this problem does not arise, because as the input vector states change, so too does the corresponding output. However for unchanging input vector states, if the system requires a changing output, then the CMAC is unable to produce a smoothly changing output if the fixed input vector states do not map to a continuous set of changing weights.

Without using a feedback vector, a desired square wave trajectory produces an example of this problem, and is illustrated in Figure 5.4.

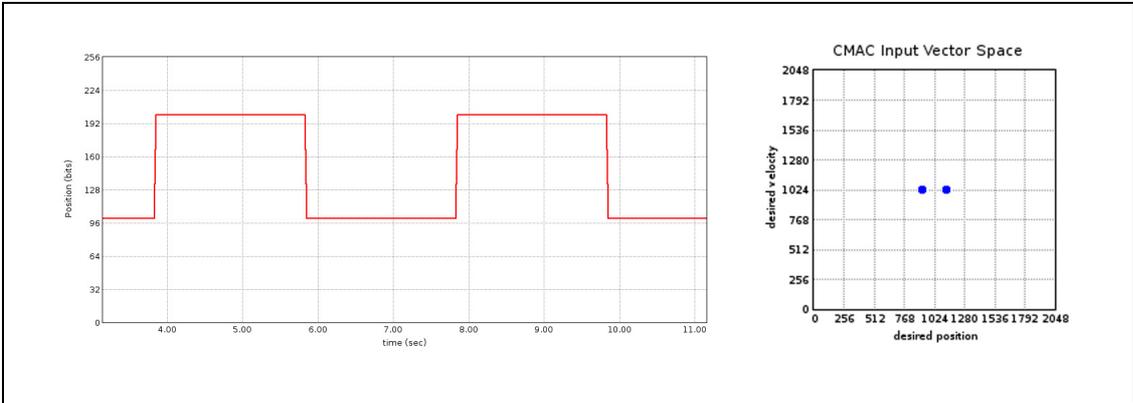


Figure 5.4 – Desired Square Wave Input Maps to Only Two CMAC Input Space Vectors

The plateaus of Figure 5.4 produce unchanging CMAC input states. Desired velocity is zero over the plateaus and the position remains constant. Continuous periods of the square wave will map to only two CMAC input states as illustrated on the 2-D CMAC Input Vector Space map.

The system will still be required to respond but if the input vector states are not changing, the CMAC output will be fixed. System limitations mean that controlled motion will still be required when moving between stationary states, and with using only desired position and velocity input vectors, this control problem will arise.

The inclusion of the actual position feedback vector parameter partially solves this problem, as there will be a path of changing input vector states occurring in the actual position dimension as the position of the system moves from one desired state to the next. However, there is no guarantee that there will be a continuous stream of changing input vectors to produce a stabilised output to drive the system to its stationary state of motion. In addition, there is no guarantee that when the actuator motion is held stationary that the driving signal should also remain fixed. For the actuator to be held stationary, a changing output may be required.

In the case of the system here using an air-muscle as the output actuating device, it is possible that the actual position will drift slightly over time for a particular output due to the elastic and other physical properties of the air-muscle system. Thus a changing output response will be required to vary both smoothly and continuously, to keep the actual position of the air-muscle constant. To produce a controller which can handle this, a motion-state independent input vector is required.

To solve this problem, time has been used as the fourth input vector for the CMAC. It is important to note that time should be used with caution, as it is an unbounded variable. For this reason, time is used ONLY when it is required. When the system lacks a continually changing desired state path, this triggers the time vector to be used. In other words, the CMAC input vector space is being created nominally within its three-dimensional space, only moving to the fourth dimension when there is a lack of continually changing desired states.

The time input vector has been designed to start incrementing only after a specified time if and only if the absolute desired velocity is below a threshold value. Thus the actual position feedback vector will still provide the required control as the system moves from one desired state to the next, and only when the absolute desired velocity

drops below a threshold value, the time vector will start incrementing and provide additional control resolution for the slow-moving system.

Using a bounded input space of 2048 units per dimension means that the time input vector needs to be capped eventually. At the control sampling rate of 100Hz, approximately 20 seconds of data can be stored in the time vector dimension before this capping will occur. Thus for a desired square wave, the maximum length of a plateau can be no longer than 20 seconds. If the system is required to be held in its desired static state for longer than this period of time, then there is no guarantee of system behaviour past this point. A non-linear mapping function however can be used to prolong the time vector output, slowly decreasing the smoothness at the CMAC output as time increases past 20 seconds if required. This method has not been implemented here but would suffice if the actuator requires a gradually less fluxuating drive signal to keep it stationary as the time vector continues past the CMAC input space boundaries.

For most changing and even static desired system states, using the above mentioned four input vector parameters should produce a controller that can differentiate between the required states the system may find itself in, whilst stably converging.

5.2.2 GENERALISATION AREA

The generalisation area for a CMAC is a combination of two factors: the number of receptive fields per input vector and the quantisation level of the input. The choice for the generalisation area mapping is based on the UNH Uniform Distribution Mapping as discussed in Section 2.8.7.2.

Generalisation areas specify how much a CMAC will generalise. In other words, how much a particular input vector will influence neighbouring input vectors. The quantisation level here refers to the truncation of a large input vector resolution space into a smaller one. For example, an input space of 2048 elements with a quantisation level of 2 means that every 2 input vectors will map to the same CMAC input, producing a CMAC input vector resolution of size 1024. This is a rough way to increase the generalisation area, as it simply reduces the input vector resolution to a level set by the quantisation level. Figure 5.5 illustrates the area and shape of a non

quantised mapped input vector, and Figure 5.6 illustrates a quantised mapped input vector where the quantisation factor is equal to 2.

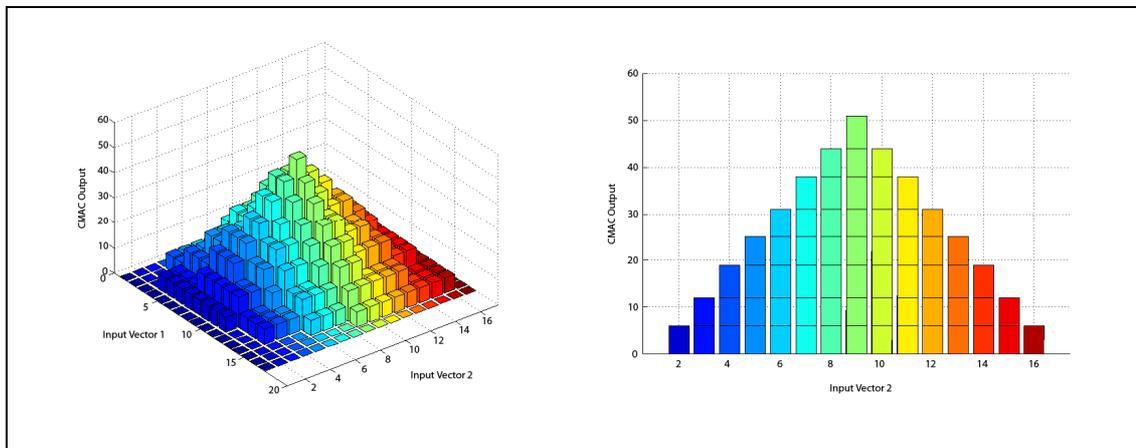


Figure 5.5 – Non-Quantised CMAC Input Space to Output Space Relationship (#RF = 8)

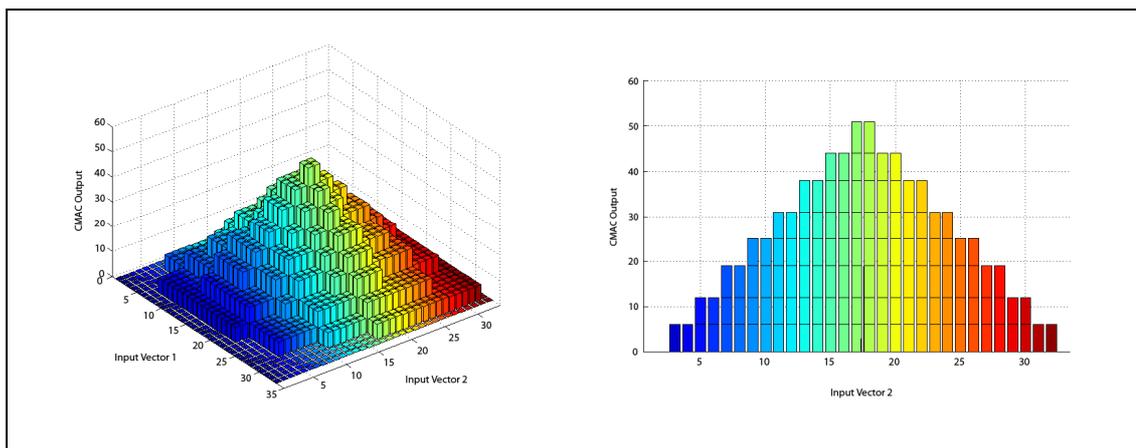


Figure 5.6 – Quantised CMAC Input Space to Output Space Relationship (#RF = 8)

At first it may seem that the space of the quantised generalisation area is larger than that of the non-quantised area, and this is true for the physical mapping space, but not for the CMAC input vector space. The CMAC input vector space area used is dictated directly by the number of receptive fields used and not by the quantisation parameter. As such, Figure 5.6 covers twice the physical mapping space as does Figure 5.5, while they both map to the same area in the CMAC input vector space with identical uniform spatial distribution patterns of the receptive fields. Thus the generalisation area of Figure 5.6 is twice the size of that in Figure 5.5, however the generalisation is coarser.

The other way to increase the generalisation area is by increasing the number of receptive fields used. Doubling the number of receptive fields in turn doubles the generalisation area of the physical mapping space, but in doing so also doubles the area used per dimension in the CMAC input vector space. The result is a smoother generalisation over the physical generalisation area. This is illustrated in Figure 5.7 where the number of receptive fields has doubled to 16 over that of only 8 used in Figure 5.5. Here the physical generalisation area is equal to that of Figure 5.6 whilst the actual CMAC memory space area per dimension has doubled.

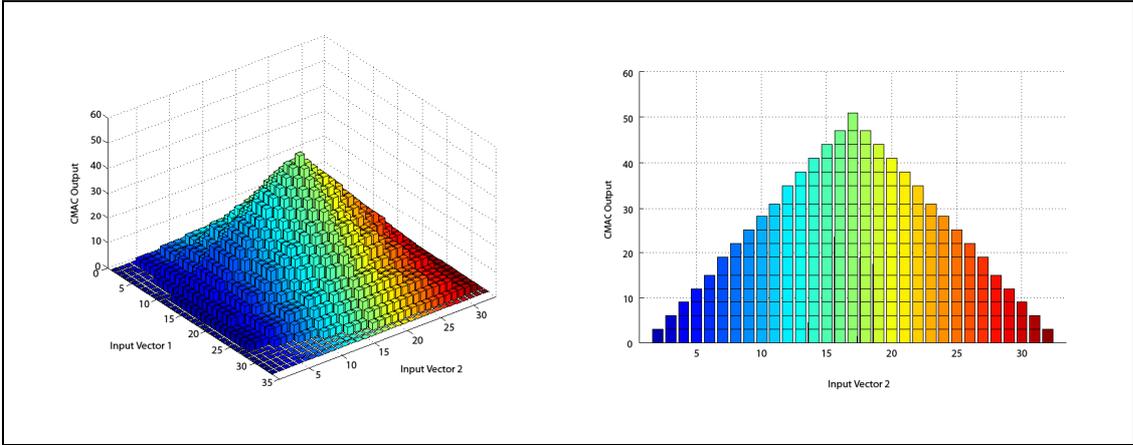


Figure 5.7 – Non-Quantised CMAC Input Space to Output Space Relationship (#RF = 16)

In terms of memory requirement for each generalisation method, building on from Equation 2.8 by incorporating the quantisation factor (q), we obtain:

$$n = K \left(\frac{Q}{q} \right)^N \tag{Equation 5.2}$$

Equation 5.2 indicates that increasing the number of receptive fields decreases the overall memory requirement, though not by as much as the equivalent quantisation factor (for a given generalisation area size). Thus there is a trade-off; a larger but smoother generalisation area using more memory or a larger but coarser generalisation area using less memory.

For this project, since minimal memory usage is not an issue, increasing the number of receptive fields to obtain a larger yet smoother generalisation area has been chosen over using a quantisation factor greater than 1.

The CMAC's local generalisation property not only plays a role in being able to generalise an output based on its current weights for a particular given input vector, but also helps to stabilise a system that suffers from time delays. An explanation of how the CMAC automatically overcomes small time delays will now be presented.

5.2.3 ACTUATOR RESPONSE TIME DELAY

A time delay in sampling periods from when the controller output is produced until when the system responds usually exists for most motion control problems. Time delay should be compensated for without expecting the CMAC to provide the time delay compensation automatically as explained in Section 2.8.6. There is no guarantee that time delayed generalisation areas will overlap, even when using large generalisation areas, and thus the system could become unstable.

To compensate for this time delay, first the time delay must be known and then a buffer used to store the input vectors used at time t to create the error at time $t+n$.

The time delay for the air-muscle system was found using the following method. A sequence of step input functions were presented to the actuator. The time was then measured from when the output signal was outputted to the actuator until the system responded to this input. The average of these times was then used to obtain the time delay of the system in control sampling periods n .

The next step involved storing the CMAC input vectors at time t in a First-In First-Out (FIFO) buffer of size n sampling periods long. For each subsequent sampling period, the data is moved progressively down the buffer. This way the input vectors at the end of the buffer will be those that caused the current error of the system. Using these input vectors, the CMAC can be trained with the current error which directly relates to the input vectors that created this error, and the time delay is no longer a problem in CMAC training. Figure 5.8 illustrates this process.

In Figure 5.8, the CMAC input vectors at time t create the CMAC output immediately, and the input vectors enter the time delay FIFO buffer. Due to the time delay, when the system finally responds at time $t+n$, the error of the system is used to

train the CMAC with the input vectors exiting the FIFO buffer. This way the corresponding error is used to train the input vectors that produced that error.

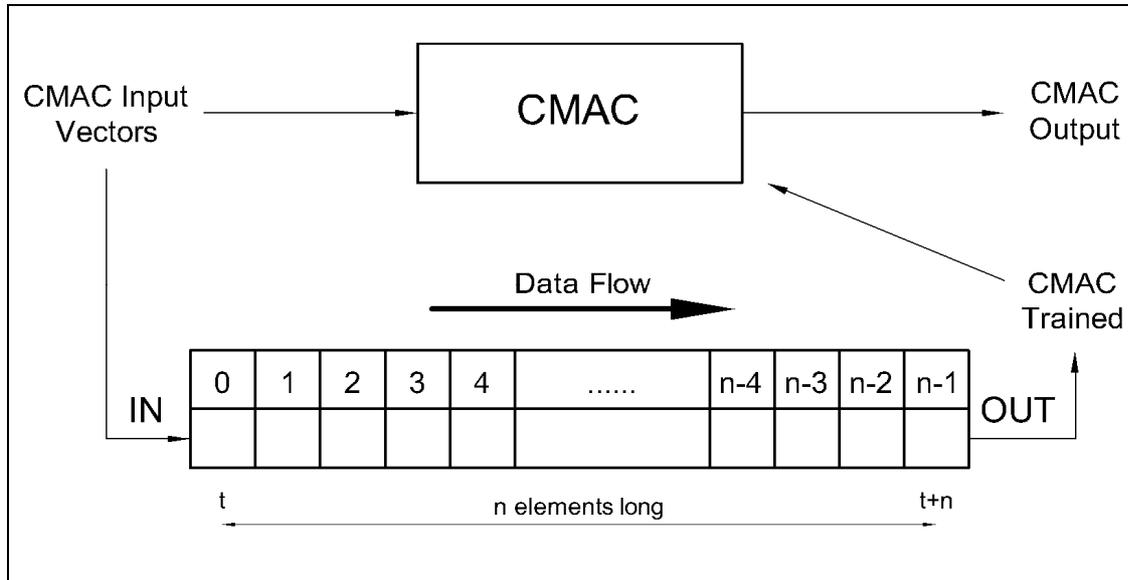


Figure 5.8 – FIFO Input Vector Buffer for Time Delay Compensation

5.2.4 WEIGHT SMOOTHING

The next important aspect in achieving stable CMAC control involves the concept of weight smoothing. Briefly discussed in Section 2.8.7.4, the weight smoothing implementation used here is that proposed by Miller and Glanz (1996). To implement weight smoothing in the Albus CMAC, the weight adjustment equation (Equation 2.9) is replaced by Equation 5.3.

$$\Delta W_i = g_1 \left(\frac{\hat{p} - p}{|A^*|} \right) + g_2 \left(\frac{p}{|A^*|} - W_i \right) \quad \text{Equation 5.3}$$

where g_1 and g_2 are separate gain values between 0 and 1, and ΔW_i is the weight adjuster to be added to active weight W_i .

To recap, weight smoothing involves restricting the deviation from the mean of the weights used in producing a CMAC output for a given input vector. It was mentioned in Section 2.8.7.4 that weight smoothing reduces residual error and sensor noise from creating spikes in the CMAC output. Testing revealed however that as weight

smoothing restricts the deviation from the average of a set of input vector mapped weights, an overall smoothness results in the CMAC output when following a continuous input trajectory. The gradual decline from the weight average at any one time means that adjacent input vector weights cannot deviate excessively from this average also. Thus sharp transitions in the output function are not possible if weight smoothing is implemented for a learnt trajectory. Figure 5.9 illustrates the effect weight smoothing has on a CMAC's output, removing sharp transitions in the controller output signal.

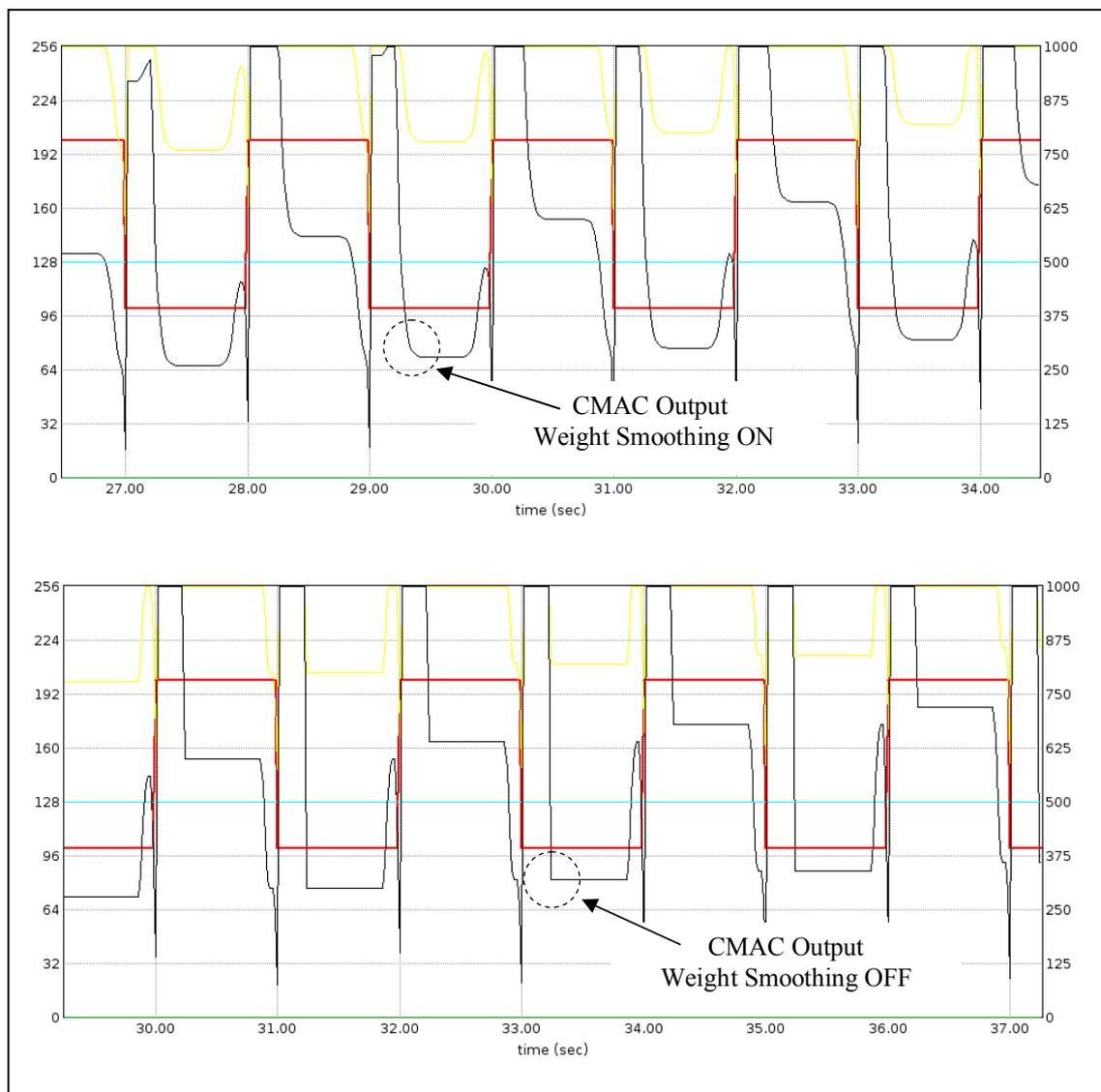


Figure 5.9 – CMAC Weight Smoothing ON and OFF

The amount of weight smoothing controls the overall smoothness of an output function, possibly limiting sharp transitions from occurring even if required. Weight

smoothing also increases controller stability, as it slows the chance of the system becoming unstable from large and rapid changes occurring throughout the CMAC's weight memory.

Weight smoothing also automatically implements its own level of eligibility. The smoothness in output transitions means that any particular weight will influence nearby weights. Thus a required large transition (e.g. a step function) in desired trajectory will cause the weights that lead up to the transition to move in the direction of the transition before the transition even occurs. Even though the input vectors on either side of the desired large trajectory transition may not lie nearby in input vector space and share no common weights, the compensated time delay means that weights earlier in time are those responsible for the error after the desired large transition has occurred. The overall effect is the CMAC moving the output of the system in the direction of the large future desired state transition, before the desired state transition even occurs.

5.2.5 CMAC WITH PID ISSUES

The combination of the CMAC feedforward controller with the PID feedback controller means that two controllers are providing independent driving forces into the system. In theory, if the CMAC controller can produce zero error at the output, then the error used to drive the PID equation will be zero. Thus the PID controller will cease to produce a changing output and remain as a fixed constant. As soon as there is an error, the PID controller will start to produce an output to drive the system error to zero. This sounds good in theory, however in practice there exists possible variables that will prevent the PID output from becoming fixed indefinitely. When using periodic input trajectories, it is possible that the summation over time of these residual errors will accumulate and drive the PID response to some very large value. For periodic signals there is no guarantee that the summation of negative and positive errors will cancel each other out over time. Thus it is required to implement systems into the controller to deal with these ever growing residual errors.

5.2.5.1 *Harmonious CMAC and PID Interaction*

Both the CMAC and the PID controller should be able to provide the full range of control values (set to 1000bits, see Section D.1 in Appendix D) to the system

regardless of what the other controller is outputting. If the CMAC controller is outputting zero, then the PID controller should be able to provide the system with the full control range of motion. In turn, if the PID equation is outputting zero, the CMAC should be also able to drive the system over its full output range.

As the output to the electronic hardware ranges from 0 to 1000 bits, it makes sense to limit the PID controller output to be capped at this range as well. Thus it is reasonable that the CMAC should be able to either add or subtract from the PID output and provide control compensation in both directions as required. Thus the CMAC output needs to provide an output of twice the range of the PID controller output, offset in either direction from a CMAC output of zero (i.e. CMAC output range is ± 1000). This way if the PID output is locked for some reason at 1000 bits, the CMAC can fully compensate the combined output all the way to 0 bits.

Capping the CMAC output is also necessary, as continuous errors that the CMAC cannot minimise will continue to build up to an infinite value. This leads to problems, especially when weight smoothing is applied to the CMAC. As discussed in Section 5.2.4, when weight smoothing is applied it takes longer for the CMAC weights to change from large weight values to small weight values throughout adjacent weights in the CMAC memory. This can be directly seen in Figure 5.9 where the slope of the CMAC output takes longer to decrease to a stable fixed value compared to when weight smoothing is not used. The greater a weight value becomes, the longer it takes for the adjacent weights to stabilise at the next required stable value.

This means that the combination of the PID and CMAC output values should in total never be greater than the 1000 bit control range. This approach above, whilst it caps the CMAC weight values to reasonable limits, still means that the maximum output for the system can be -1000 bits to $+2000$ bits, wasting 1000 bits either side of the 0 to 1000 bit control range.

To solve this, the CMAC training algorithm required an additional term when calculating the CMAC weight value. This value is the PID control value produced at time t , stored in the time delay buffer until training occurs at time $t+n$. The maximum

CMAC output possible is then based on the difference between the PID output value and the limits of the 1000 bit control range. This produces a final combined CMAC + PID control output which will always be within the 1000 bit controllable range with no final capping needed. The advantage is that there will be no time wasted moving from redundantly large CMAC weight values to lesser required weight values, whilst still permitting the CMAC to provide full output control compensation to the total output range of the system.

5.2.5.2 Resetting the PID Output

Capping the outputs from the CMAC and PID controllers to limit the overall output only partially solves the problem associated with the PID output drift resulting from the combination of residual PID input errors associated with the learning limits of the controller.

Another much more immediate problem faced by the controller from these accumulated residual errors is that as the PID output slowly drifts, the CMAC has to consistently compensate the drifting output. Thus the CMAC cannot converge to a stable value as the required output value is constantly changing, until the PID output is limited by the capping. As this is an unacceptable solution, and is only a side-effect of the capping and not its original intent, another solution is required to prevent the PID input errors from accumulating over time when using periodic signals.

The solution used here is rather simple and effective and was developed by the author. At the start of a new desired trajectory such as a hand sign gesture, the accumulated error in the PID controller is reset to zero. This way the error is unable to accumulate with the addition of each successive hand sign trajectory, and thus the PID output cannot drift indefinitely over time. This method is easy to implement and effective, and allows the CMAC to converge in the learning of each new hand sign trajectory.

The next chapter looks at a theoretical modification to how the CMAC is trained, adding an intelligent Situation Pre-empting training signal to the CMAC similar to

that of the 'eligibility' concept by Smith (1998), though using a totally different approach, specifically designed to minimise the error associated with the joining of deafblind hand sign trajectories.

6.0 THEORETICAL EMGC MODEL

With the CMAC+PID control foundation presented, the next stage involves the theoretical explanation of the Error Minimising Gradient Controller (EMGC) which precedes the CMAC+PID controller, providing error minimising control with respect to time to the overall system output. The EMGC with respect to the overall control system is illustrated in Figure 6.1.

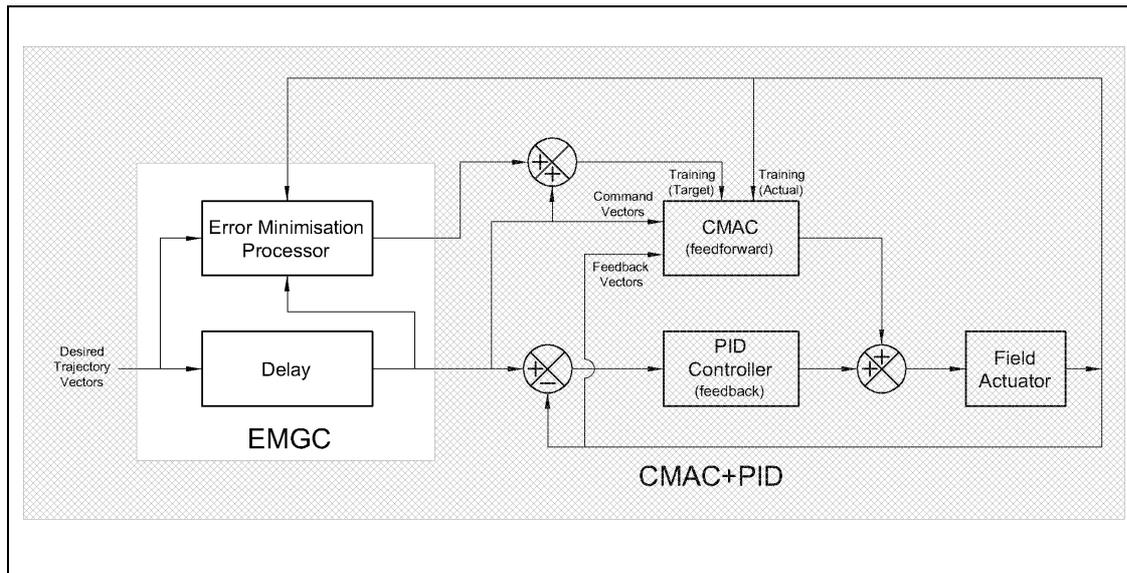


Figure 6.1 – EMGC in the Overall Control System

The EMGC in Figure 6.1 accepts two independent inputs and produces two independent outputs. The first input consists of the desired trajectory command input vectors. The second input consists of the actual field feedback vectors. The first output is the desired input vectors passed through a delay. The second output consists of the error minimising gradient, producing an error minimising signal which alters the training target in the subsequent CMAC (in the CMAC+PID controller).

6.1 PURPOSE OF THE CONTROLLER

Recapping from the end of the Literature Review in Section 2.9, errors between the desired and actual position response of a system are often compensated for by a controller at any particular point in time, as shown in the upper half of Figure 6.2. A CMAC+PID controller alone can produce such a response. The purpose of the EMGC together with the CMAC+PID controller goes one step further, minimising

the error in terms of both position and time, as illustrated in the bottom half of Figure 6.2. D_p and A_p represent the Desired Position and Actual Position trajectories.

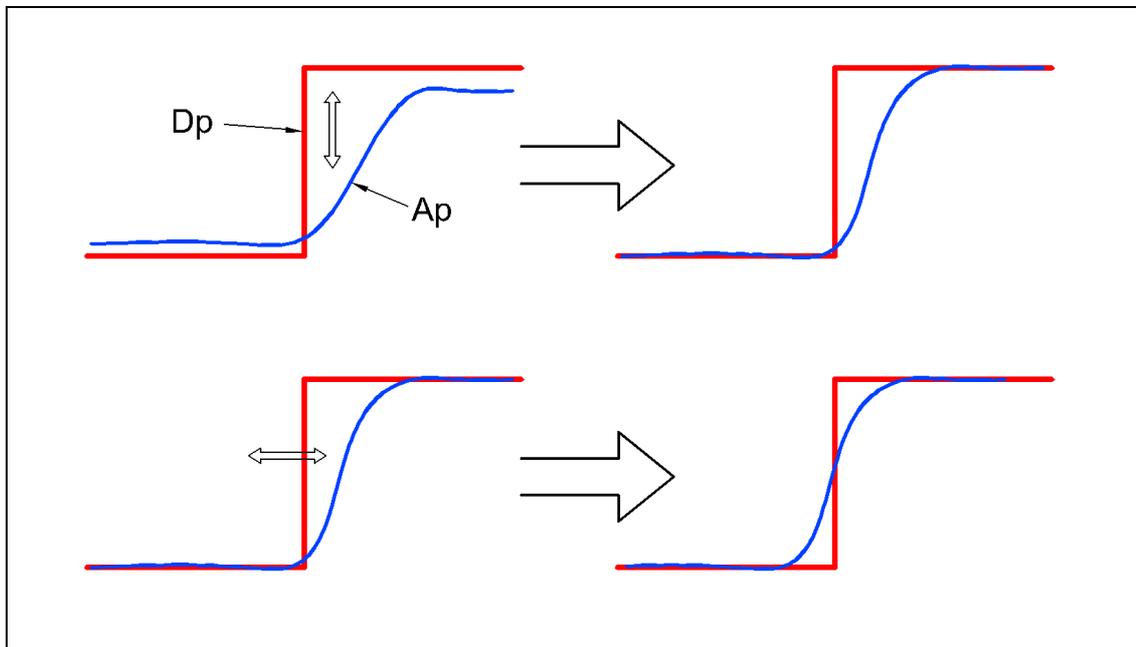


Figure 6.2 – Error Minimisation over Time (Identical to Figure 2.27)

The EMGC was specifically designed to minimise the error created by joining hand signs sequentially. Each hand gesture trajectory can start and end at any position throughout the actuator's range of motion, creating a rapid desired transition in the required motion from one gesture to the next. The transition from one trajectory to the next will be separated by a discontinuity, creating a 'situation'. A situation can be defined as a large transition between two desired trajectories, either positive or negative in direction, to which the actuator is not able to immediately respond. The desired step function in Figure 6.2 represents a situation in the desired trajectory. A situation exceeds the maximum physical velocity and/or acceleration limits of the actuator being controlled, and produces large error regions after the situation has occurred. This is illustrated in Figure 6.3.

In terms of using the EMGC to reduce the error over time associated with a situation, by using the concepts illustrated in Figure 6.2 and applying it to the trajectory response in Figure 6.3 will produce an error minimised trajectory for both position and time errors, as illustrated in Figure 6.4.

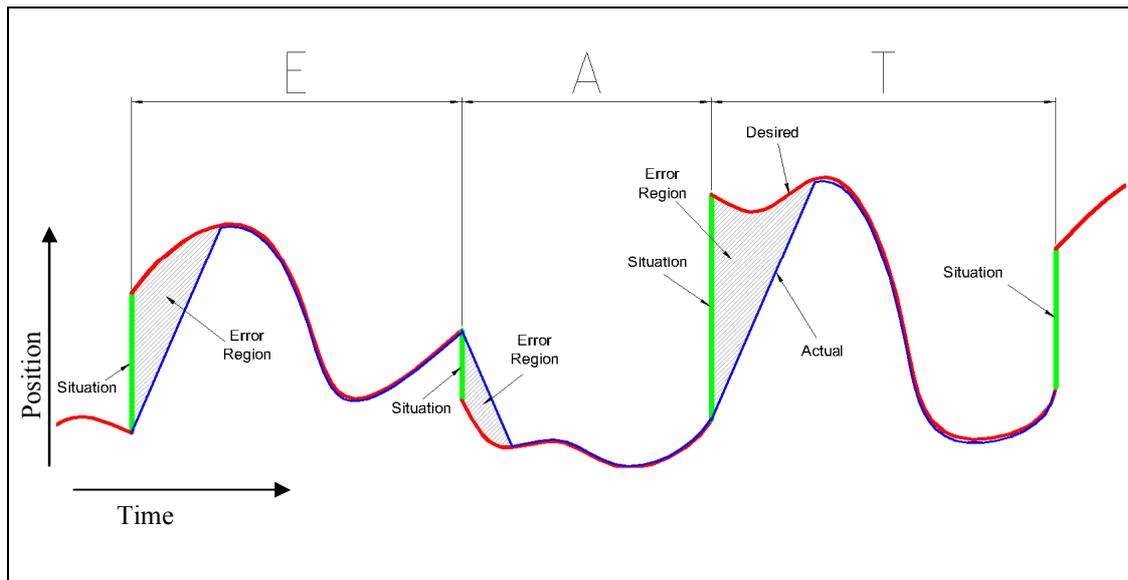


Figure 6.3 – Possible CMAC Control of Actuator for Desired Concatenated Trajectories (Identical to Figure 2.26)

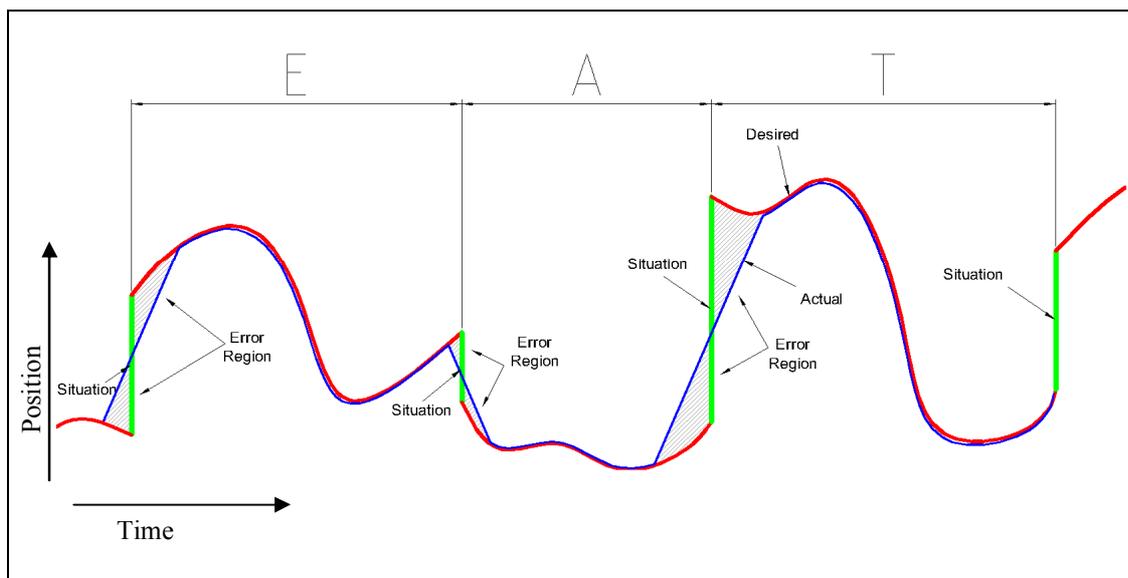


Figure 6.4 – Error Minimised Control of Actuator for Desired Concatenated Trajectories (Identical to Figure 2.28)

It is the purpose of the EMGC to take a trajectory response such as that in Figure 6.3 and to convert it into the error minimised response of Figure 6.4. This process allows the language information to be preserved as much as possible. Instead of losing a greater amount of information conveyed at the start of every new trajectory, a lesser amount of the ending of the previous letter trajectory and a lesser amount of the beginning of the subsequent letter trajectory are lost instead. This creates a better

approximation of the desired response by producing a smoother and clearer depiction of the word being communicated.

The basic definition of a situation has been presented, in context, to show how situations occur when joining together individual deafblind fingerspelled trajectories to form a word. The EMGC must be able to detect situations and analyse trajectory data before and after a situation occurs if the error created by a situation is to be minimised. A deeper look at situations will now be presented.

6.1.1 SITUATION DEFINITION

The situation examples given in Figure 6.4 illustrate a situation as an instantaneous change in desired position, followed and preceded by a smooth random trajectory. This instantaneous change in desired position is representative of the instantaneous change in desired position of a step function. The step function or square wave will be used as the classic example for situations throughout this thesis, as it produces a clear situation and creates a large and obvious error region following the situation, bounded by the desired and actual trajectory responses. This is illustrated in Figure 6.5, using the rising edge of a square wave desired trajectory to produce a clear situation.

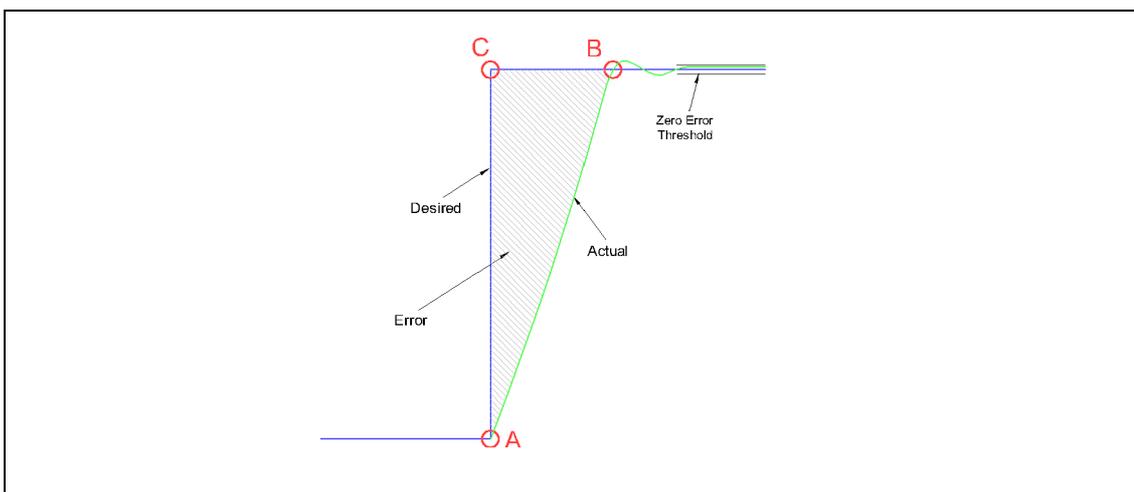


Figure 6.5 – Square Wave Trajectory Situation

Figure 6.5 shows a square wave desired input trajectory to the system. It is impossible for an actuator to accelerate at the rate needed to change its actual position as quickly as the desired square wave trajectory requires. The actual

trajectory response of the system starts to accelerate at its maximum capabilities (A), reaches its maximum velocity and remains there until (or close to) the actual trajectory is equal (or close to) to the desired trajectory (B). Once settled, the error between the desired and actual trajectory signals is below some threshold value, and the system no longer needs to drive the actuator.

From the time when the desired trajectory velocity exceeds that of the maximum actuator velocity (A) until when the actuator position meets the desired trajectory position, a large sum of error exists between the desired and actual trajectory response plots. Point (C) on the graph is the point at which the velocity of the desired trajectory drops below the maximum velocity threshold, which is within the maximum actuator velocity limits. From point (C) (after the system has recovered from the situation) to the next situation detected, the system should be able to learn and follow the desired trajectory, so long as the desired trajectory’s dynamic requirements are within the actuator’s limits.

Situations can also occur in smooth continuous trajectories. The same points illustrated for the square wave situation can also be applied to a smoother, more natural-like trajectory, such as that illustrated in Figure 6.6.

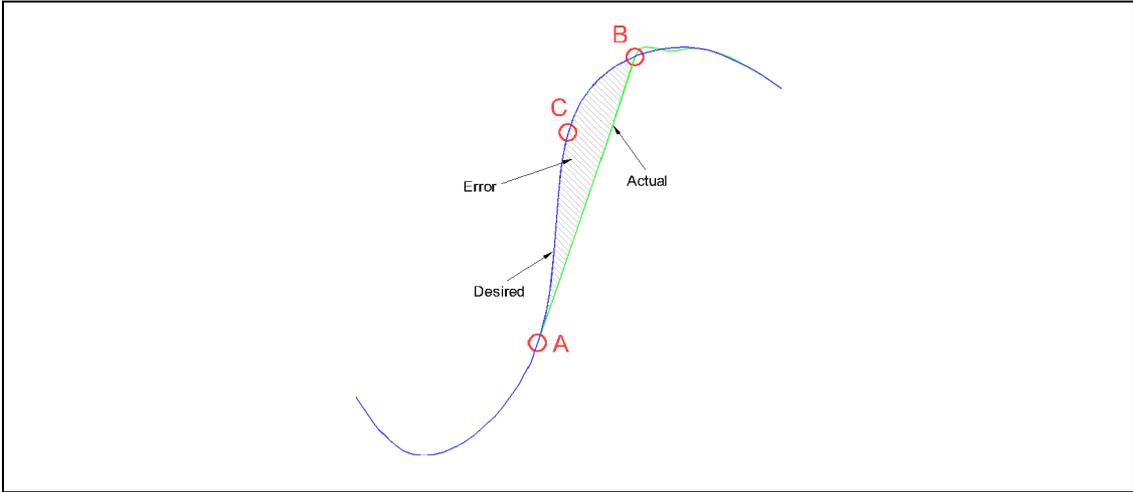


Figure 6.6 – Natural Trajectory Situation

Figure 6.6 illustrates another situation, however this is a more natural situation that can occur, as there are no sharp and immediate changes in the desired trajectory. Thus a situation can also be defined as part of a desired trajectory which exceeds the

maximum response limitations of the driven actuator. At the point where the actual response cannot match the requirements of the desired response, the desired and actual trajectory plots separate until the trajectories meet at point (B). This illustrates that even for smooth trajectories, situations can occur, and the detection of a situation is at Point (A), where a velocity threshold is exceeded by the desired trajectory. Point (C) is more clearly illustrated here as it is evident to see that the desired velocity starts to decrease below the maximum velocity threshold value. The situation is not resolved however until point (B) is reached.

6.1.2 ADVANCED KNOWLEDGE OF A SITUATION

It is impossible for a system to deal with minimising the errors in Figure 6.5 beyond the physical limitations of the actuator, even with the best control algorithms using classical or adaptive approaches, unless knowledge of the upcoming situation is known prior to the situation occurring. To illustrate this, think of a blind person walking towards a staircase down a path they have never walked before. The blind person has no firm knowledge that the staircase exists, unless they have been down this path before and remember where the staircase is in relation to some previous marker point (e.g. 10 steps away from a door). Only their walking cane will provide them knowledge about the staircase moments before they step onto the first step of the staircase. Without their walking cane, the person cannot make a decision about how to deal with the staircase until they reach it with their foot. At this point they will either pause, think, and step up onto the first step or fall down the step (depending on the desired trajectory direction). Thus it is vitally important that they know something about the staircase before they encounter it with their foot.

With control systems it is much the same. A control system cannot anticipate and make an educated decision about how to handle a situation (such as the blind person's upcoming staircase) before the situation is reached unless the system has prior knowledge of what situation is going to occur, and when it is going to occur.

Humans and animals make educated decisions and approximations based on prior learnt knowledge which has been classified into similar situations. Our visual and aural senses allow us to see and hear things before our body actually arrives at an event. When conscious we always know where we are travelling to or what is

travelling towards us, and thus we can make educated guesses about how to best navigate our way through our surroundings. We do not intentionally walk into tables and chairs. We step around them because we know where the table is and take corrective planned action before we physically encounter the table. The same goes for the Olympic hurdler. They see the hurdle and using an educated decision based on the height of the hurdle, distance to the hurdle and their running velocity, they jump 'before' they physically encounter the hurdle.

Again in the blind person's case, they will feel the staircase with their walking cane and lift their foot while in mid-stride to step onto the first step 'before' their foot reaches the step. They pre-empt. Thus this action means that their foot will land down on the first step without the delay that would be caused if they did not have their cane. They take corrective action before the situation occurs by looking forward in time with their cane. We will represent their present state as the position where their feet are, and their future state as the position where their cane is (in front of them).

Without this future awareness, it is impossible to take accurate corrective action before a situation occurs. As we are dealing with natural trajectory motion systems, it is highly desirable for there to exist a higher level of control to provide a system with a predictive element, and for a classification system to classify situations based on past experiences.

In this case, as we are learning fingerspelling trajectories, each fingerspelling trajectory is already previously known. The trajectory in itself is known and in addition, the fingerspelling trajectory that will follow the current trajectory will also be known. Sentences are made up of words which follow a known sequence. At any stage, there is previous planning which has produced a string of information which will be outputted as gestures in a sequence. Even the very first trajectory of the first sequence will be anticipated before it is executed. Thus for these systems, where trajectory planning is initially performed, there is knowledge available about 'situations' throughout the planned trajectory. Unless there is an instant change in desired output, situations can be seen ahead and dealt with accordingly. For example, if someone pushes the blind person while they are walking, a situation that they have

no future indication of, they will only start to correct their action once the situation has occurred (the unanticipated situation being the push). They have no prior indication that the push will occur and therefore have no way of starting to provide corrective action against the push until after it has occurred.

Thus for systems that allow it, having prior knowledge of the desired trajectory can be immensely helpful in planning what to do ‘before’ an imminent situation occurs. This is illustrated by considering the desired and actual trajectories in Figure 6.7.

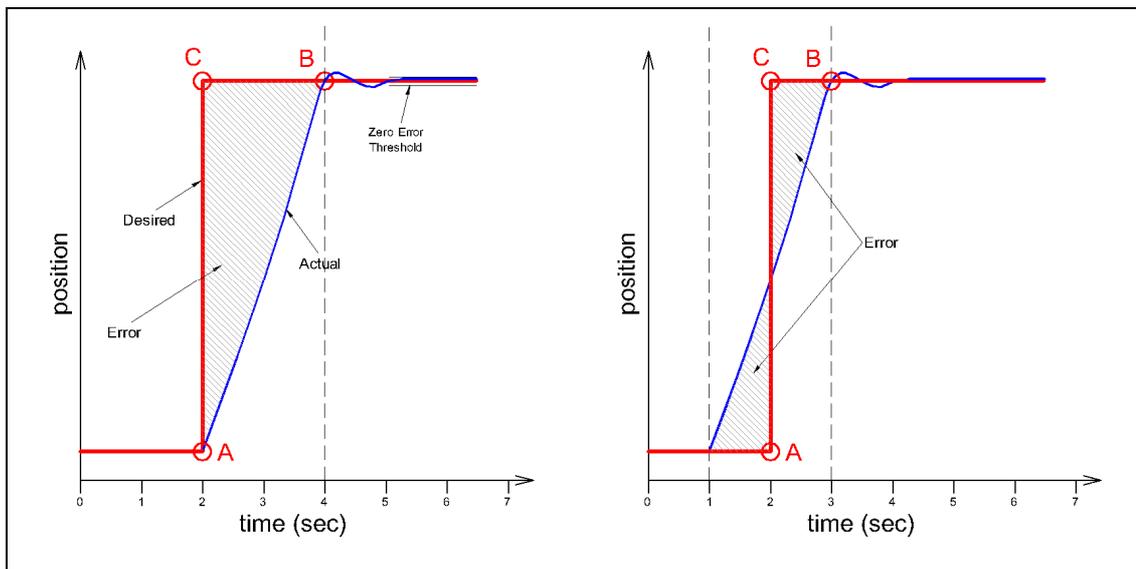


Figure 6.7 – (a) Square Wave without Error Minimisation, (b) Square Wave with Error Minimisation

Figure 6.7(a) replicates Figure 6.5 but now adds a time scale. This plot is equivalent to the blind person without their walking cane. There is no future indication that a situation will occur and therefore the system can only start correcting the sharp change in desired position once the situation has occurred, at Point (A).

Figure 6.7(b) shows how the system can control the actuator if the information about the situation is known before the situation actually occurs. Similar to how the blind person uses their cane to feel the staircase before their foot meets the first step, the system knows that a sharp change in position is going to occur and therefore starts to move the system in the desired direction ‘before’ the situation has even occurred. By the time the situation has occurred, the actual position is already half way to the desired position, arriving at Point (B) one second earlier than in Figure 6.7(a). While

the actual position has arrived at the desired position one second earlier, it has also started to deviate from the desired position one second earlier. Thus from a time perspective there is no gain in performance, as the total time where the desired doesn't equal the actual position is the same in both Figure 6.7(a) and Figure 6.7(b).

The benefits from starting to correct for a future situation before the situation has occurred exists in the minimised sum of errors between the desired and actual trajectories. This is evident when comparing the sum of the error areas between Figure 6.7(a) and Figure 6.7(b).

By moving the actual trajectory plot backwards in time by one second, the sum of the error magnitudes has decreased. Also the maximum error magnitude at any point in time is reduced by almost half. If moved back to two seconds in time, the sum of the error magnitudes start to grow again and so too does the maximum error magnitude. Therefore there is a point between 0 and 2 seconds prior in time to the situation, where moving the actual trajectory motion earlier will provide an optimised sum of errors for the situation.

While this point is not explicitly known for an unmodelled physical system, adaptive learning from experience can find this point. Thus the system needs some way of taking the prior knowledge of a situation and using it to provide an additional signal to the trained output to drive the system in the direction of the situation before the situation occurs.

6.2 EMGC THEORETICAL MODEL

The internal workings of the EMGC model will now be presented. As illustrated in Figure 5.1, the EMGC precedes the CMAC+PID controller, delaying the input signal to the CMAC+PID controller and providing error minimisation training signals to the succeeding CMAC. The EMGC section of Figure 5.1 is expanded upon and presented in further detail in Figure 6.8.

The following presentation of the EMGC theoretical model exists in two main parts. The first section deals with the situation processing phase, involving all aspects of how situations are detected, classified and processed. The second section deals with

the processing of an Error Minimising Gradient (EMGC), involving how an EMGC is created and used via the use of the data captured from the situation processing phase.

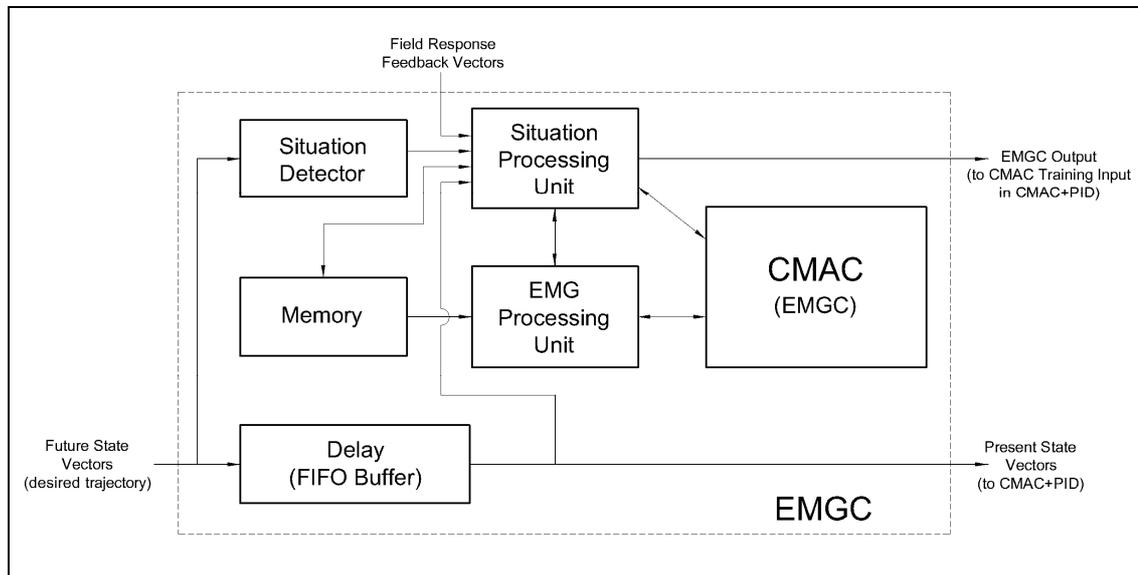


Figure 6.8 – EMGC Internal Theoretical Model

6.2.1 SITUATION PROCESSING PHASE

The EMGC allows the system to provide situation error minimisation, converting the response in that of Figure 6.7(a) to that of Figure 6.7(b). In effect it basically gives the blind person a cane. As explained in Section 6.1.2, having prior knowledge of an upcoming situation can allow a controller to produce an output signal that moves the actuator in the direction of the situation before the situation even occurs in present time. In much the same way as the blind person feels ahead with their walking cane, the input signal needs to be predicted or known before it is needed. This is actually the job of a system preceding the EMGC, which chooses what the actuator will do before the actuator is required to complete the task. Since this is beyond the scope of this work, the EMGC will be provided with a desired trajectory before it is required to be produced at the output (actuator) to allow the EMGC to anticipate what to do before the output is required by use of a delay. The delay is illustrated as part of the EMGC model in Figure 6.8.

Using a delay to allow the EMGC to anticipate means that two different time variables exist: future time and present time. The purpose of having future time variables is to enable the EMGC to see an upcoming situation before the present time

needs that situation to be handled. The present time variables are needed for comparison with the actual response of the system and also to provide the CMAC+PID controller with the present desired input trajectory data.

6.2.1.1 Delay – FIFO Buffer

Working with a sampled input trajectory, the EMGC uses a FIFO (First-In First-Out) buffer, storing position and velocity data about the desired input trajectory until the present time for that data arrives. This is shown as the Delay in Figure 6.8. The size of the FIFO buffer depends on the controller sample rate and the horizon distance. The horizon distance is defined as the distance in time between the start and end of the FIFO buffer. Figure 6.9 explains this concept.

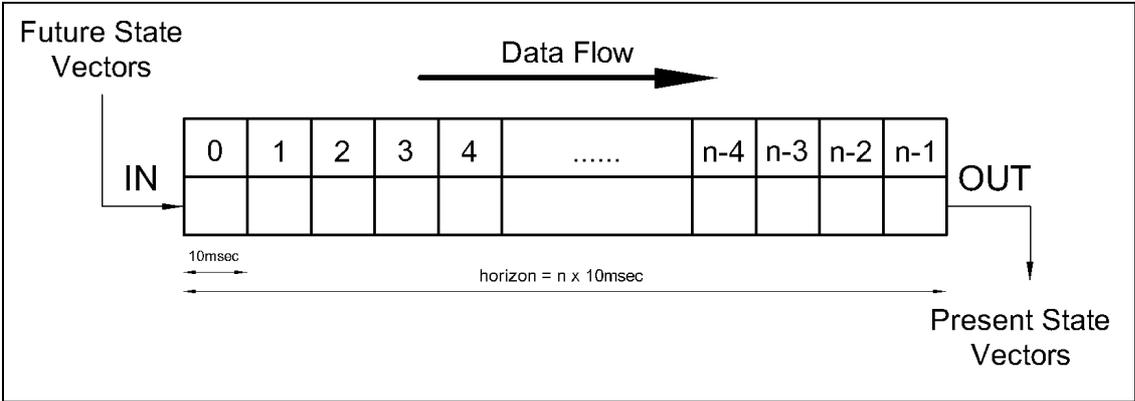


Figure 6.9 – FIFO Buffer

Future time trajectory parameters enter the buffer. As time progresses, this information makes its way to the end of the buffer. In this case each cell in the buffer holds data sampled once every 10msec. The buffer horizon is one second wide, meaning it will take one second from when data enters the buffer until the same data exits the buffer. One second worth of 10msec samples makes the FIFO buffer 100 samples wide (n). The value of n is customisable.

6.2.1.2 Situation Detection

The first processing step in the EMGC involves detecting a situation. This is determined by checking the future state data (velocity) and comparing it to a threshold (EMGC Desired Velocity Threshold). The threshold is a signed velocity value greater than the physical velocity limitations of the actuator for both directions of motion.

The detection method/algorithm for situations is not set in stone, and can use any number of techniques to determine when a situation occurs. For example, the rate of change of trajectory error may trigger a situation. For this thesis, only a simple comparison between the future state velocity and some velocity threshold which the actuator cannot match is used to detect a situation. In an initial calibration step for the EMGC, the step response of the actuator is tested in both positive and negative directions, and the EMGC remembers the maximum instantaneous velocity the actuator is capable of. These values are then used as the threshold values. This is illustrated in Figure 6.10(a).

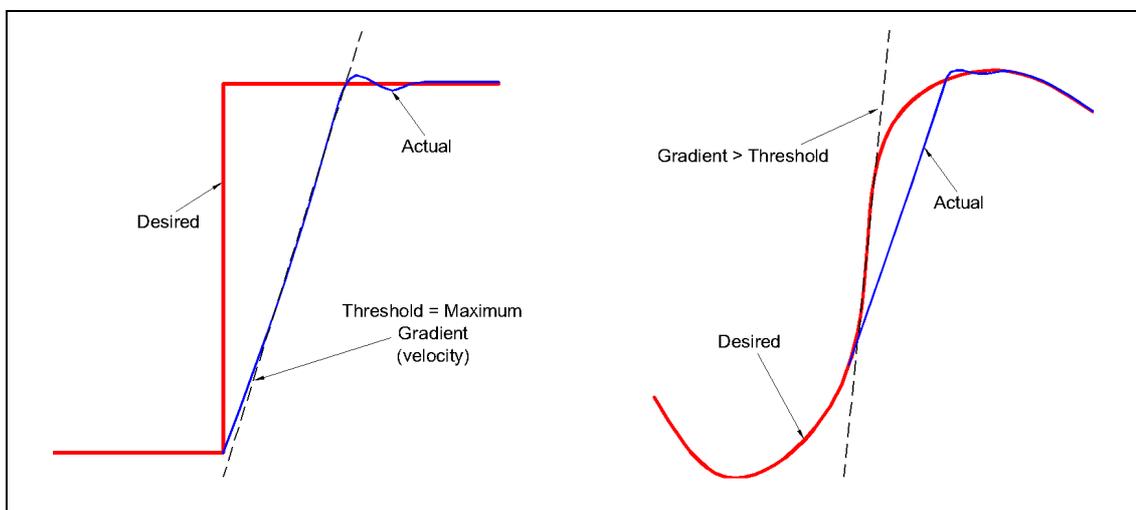


Figure 6.10 – (a) Threshold Gradient Determination, (b) Situation Detection using Gradient Comparison

Future instantaneous state velocities are compared against these values, and if the future state velocity is greater than the maximum velocity the actuator is capable of, it determines that a situation has occurred as illustrated in Figure 6.10(b). This method works sufficiently well for the test examples here, however a more robust situation detection method would be preferable in the long run, especially if the step response velocity of the actuator is less than approximately uniform.

6.2.1.3 *Situation Memory Variables*

After a situation is detected, current active situation variables need to be recorded so they can be dealt with later by the EMG Processing Stage of the EMGC. The Memory unit, as illustrated in Figure 6.8, stores this data. At every sampling interval

(every 10msec) five variables are saved in memory, assigned for the particular situation until successful situation completion occurs. These variables include:

Timestamp: A high resolution timestamp is stored in microseconds. This variable assists in determining the elapsed time between sampling periods. When comparing how far position feedback data has moved, i.e. finding the rate of change, etc., it is critically important that an accurate time base is used. Thus for every sample taken, a high resolution timestamp variable is recorded.

Actual Velocity: The current actual velocity of the system is stored at each time sample interval to later find the maximum velocity the actuator achieved. This value is critically important as it allows the EMGC to determine what Error Minimising Gradient (EMG) (see Section 6.2.2.1) to use to compensate the original desired trajectory with.

Desired Velocity: The desired velocity at the present time (as opposed to future time) is stored at each sample interval to later find the point on the desired trajectory plot where the desired velocity drops below the situation detection threshold. The difference in time from the end of the situation to this point is used in calculating the time factor to move the EMG forwards and backwards in time by (producing the prediction effect of the EMGC).

Desired and Actual Position: The desired position at the present time together with the actual position is stored at each sample interval to find the error at each interval. The summation of these errors over the situation is then used in calculating the time factor to move the EMG forwards or backwards in time by, based on the previous EMG calculation (from the last time the situation occurred).

These variables are post-situation processing variables, and the use of these variables will be explained later throughout Section 6.2.2.

6.2.1.4 Situation Classification

Once a situation is detected, the situation must then be classified. Classification is necessary to allow the EMGC to assign the updated corrective parameters to the

same situation the next time it is encountered. If a string of different situations occurs more than once in a random sequence, the EMGC should be able to handle each situation separately based on the learnt data for that particular situation from past encounters.

Classification can be achieved in a number of ways. In our case, we will simply use two vectors to classify a situation: the desired position and velocity 100msec after the situation has been detected. This is illustrated in Figure 6.11, with position and velocity values taken at Point (H).

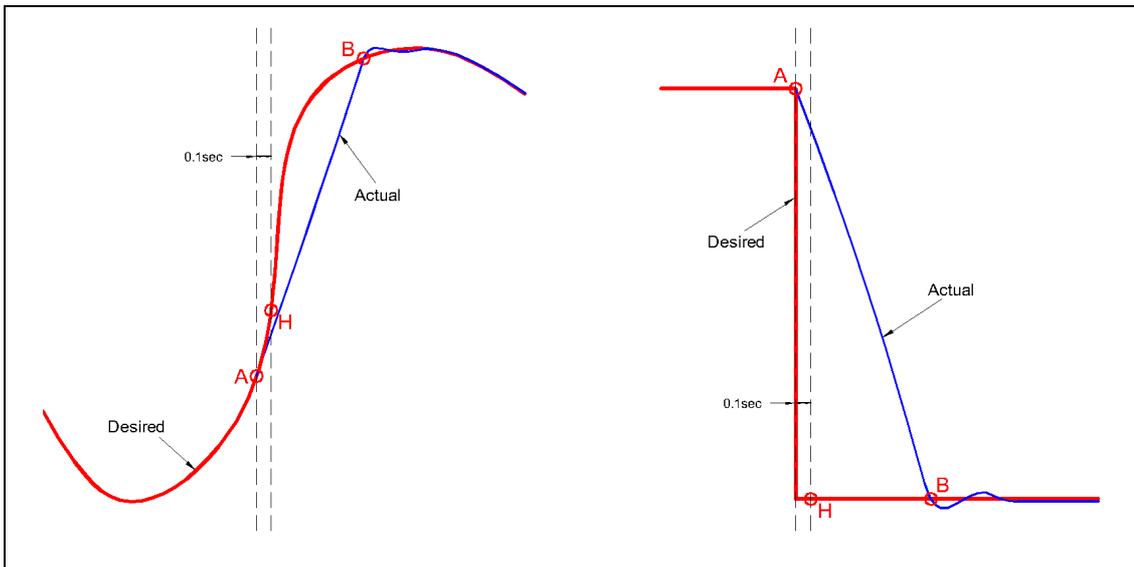


Figure 6.11 – Situation Classification Vectors for Two Different Situations

This classification approach is far from an optimal classification solution, and different situations can produce the same classification vectors using this method. However for our tests this method proves fairly useful and successful. A far more robust classification approach would be highly desirable, but robust trajectory classification is beyond the scope of the work here. However the robustness of the EMGC largely depends on both situation detection and correct situation classification, and thus future work in this area would be greatly beneficial.

However, as long as the situation vectors for a situation are the same the next time the situation is encountered, the EMGC will be able to build on the knowledge already learnt from encountering the situation previously. Thus for periodic signals or repeatedly encountered trajectories, this method should work extremely well.

6.2.1.5 Situation Assignment

Once a situation is detected, it is also assigned an identification number. An ID number is required because the system is designed to be capable of handling more than one situation at a time. This may seem pointless at first, but take an example where within the horizon time more than one situation occurs. The EMGC will be handling more than one situation at once. While only one situation at a time will physically be compensated by the EMGC output, the EMGC will still be recording data for the next situation.

An example of this is illustrated in Figure 6.12, demonstrating the need for the EMGC to be able to handle any number of situations that occur during the horizon, up to a user-defined limitation.

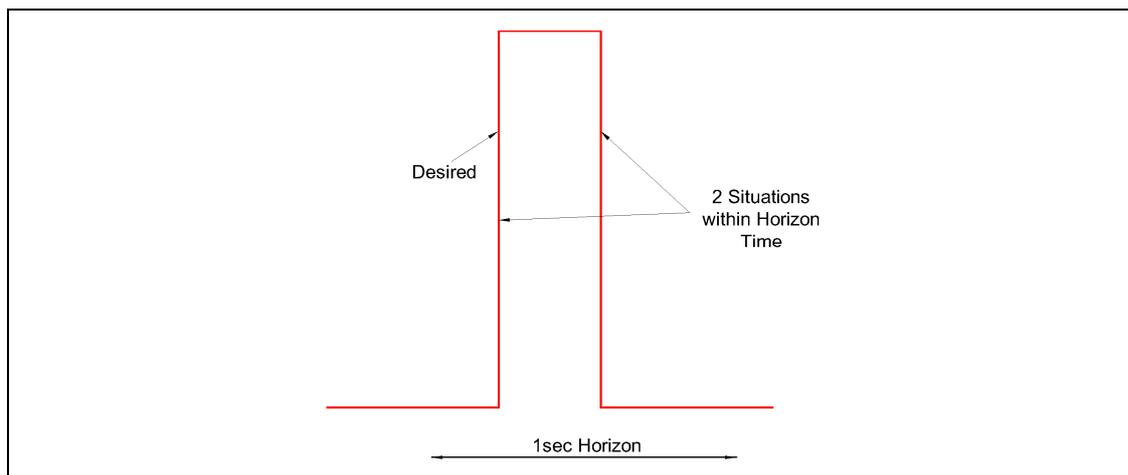


Figure 6.12 – Multiple Situations within Horizon Time

Figure 6.12 illustrates two situations occurring within the horizon. The first situation is detected first and thus becomes the ‘active’ situation. The variables for this situation are recorded in memory at each sample interval, and the situation is also providing EMGC trajectory compensation in real-time. Once the second situation is detected, even though the first is still the current active situation, the EMGC keeps track of this situation, storing variable data at each time sample interval, waiting for the active situation to complete. This ‘pending’ situation is not currently active, and thus is not providing EMGC trajectory compensation yet until the first situation completes and gives its active status to the pending situation. The first situation moves from ‘active’ status to ‘training’ status, while the second situation moves from

‘pending’ status to ‘active’ status. When the second situation completes, it too will move to ‘training’ status, waiting to train the EMGC with its situation data stored in memory.

Once the EMGC is completely finished with a situation (see Section 6.2.1.6), meaning the situation has completed training the EMGC with its situation data, the EMGC will release the ID number of the situation, ready to be assigned to the next situation that is detected.

6.2.1.6 *Situation Completion*

Successful completion of a situation relies on the occurrence of a single factor, though only if a particular criterion is satisfied first.

The situation completion factor when detected, allows the situation to be completed and moved into the EMG Processing Unit. This factor occurs when the actual position arrives at the desired position, Point (B). When Point (B) is reached, the error region area is closed and the EMGC has enough data to start processing the situation.

This factor however will only trigger a true situation completion if and only if a criterion is satisfied first. This criterion involves detecting the output value of the controller at the actuator and comparing it to two thresholds: the EMG Minimum and Maximum Trigger thresholds. The EMG Minimum and Maximum Trigger Thresholds are threshold values in the CMAC+PID controller output scale. These values are usually set to levels just before saturation of the output signal is reached.

The CMAC+PID output is checked to see if the CMAC+PID controller is pushing the output to its limits or not. As the purpose of the EMGC is to reduce the error associated with a situation, the CMAC+PID controller will be producing saturated maximal/minimal outputs when a situation occurs. When the CMAC+PID controller can no longer push the actuator any further once its outputs are saturated, the EMGC can then start working to reduce a situation’s error. It is the consistently large or small controller output for a situation that can be detected and used to tell the EMGC

to complete an active situation and thus train the EMGC CMAC with the acquired data.

Thus if the CMAC+PID output lies outside the EMG Minimum and Maximum Trigger thresholds meaning the CMAC+PID output is maximally stressed, and Point (B) is reached, then a situation has been successfully completed. Memory variables for the situation are locked and stored, and the situation is ready for processing by the EMG Processing Phase.

A situation is terminated and the memory variables for a particular situation discarded if the situation completion factor and criterion are not reached within a specified timeframe.

6.2.2 EMG PROCESSING PHASE

Once a situation is successfully completed, the situation moves into the EMG Processing Phase handled by the EMG Processing Unit. All vectors for the situation cease being recorded at each sample interval and it is time for the recorded vectors to be processed by the EMG Processing Unit. Figure 6.8 illustrates the EMG Processing Unit in relation to the rest of the EMGC.

Ideally the EMG processing unit should run in parallel to the Situation Processing Unit as it is not required to execute EMG tasks in real-time. However in this case, as the PC processor executes code sequentially, code executing within either the Situation or EMG processing units can only run at any one time. This is generally not a problem as the Situation Processing Unit has priority and executes whenever needed in real-time. The EMG processing unit processes data whenever the Situation Processing Unit is not needed (i.e. in between sample intervals when the processor is idle). However, since both the Situation and the EMG processing units use the same CMAC for data retrieval and storage, if the EMG processing unit is writing to the CMAC, the Situation Processing Unit cannot access data from the CMAC. This is usually not a problem as the speed of either processing unit execution is much higher than the controller sampling frequency. However on the odd occasion that the Situation Processing Unit needs to access the CMAC while it is being trained by the EMG processing unit, then the Situation Processing Unit will skip a sample interval

output. The EMGC output value will remain the same as for the previous sample interval, and change again for the following sample interval. As this data retrieval skip will only occur on the odd occasion and be only 20msec long (two sample interval periods), no noticeable effect should occur on the overall controller output. Such cases could exist however on a heavily loaded system, with many CMACs requiring processing at once. Further testing is required to obtain hardware processor limitations here.

For the following explanations of the EMG processing unit, the trajectory plot of Figure 6.13 will be referred to. For consistency, the points on Figure 6.13 also correspond to those of early trajectory plot figures.

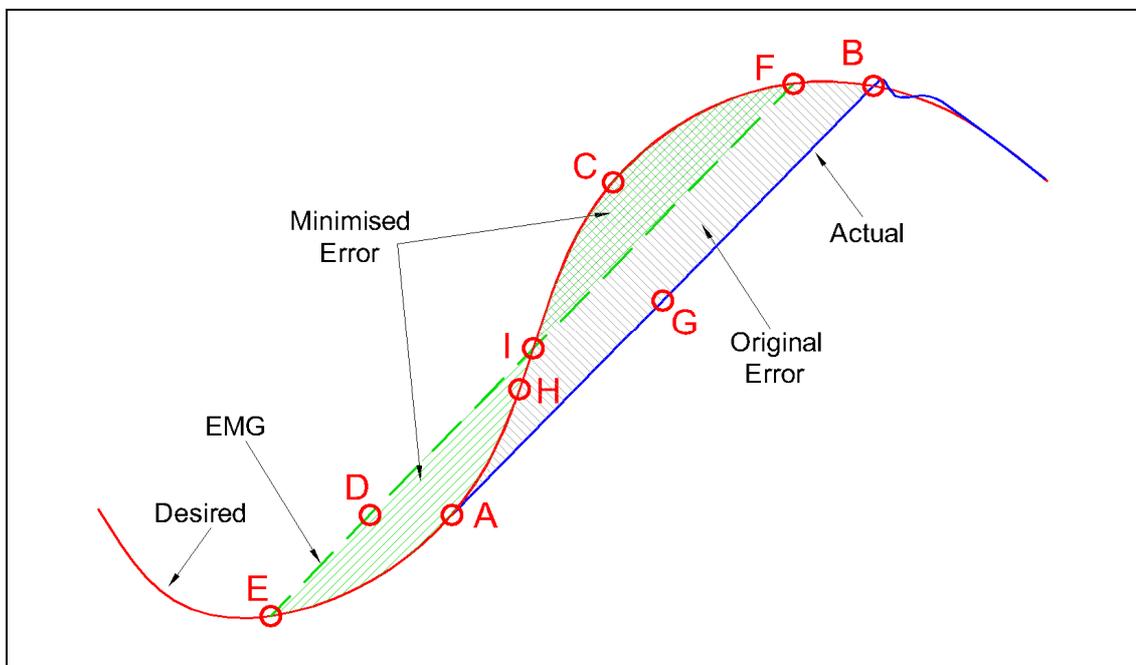


Figure 6.13 – Standard Trajectory Plot Points

The following points are illustrated on the plot in Figure 6.13.

- (A) Start of Situation: Desired velocity exceeds maximum velocity threshold.
- (B) End of situation: Actual Position = Desired Position.
- (C) Desired velocity drops below maximum velocity threshold.
- (D) EMG Origin.
- (E) Error Minimising Gradient (EMG) = Desired Position (start point)

- (F) Error Minimising Gradient (EMG) = Desired Position (end point)
- (G) Maximum Actual Velocity.
- (H) Classification Point: Desired Position and Velocity 0.1sec after Point (A).
- (I) Error cross-over point.

6.2.2.1 The EMG

The Error Minimising Gradient (EMG) is a superposition of the original trajectory whereby an additional trajectory gradient is added before and after a situation to minimise the error summation associated with the situation. For example, Figure 6.14 shows how the addition of an EMG minimises the error associated with a situation.

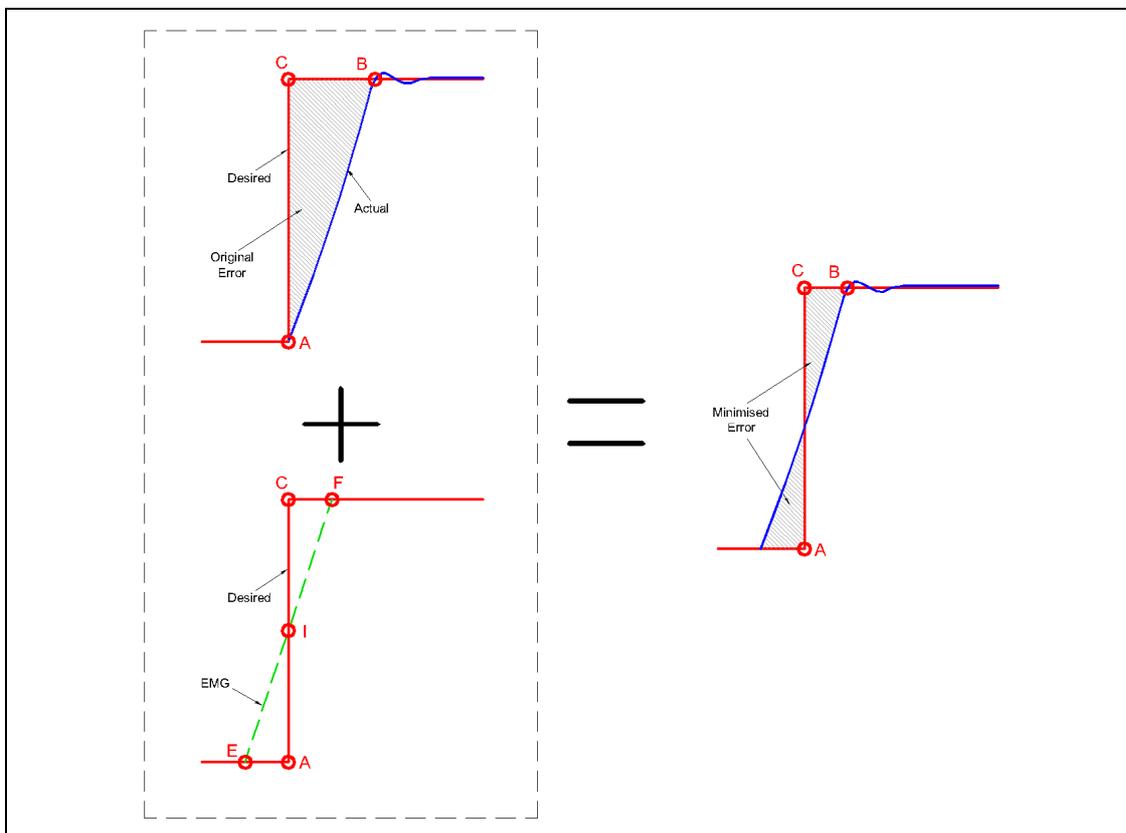


Figure 6.14 – Original Situation with the Addition of EMG

6.2.2.2 Recall Previous EMG Origin

The first process in the EMG processing unit is to recall from the CMAC the previously stored EMG origin from previous occurrences of the situation being handled by use of the stored Situation Classification Vectors. The purpose of the EMG origin is to locate the EMG some distance in time prior to when the situation

occurred at Point (A), illustrated in Figure 6.13 as Point (D). As the EMGC progressively learns a situation, the EMG origin is moved to the optimal position in time to produce the least amount of output error from a given situation to produce the final response illustrated in Figure 6.14. The EMG origin is critical for the system to iteratively adapt, as the EMGC can build upon knowledge learnt from previous encounters of a situation.

The EMG origin is stored in the same CMAC as the rest of the EMGC data, utilising an unused portion of the CMAC to store this data. The maximum number of input vectors for a given dimension is approximately 2048 (minus receptive field edge limits). For a horizon of one second producing a maximum situation memory length of twice that to 2 seconds, at a sampling frequency of 100Hz, this uses a total of 200 input vectors for the CMAC's 'Time' input dimension. Thus a substantially large portion of the CMAC is not used. We exploit this design decision and make use of it by storing the situation EMG origins in this portion of the CMAC (i.e. a 'fake' Time vector is used as a lookup vector for the EMG origins). The vectors used to relocate the EMG origin are those used to classify the situation (i.e. the classification vectors at Point (H), 0.1 seconds after the situation is detected).

6.2.2.3 A Practical Consideration: The EMG Origin Offset

The EMG Origin Offset is an offset applied to the EMG origin position so the EMG always lies prior in time to when the actuator starts to move in the direction of the situation. It is critically important in allowing the EMGC to work from the very first EMGC training iteration. Depending on the type of base controller used, here it is a CMAC+PID controller, whereby the CMAC is utilising weight smoothing together with a time-delay, a small amount of inherent Situation Pre-empting already exists. This means that the native output from the CMAC+PID controller without the use of the EMGC already starts to drive the output in the direction of a step change before the step change actually occurs. This is due to the large increase in CMAC output and the weight smoothing causing the weights that lead up to the step change in desired output to change gradually. (See Section 5.2.4 on Weight Smoothing.)

What this means when this type of controller is being used with the EMGC is that the conceived EMG origin is not actually at the point on the desired trajectory where the

slope of the desired trajectory exceeds the EMGC Desired Velocity Threshold (see Section 6.2.1.2). The figures throughout this chapter indicate that the actual initial output of the system, before the EMGC adds its EMG superposition, will look like that in Figure 6.14. For our system however, this is incorrect as there is a slight level of Situation Pre-empting occurring, producing an initial response like that illustrated in Figure 6.15.

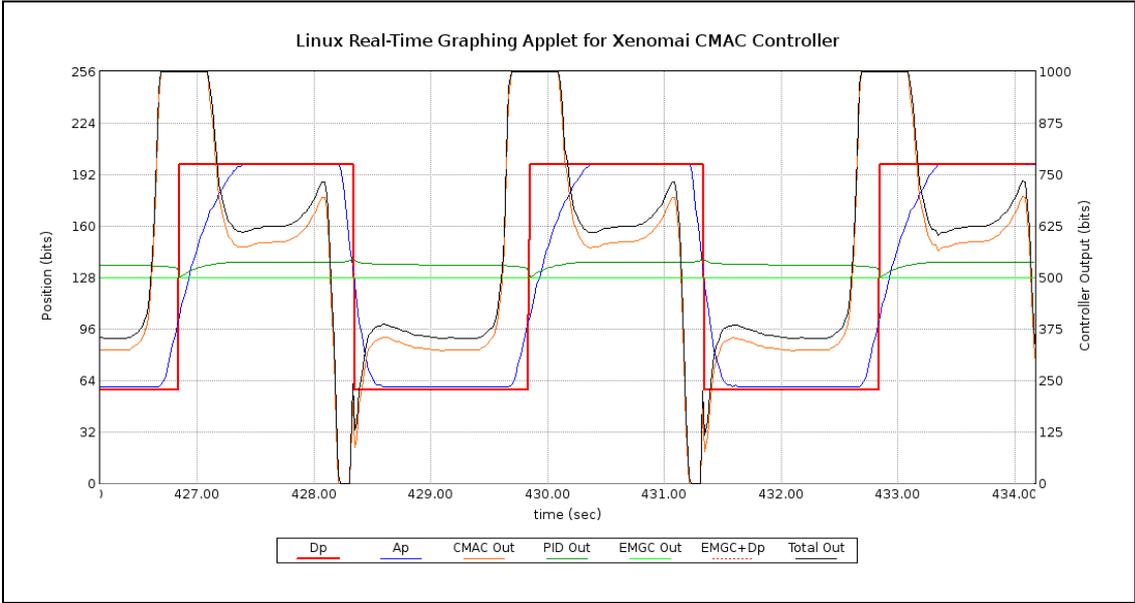


Figure 6.15 – Initial Physical Response of the Air-Muscle before the EMG Superposition Occurs

Note the differences between the initial actual position response in Figure 6.14 and that of Figure 6.15. It is clear that the original EMG origin cannot be at Point (A) like it would be in Figure 6.14. If it were, it would not be able to push the actual trajectory backwards (earlier) in time since the EMG would occur after (later than) the actuator has already started to move in the required direction. The new EMG origin obtained from Eq. 6.3 will produce an EMG origin which will be offset earlier in time than the original EMG origin, but there is no guarantee that it will exist prior to the start of the actual trajectory’s state transition. And this of course defeats the purpose of the EMG superposition. The EMG is used to push the actual response of the system backwards (earlier) in time to reduce the error associated with a situation.

Thus what is needed is an offset to be always applied to the EMG origin position so the EMG always lies prior in time to when the actuator starts to move in the direction of the situation. This way the EMG is always driving the actual response of the

system in the direction required to minimise the situation's error magnitude summation. Without the EMG Origin Offset, the EMG will be driving the system in the wrong direction, as the EMG will always be on the wrong side of the actual response (i.e. the EMG must always initially exist prior in time to the actual system response for a situation).

6.2.2.4 Find the EMG Slope with EMG Offset Percentage

The next step is to obtain the slope of the EMG. The EMG slope is based on the maximum velocity obtained by the actuator for a situation, shown as Point (G) in Figure 6.13, and the EMG Offset Percentage.

The EMG Offset Percentage is used to create an EMG with a satisfactorily steep gradient. The EMG Offset Percentage is a percentage of the maximum attained gradient (of the desired trajectory over the time span of an active situation) added to itself, producing Equation 6.1.

$$EMG = (1 + EMGOffset\%) * MaxVel \quad \text{Equation 6.1}$$

The EMG Offset Percentage is required is so that the controller output continues to be saturated, in much the same way as the original situation saturates the controller output. Situations require more from the actuator than is physically possible, and it is the EMGC's purpose to move the same response earlier in time to minimise the overall error associated with a situation. EMG offset percentages greater than zero are required to obtain the best performance from the EMGC, and this is demonstrated in the result section, Section 8.1, where different EMG Offset Percentages are shown to produce different situation error magnitude summation values, using EMG Offset Percentages between 0% to 100%.

The creation of an EMG with a 50% EMG Offset Percentage is illustrated graphically in Figure 6.16. The greater the EMG Offset Percentage, the greater the slope of the EMG. This value rotates the EMG about the EMG origin, with EMG Offset Percentages greater than zero rotating the EMG towards the vertical, and EMG Offset Percentages less than zero rotating the EMG towards the horizontal.

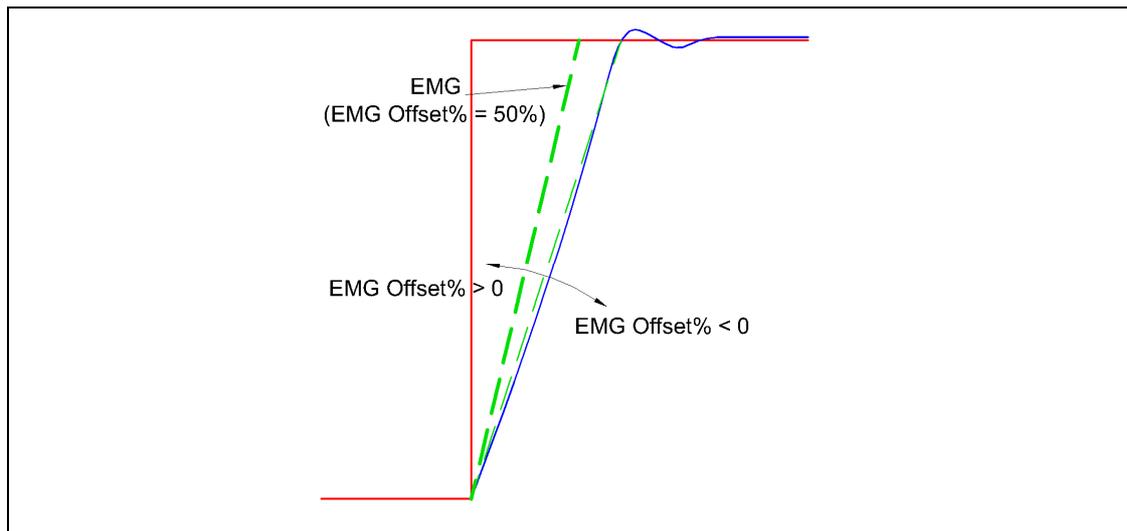


Figure 6.16 – Creation of an EMG Using a 50% EMG Offset Percentage

6.2.2.5 *The Error Associated with the Situation*

As the purpose of the EMGC is to minimise the error associated with the occurrence of a Situation, an EMG must be produced which moves the actual trajectory response backwards (earlier) in time. When this happens two smaller errors will result as opposed to a single larger error occurring only after the Situation has occurred as illustrated in Figure 6.13. Thus the EMGC is required to move back the EMG to a position in time where the two smaller errors equal each other. Figure 6.17 illustrates the common errors associated with a Situation.

When a situation is encountered by the EMGC for the very first time, only the 1st Error Region is guaranteed to exist, as there is no EMG yet to move the actual response backwards in time. However when the situation occurs again, the addition of an EMG for a particular situation will result in the creation of a 2nd Error Region as illustrated in Figure 6.17. Note that with that addition of the EMG, Points (B) and (F) become the same point, however Points (A) and (E) still remain separate.

Therefore the next step in calculating the EMG is to calculate the error associated with the Situation. Starting from the end of the Situation at Point (B) and working back in time, the difference between the desired and actual trajectory responses is calculated for each sample interval.

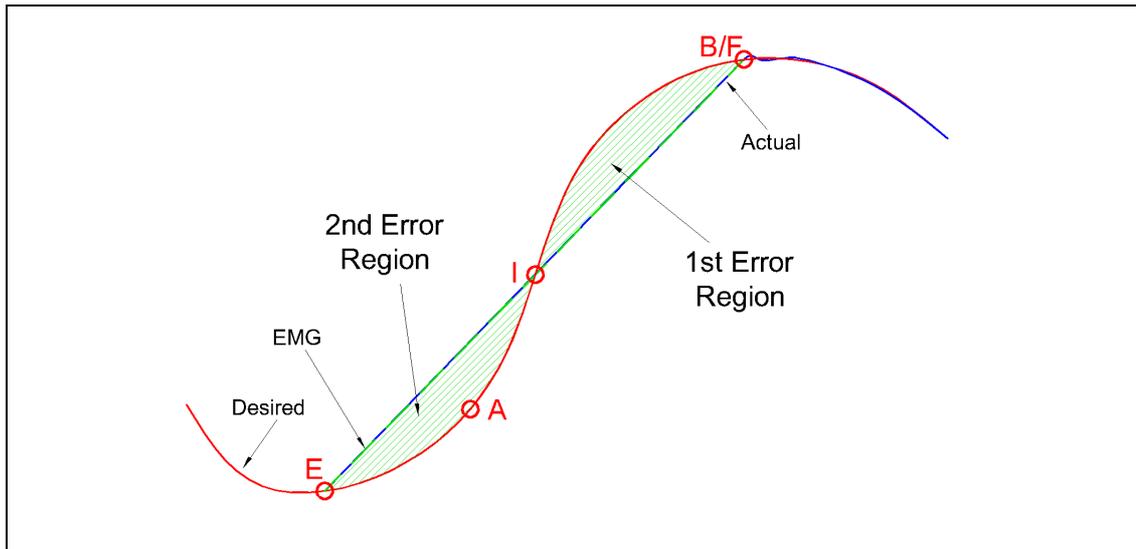


Figure 6.17 – Situation Error Regions

This process is continued down the sample interval memory until Point (I) is reached. Point (I) is the Error Cross-Over point where the error between the desired and actual trajectory responses changes sign. For example, the difference between the desired and actual response positions producing the 1st Error Region will result in positive error values. However the difference between the desired and actual response positions producing the 2nd Error Region will result in negative error values. It is this change in sign that allows the EMGC to detect the boundaries of the 1st and 2nd Error Regions. The summation of the sampling magnitude errors for each region produces the ‘Total Error’ for each region.

The difference between the two error regions is then computed using the ‘Error Ratio’. The Error Ratio is computed by the equation:

$$ErrorRatio = 1 - 2 * \frac{2ndError}{1stError + 2ndError} \quad \text{Equation 6.2}$$

Producing: 0 if the errors are equal,
 1 if the Total Error is all 1st Error,
 -1 if the Total Error is all 2nd Error.

The EMGC tries to minimise the Total Error associated with a situation by equalising the magnitudes of a situation’s first and second error regions. The EMGC does this

by converging the Error Ratio of a situation to zero. Using this 1st and 2nd Error Area Magnitude Equalisation approach to minimise a situation's Total Error, produces an approximately minimal error over time for a variety of situations tested in this thesis. This argument is explained in further detail in Appendix E.

The Error Ratio allows the EMGC to know whether the EMG origin should be moved backwards or forwards in time with reference to the previous EMG origin. However another factor is also used in this procedure. This is the EMG origin time-shift factor.

6.2.2.6 *EMG Origin Time-Shift Factor*

The Error Ratio is used together with the EMG origin time-shift factor to move the EMG in the direction required to minimise the overall error associated with a Situation. The purpose of the time-shift factor is to allow situations of different time lengths (the time from Point (A) to Point (B)) to be converged at a similar rate.

The first step in obtaining the time-shift factor involves finding Point (C) in Figure 6.13. Point (C) is the point on the desired trajectory where the desired velocity drops below the situation activation threshold.

Point (C) is found by checking each Desired Velocity value in the Situation memory at each sample interval, by working backwards in time through the samples starting at Point (B). This process is ceased when the Desired Velocity values become greater than the Situation activation threshold, and the position of the desired trajectory plot at this sample interval is labelled as Point (C).

Once Point (C) is found, the time difference between Point (C) and the end of the Situation at Point (B) is calculated. This time difference is called the time-shift factor.

6.2.2.7 *EMG Shift Rate Value*

The EMG Shift Rate Value determines how fast to shift the EMG forwards or backwards in time. As the purpose of the EMGC is to minimise the error associated

with a situation by equalising the first and second error regions, the EMG must move to the optimal position in time until the error regions are equalised and the Error Ratio is zero. The speed at which the EMG moves to find this position is adjusted by the EMG Shift Rate value.

A low rate will produce a slower EMG movement, producing a slowly yet stable convergence of the error regions until an Error Ratio of zero is produced. Too great a shift rate and the EMG will jump around in time, never converging the Error Ratio to zero. The shift rate value n should be between 0 to 1.

It is important to note that the CMAC in the CMAC+PID controller best adjusts to small changes in the new desired trajectory training signal based on the superposition of the original desired trajectory together with the EMG. This way the output slowly changes, reducing the chance of dangerous and unstable driving forces at the output. Unstable changes at the output can also cause the error associated with the situation to fluctuate rapidly, increasing the chance of unstable Error Ratio convergence.

Figure 6.18 illustrates the effect the EMG shift rate has on the EMG.

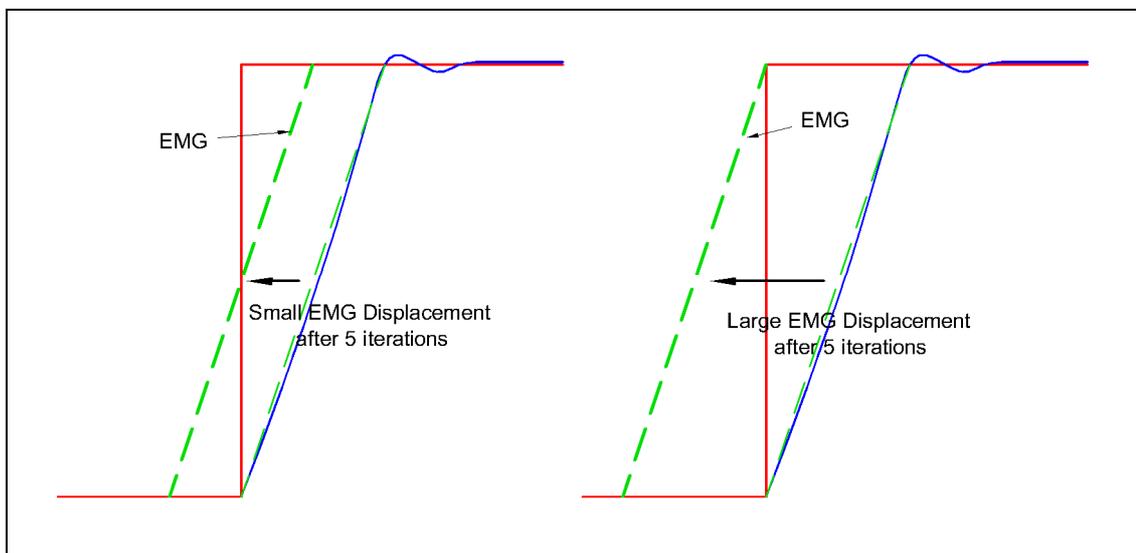


Figure 6.18 – (a) EMG Shift Rate Small (b) EMG Shift Rate Large

Figure 6.18 illustrates an example of how far an EMG has moved after 5 training iterations. With a small shift rate, the EMG has only moved backwards (earlier) in time by a slight amount compared to when a large shift rate is used. Clearly with a

larger shift rate, the EMG has overshoot the optimal EMG position, driving back the EMG as far as physically possible (any further and the later convergence point on the EMG does not converge with the desired trajectory).

6.2.2.8 New EMG Generation

The next stage in the EMG Processing Phase involves calculating the position of the EMG origin from the acquired knowledge. The time-shift factor T is multiplied by the EMG shift rate n which is then multiplied by the Error Ratio, and finally added to the old EMG origin position. This produces the new EMG origin, Point (D).

$$\text{NewEMGOrigin} = \text{EMGOriginOffset} + \text{OldEMGorigin} + \text{ErrorRatio} * (T * n) \quad \text{Eq. 6.3}$$

The slope of the EMG was determined earlier as a product of the maximum recorded velocity of the actuator associated with the situation and the EMG Offset Percentage. Together with the EMG Origin Offset and the new EMG origin product, the generation of the new EMG is illustrated in Figure 6.19.

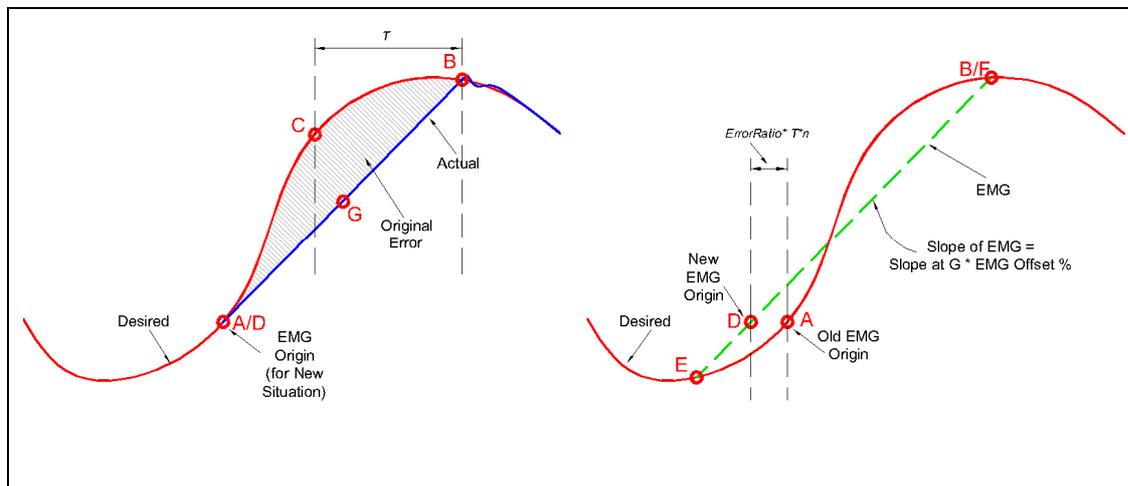


Figure 6.19 – Generation of the New EMG Origin at Point (D)

The new EMG origin is then stored in the EMGC CMAC at the same location where the previous EMG origin for the Situation was stored, overwriting the previous EMG origin value.

6.2.2.9 EMG Limits

The last step before training the EMGC is to find the limits of the EMG. Figure 6.19 shows Point (E) and Point (F) as the early and later EMG limits respectively. These points are simply the intersections between the EMG and the desired trajectory curve, and are required to define the EMGC training limits. A simple sequential search algorithm is used to find the intersections of the EMG with the desired trajectory, using knowledge about the gradient of the EMG and the velocity sign of the situation. At each time interval the position of the desired trajectory is compared to the position value of the EMG, until the minimum and maximum intersections are found.

Once the EMG limits are found, the final stage of the EMGC can begin: CMAC training.

6.2.3 EMGC CMAC TRAINING

Training the CMAC inside the EMGC is the final stage in the whole EMGC process. All information about the Situation has been obtained and processed together with the EMG gradient, origin and limits calculated. This all leads to training the EMGC CMAC with the error difference between the EMG and the desired trajectory curve to produce the error minimising compensation output.

For every sample interval between the EMG limits, the difference between the EMG position and the desired trajectory position is calculated. This value is then used as the desired training value used to train the CMAC. This is illustrated in Figure 6.20.

Figure 6.20 also shows the training data used in training the CMAC and where these input vectors originate from. The sample interval m is required to specify which training interval is being trained in the CMAC input space. Using a one second horizon means the EMGC takes a maximum of 200 samples for each situation (see Section 6.2.2.2). Only the sample intervals between the EMG limits will be used as input vector m , often utilising a smaller portion of the total 200 input vectors possible for this input dimension.

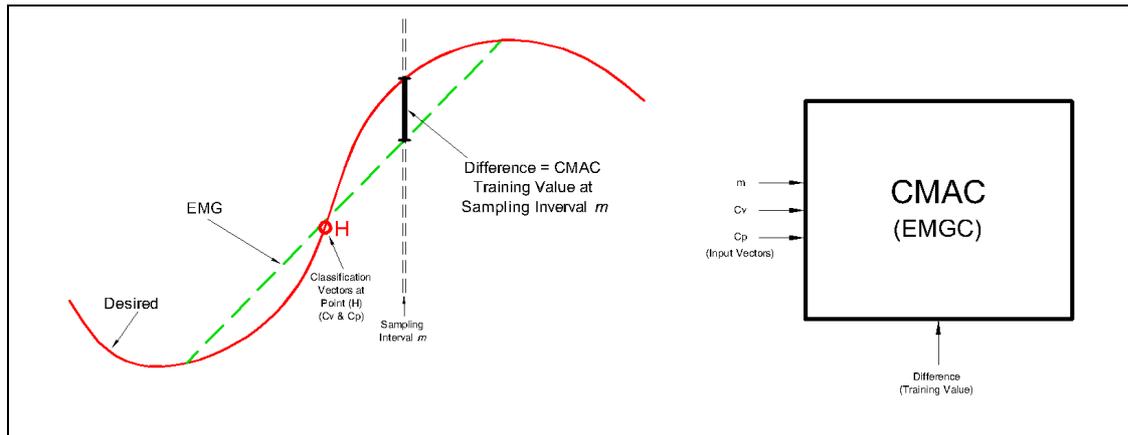


Figure 6.20 – EMGC CMAC Training Data

The situation classification vectors obtained from Point (H) are also used as input vectors to the CMAC shown as C_v and C_p in Figure 6.20. These classify the situation in a two-dimensional input space, allowing the Situation to be adequately classified without excessive cause for situation overlap (i.e. when two different situations are classified by the same classification vectors). Due to the simplistic nature of the classification method, and due to the fact that the number of situation possibilities outweighs the classification input space, there is still the possibility of situation overlap. However unless several situations with very similar classification properties occur, the chance of this occurring in this test system is slim.

EMGC CMAC training completes once the EMG upper limit sample interval is reached by the training process.

Referring back to the example used in Figure 6.14, what the CMAC is in fact learning is illustrated in Figure 6.21.

Figure 6.21 illustrates a two-dimensional input space, showing only two out of the three input vectors: m and C_v . The third dimension shows the difference value between the EMG and the desired trajectory positions which is taught to the CMAC. Thus this plot represents the output space for the CMAC for input vector dimensions m and C_v . Note that it is irrelevant whether input vector dimensions C_v or C_p were used for this representation for Figure 6.21 as both remain static throughout the duration of the particular situation.

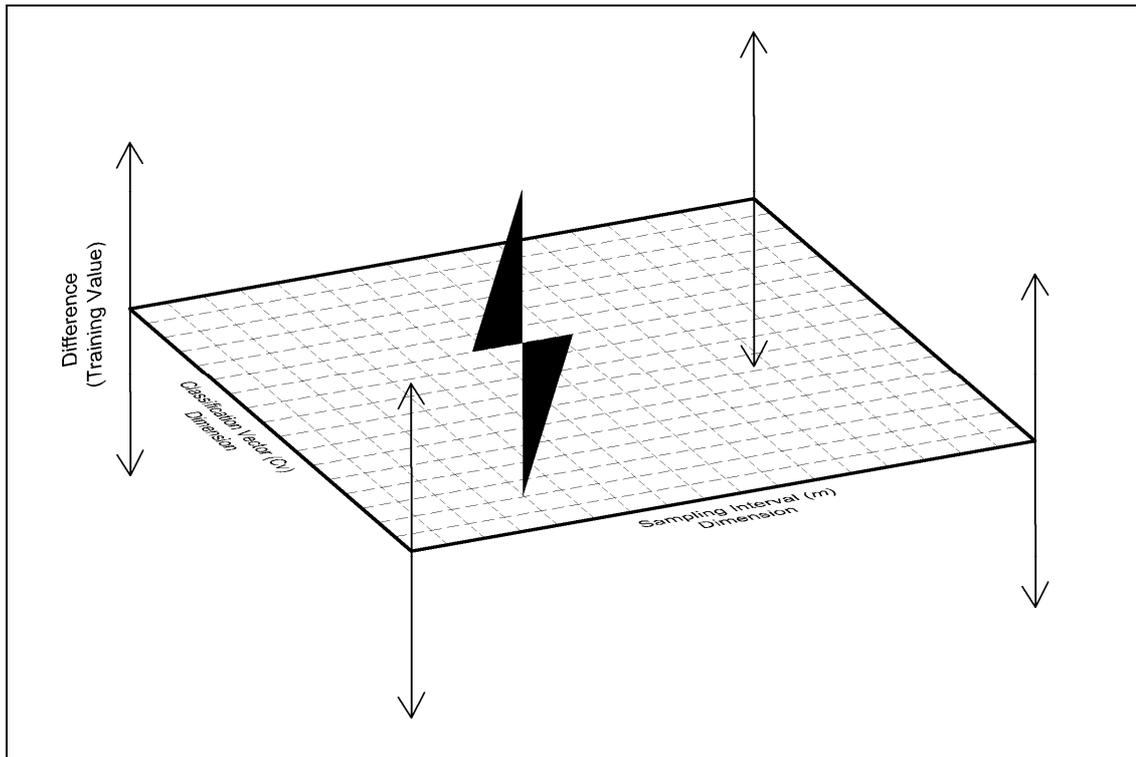


Figure 6.21 – EMGC CMAC Output Space for Input Vector Dimensions m & C_v .

Now that the EMGC process has been presented, the EMGC's EMG output effect on the subsequent CMAC+PID controller will be discussed.

6.3 EMGC INTEGRATION WITH CMAC+PID CONTROLLER

Once the EMGC CMAC has been trained for a particular situation, if that same situation is detected again, the EMGC will produce an EMG output to minimise the error in time associated with a situation. This output will be produced every time the EMGC detects the same situation again.

The EMGC output is a trajectory superposition to the desired trajectory. These two signals are summed together as illustrated in Figure 5.1, producing a new training target signal for the CMAC in the CMAC+PID controller to learn. This new training signal eventually moves the actual position trajectory backwards in time, until the Error Ratio between the first and second error regions equates to zero. This process is illustrated in Figure 6.22.

Figure 6.22 illustrates how both the EMG and the desired trajectory are summed together to produce a new target training trajectory signal for the CMAC.

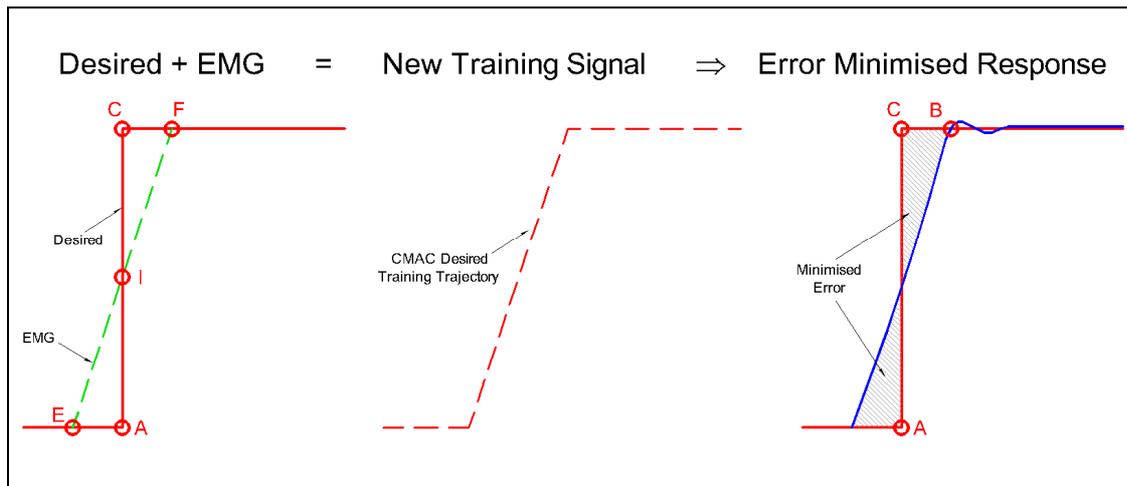


Figure 6.22 – EMGC Output Providing New Desired Training Signal to CMAC

The CMAC, after several situation iterations, tries to produce an output that matches as closely as possible the new desired training signal. Thus the final output from the CMAC+PID controller over time will be the error minimised response, with the actuator response moving in the situation's direction before the situation occurs, using the predictive effect of the EMGC.

It is important to note that the EMGC output signal is not summed together with the original input vectors leading to both the CMAC and PID controllers. The reason for this is that if the desired trajectory input vectors were summed together with the EMGC output, this would cause an immediate drastic change to the input vector path through the CMAC input vector space. New input vectors would be mapped to previously untrained weights, producing unstable effects at the CMAC output.

It is desirable that once the EMGC starts to produce an error minimising output for a particular situation, a gradual change at the CMAC output occurs. This way the system remains stable and slowly changes the actuator's input in the direction to minimise the situation's error. To do this, the EMGC output is used together with the desired trajectory signal to produce a new training signal for the CMAC. This way, all the input vectors used in the CMAC for a situation remain the same. No new weights are immediately mapped to by the CMAC input vectors and thus the output is only a result of the slowly changing weights already used to control that situation. Thus the input vectors for the situation do not change; only the values of the weights change mapped to by the same input vectors. Hence the output change is gradual,

allowing the system to slowly adjust to the new training signal, remaining stable. This process takes into account the new weights mapped to by the new Actual Position input vectors, though this process is as gradual as when learning any new trajectory function.

Before testing of the EMGC and CMAC+PID controller can be performed, the hardware/software system the controller will run on must be discussed. This topic is a critical part of the project, as the requirements needed to produce a system that is capable of handling many CMACs all running sequentially at a high sampling rate, with enough I/O and memory requirements, a GUI, etc, all in a real-time environment, limits the hardware/software choice immensely. These aspects of the project will now be presented in the following chapter.

7.0 TESTING SETUP

This chapter explains the hardware setup used to provide a stable hardware system to produce and test a functional software environment suitable to develop the controller concepts discussed in this thesis. This area was critical in allowing the CMAC code to execute in real-time, with enough processing power, enough memory and enough hardware I/O to incorporate future functionality to run a 25 degree of freedom hand while simultaneously running a highly graphical GUI.

7.1 ACTUATION TEST RIG

The actuator of choice to be used in the creation of a deafblind fingerspelling hand is the air-muscle. Previous humanoid hand actuation using air-muscles is discussed in Appendix A. The air-muscle test rig used for controller testing in this thesis is illustrated in Figure 7.1.

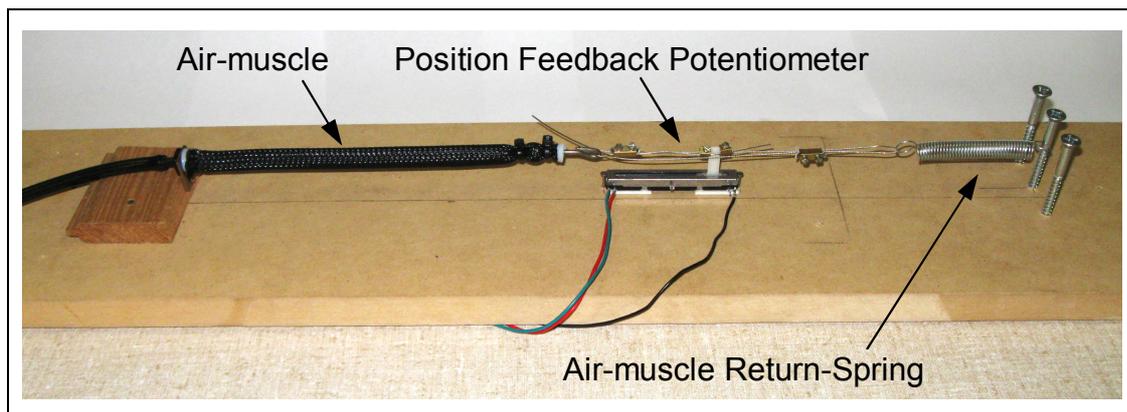


Figure 7.1 – Picture of Air-Muscle Rig used for Testing the EMGC

Figure 7.1 illustrates the air-muscle test rig used in the generation of all test results obtained in this thesis, corresponding to the tests outlined in Appendix F. The air-muscle test rig consists of a single air-muscle, with a single return-spring. (Spring constants for all tests are included in Appendix F.) A linear potentiometer is used to measure the position of the air-muscle.

The air-muscle is connected to the EMGC and CMAC+PID controller through custom made electronic and pneumatic hardware. The hardware used for the tests in this thesis is the same hardware used in Scarfe (2004), with several minor improvements made. Appendix D contains a description of these modifications.

7.2 REAL-TIME CONTROL OPTIONS AVAILABLE

Several options existed which would allow real-time control using the CMAC software to control the air-muscle hardware. However, finding a system that is totally integrated, producing 25 independent PWM outputs operating at deterministic microsecond intervals, while incorporating a highly customised graphing GUI, sitting on top of a real-time CMAC controller with access to large amounts of RAM and data logging storage mediums, while still remaining cost-effective, narrowed the choices down dramatically.

A system as described above can be easily implemented on a modern x86 PC (with a suitable digital I/O card) which these days are common, inexpensive and powerful. However the operating system necessary to obtain the required functionality from the hardware is crucially important to achieve real-time control.

The two dominant operating system choices for x86 hardware are Microsoft Windows[®] operating systems and Unix Based operating systems such as Linux. Either operating system alone, however, will not provide real-time control over the hardware.

Options exist which allow both MS Windows or a Linux distribution to execute custom developed software in real-time. These options are ‘operating system extensions’, which run alongside the main operating system, providing control over the hardware at a lower level than the main operating system. Tasks that need to be executed on time at microsecond intervals are given priority over all main operating system tasks. In effect, the main operating system runs in the operating system extension’s idle time. On powerful enough hardware, the operating system and its extension can run together seamlessly, allowing real-time functionality to be obtained from x86 hardware without noticeably effecting the main operating system’s performance.

For MS Windows, such operating system extensions include RTX (IntervalZero, 2008), and INtime (TenAsys, 2008). However such extension development packs are both commercial products. For Linux distributions, two main operating system

extensions also exist. They are RTAI (RTAI, 2008) and Xenomai (Xenomai, 2008). Both extensions are licensed under the GNU General Public License (GPL), making them freely available to integrate into a Linux distribution of choice. Together with most Linux distributions also being under the GNU GPL, a totally free, highly supported real-time operating system for x86 hardware can be implemented.

Either RTAI or Xenomai would produce the functionality required. Xenomai was chosen from the two, as it is widely supported. The Linux distribution chosen was Ubuntu because of the popularity and support currently available.

7.3 XENOMAI CAPABILITIES

Specifically, Xenomai with Linux allows the following:

- Real-time operating system functionality. Priority thread creation with high frequency timers, with worst case thread activation delay of 25usec.
- Powerful yet flexible PWM generation in software.
- Powerful x86 architecture to utilise as the hardware framework.
- Access to PC peripherals including fast and powerful multi-core processors, large memory allocation with both RAM and hard-disk, and a large variety of digital or analog I/O options, serial communication devices, etc.
- Separate prioritised real-time algorithms running alongside standard Linux Applications.
- Rapid communication between Linux and Xenomai applications by use of POSIX pipes.
- Powerful hardware accelerated graphical capabilities running alongside real-time code on the same platform, producing highly flexible GUI functionality for the real-time applications.

- GNU General Public Licensing for both Linux and Xenomai and a supportive online community.

The list of advantages of using Linux makes it ideal for developing an adaptive CMAC control system interfacing with external hardware in real-time, while providing a seamless integrated GUI front end to the system.

7.4 HARDWARE/SOFTWARE USED

The following development framework and hardware listed in Table 7.1 was used in the creation of the CMAC control system designed in this thesis, together with the GUI Graphing Applet created. In addition, the code for the controller and GUI is contained on the accompanying CD.

Processor and Chipset	2.8Ghz Pentium D (w/Intel 945P Express Chipset, FSB 1066Mhz)
RAM	4 x 512MB (667MHz) Dual Channel Configuration
I/O Interface	Decision 8255 48 Channel Digital I/O Board, PCI.
Operating System Base	Ubuntu 7.10 (Gutsy)
Operating System Kernel	Linux Kernel 2.6.22.14
Real-time Kernel	Xenomai 2.4.2
Programming Language	C/C++
Compiler	GCC/G++ 4.1.2
IDE	CodeBlocks (SVN 5020)
GUI Development	WxWidgets 2.8.7

Table 7.1 – Processing Hardware and Software Used for Running the Control Code

7.5 SOFTWARE FRAMEWORK DEVELOPED

Not only was a real-time control algorithm required to be implemented, but a graphical user interface (GUI) was also required. The GUI was required to run alongside the real-time control software, to provide the user with direct control over the hardware and real-time control system parameters, and to provide direct and smooth graphical feedback data from the pneumatic system. The GUI is interfaced to the hardware through the real-time control software. Together the two software

systems should seamlessly interact, passing data back and forth at deterministic intervals. Figure 7.2 illustrates the software framework developed.

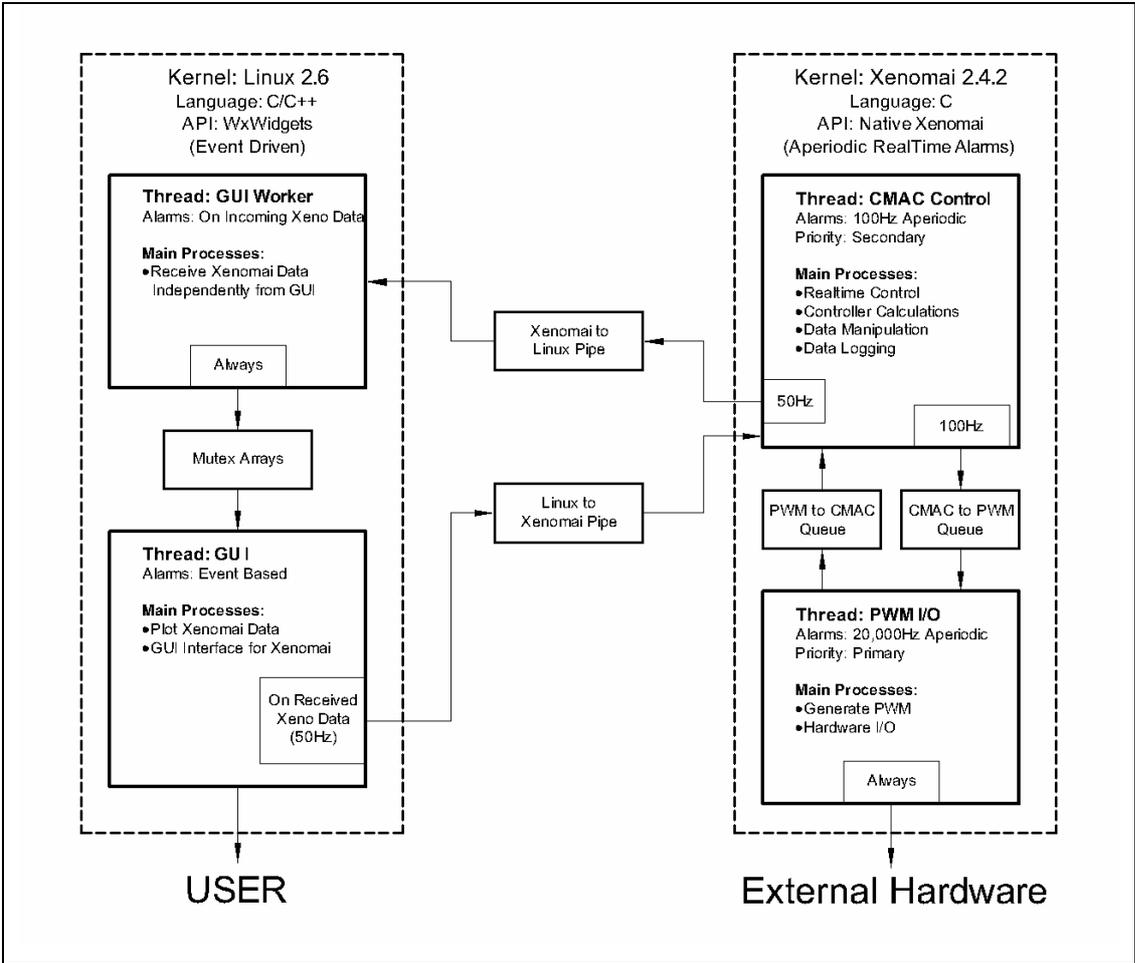


Figure 7.2 – Linux/Xenomai Software Framework

The software framework illustrated in Figure 7.2 will now be discussed starting with the Xenomai threads.

7.5.1 XENOMAI THREADS

Xenomai allows prioritised threads to be created, providing the programmer with control over which real-time thread should execute most urgently. Xenomai also allows threads to be executed based on either aperiodic or periodic modes. Aperiodic threads are based on the CPU clock time, and thus do not drift over time. Periodic threads on the other hand are based on the time interval since a previous thread has executed (Xenomai, 2008). Thus when using periodic threads, drifting is possible. To obtain accurate thread execution, aperiodic threads were used.

The most important thread created, given the highest execution primary priority, was the PWM thread. This thread is responsible for creating the PWM signals required to run the electronic hardware output board external to the 8255 Digital I/O board. This thread must execute as accurately as possible at a predetermined frequency to produce a stable PWM signal, with the PWM duty cycle directly corresponding to the required final DC output voltage. This thread is also responsible for controlling the ADCs used on the electronic hardware input board, and reading the ADC data back from the hardware. Instabilities here can result in missing and inaccurate feedback data from the air-muscle position potentiometer, resulting in incorrect training data to the CMAC algorithm. Thus it is crucial that this thread executes when required, ahead of any other threads also used.

The PWM thread was chosen to run at a fast yet stable frequency, chosen to be 20,000Hz. Refer to Appendix D for reasonings behind this chosen frequency. No unnecessary code is executed in this thread apart from the setting and reading of output bytes on the various ports of the 8255 Digital I/O card. This thread communicates to another Xenomai thread, the CMAC Control thread, using two Xenomai Data Queues, one from the PWM thread, the other to the PWM thread. Data to the PWM thread consists of the required PWM duty cycles. Data from the PWM thread includes the ADC 8bit feedback data.

Data transmitted between the PWM thread and the CMAC Control thread is sent and received through data queues operating at 100Hz, based on when the CMAC Control thread sends the data to the PWM thread. When data is received from the CMAC Control thread, data is also sent back to the CMAC Control thread in the same thread instance.

The CMAC Control thread executes at an aperiodic 100Hz, thus data to and from the CMAC Control thread to the PWM thread occurs at every thread instance. The CMAC Control thread is responsible for all of the real-time control algorithms, for communication to and from Linux and for all data logging functions. This thread is given secondary priority, and executes when required, only if the CPU is not required to execute the PWM thread. CPU task handling is automatically handled by Xenomai once thread priorities have been set. If the CMAC Control thread is currently

executing code and the PWM thread is required to execute, the CMAC Control thread will be suspended temporarily while the PWM thread is handled, and then resumed once the PWM thread has finished executing its code.

Communication from Xenomai to and from Linux is obtained via the use of Xenomai Pipes. Xenomai Pipes allow data to be exchanged from standard Linux applications to real-time Xenomai applications. Thus two pipes exist, one to Xenomai and one from Xenomai, allowing the exchange of data to be achieved between the Linux GUI application created. At every second CMAC Control thread instance, thus operating at 50Hz, data is sent to the Linux application. This data consists of the data from the field, as well as current controller parameters and variables.

7.5.2 LINUX THREADS

WxWidgets (wxWidgets, 2008) was used as the main programming library to create the GUI application in Linux. WxWidgets allows easy form creation to be achieved and written in C++. In addition, WxWidgets in Linux makes thread creation simple. By default all GUI functionality runs in a single thread when a WxWidget application is created. However additional threads can also be created, allowing independent code to be executed behind the GUI application. Thus if the GUI thread is waiting for user input or an event occurs, the external thread can still execute independently without waiting.

An additional thread named the GUI Worker thread was created specifically to wait for data to be sent from Xenomai. This thread waits until it receives new data from the Xenomai CMAC Control thread. When new data is received, it sends the data from Xenomai to the main GUI thread via the use of a mutex array. A mutex is a safe method of locking memory to a single thread, so that simultaneously running threads cannot access the data at the same time, which could possibly cause the application to crash.

When the main GUI thread receives data from the GUI Worker thread, the data is updated to the screen. In addition, any data waiting to be sent to Xenomai is also sent. Thus data is sent to Xenomai when it receives new incoming data from Xenomai (at 50Hz).

7.6 DATA LOGGING

The Xenomai CMAC Control thread is responsible for logging all data that is generated by the EMGC and CMAC+PID controller. Data log files are written as ‘space separated value’ text files, suitable for future manipulation.

Two text files are created. “DataHistory_x_y.txt” and “ErrorHistory_x_y.txt”, where x and y are the date and time the file was created respectively, in YYMMDD_HHMMSS format. The data contained in each file contains the following, based on their column number.

For “DataHistory_x_y.txt”, the data is logged every other sampling period and includes:

Column Number	Data Description
1	CMAC Training Time (in sampling periods)
2	Desired Position
3	Actual Position
4	Combined CMAC outputs
5	PID Controller output
6	Not Used
7	Not Used
8	CMAC output
9	CMAC + PID combined output
10	Iteration number of the gesture

Table 7.2 – DataHistory_x_y.txt File Data Description

For “ErrorHistory_x_y.txt”, the data is logged once per ‘situation’, so long as the situation training requirements are satisfied, and includes:

Column Number	Data Description
1	EMGC On/Off Status
2	Situation Classify Vector 1
3	Situation Classify Vector 2
4	Controller time (in sampling periods of when the situation occurs)

5	EMG slope
6	Summation of Error 1
7	Summation of Error 2
8	Total Error
9	Error Ratio
10	EMG slope offset percentage
11	EMG offset Origin Positive
12	EMG offset Origin Negative
13	EMG Shift Rate Value

Table 7.3 – ErrorHistory_x_y.txt File Data Description

Data log files are automatically saved together with the CMAC memory data. If a CMAC is restored from file, then the data log files are automatically restored too, with new data appended to the end of the file.

7.7 GUI INTERFACE

The GUI interface was built in WxWidgets, to provide a front end of the complete system. The user can directly see what the physical system and the controller are doing graphically and numerically, with access to commonly used variables updating within the real-time controller.

The GUI developed to interface to the Xenomai CMAC control code is illustrated in Figure 7.3.

The developed GUI is made up of five main areas on screen. The purpose and functionality of each will now be explained.

7.7.1 CONTROL BUTTONS

At the top of the screen are the Control Buttons, each carrying out a specific real-time online task. The top of this section displays the status of the Xenomai application itself and the status of the communication pipes between Xenomai and Linux. An explanation of the individual buttons will now be presented.

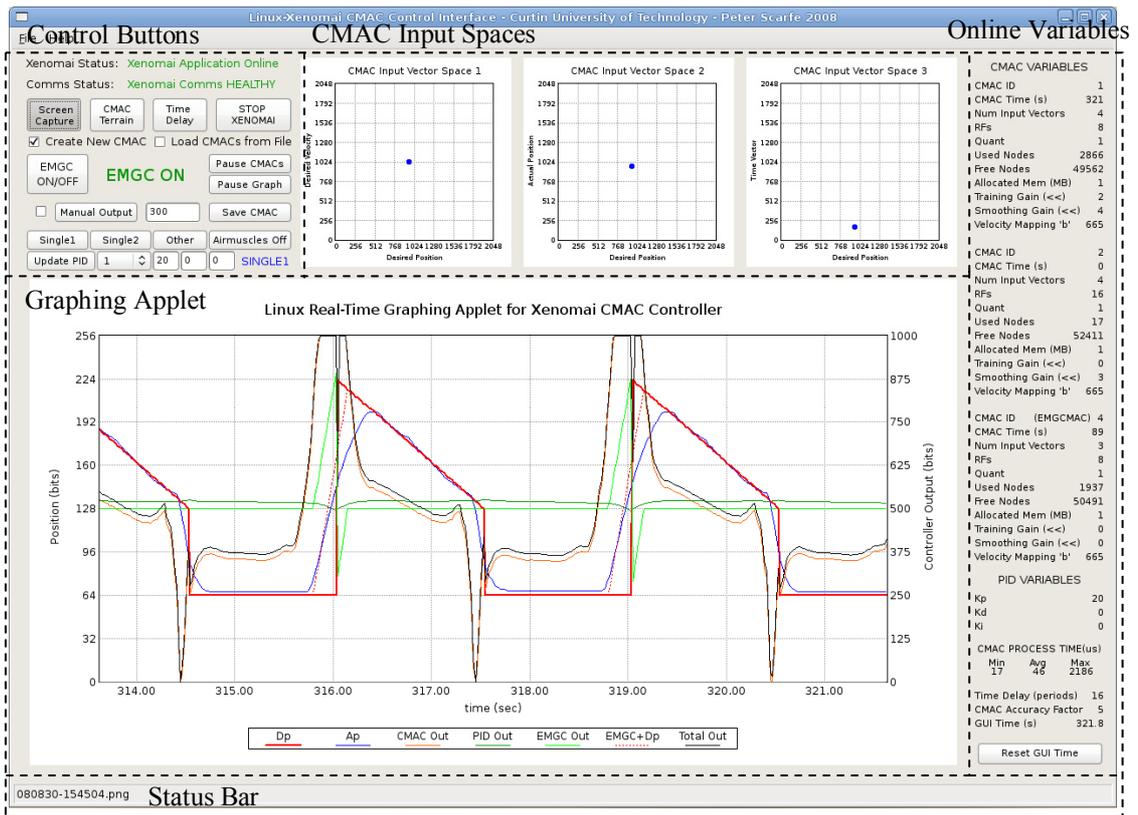


Figure 7.3 – Linux Graphical User Interface Created with WxWidgets for Xenomai CMAC Controller

The screen capture button will take a full screen shot of the GUI (like that illustrated in Figure 7.3) and save it to a ‘png’ file, with the file name stamped in the Status Bar at the bottom of the GUI. It will also take a cropped screen capture of the Graphing Applet and save it to a separate ‘png’ file.

The CMAC terrain button will (if using a two-dimensional CMAC) create a text file, producing a two-dimensional terrain map of the CMAC weight memory. The text file can then be manipulated with Matlab to produce a 3-D representation of the CMAC memory, such as that in Figure 5.5 to Figure 5.7.

The Time Delay button will execute a separate piece of code. This code produces a large step function for the system, and calculates the time the system takes to respond. A series of step functions are produced in both positive and negative directions and printed to the Xenomai console on screen. The time delay of the system is required to match the CMAC weights to the correct error associated with them, as described in Section 5.2.3.

The STOP Xenomai button causes a clean termination of the Xenomai application, deleting any temporary CMAC memory and data log files created.

Below the top row of buttons are two checkboxes, allowing the choice of either a new CMAC memory and log files to be generated, or for previously saved CMACs to be recalled from file in the appropriate date-time stamped folder. The specification of the folder must be manually entered in the Xenomai code.

The EMGC ON/OFF button turns the EMGC either on or off, with the status of the EMGC boldly displayed next to it.

Two buttons are provided for independently pausing the animated CMAC Input Space plots or the Graphing Applet plot. These buttons only freeze the data being updated to the screen and do not affect the performance of the controller or the updating of any other variables to the screen in any way.

The next line of buttons allows the overriding of the controller output to be attained. The checkbox must be selected for the overriding to be active. When checked, the numerical integer entered into the textbox, between 0 to 1000 bits is directly outputted to the air-muscle valve as a corresponding DC voltage.

The Save CMAC button on this line saves the current CMAC weight memory, the EMGC weight memory and the data log files created for the life of the CMAC, to individual files. A screen capture is also taken of the GUI, with the location and date-time stamp written to the Status Bar for easy future system state reference.

The next row of buttons select which test rig to run the controller algorithm on, or send the output override signals to. These buttons act as a switch for the currently used output and input signals, deactivating any high outputs to any unused output lines. The Air-muscles Off button sends a low signal to all output lines. The status of the particular rig being used is displayed on the line below.

The bottom row of buttons allows the user to modify the PID gains used in the PID section of the controller algorithm. The chosen values are sent to Xenomai when the Update PID button is clicked.

7.7.2 CMAC INPUT SPACES

The CMAC input spaces provide the user with a direct representation of the input vectors used in the CMAC input space (for the CMAC+PID CMAC). A total of four input vector variables are plotted on three discrete CMAC input space maps, each representing a two-dimensional representation of two input vector variables.

CMAC Input Vector Space 1 plots the Desired Velocity input vector value against the Desired Position input vector value in CMAC input vector space coordinates. CMAC Input Space 2 plots the Actual Position input vector value against the Desired Position input vector value. CMAC Input Space 3 plots the Time Vector input space vector value against the Desired Position input vector value.

The scale on each axis ranges from 0 to 2048, the maximum range that a CMAC input vector parameter is limited to. The representation of these input vectors in this way allows the user to see directly where the current input vector is mapping to within the CMAC's multidimensional input space.

7.7.3 ONLINE VARIABLES

On the right side of the GUI there are several variables which are all updated in real-time continuously.

The first set of variables are CMAC variables, showing either fixed parameters about a particular CMAC or the real-time updating of changing CMAC variables. There are three sets of repeated CMAC variables displayed, each corresponding to a different CMAC. The variables displayed include:

CMAC ID: This is the CMAC identification number, assigned to a CMAC by the Xenomai Code. Each CMAC being used within the Xenomai code at any time has a unique ID number, allowing for multiple CMACs to be used online simultaneously.

CMAC Time (s): The CMAC time in seconds. This increments every time the CMAC is trained, at the sampling frequency of the control code (set here to 100Hz). Thus every time the CMAC is trained, the CMAC Time is incremented by 10msec. If a CMAC is not being trained, the CMAC Time ceases to increment.

Num Input Vectors: The number of input vector parameters the CMAC is using.

RFs: The number of receptive fields used in the CMAC.

Quant: The quantisation level used in the CMAC (see Section 5.2.2).

Used Nodes: The total number of Binary Search Tree nodes that have been assigned weights in the CMAC weight table.

Free Nodes: The number of free nodes currently available to be used in the CMAC's weight table until automatic online reallocation of new nodes occurs.

Allocated Memory (MB): The number of megabytes used by the CMAC weight table. This value is the combination of memory used by the CMAC's Used Nodes and the CMAC's Free Nodes.

Training Gain (<<): The training gain of the CMAC, represented as the inverse of a left binary shift value. (I.e. A value here of 0 means a training gain of 1. A value of 1 means a training gain of 0.5, etc.)

Smoothing Gain (<<): Same as for the Training Gain, however this is the weight smoothing gain value of the CMAC.

Velocity Mapping 'b': Defines the shape of the velocity mapping curve for the CMAC Desired Velocity input vector, as outlined in Appendix C.

The second set of variables are the PID gains for the PID algorithm in the CMAC+PID controller.

The third set of variables show the CMAC Process Time in microseconds. This represents the time taken by the Xenomai code to execute a single occurrence of the complete control algorithm. The minimum, average and maximum times taken for the control algorithms to occur once (within the 10msec sampling period) are displayed here to provide some indication of how much processing time is required by the control code, for diagnostic purposes.

The last set of variables includes the Time Delay (in sampling periods) used by the control code to overcome the time delays taken from when the controller produces an output until the system physically responds. This allows the correct training of CMAC weights to occur, corresponding to the input vectors that produced the associated output.

The CMAC Accuracy Factor is a multiplier that provides the CMAC error margin to increase beyond its natural value. This is used to increase the CMAC output resolution.

The GUI Time, in seconds, represents the active time of the GUI. This value can be reset by use of the 'Reset GUI Time' button below.

7.7.4 GRAPHING APPLET

The Graphing Applet provides a real-time scrolling graph to the user, to show the progression of changing input and output variables, together with changing controller parameters.

The graphing applet can show a maximum of seven different plots simultaneously. Two separate scales are included for these parameters, with Position in bits ranging from 0 to 256 bits indicated on the left of the graph, and Controller Output in bits ranging from 0 to 1000 bits on the right side of the graph. A scrolling incremental time scale is included on the bottom of the scrolling graph, in seconds. The different plots indicated on the graph include:

Dp (Desired Position): The current desired position of the actuator in Position Bits.

Ap (Actual Position): The current actual position of the actuator in Position Bits.

CMAC Out (CMAC Output): The current output of the CMAC in the CMAC+PID controller, in Controller Output Bits. (Offset by +500 bits. i.e. 500 on the graph represents a CMAC output of 0.)

PID Out (PID Output): The current output of the PID algorithm in the CMAC+PID controller, in Controller Output Bits. (Offset by +500 bits. i.e. 500 on the graph represents a PID output of 0.)

EMGC Out (EMGC Output): The current output of the EMGC in Position Bits. (Offset by +128 bits. i.e. 128 on the graph represents an EMGC output of 0.)

EMGC+Dp (EMGC Output + Desired Position): The summation of the Desired Position and the EMGC output, representing the new training trajectory the CMAC in the CMAC+PID controller must train to. Represented in Position Bits.

Total Out: The total summation of the CMAC and PID controller algorithms, in Controller Output Bits. This is the output value that is sent to the actuator.

7.7.5 STATUS BAR

A status bar is included at the very base of the GUI screen, to provide screen capture file names to the screen captures for easy tagging purposes.

This finalises the hardware/software system description designed specifically to test and design the adaptive CMAC controller to control the air-muscle hardware. While future work will require several air-muscles to be controlled simultaneously, the work presented in the following chapters looks at a specific implementation of the CMAC with custom adaptations for a single air-muscle. As the future plan is to operate several independently operated air-muscles for a robotic hand, the duplication of algorithms are all that is required to operate a fully functional robotic hand, as each degree of freedom is operated by an independent CMAC controller running on the same hardware.

With the EMGC, CMAC+PID controller and the physical test hardware system presented, EMGC performance testing remains. The following two chapters present and discuss a series of tests to see if the addition of the EMGC decreases the overall error associated with a situation and by how much this error is reduced. In addition, the physical effect the EMGC has on waveform trajectories is explicitly shown, confirming the feasibility of the controller.

8.0 EMGC INITIAL SQUARE WAVE TESTS

This chapter deals with testing the EMGC to demonstrate and understand how it performs on a single air-muscle spring return actuator, and to understand the effect different EMG gradients have on a situation's minimised error. This section also deals with testing the EMGC over a variety of different Total Error magnitudes based on modifying the desired input trajectory and changing the air-muscle's return-spring constant.

It is important to point out here that the testing of the EMGC with CMAC+PID controller is being performed on a single air-muscle actuator with a single degree of freedom. Designed as an error minimising controller for performing deafblind sign language on a fingerspelling hand, each degree of freedom would be actuated individually, and thus for this reason, only a single degree of freedom is being used throughout the EMGC tests.

Square wave step functions of varying magnitudes will be used to test the EMGC in this chapter. In terms of fingerspelling gestures, a square wave stimulates stationary finger joint gestures, such as the letter 'A' in the deafblind manual (see Figure 2.1). Square wave functions also produce a strong and clear situation, and provide a clear comparison between the sharp desired trajectory and the smooth actual system response, together with being able to vary situation magnitude while holding all other parameters constant. Non-square wave functions will be tested in Chapter 9.0.

8.1 EMG OFFSET PERCENTAGE DETERMINATION

Critical to the performance of the EMGC is determining a suitable EMG to use to minimise the error over time. The slope of the EMG is important as too low a gradient will cause the overall error for a situation to increase rather than decrease. While the error on both sides of the situation will eventually be balanced due to EMGC Error Ratio convergence, the Total Error for the situation will grow if the EMG is not suitably steep. The first test here involves determining a suitable EMG offset percentage to add to the maximum actual velocity associated with a situation.

This test involved creating an obvious situation using a square wave input trajectory signal, of suitable magnitude so that the air-muscle response would generate a suitably large initial Error Ratio in either positive or negative motion directions. A variety of EMG offset percentages were then used throughout a series of tests to obtain a comparison of the effect different EMG offset percentages had on the Error Ratio and Total Error for a situation.

Figure 8.1 illustrates the training function for the EMG Offset Percentage Determination (OPD) Test, and the response of the system with the EMGC turned off.

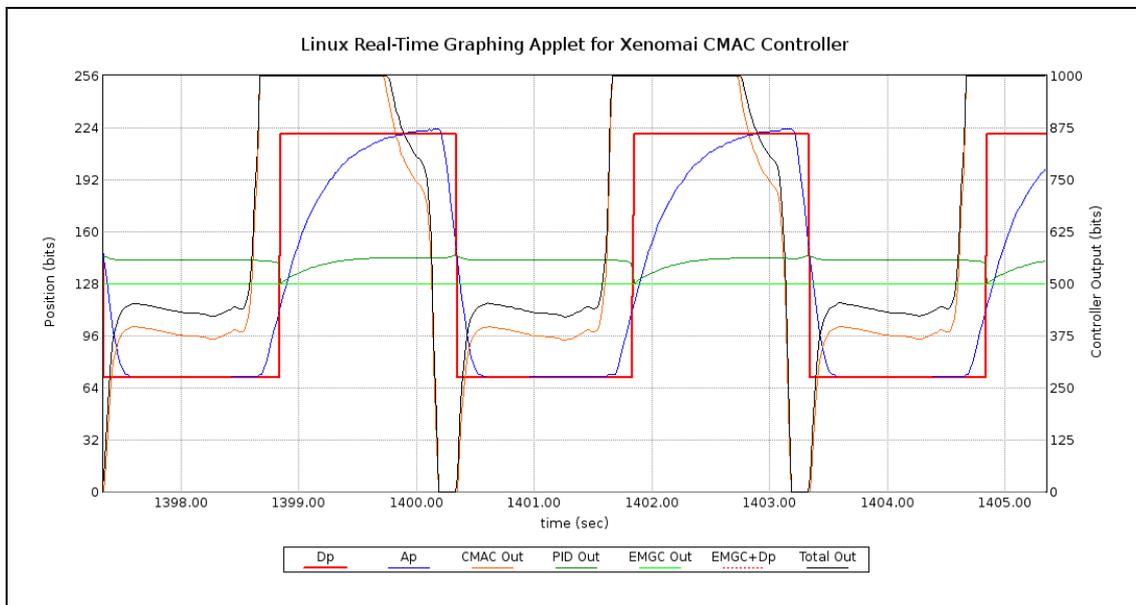


Figure 8.1 – EMG OPD Test. Graphing Applet Response. EMGC OFF.

Figure 8.1 illustrates the desired and actual response of the system, along with several other traces outlined in the key below the graphing area. A desired square wave trajectory of 50% duty cycle over a three second period was used to create substantially large situations, indicated by the large lag time taken for the actual response of the system to finally reach the target position for the positive direction of motion. As the positive motion direction creates a far greater Error Ratio in this test (due to the large Total Errors associated with positive situations for the test rig used), it will be used in determining which EMG offset percentage is most useful in minimising the situations errors. The air-muscle rig used a single air-muscle with a spring return, of spring constant $k = 608N/m$.

Using a range of EMG offset percentages from 0 to 100% of the maximum actual velocity obtained from the situation occurrence, comparisons of the Error Ratio, the Total Error, and the maximum actual velocities, obtained after the EMG was applied, will now be presented.

Figure 8.2 illustrates the error associated with the positive situation in Figure 8.1 for different EMG offset percentages. All errors in Figure 8.2 and throughout the remaining tests are represented as a summation of the magnitude error between desired and actual trajectories, for each time sampling period. The best performance illustrated in Figure 8.2, determined by the consistent minimal Total Error, is obtained by using an EMG offset percentage of 50%. The least useful EMG offset percentage is that of 0%, where the EMG is equal to the maximum actual velocity obtained by the actuator for the situation.

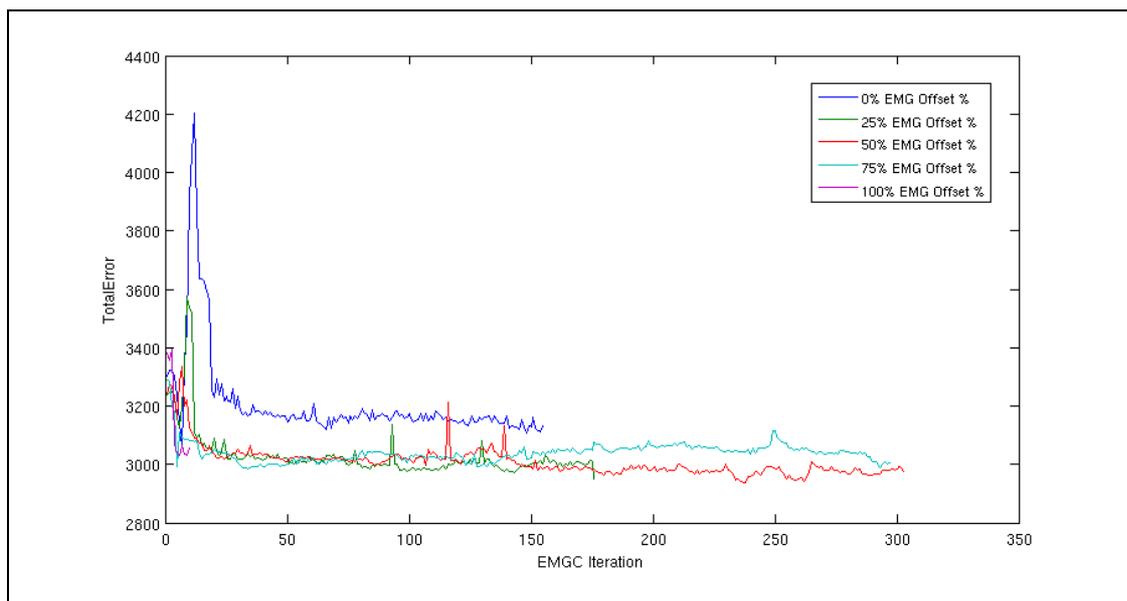


Figure 8.2 – EMG OPD Test. Total Error for EMG Percentage Offset Comparison Test.

The next figure, Figure 8.3, illustrates the Error Ratio for the range of EMG offset percentages for the EMG OPD Test. The EMGC converges all Error Ratios to zero for each of the EMG offset percentages within the first 50 EMGC training iterations. Again the worst performance here is that of the 0% EMG offset percentage, producing a severely negative Error Ratio before it converges to zero. The pattern seen here is that greater EMG offset percentages converge faster, however note that at 100%, only very few training iterations exist.

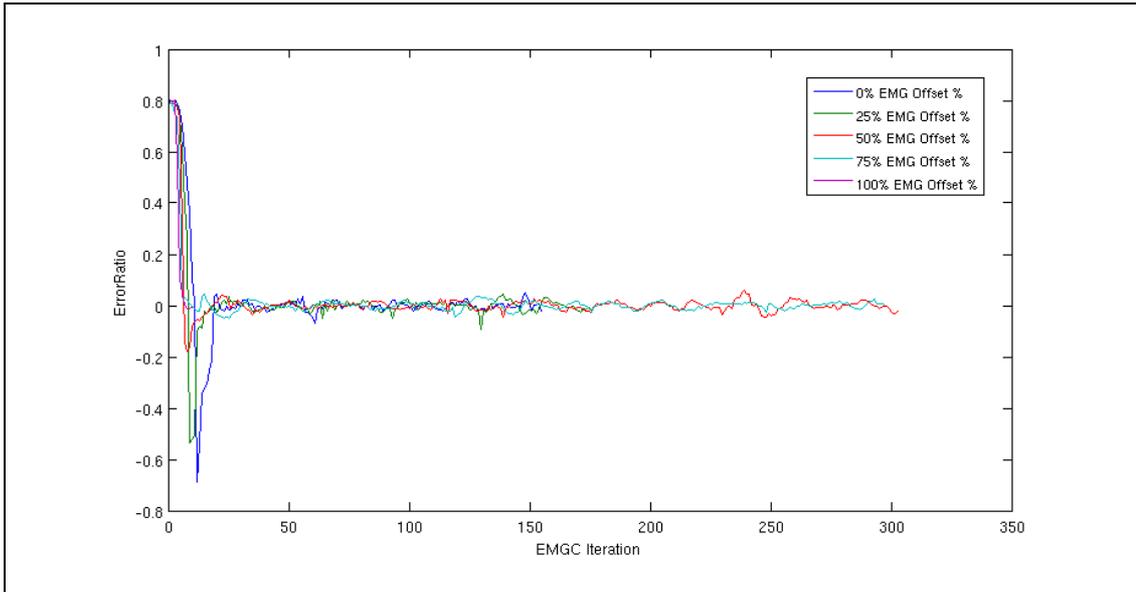


Figure 8.3 – EMG OPD Test. Error Ratio for EMG Percentage Offset Comparison Test.

Figure 8.4 illustrates the maximum actual velocity obtained throughout the EMG training process. Again, using an EMG offset percentage of 0% produces the slowest response. Initially the maximum actual velocity of the system starts at around 270 to 280 bits per second. To minimise the situation error optimally, the maximum actual velocity after the EMGC converges should be equal to the maximum actual velocity before EMGC training occurs. The data here relates closely to the Total Error plot in Figure 8.2. Larger maximum velocities produce smaller final errors as expected, and consequently, an EMG of 50% and 75% produce the minimal Total Error.

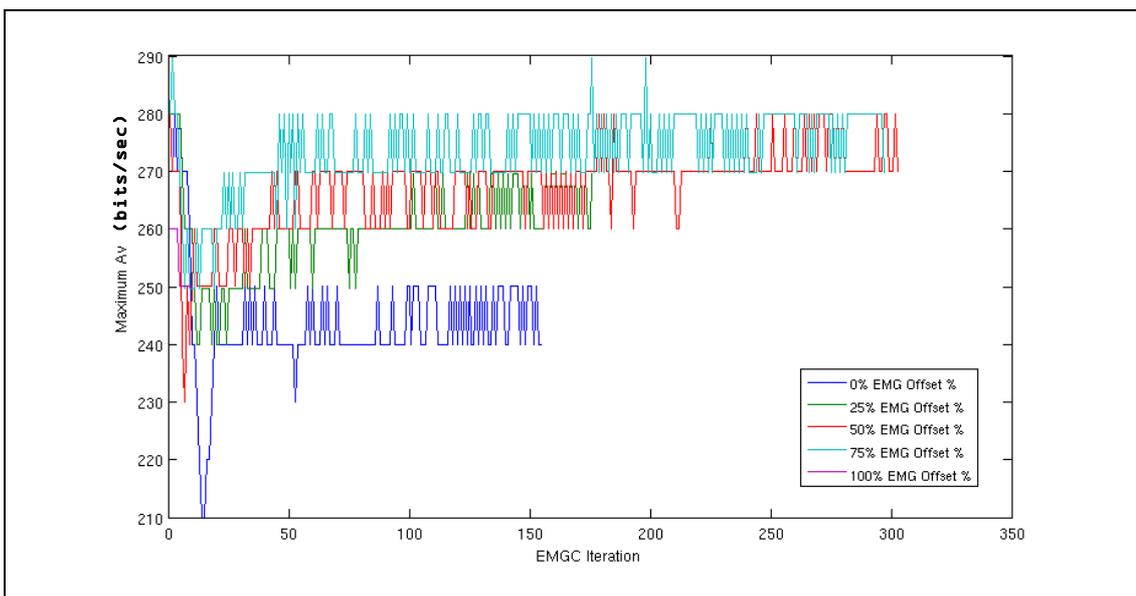


Figure 8.4 – EMG OPD Test. Maximum Av for EMG Percentage Offset Comparison Test.

The test results above show that the EMG offset percentage of 100% produces very few iterations, unable to produce a suitable convergence profile compared to the other EMG offset percentages. This is clearly illustrated in the EMG comparison plot shown in Figure 8.5. An EMG offset percentage of 50% comes out slightly in front of the rest, undergoing the most iterations. The larger number of iterations is also attributed to the convergence time of the EMGC for the situation, producing a stable converged EMG requiring less training of the CMAC+PID controller due to the lower number of new Actual Position input vector states created.

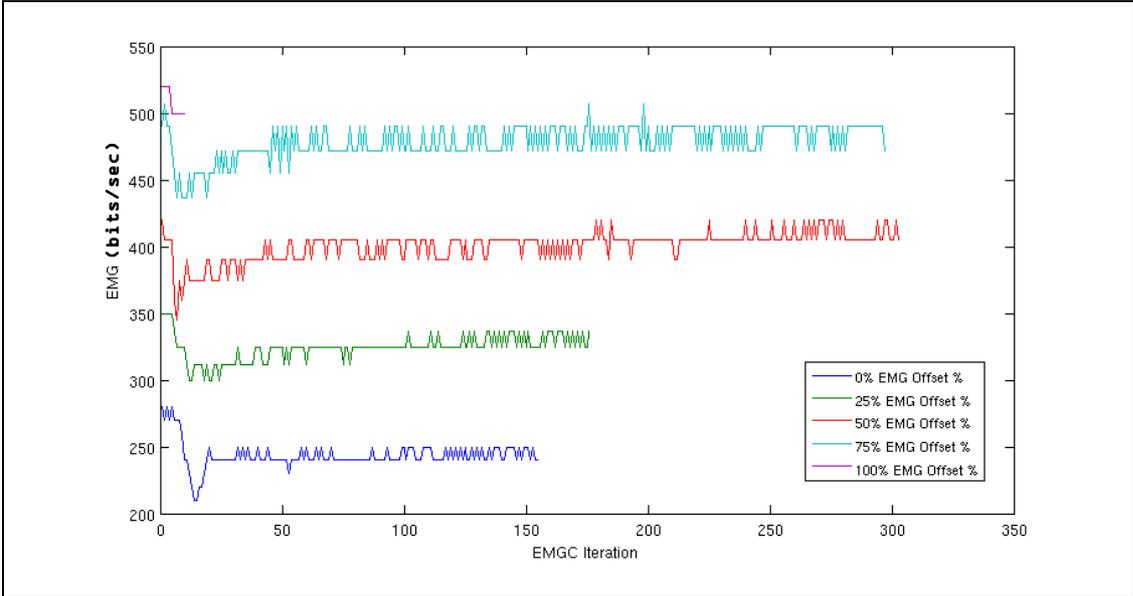


Figure 8.5 – EMG OPD Test. EMGs for EMG Percentage Offset Comparison Test.

The reason for the very low number of EMGC iterations for the EMG offset percentage of 100% can be seen in the following figure, Figure 8.6.

In Figure 8.6, the EMG addition to the Desired Position plot (EMGC+Dp) shows that the EMG has moved back in time as far as the controller allows. If moved back any further in time, the EMG will not intersect the desired position trajectory. Thus, this is the limitation of the EMG time shift factor. The reason for the low number of iterations for this EMG offset percentage is that the EMG time shift factor wants to continue to shift back further in time to converge the Error Ratio to zero. If however this limit is reached and the Error Ratio has not dropped to or below zero, then each successive iteration will produce an EMG with a time shift factor that will cause the

EMG to not intersect the desired position trajectory. Thus a suitable EMG cannot be created and the EMGC will not be further trained.

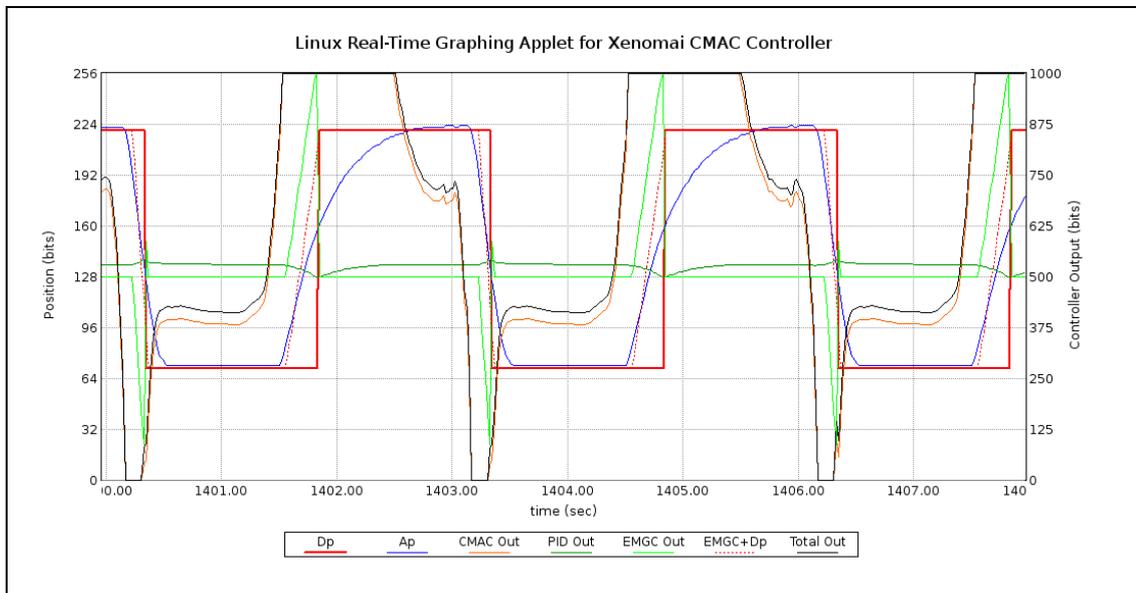


Figure 8.6 – EMG OPD Test. EMG Offset Percentage at 100%.

This interesting occurrence illustrates the reason why the EMG offset percentage cannot be too large. Together with the previous results, the EMG offset percentage cannot be too small. Thus the best EMG offset percentage to use, indicated from the results illustrated, is an EMG offset percentage of approximately 50%, producing the lowest converged Total Error in Figure 8.2. While there is clearly a range of EMG offset percentages that could be used (40%, 60%, etc) the results above indicate that approximately 50% performs well. This EMG offset percentage will thus be used as the default value throughout the remainder of the following tests.

8.2 EMGC SITUATION ERROR PERFORMANCE

The performance of the EMGC will now be tested for a variety of situation sizes the system may encounter. The tests will be conducted using a series of desired step wave functions of differing magnitudes to represent a variety of situation sizes. A single air-muscle with a spring return will be used as the test actuator, using two different return-spring constants to produce actuators of differing dynamic responses.

The purpose of these tests is to create a representative range of situation errors of different sizes, which could realistically be encountered by an air-muscle driven

system, to see how the EMGC performs over a wide variety of encounterable situations. Note that while the system is only presented with square wave functions in these tests, further tests presented in Chapter 9.0 demonstrate how other gradient functions approaching and departing a situation fare with use of the EMGC.

Each test undertaken was conducted for approximately 1300 seconds producing a stably converged error response, obtaining approximately 400 training iterations of each periodic function. Each test was first conducted on a standalone CMAC+PID controller for benchmarking purposes, followed by a comparison test using the EMGC in addition to the CMAC+PID controller. An example input training trajectory used in these tests is illustrated in Figure 8.7.

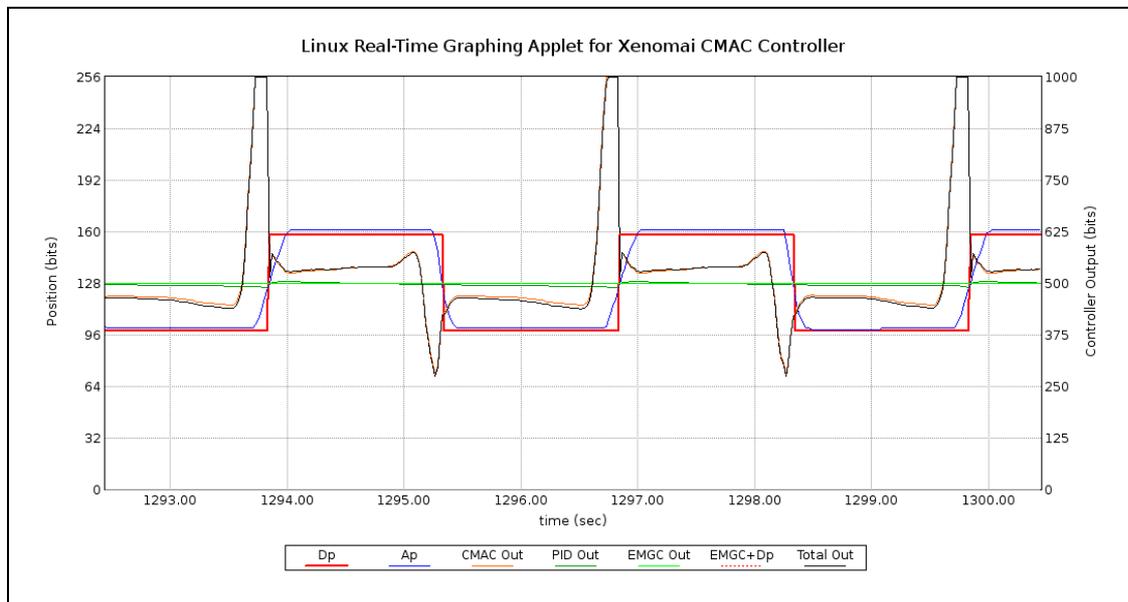


Figure 8.7 – Test Training Function. Three Second Period, 60 Bit Magnitude, Square Wave.

Eight tests were conducted, using four different training trajectories, identical to that in Figure 8.7, but with various magnitudes from 60 bits to 170 bits, resulting in 12 usable situation error log files. For test number descriptions used throughout the following tests, please refer to the test log descriptions contained in Appendix F.

For each range of input function step sizes, two separate air-muscle return-spring constants were used to produce actuators with different response dynamics. The first return-spring had a spring constant of $k=608N/m$. The second return-spring had a spring constant of $k=912N/m$. Each spring was used separately with the air-muscle.

The following sections will refer to the tests as either the ‘first’ or ‘second’ spring tests to differentiate between the actuator setups used in the tests.

The test results for each spring constant will be presented separately followed by a comparison of the test results.

8.2.1 SQUARE WAVE SITUATION TESTING (FIRST SPRING: $k=608N/m$)

The first series of square wave tests was conducted on an air-muscle rig consisting of a single air-muscle with a return-spring constant of $k=608N/m$. A range of different step sizes were used, ranging from 60 bits to 170 bits, with each test conducted both with the EMGC On and Off.

The representation between Total Error, Error Ratio and Training Iteration for the first return-spring test is illustrated in Figure 8.8. Note that the tests referred to in Figure 8.8 and subsequent figures refer to the test data used and obtained from the tests carried out in Appendix F.

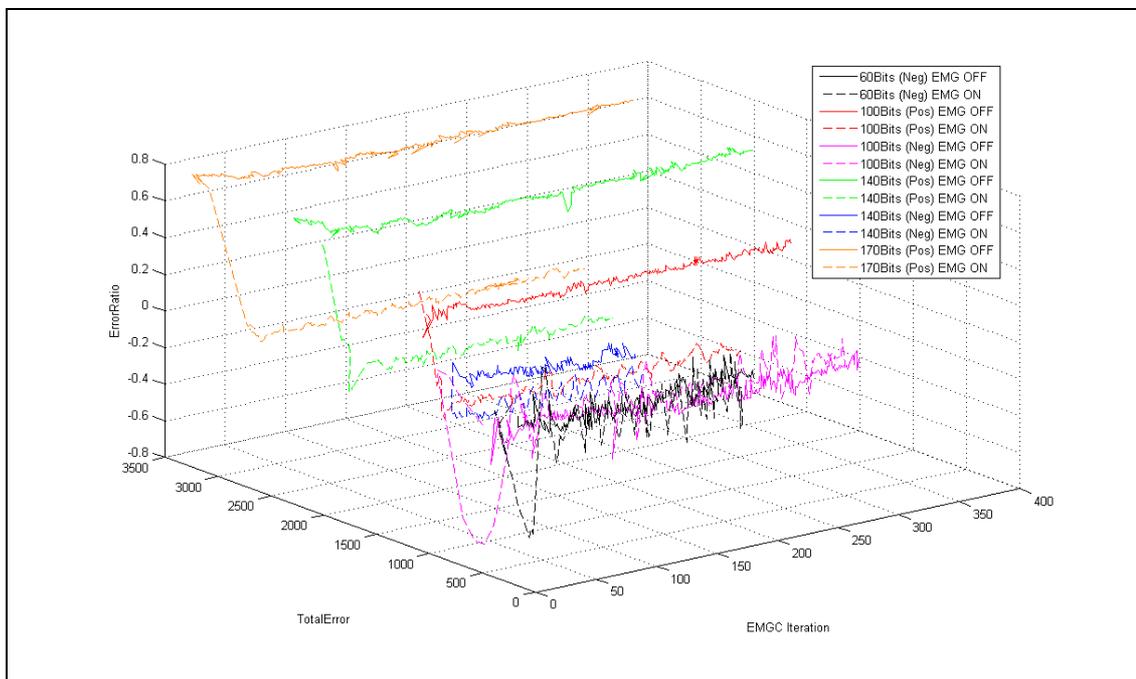


Figure 8.8 – First Spring: Error Ratio vs. Total Error vs. Iteration 3-D (Test 08 to Test 15)

Figure 8.8 illustrates the relationship between the Error Ratio, the Total Error and the number of successful situation iterations completed over the time span of the tests.

Note that each trace is identified by the information relevant to the situation it is representing in the legend. The step size in position bits is the value initially given. The positive or negative direction of the situation (rising or falling edge of the square wave respectively) is then given, followed by whether the EMGC was used or not, providing an EMG overlay to the training function. For example, ‘60bits (Neg) EMG OFF’ means a square wave step magnitude of 60 bits was used, with the data recorded for the negative associated situation (falling edge) with the EMGC not being used.

For clarity of representation, the pairwise relationships between the variables from Figure 8.8 will now be presented, starting with the Total Error vs. the EMGC iteration in Figure 8.9.

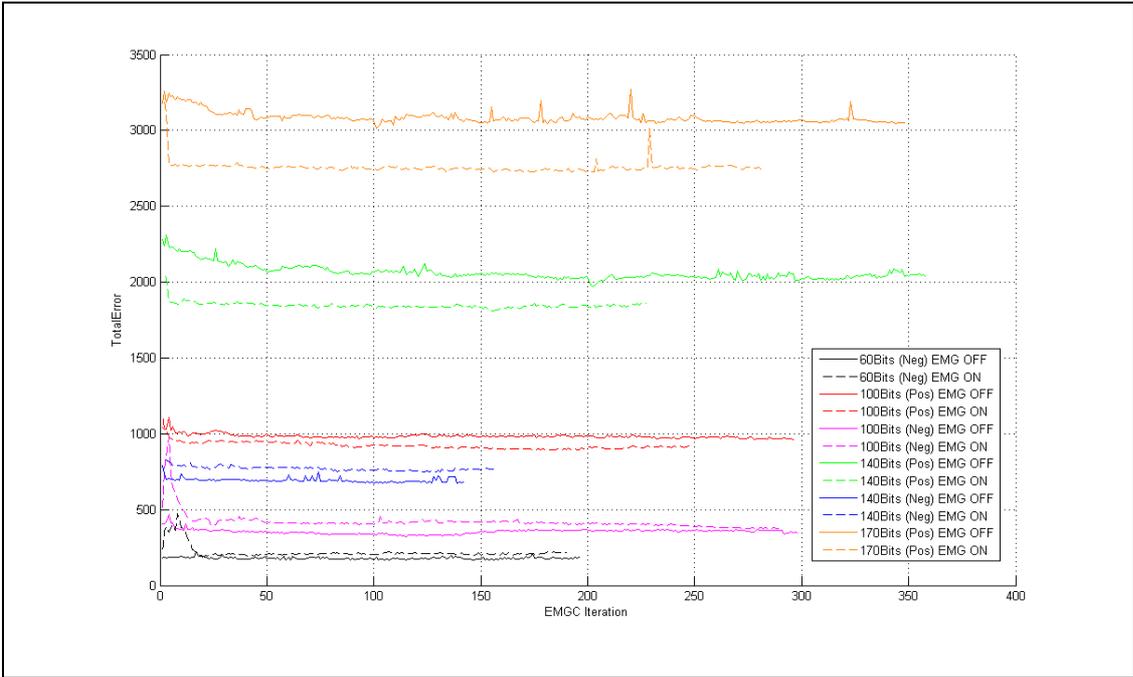


Figure 8.9 – First Spring: Total Error vs. Iteration (Test 08 to Test 15)

Figure 8.9 illustrates the Total Errors associated with each test successfully conducted. Note that the number of successfully recorded situations varies between tests. For a situation to be completed successfully it must meet a number of criteria, which were explained in Section 6.2.1.6. Thus if the criteria are not met, the data for a situation will not be logged and therefore will not appear in the corresponding log

file for the acquired data. Sufficient data was collected for each trace above, however each noticeably converging to a stable error value over the 1300 sec test.

Spikes are evident throughout the training data, and are especially noticeable for the larger Total Error traces. The physical dynamic nature of the air-muscle system means that there is no guarantee that each successive iteration will contain a decrease in Total Error from the previous iteration. Accumulated throughout the range of iterations, it is clear that the Total Error does indeed minimise and stabilise, however for any iteration throughout, sensor noise, physical disturbances, new CMAC input space trajectory outputs, etc, can cause the error to slightly increase above the stabilised value. The dynamic and robust nature of the CMAC however shows that even though these disturbances to the system exist, the overall error of the system stabilises for each trace as in Figure 8.9.

The illustrative purpose of Figure 8.9 is to compare the performance of the CMAC+PID controller with and without the addition of the EMGC. For each test, two traces exist, one with the EMGC on and the other with the EMGC off. Comparing each pair of traces in Figure 8.9 shows that the use of the EMGC does not always result in a final stabilised lower Total Error than when using the CMAC+PID controller alone. For Figure 8.9, the situations that produce a lower error when using the EMGC are: 100 (Pos), 140 (Pos) and 170 (Pos).

To compare the performance gain of using the EMGC for the situations in Figure 8.9, the ratio of the errors between traces for identical situation magnitudes will now be presented. The Error Gain, expressed as a percentage change from the original error (i.e. the error when the EMGC was Off), was obtained using the following equation for each iteration:

$$ErrorGain = \left(\frac{TotalError_{EMGC_ON}}{TotalError_{EMGC_OFF}} - 1 \right) * 100 \% \quad \text{Equation 8.1}$$

The Error Gain vs. the EMGC iteration relationships for the first spring tests are illustrated in Figure 8.10.

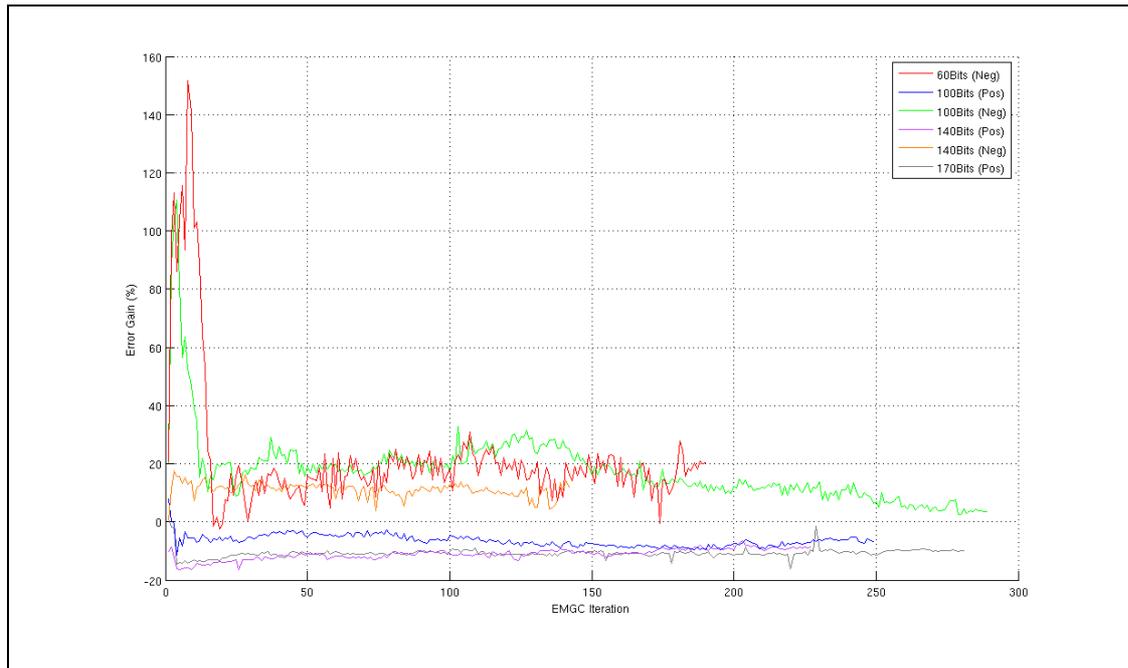


Figure 8.10 – First Spring: Error Gain vs. Iteration (Test 08 to Test 15)

Figure 8.10 directly represents the Error Gains for each situation encountered by the EMGC. The blue 100bit (positive), the purple 140bit (positive) and the grey 170bit (positive) situations all result in an Error Gain below zero when the EMGC is being used. An Error Gain below zero means that the Total Error of the situation has been reduced by the EMGC compared to when the EMGC was not used (i.e. when only the CMAC+PID controller was used).

Figure 8.10 illustrates directly the extent of error reduction the EMGC provides to situations produced on the air-muscle with the first return-spring. All error reduction here is completed by the EMGC, by minimising the first and second errors associated with a situation, converging the Error Ratio of the two errors to zero. The EMGC's ability to do this, taking the Error Ratio and Total Error variables from Figure 8.8, produces Figure 8.11.

Figure 8.11 suggests that no matter which situation is encountered, the use of the EMGC always converges the Error Ratio to zero. This can be seen by noting that the situations used with the EMGC On, have situation convergence regions lying around the zero Error Ratio line.

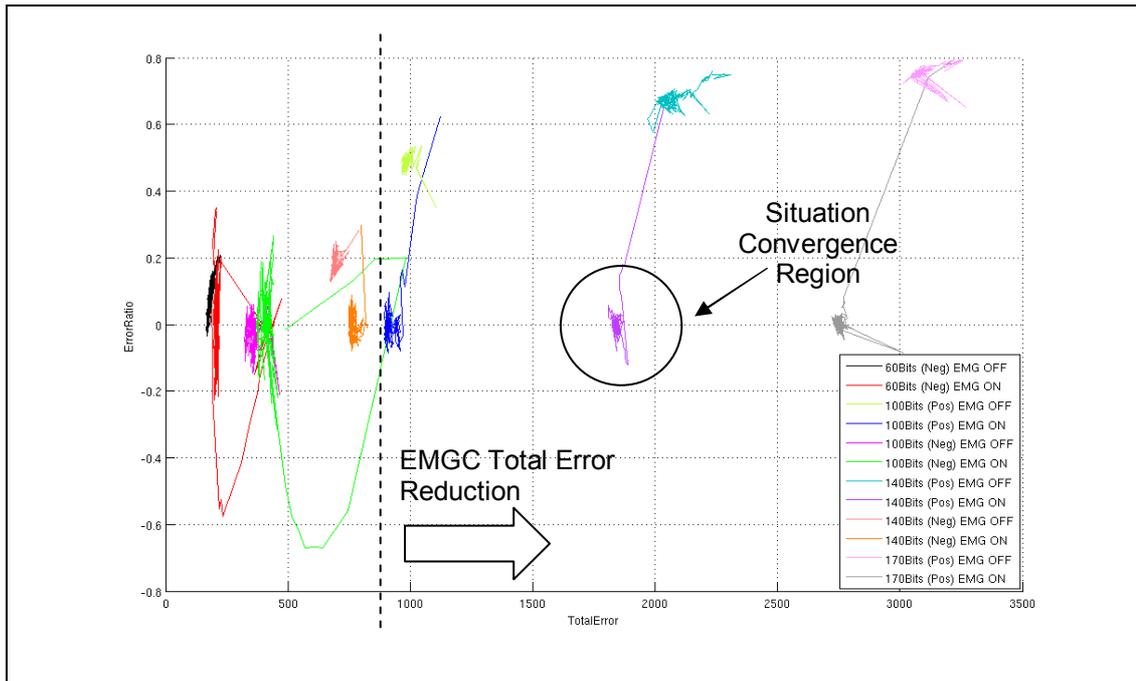


Figure 8.11 – First Spring: Error Ratio vs. Total Error (Test 08 to Test 15)

The situation convergence regions which do not lie around the zero Error Ratio line correspond to situations encountered when the EMGC was off, and the Error Ratio was unable to be reduced to zero. Note that for the situations with Error Ratios that have been reduced to zero, initially the Error Ratio for these situations lie somewhere near their corresponding situation convergence regions for when the EMGC was not used. This is indicated by the lines extending between convergence regions for equivalent situations, illustrating that initially, before the EMGC reduces a situation's Error Ratio to zero, the Error Ratio was much larger than the final converged value.

Figure 8.11 also illustrates that the three situations with the greatest initial Error Ratios are the same situations that correspond to the situations that produced negative Error Gains in Figure 8.10. In Figure 8.10, the blue 100bit (positive), the purple 140bit (positive) and the grey 170bit (positive) situations, all produced a lower Total Error with use of the EMGC. This trend can also be seen in Figure 8.11, whereby the situation convergence regions produced by use of the EMGC all lie to the left (having a lower Total Error) than their equivalent EMGC Off trace.

The remaining traces, as illustrated in Figure 8.10, all produce a greater Total Error with use of the EMGC. In Figure 8.11 two main relationships can be seen. The first relationship is: situations with initial larger Error Ratios produce EMGC minimised Total Errors, and the larger the initial Error Ratio, the greater the reduction in Total Error. The second relationship is: an increase in Error Ratio (when the EMGC is Off) equates to larger Total Errors.

Before conclusions are made, the situations presented above will now be tested on the same air-muscle using the second spring, having a return-spring constant of $k=912N$, to compare the differences in situation responses and relationships on differing dynamic systems.

8.2.2 SQUARE WAVE SITUATION TESTING (SECOND SPRING: $K=912N/M$)

The second series of square wave tests was conducted on an air-muscle rig consisting of a single air-muscle with the second return-spring, of spring constant value $k=912N/m$.

The representation between Total Error, Error Ratio and Training Iteration for the second return-spring test is illustrated in Figure 8.12.

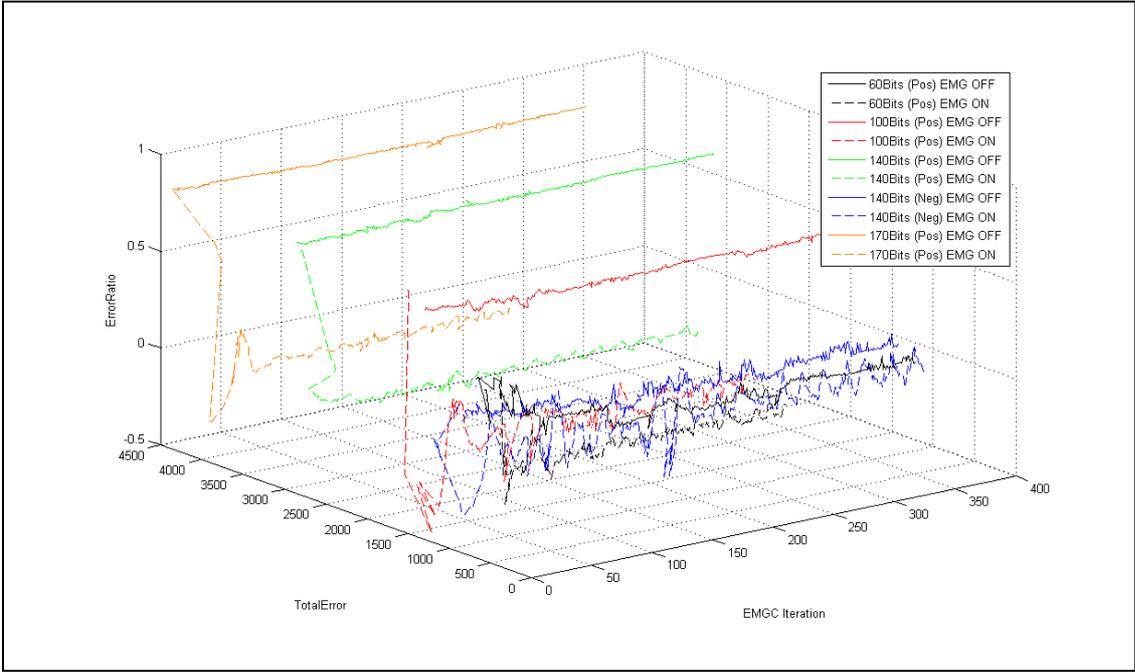


Figure 8.12 – Second Spring: Error Ratio vs. Total Error vs. Iteration 3-D (Test 21 to Test 28)

Figure 8.12 illustrates the relationship between the Error Ratio, the Total Error and the number of successful situation iterations completed over the time span of the tests for the air-muscle actuator using the second return-spring. Comparing Figure 8.12 to its equivalent first return-spring response plot in Figure 8.8, there appears to be no discernable differences apart from the larger Total Errors produced from the second return-spring system. The greater spring force opposing the contraction of the air-muscle is the reason for this, as a greater force is required to move to the desired position in the positive direction of motion. This in turn produces a slower response which creates larger Total Errors.

Comparing the Total Error and EMGC iteration data from Figure 8.12 produces Figure 8.13 below.

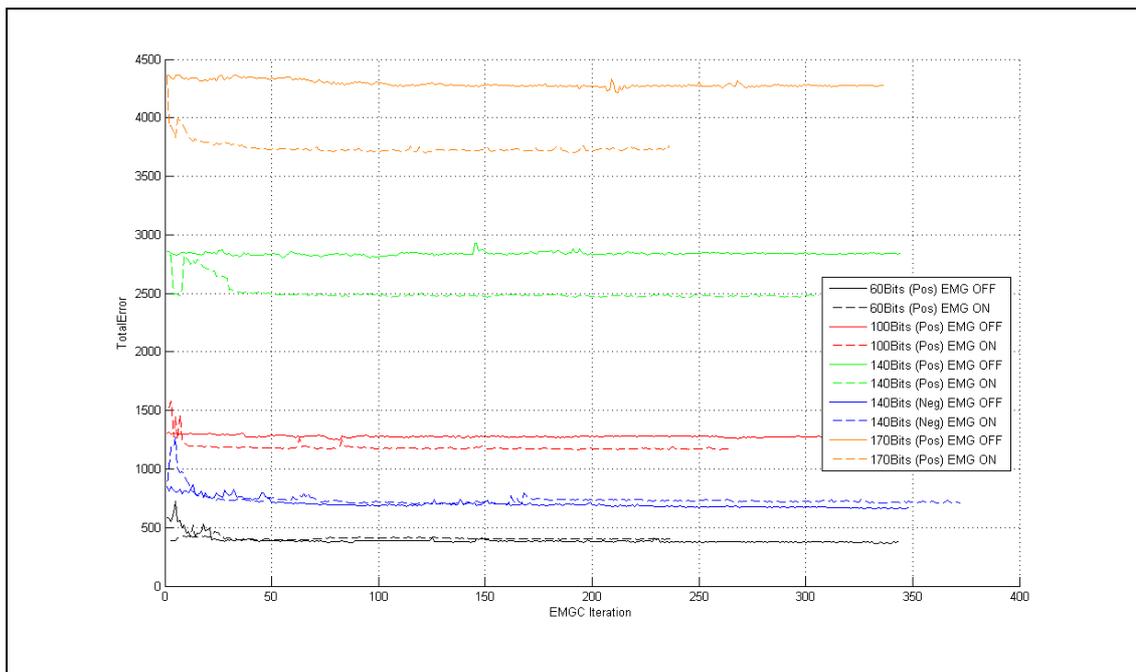


Figure 8.13 – Second Spring: Total Error vs. Iteration (Test 21 to Test 28)

Figure 8.13 illustrates the Total Errors associated with each test conducted for successfully completed tests on the second return-spring actuator. The responses appear much the same as their corresponding response plots for the first return-spring tests. Convergence of the errors is achieved for each test, whether or not the EMGC is used. Again the EMGC produces a greater error minimising effect for situations with larger Total Errors, as opposed to smaller Total Error situations.

To compare the performance gained by using the EMGC for the situations tested above, the ratio of the errors plotted will now be presented. The Error Gain was obtained using Equation 8.1. The Error Gain vs. the EMGC iteration relationship is illustrated in Figure 8.14.

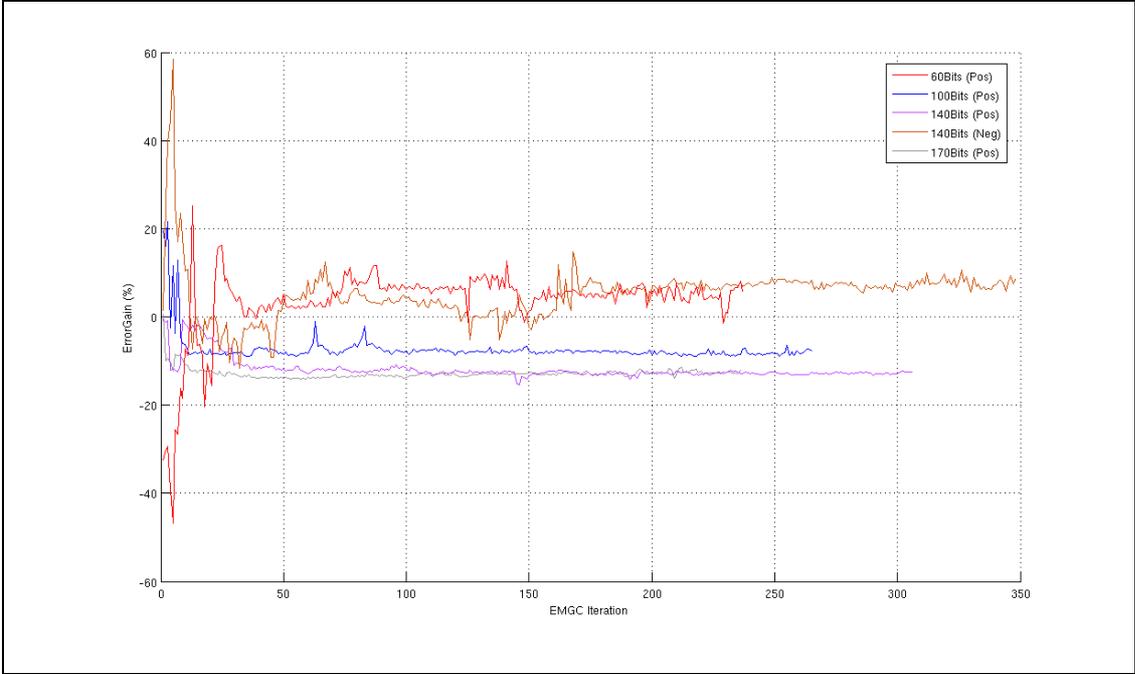


Figure 8.14 – Second Spring: Error Gain vs. Iteration (Test 21 to Test 28)

Again, the traces in Figure 8.14 corresponding to Figure 8.13 illustrate that the situations with the larger Total Errors benefit far more from the use of the EMGC than do the situations that create smaller Total Errors. The blue 100bit (positive), the purple 140bit (positive) and the grey 170bit (positive) situations, once again all result in a negative Error Gain when the EMGC is used. Note that stabilisation of the Error Gain for situations that produce negative Error Gains converge smoother than the situations that produce positive Error Gains. Figure 8.14 shows a stabilised Error Gain of up to approximately -13% as shown in the 140bit (positive) and 170bit (positive) traces.

Comparing the Error Ratio and Total Error data from Figure 8.12 produces Figure 8.15.

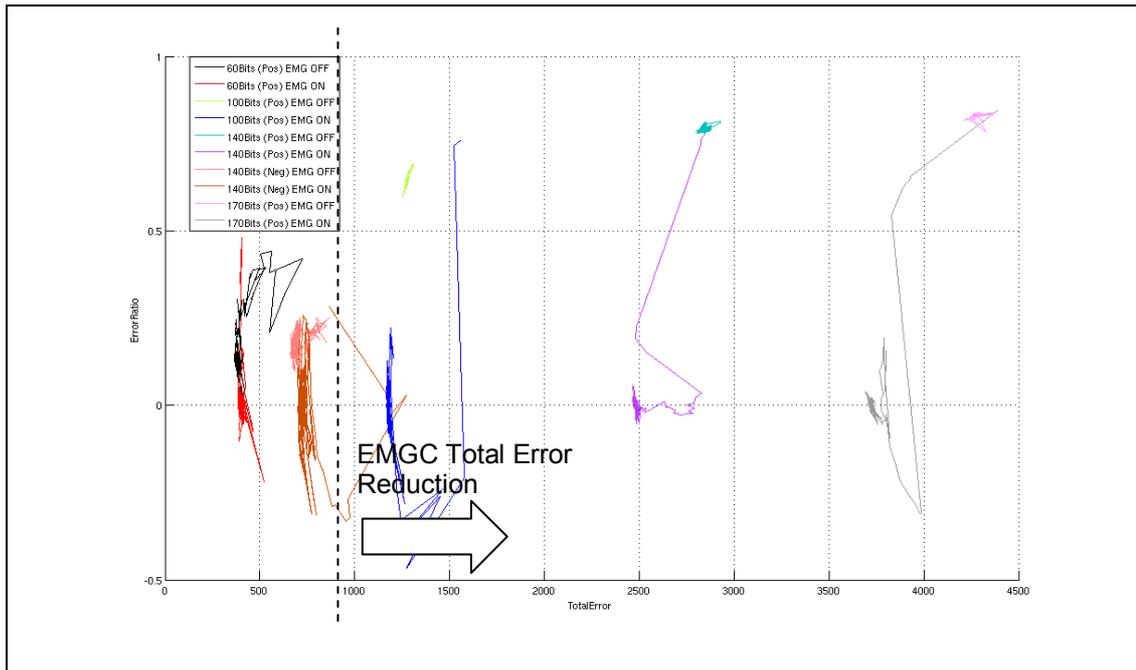


Figure 8.15 – Second Spring: Error Ratio vs. Total Error (Test 21 to Test 28)

Figure 8.15 illustrates that no matter which situation is encountered, the use of the EMGC always converges the Error Ratio to zero. The situation convergence regions produced by the EMGC all lie around the zero Error Ratio line, a decrease from the initial Error Ratio associated with their corresponding non-Error Ratio minimised situations (EMGC Off).

Similarly to the single spring tests, the three situations with the greatest initial Error Ratios are the same situations that produce negative Error Gains in Figure 8.14. In Figure 8.14, the blue 100bit (positive), the purple 140bit (positive) and the grey 170bit (positive) situations, all produced a lower Total Error with use of the EMGC. This trend can also be seen in Figure 8.15, whereby the situation convergence regions produced by use of the EMGC all lie to the left (having a lower Total Error) than their equivalent EMGC Off trace.

The remaining traces in Figure 8.15 all produce a greater Total Error with use of the EMGC. Much the same as from the first spring tests, the same relationships can be seen in Figure 8.15. The situations with initial larger Error Ratios are those which produce a greater reduction in Total Error with the EMGC. In addition, larger Total Errors produce an increase in Error Ratio (when the EMGC is Off).

A discussion of all situations tested in both series of tests above using the first and second air-muscle return-springs, will now be presented, providing explanations for why the EMGC can both reduce or increase a situation's Total Error.

8.2.3 COMPARISON OF FIRST AND SECOND RETURN-SPRING TEST DATA

Combining the data extracted from both the first and second return-spring tests presented above for different step sizes, will allow a direct comparison of the overall data to be discussed. This is important as it shows what similarities exist between tests conducted on two actuators with differing dynamic properties.

From Figure 8.11 and Figure 8.15, it was shown that situations with larger Error Ratios had their situation's Total Error reduced by use of the EMGC. For situations that produced an initially small Error Ratio, the EMGC increased a situation's Total Error. Combining the data from Figure 8.11 and Figure 8.15 together produces Figure 8.16 below.

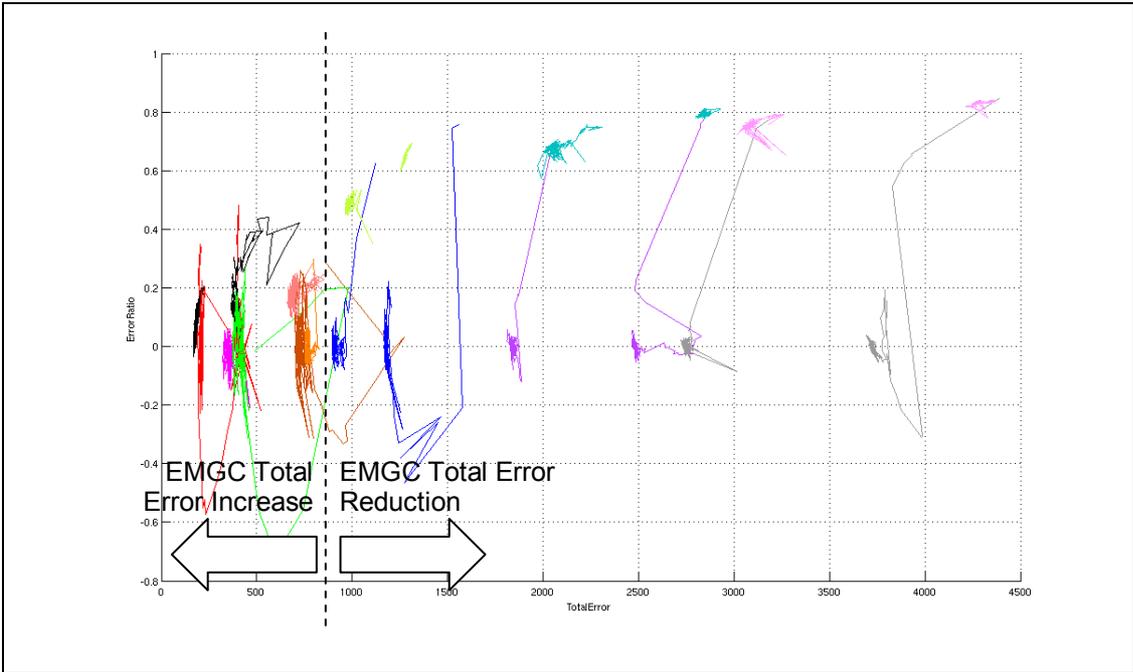


Figure 8.16 – First and Second Spring Tests: Error Ratio vs. Total Error (Test 08 to Test 28)

Figure 8.16 illustrates the Error Ratio against the Total Error data for both the first and second spring tests. The data is a reproduction of the data illustrated in Figure 8.11 and Figure 8.15, however combined together to produce an overall visible trend.

The six situations to the right in Figure 8.16 produce large initial Error Ratios, and it is these situations which benefit most from use of the EMGC in reducing their situation's Total Error. This is indicated in Figure 8.16, where all Total Error reduced situations lie to the right of the overlaid dashed line, and all have greater Total Errors than approximately 850 bits. These situations also have a clearer situation convergence region, and it can be noticed that for situations with larger Total Errors, the situation convergence regions are tighter and more defined. The reason for this is that smaller changes in the system response have a smaller effect on large error regions than they do on small error regions.

A clear separation of the situation convergence regions, illustrating the relationship between the Error Gain of a situation against its Total Error, for the data collected from both the first and second spring tests, is illustrated in Figure 8.17.

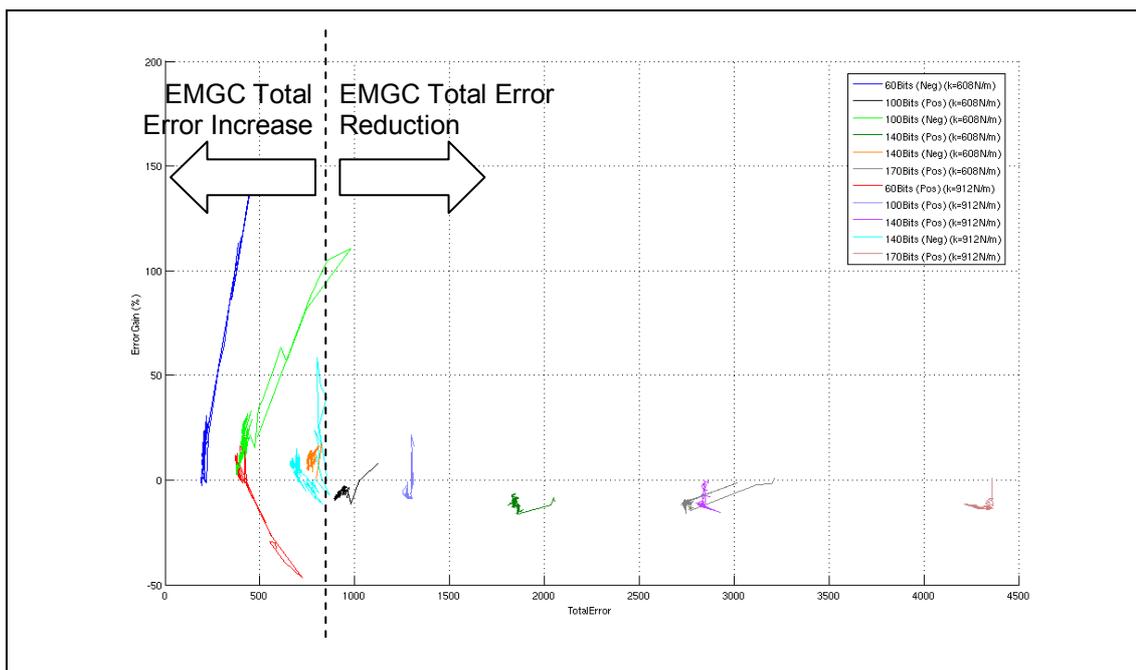


Figure 8.17 – First and Second Spring Tests: Error Gain vs. Total Error (Test 08 to Test 28)

Figure 8.17 illustrates the Error Gain against the reduced Total Error for the first and second spring tests. The situation convergence regions lying to the right of the dashed line overlay (greater than 850 bits) all lie below the 0% Error Gain line, meaning that the Total Error of the situations have decreased with use of the EMGC. Situation convergence regions lying to the right of the line all have an increased

Error Gain, meaning that the use of the EMGC on these situations has increased the situation's Total Error. Thus from these results, it can be concluded that the EMGC does not always reduce the error associated with a situation.

The situations which produce an initial Total Error above 850 bits in Figure 8.16 correspond to those situations in Figure 8.17 which produce negative Error Gains. While situations with Total Errors greater than 850 bits all result in a reduced Total Error with use of the EMGC, the Total Error itself cannot be used as a critical variable in determining the performance of the EMGC on a situation, as it is a system dependent output variable.

A trend is also noticed in Figure 8.16, where situations with Total Errors greater than 850 bits also have far greater initial Error Ratios than do the situations with Total Errors less than 850 bits. The first situation with a Total Error greater than 850 bits has an initial Error Ratio of approximately 0.4. All situations with initial Error Ratios greater than 0.4 result in a lower Total Error with use of the EMGC. Situations with initial Total Errors less than 0.4 result in greater Total Errors with use of the EMGC.

Error ratio, unlike the Total Error, is system independent. Error ratios will exist between -1 to +1 for any situation on any system where situations exist, so long as a clearly defined first and second error region exist. Thus the Error Ratio is a very useful variable to use in determining whether or not the EMGC will be suitable in reducing a situation's Total Error.

8.2.4 EXPLANATION OF RESULTS

From Figure 8.17, it has been shown that the EMGC does not always reduce a situation's Total Error. This was the original intent of the EMGC, however on a practical system, this has been demonstrated to be not always true.

The EMGC's limited effectiveness, demonstrated by the existence of the 850 bit Total Error crossover point, exists because the EMG overlay produces a final training target which is less aggressive than the original trajectory situation. This is illustrated in Figure 8.18, where the original desired step function is replaced with the new training signal, which has a gradient less than the initial step function.



Figure 8.18 – EMGC Output Providing New Desired Training Signal to CMAC (Identical to Figure 6.22)

Figure 8.18 illustrates how a step function was originally used to move the response of the system from one trajectory position to another. The EMGC however produces an EMG to replace the step function (which has a very large desired velocity), and the EMG has a desired velocity of far less magnitude than the initial step function. The slope of the EMG is determined by the maximum response of the actuator for a situation, and by the EMG Offset Percentage. As explained in Section 8.1, the EMG cannot have too steep a gradient, and thus the EMG (gradient) will also be less than the original maximum desired velocity of a situation. Thus the use of the EMG produces a final training target which is less aggressive than the original trajectory situation.

In addition, an inherent level of Situation Pre-empting is already implemented from the use of weight smoothing, as discussed in Section 6.2.2.3. This is illustrated in Figure 8.19, showing the system response (A_p) produced without the use of the EMGC. The inherent level of Situation Pre-empting allows the actual response of the system to start moving in the direction of the situation before the situation occurs, for both the rising and falling edges of the desired square wave function.

Figure 8.19 illustrates the initial response of the system to a desired square wave input function. The inherent level of Situation Pre-empting means that the initial Error Ratio of the situation will never be equal to 1. If no inherent Situation Pre-empting existed, then the second error region would not initially exist (the error

region to the left of a situation) and thus the initial Error Ratio would be equal to approximately 1.

This would be the case for all situations, regardless of the initial Total Error size. However in practice, some degree of Situation Pre-empting does initially exist, producing initial Error Ratios anywhere between 0 to +1, meaning that the EMGC is not always needed.

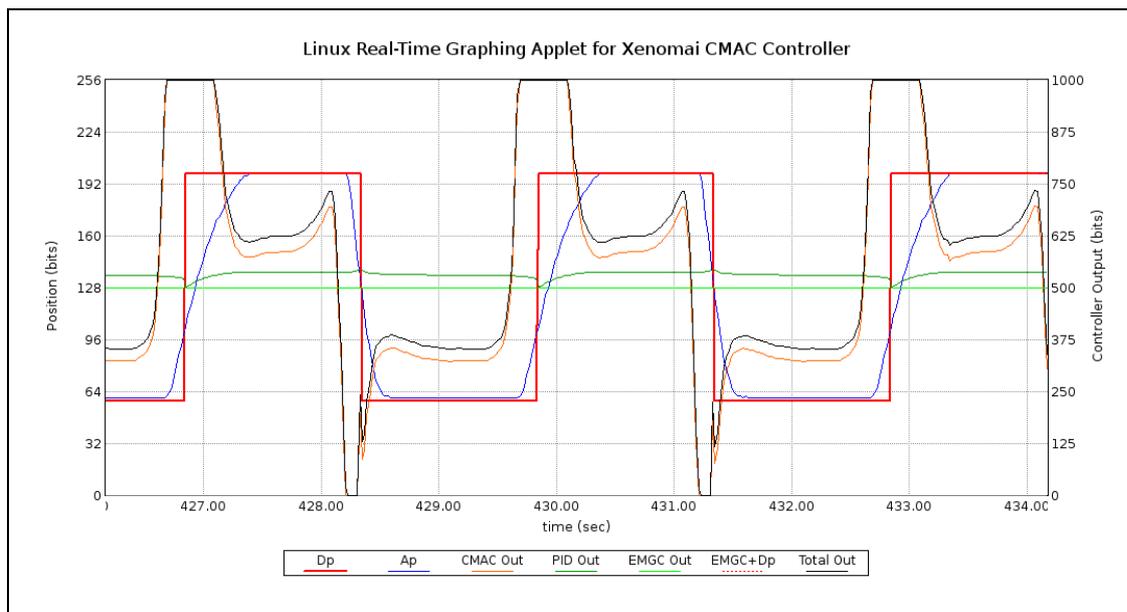


Figure 8.19 – Initial Physical Response of the Air-Muscle before the EMG Superposition Occurs

Combining both factors together, that is the less aggressive EMG training signal and the inherent level of Situation Pre-empting, produces Figure 8.20.

Figure 8.20 helps explain why the EMGC is useful in reducing the Total Errors for some situations and not for others. Situations with initially small Total Errors, where the first and second error regions are approximately equal, do not benefit from use of the EMGC. The inherent level of Situation Pre-empting produced by weight smoothing will produce an initial small Error Ratio. If an EMG is applied to this situation, the Error Ratio of the situation will continue to decrease until approximately zero. However the new less aggressive training target produced from the EMG by replacing the step function with a function of a less step gradient, results in an increased Total Error for the situation. This is illustrated in Figure 8.20(a).

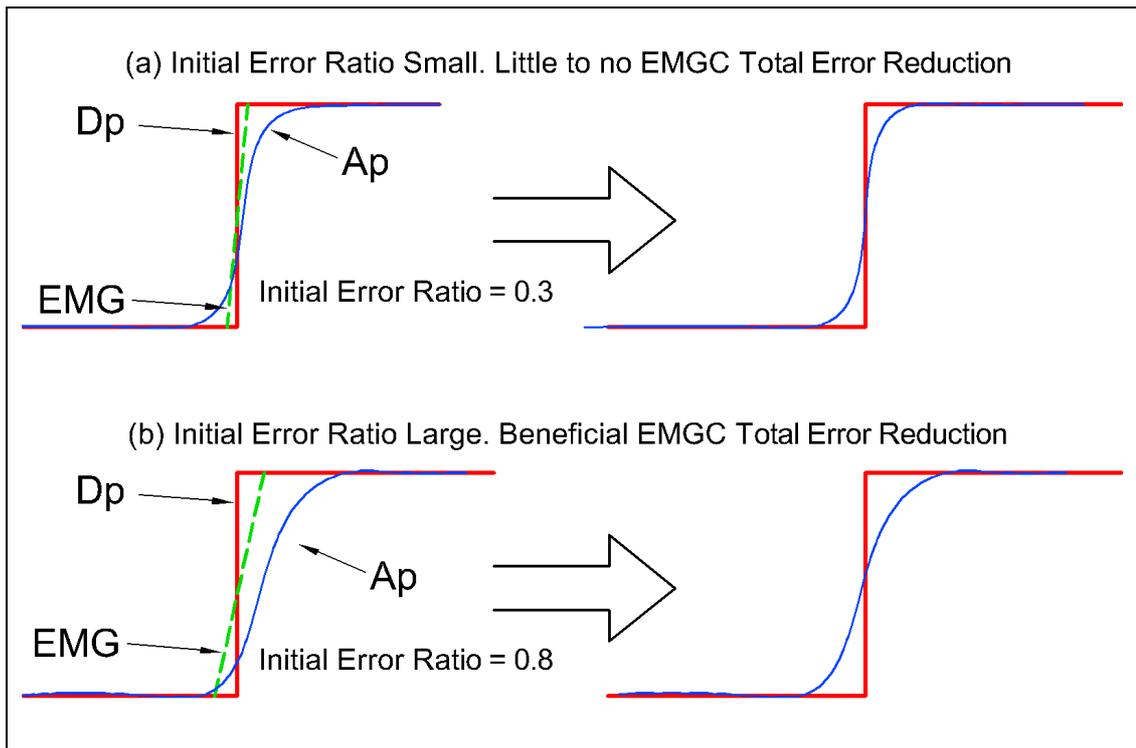


Figure 8.20 – (a) EMGC Not Beneficial for Small Initial Error Ratios (b) EMGC Beneficial for Large Initial Error Ratios

The situations that benefit most from use of the EMGC are those with initially large Error Ratios. Such a situation is illustrated in Figure 8.20(b), where initially the first error region is much larger than the second error region, producing a large positive Error Ratio. For this situation, replacing the desired step with an EMG at the appropriate point in time will move the actual response of the system back in time to reduce the Error Ratio to zero. This will also reduce the overall Total Error, as the benefits of using the EMG outweigh the disadvantages.

From Figure 8.16, a value will exist on the Error Ratio scale which will provide a crossover point for when the EMGC is beneficial in reducing a situation's error and when it is not. Situations with initial Error Ratios above this point should be used with the EMGC to produce a minimised Total Error. Situations with initial Error Ratios below this point should not be used with the EMGC, as the EMGC will either have no effect or increase the situations Total Error.

All situations with initial Error Ratios above approximately 0.4 produced reduced Total Errors with the EMGC. This initial Error Ratio crossover point of 0.4,

empirically derived from Figure 8.16, can be used as a threshold to allow the EMGC to choose when it first encounters a situation whether or not to produce an EMG for the situation and reduce the situation's Error Ratio to zero. For initial Error Ratios below 0.4, the EMGC can ignore the situation as it cannot reduce a situation's Total Error. This crossover point will vary from system to system depending on the inherent level of Situation Pre-empting, but nonetheless will exist for some Error Ratio value as it does here. For the most beneficial use of the EMGC in a control system, using the EMGC on situations where it is guaranteed to reduce the error will produce an overall error minimised response for the system.

8.3 EMGC DIRECT EFFECT ON ACTUAL TRAJECTORY RESPONSE

While the previous sections in this chapter have dealt with the error reduction magnitudes produced by the EMGC, the physical effect directly seen on an actual trajectory response has not been presented. The reason for this is that it is difficult to directly show the change in actual trajectory response (such as that illustrated in Figure 8.21) over each EMGC iteration for each test taken, and prevents in-depth controller error-reduction comparisons. However, visually providing an illustration of the desired and actual response differences for a single iteration provides a clear understanding of the EMGC's effect on a system's output. Figure 8.21 compares the air-muscle's response with and without using the EMGC together with the CMAC+PID controller. Note that the EMGC is only being used for positive gradient situations (i.e. the rising edge of the square wave) in this example. An animated version of Figure 8.21 illustrating the CMAC+PID with EMGC convergence is included on the Thesis CD.

For the EMGC Off response (meaning the CMAC+PID controller is producing the response alone), the error has been minimised for a single instance in time throughout the periodic response, with the actual response starting to move in the direction of the situation before the situation occurs. As explained in Section 5.2.4, weight smoothing together with the feedback time delay compensation moves the system output in the direction of the large desired step change before the situation occurs. A large first-error region for the situation exists however, clearly being much larger than the situation's second-error region. Thus the Error Ratio for the situation is positive.

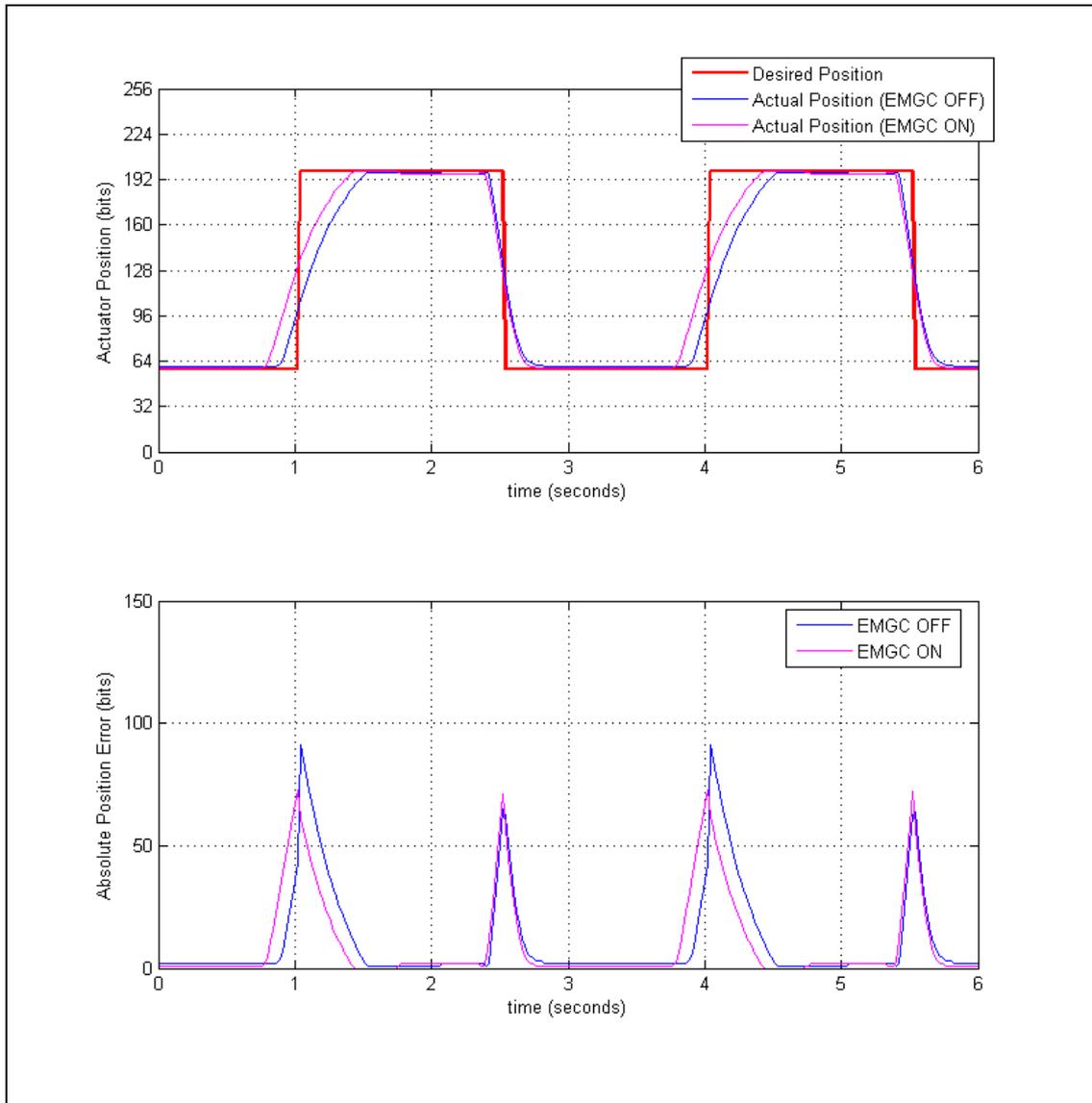


Figure 8.21 – Position and Error Response Comparing EMGC to CMAC+PID Control.

For the EMGC On response, the error has been minimised not only for a single instance in time, but also over the time surrounding a situation. The most obvious difference between the EMGC On response plot compared to the EMGC Off response plot, is that the actual position starts to move in the direction of the situation well in advance before the situation occurs. It is the EMGC which produces this difference, starting to move the actual response of the system in the direction of the situation, towards the point in time which will minimise the Total Error of a situation (i.e. when the Error Ratio converges to zero).

8.4 CHAPTER CONCLUSIONS

In conclusion of the square wave tests undertaken throughout Chapter 8.0, the following points can be made for desired square wave trajectories:

1. An EMG offset percentage of approximately 50% produces a lower converged Total Error compared to when using the maximum attained velocity of the actuator (i.e. an EMG offset percentage of 0%). However, too great an EMG offset percentage and the EMG may not intersect the original desired trajectory's path with a great enough pre-emption time shift, causing the Error Ratio to be unable to converge to zero.
2. For situations where the EMGC was able to repetitively produce an EMG, the EMGC always converged a situation's Error Ratio to zero.
3. The EMGC can either increase or decrease a situation's Total Error. Situations with initial Error Ratios greater than 0.4 all result in a reduced Total Error with use of the EMGC. Situations with initial Error Ratios less than 0.4 result in an increased Total Error with use of the EMGC.
4. The EMGC's Total Error reduction capabilities are influenced by the size of a situation's initial Error Ratio. The greater a situation's initial Error Ratio, the greater the Total Error reduction is with use of the EMGC.
5. The greater the level of inherent Situation Pre-empting provided by the CMAC+PID controller, the lesser the error reduction effect of EMGC on a situation's Total Error. The reason behind this is due to the inherent Situation Pre-empting reducing the initial Error Ratio to a value between 0 and 1, moving the output in the direction of a situation before the situation occurs. This lowers the error reduction effect provided by the EMGC.
6. The EMGC produces an EMG to replace a situation with a less steep gradient in the desired trajectory. This reduces the aggressiveness of the desired trajectory at the time of the situation, and thus produces an actual response which is slower than the initial response (without the EMG) of the system. It is this reason why situations with initial Error Ratios below 0.4 do not benefit by use of the EMGC.
7. The above points allow a final conclusion to be made: the initial Error Ratio of 0.4 is a result of the EMG offset percentage used. The greater the EMG offset percentage, the lower the initial Error Ratio threshold would be. The lower the

EMGC offset percentage, the greater the initial Error Ratio threshold would be. Thus the effectiveness of the EMGC is a result of the EMG offset percentage. An EMG which has the same gradient of the situation would produce the greatest error minimisation from the EMGC, resulting in an initial Error Ratio threshold of approximately zero. However using too steep an EMG is not possible, based on the reasoning supported by Figure 8.6.

The final series of tests to be conducted involves testing the EMGC over a variety of dynamic fingerspelling trajectory combinations. These trajectories can end and start with any possible gradient, and can start and finish at any position in the air-muscle stroke. Thus a series of tests needs to be conducted, simulating the joining of trajectories of varying gradients and varying position magnitudes. The following chapter will study these situation conditions with and without the use of the EMGC.

9.0 APPROACH AND DEPARTURE GRADIENT TESTS

This section involves a series of tests which have been designed to assess the functionality of the EMGC for given situations with varying approach and departure gradients. The tests throughout Chapter 8.0 involved training the EMGC and CMAC+PID controller with a square wave desired trajectory of varying magnitude. However the joining of hand gesture trajectories will not only involve stationary finger joint trajectories with situations of various magnitudes, but will also consist of a variety of situation approach and departure gradients representing dynamic finger joint motions, such as in the letter ‘C’ of the deafblind manual (see Figure 2.1).

Situation approach and departure gradients refer to the slope of the desired trajectory immediately before and after the situation occurs. A hand gesture trajectory could be any motion profile recorded by a data glove, stored as a set of sampled data of fixed frequency, and then used directly or interpolated in terms of position resolution to train the EMGC and CMAC+PID controller. Combining these individual gestures results in a continuous stream of natural hand gesture trajectories separated by situations, as illustrated in Figure 9.1.

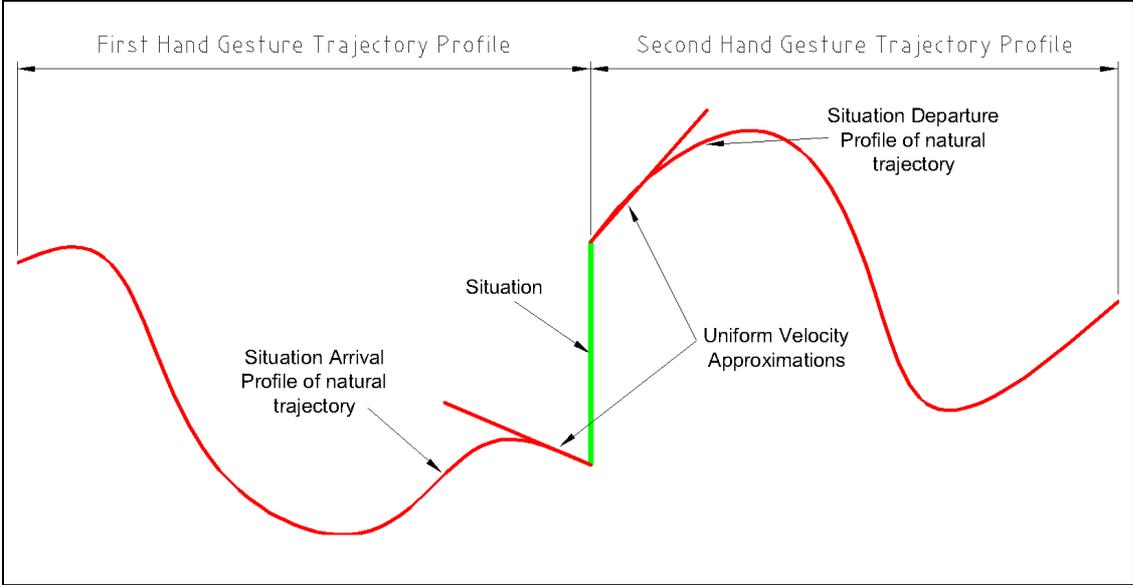


Figure 9.1 – Natural Hand Gesture Trajectory Approach and Departure Profiles for a Situation

Figure 9.1 illustrates a natural motion profile for the joining of successive hand gesture trajectories, together with the uniform velocity approximations for the

situation's approach and departure trajectory profiles. Approximating the non-uniform approach and departure velocities as a fixed velocity allows testing to be carried out using a variety of fixed approach and departure velocities, rather than an infinite amount of non-uniform profiles.

Presenting these approximations to the controller in this way allows the EMGC to be tested on a wide variety of possible profiles that may occur when combining natural hand gesture trajectory profiles together. These tests will not only show the performance of the EMGC when working with non-square wave desired trajectories, but will also demonstrate that the EMGC detection algorithm and EMG creation algorithms work in practice for situation approaches and departures of varying gradients.

9.1 APPROACH AND DEPARTURE GRADIENTS METHODOLOGY

For the following tests, a variety of approach and departure gradients will be simulated for a positive situation, on a single air-muscle system with a return-spring constant of $k=604N/m$. The desired training trajectory will simulate a variety of fixed approach and departure gradients for a given situation, by use of the periodic trajectory profile illustrated in Figure 9.2.

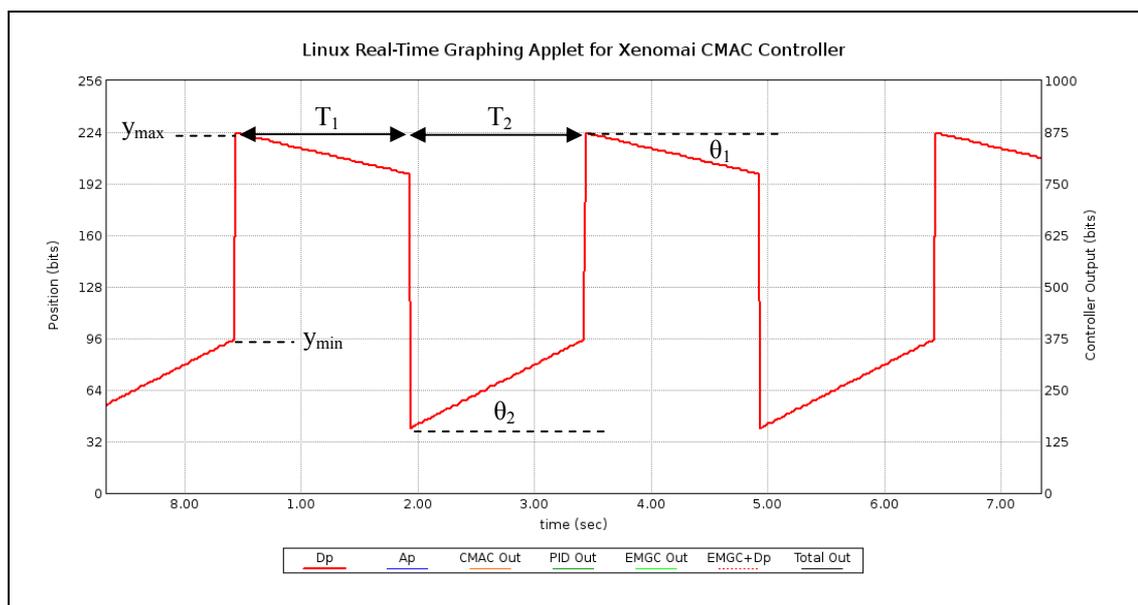


Figure 9.2 – Desired Training Trajectory for Simulated Situation Approach and Departure Gradients

Figure 9.2 illustrates a training trajectory used for the tests in this chapter, with the important parameters labelled. A single training period is made up of two parts, of time lengths T_1 and T_2 in CMAC sampling iterations. A situation's magnitude is defined by the difference between the situation's start and end positions labelled y_{min} and y_{max} respectively. The approach and departure gradients of a situation are defined by angles θ_2 and θ_1 respectively. These parameters together make up a single training period, which is continuously repeated.

The positive and negative approach and departure gradient profiles are illustrated in Figure 9.3.

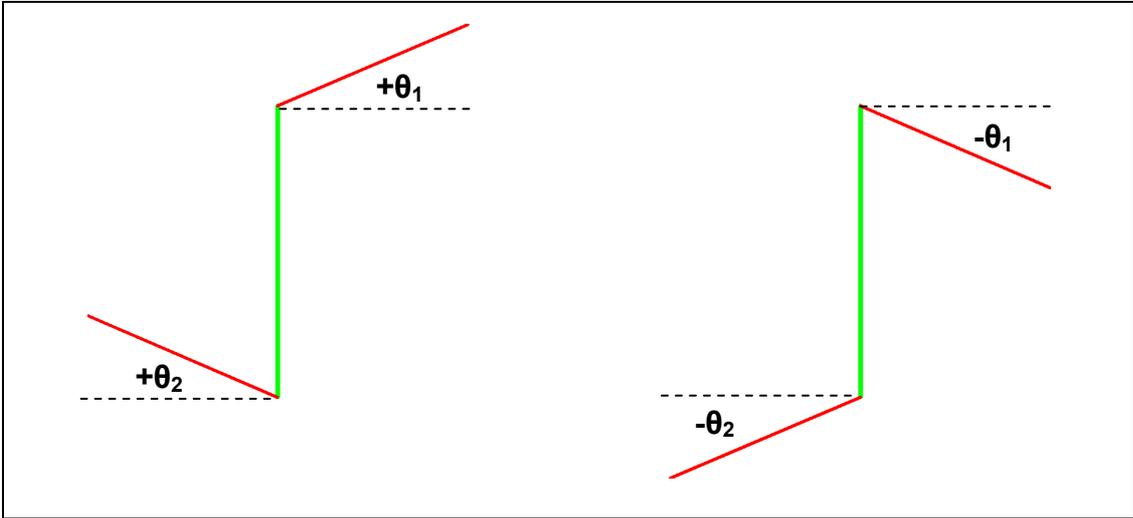


Figure 9.3 – Positive and Negative Approach and Departure Gradients

The remainder of this chapter will now test the EMGC with varying approach and departure gradients.

9.2 APPROACH AND DEPARTURE GRADIENT TESTS

A series of tests was conducted using a varying combination of approach and departure gradients, to fully simulate a realistic range of desired trajectory possibilities. The situation size was fixed at 116bits and both the approach and departure gradients were varied.

The results presented in this section use an average Error Gain, obtained from the mean of the converged portion of the Total Errors obtained for each test. Unlike all

previous tests, this was required to omit the EMGC Iteration count as a variable in the result plots, to reduce the number of variables plotted.

The following combinations of approach and departure gradients used in Test 73 to Test 122 are illustrated in Figure 9.4.

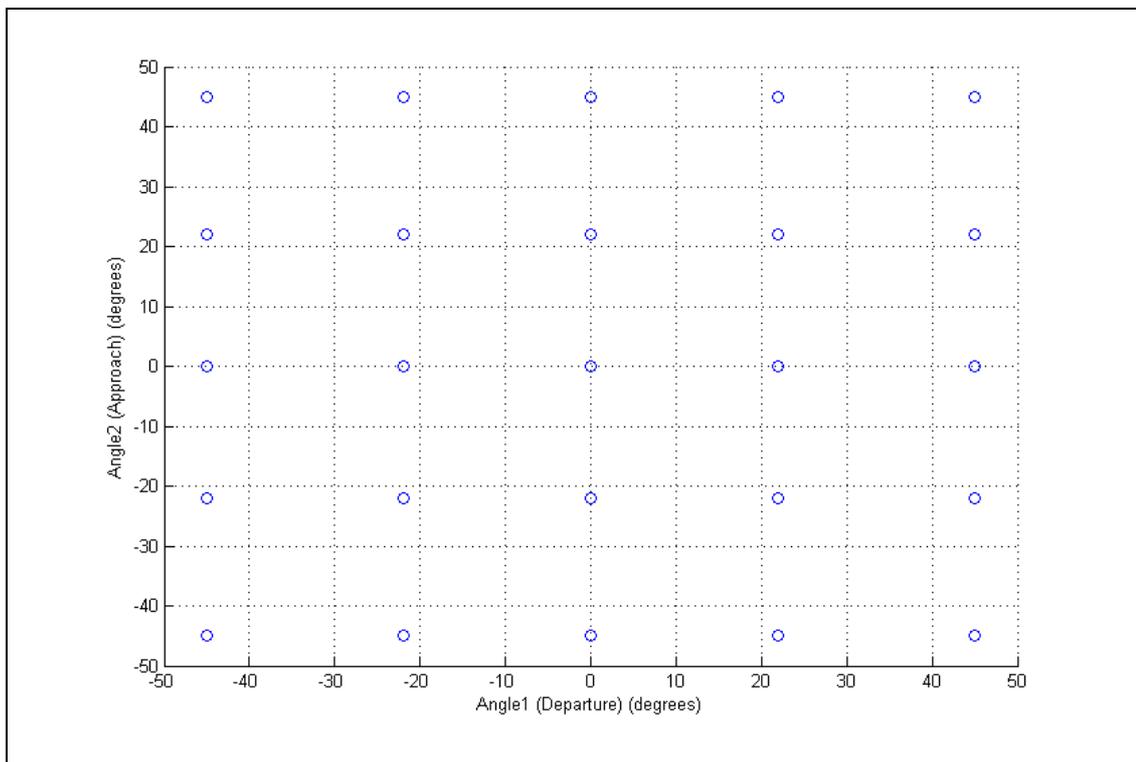


Figure 9.4 – Departure and Arrival Gradient Combinations Used in Test 73 to Test 122

Figure 9.4 illustrates a total of 25 performed tests. However each test was performed twice: a test with the EMGC On and a test with the EMGC Off, producing a total of 50 tests. The test result data is contained in Appendix G for Test 73 to Test 122. The maximum range of approach and departure gradients used is limited between -45° to 45° .

9.2.1 INITIAL ERROR RATIO VS APPROACH AND DEPARTURE GRADIENT RELATIONSHIPS

The first relationship to be studied from the varying approach and departure gradient tests involves how the initial Error Ratio changes with respect to the departure gradient for various approach gradients. These relationships are illustrated in Figure 9.5.

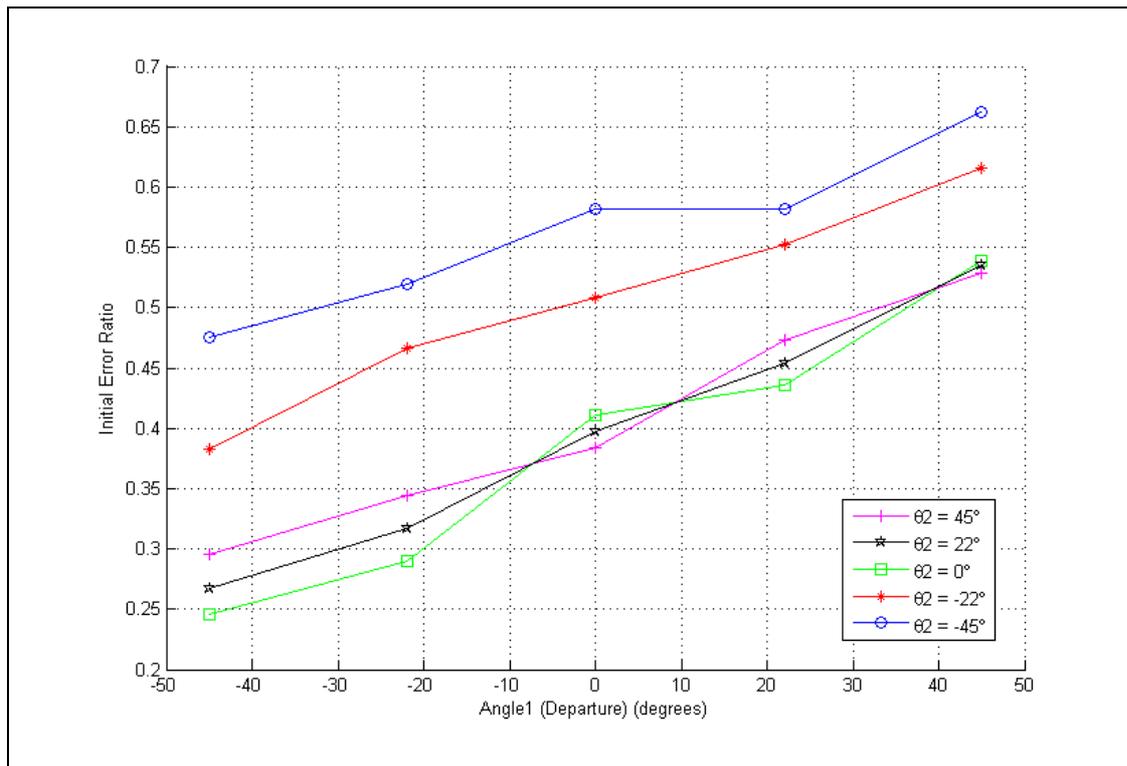


Figure 9.5 – Error Ratio vs. Departure Gradient for Various Approach Gradients (Test 73 to Test 122)

Figure 9.5 illustrates a common trend for all approach gradients: as the departure gradient increases, so too does the initial Error Ratio. Thus regardless of the approach gradient, an increase in the departure gradient will result in a direct increase in Error Ratio. For most of the approach gradient traces, this relationship is fairly linear, especially when $\theta_2 = -22^\circ$ and $\theta_2 = 22^\circ$. A continual increase in the initial Error Ratio is noted as the approach gradient moves from $\theta_2 = 0^\circ$ to -45° , but this relationship is not distinct for positive approach gradients, with Error Ratios for positive approach gradients all producing fairly similar Error Ratios, regardless of the departure angle.

The strong increase in initial Error Ratio as the departure gradient increases, for all approach gradients, is the result of the increasing departure gradient directly increasing the magnitude of the first error region. Independent to the second error region, an increase in the first error region, as defined by Equation 9.1, will directly increase the Error Ratio. This relationship is evident here.

$$ErrorRatio = 1 - 2 * \frac{2ndError}{1stError + 2ndError}$$

Equation 9.1
(also Equation 6.2)

The data represented in Figure 9.5 can also be represented in Figure 9.6 by swapping the approach and departure gradient labels.

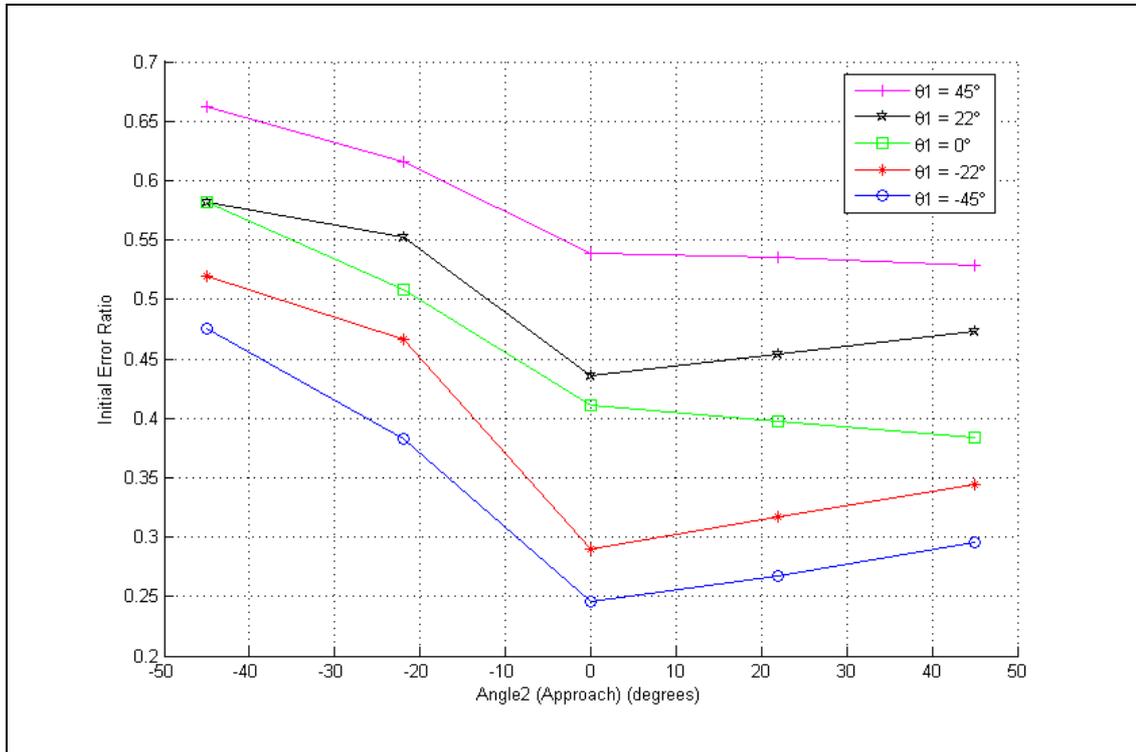


Figure 9.6 – Error Ratio vs. Approach Gradient for Various Departure Gradients (Test 73 to Test 122)

Unlike the departure gradient vs. initial Error Ratio relationship in Figure 9.5, a strong linear relationship does not exist between the approach gradient and the initial Error Ratio in Figure 9.6. For negative approach gradients there is a clear relationship, with decreasing approach gradients producing an increase in initial Error Ratio. However for positive approach gradients, in comparison there is little or no approach gradient effect on the initial Error Ratio.

The approach gradient directly has an effect on the size of the second area region (the error region before the situation) in the same way the departure gradient has a direct effect on the first error region. As the departure gradient increases, as illustrated in Figure 9.5, so too does the initial Error Ratio for all approach gradients, as the first

error region increases in size. By Equation 9.1, this means that the initial Error Ratio will increase, as the ratio of error between the second and first error regions is increased.

This relationship between the approach gradient and the Error Ratio is not as directly obvious as it is for the departure gradient. Weight smoothing and the time delay compensation denote that the inherent level of Situation Pre-empting in the CMAC+PID controller will create an initial Error Ratio less than 1. Therefore depending on the approach gradient, the CMAC+PID controller will start to move the system in the desired direction of the situation some period of time before the situation occurs.

The Error Ratio from Equation 9.1 is a product of both the first and second error regions. Thus to understand the trends in Figure 9.6, the first and second error regions with respect to the changing approach and departure gradients must be studied independently. The magnitude of the first error region with respect to the approach gradient is illustrated in Figure 9.7.

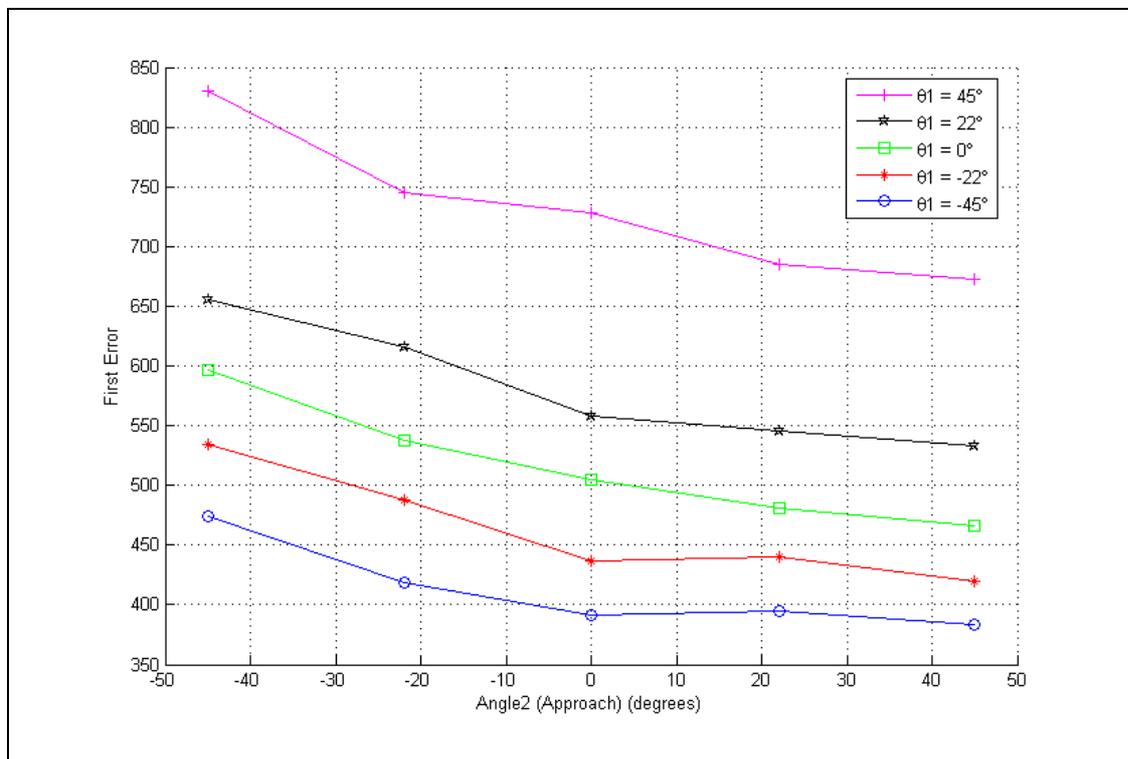


Figure 9.7 – First Error vs. Approach Gradient for Various Departure Gradients (Test 73 to Test 122)

Figure 9.7 illustrates a clear relationship between the size of the first error region and the departure gradient (θ_1). As the departure gradient increases from -45° to 45° , the size of the first error region increases. This trend is expected, because increasing the departure angle means that the actual position response will take longer to reach the desired trajectory. This results in a larger first error region.

There is no evidence in Figure 9.7 to provide reasons for the shape of the trends in Figure 9.6. The magnitude of the second error region with respect to the approach gradient is illustrated in Figure 9.8.

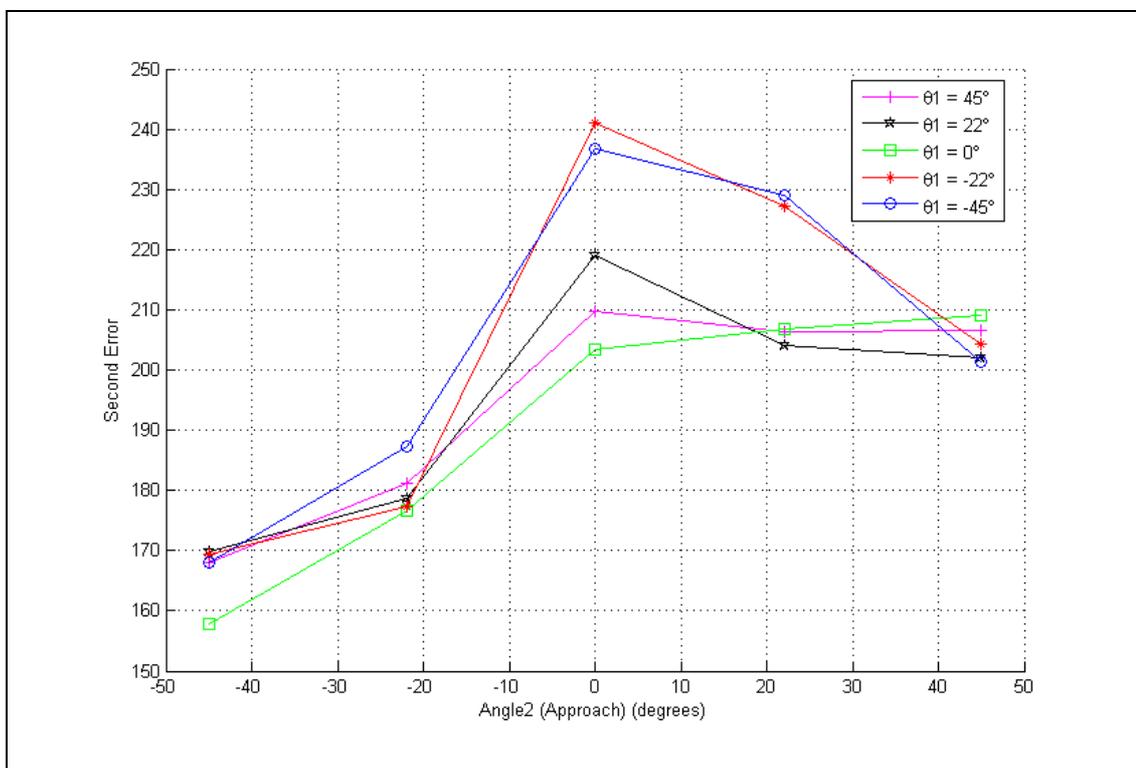


Figure 9.8 – Second Error vs. Approach Gradient for Various Departure Gradients (Test 73 to Test 122)

Figure 9.8 illustrates the effect the approach gradient has on the magnitude of the second error region. As the approach angle changes from $\theta_2 = -45^\circ$ to 0° , there is a definite trend for the second error region to increase. For the majority of departure gradient traces, the second error region peaks when the approach gradient is equal to 0° , and then slowly decreases as the approach angle increases from $\theta_2 = 0^\circ$ to 45° . This result is explained using the following reasoning.

The inherent Situation Pre-empting of the CMAC+PID controller produces a second error region for all situation occurrences. This inherent Situation Pre-empting is a result of the weight smoothing and the time delay compensation, and thus the output for a situation will be produced at roughly the same time interval prior to the situation occurrence, for situations of the same step size.

This means that for different approach gradients, the CMAC+PID output will start to move in the direction of the situation at a fixed time of n sampling periods before the situation occurs, regardless of the size and direction of the approach gradient. Figure 9.9 illustrates this concept.

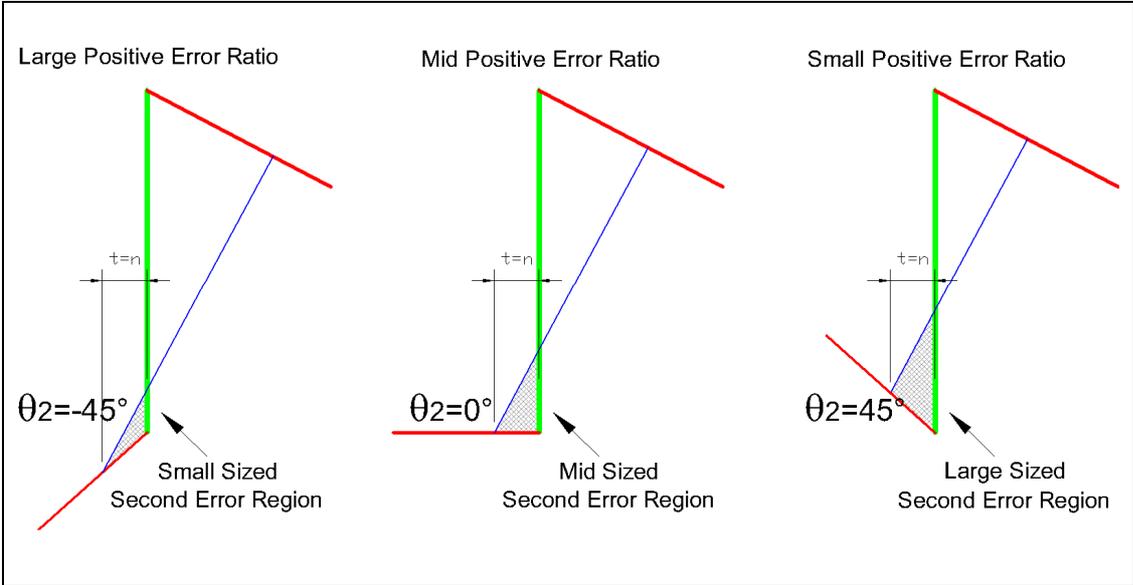


Figure 9.9 – Theoretical Actual System Response Due to Inherent Situation Pre-empting for Varying Approach Gradients

Figure 9.9 illustrates the effect that the inherent Situation Pre-empting has on the actuator output of the system. The actuator output is represented as a fixed velocity in blue, starting at a time of n sampling periods before the situation occurs, regardless of the situation approach gradient. With this value of n being a product of the weight smoothing and time delay compensation incorporated in the CMAC+PID controller, the size of the first and second error regions are directly affected by the approach gradient.

Starting with an approach gradient of $\theta_2 = -45^\circ$ in Figure 9.9, the actual actuator response starts to move in the direction of the situation n sampling periods prior to the situation occurrence, creating a small second error region and a large first error region. This creates a large positive Error Ratio. As the approach gradient moves from $\theta_2 = -45^\circ$ towards $\theta_2 = 45^\circ$, the size of the second error region increases in size. As it does, it changes the size of the first error region as well, decreasing it gradually and moving the Error Ratio towards zero.

The increase in area of the second error region as the approach gradient moves from $\theta_2 = -45^\circ$ to $\theta_2 = 0^\circ$ accounts for the increase in second error trend noticed in Figure 9.8 between $\theta_2 = -45^\circ$ to $\theta_2 = 0^\circ$.

Figure 9.9 also illustrates that the second error region continues to grow as the approach gradient moves from $\theta_2 = 0^\circ$ to $\theta_2 = 45^\circ$. The results in Figure 9.8 however show that this is not true. A slight decrease in the size of the second error region is noticed between $\theta_2 = 0^\circ$ to $\theta_2 = 45^\circ$ as opposed to the increase in the area of the second error region in Figure 9.9.

Upon close inspection of the actuator response for changing approach gradients, a common pattern in the actual response trajectory was noticed. For positive approach gradients, the inherent Situation Pre-empting of the system produces an output to move the actuator in the direction of the situation at n sampling periods prior to the situation occurrence. For negative approach gradients, the actuator is already moving in the direction of the situation, and thus the actuator only needs to start moving with a greater velocity.

For positive approach gradients however, the actuator velocity must change direction. The CMAC+PID controller output will produce a signal to drive the actuator in the direction of the situation at n sampling periods prior to the situation occurrence. When this happens, the actuator must first decrease its velocity in the current direction, stop, and then start moving in the opposite direction at its maximum velocity. This physical change in direction depends on the physical responsiveness of the system, but accounts for the decrease in the size of the second

error region as the approach gradient is increased from $\theta_2 = 0^\circ$ to $\theta_2 = 45^\circ$. This concept is illustrated in Figure 9.10.

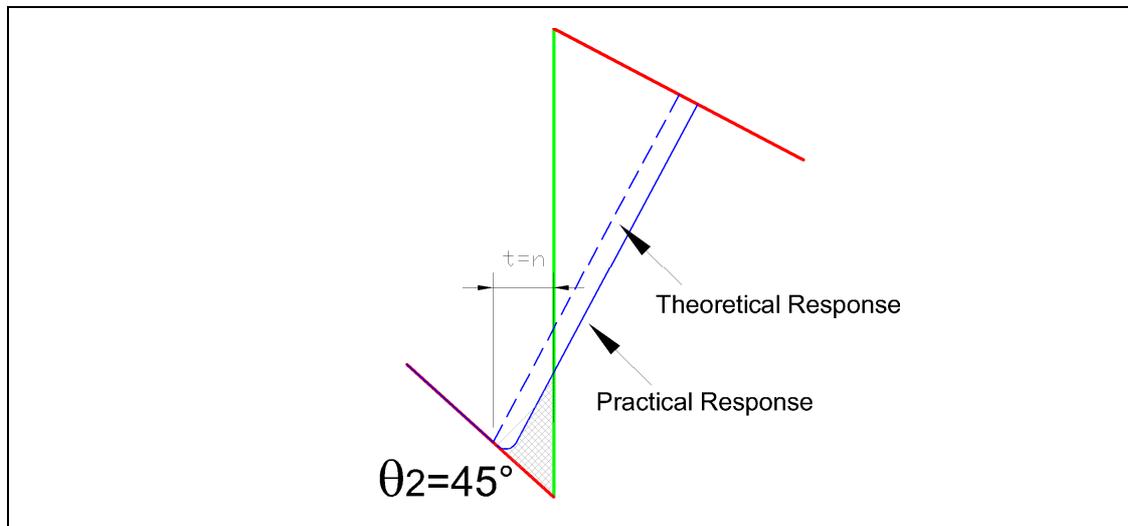


Figure 9.10 – Practical Actual System Response for Positive Approach Gradients

Figure 9.10 illustrates why the second error region in Figure 9.8 decreases from $\theta_2 = 0^\circ$ to $\theta_2 = 45^\circ$. For a system that responds infinitely fast, the trend illustrated in Figure 9.9 would be true. However for the air-muscle actuator used, the time required to change the direction of motion causes the second error region to be smaller than anticipated in Figure 9.9. The greater the positive approach gradient, the greater the deceleration time needed before the actuator motion can stop and change direction. This accounts for the decrease in the size of the second error region for positive approach gradients.

The reasoning above finally explains why Figure 9.6 appears the way it does. For negative approach gradients, the Error Ratio decreases directly as illustrated in Figure 9.9. However for positive approach gradients, the time taken for the actuator to reverse its direction causes the Error Ratio to have either little or the opposite effect compared to negative approach gradients.

9.2.2 ERROR GAIN VS APPROACH AND DEPARTURE GRADIENT RELATIONSHIPS

The Error Gain relationship corresponding to Figure 9.5 is illustrated in Figure 9.11.

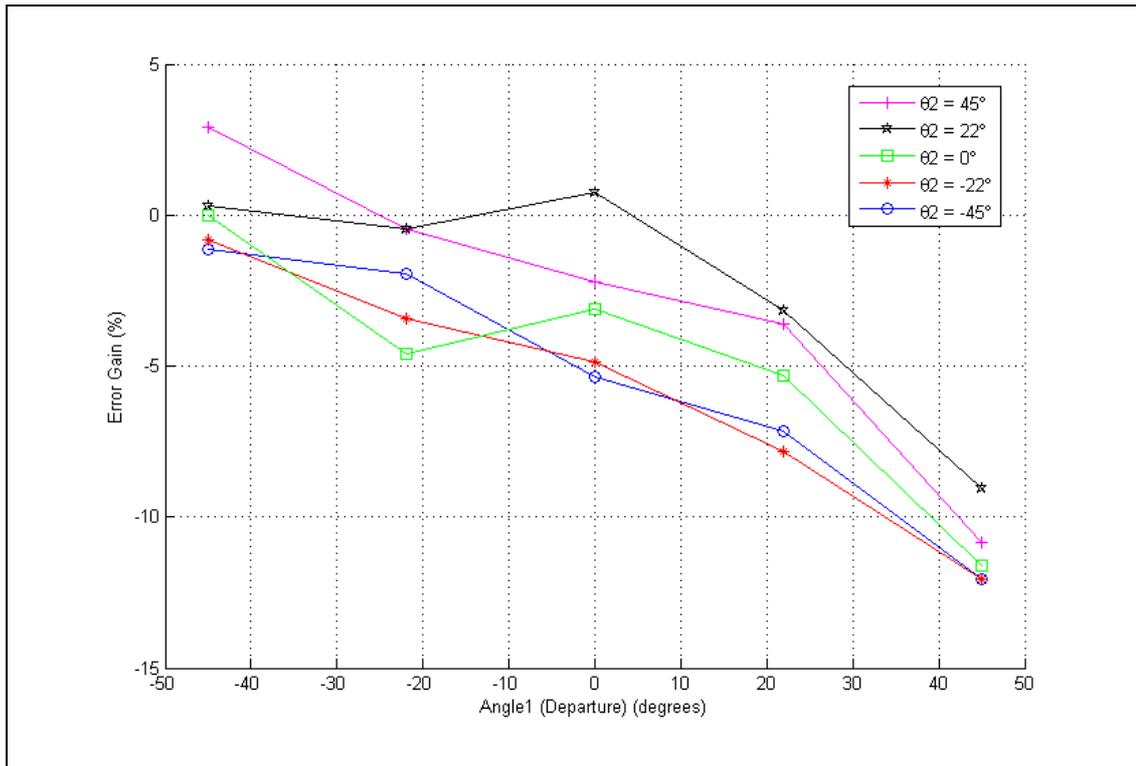


Figure 9.11 – Error Gain vs. Departure Gradient for Various Approach Gradients (Test 73 to Test 122)

Figure 9.11 illustrates a common trend for all approach gradients (θ_2): as the departure gradient increases, the Error Gain decreases, increasing the error reducing effect of the EMGC. This trend confirms that the Error Ratio is a useful predictor in how effective the EMGC will be in reducing a situation's Total Error. Figure 9.5 illustrated that as the departure gradient increased, so too did the Error Ratio. Figure 9.11 shows that as the departure gradient increases, the error minimising effect from the EMGC is also increased, producing greater negative Error Gains. Thus as the departure gradient increases from negative to positive, the Error Ratio of the situation is increased, producing a larger negative Error Gain by the EMGC.

Figure 9.11 also shows that the Error Gain is influenced less by the approach gradient than it is by the departure gradient. A clear indication exists for the negative Error Gain to increase as the departure gradient also increases in the positive direction, though the overlapping of the traces indicates that there is a less coherent relationship between the approach gradient and the Error Gain. This relationship is better illustrated in Figure 9.12.

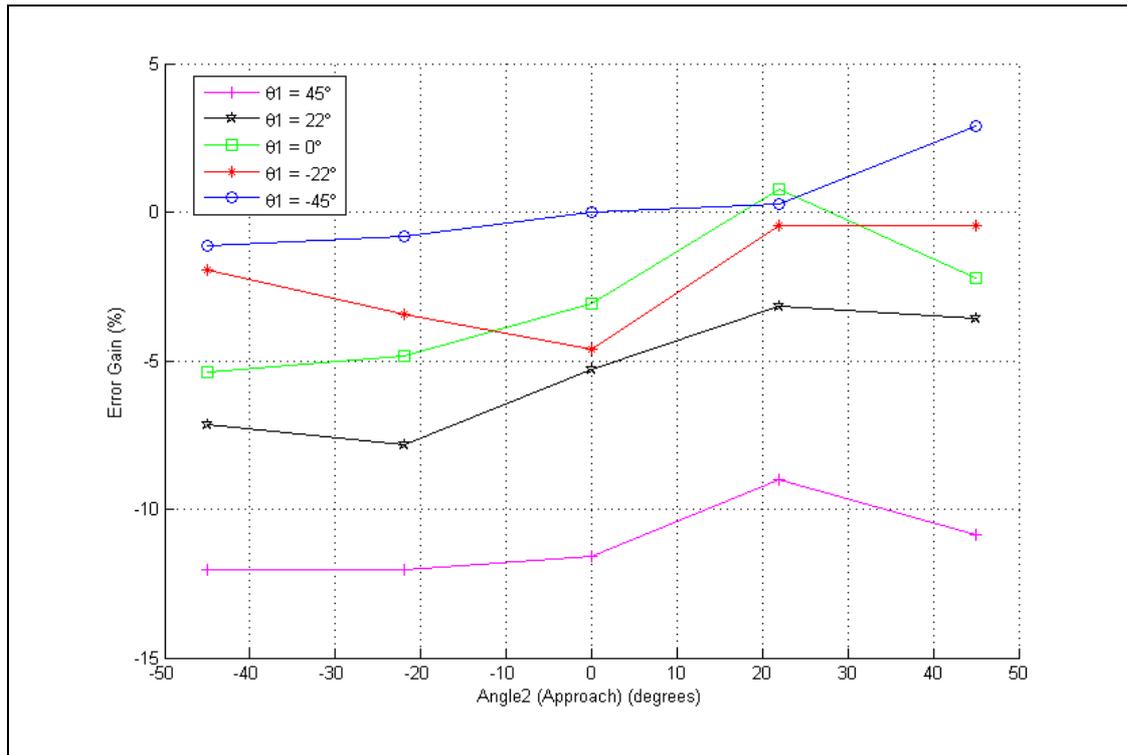


Figure 9.12 – Error Gain vs. Approach Gradient for Various Departure Gradients (Test 73 to Test 122)

Unlike the departure gradient vs. Error Gain relationship in Figure 9.11, a strong relationship does not exist between the approach gradient and the Error Gain in Figure 9.12. For example, regardless of the approach gradient for $\theta_1 = 45^\circ$, the Error Gain remains relatively consistent, with possibly a very slight overall negative Error Gain as the approach gradient increases. The increasing negative Error Gain with increase in approach gradient relationship is much more subtle than the Error Gain vs. departure gradient relationship.

The relationship between the Error Gain and the Error Ratio for Test 73 to Test 122 will now be discussed.

9.2.3 ERROR RATIO VS ERROR GAIN RELATIONSHIP

The final step in presenting the results for Tests 73 to 122 involves discussing the relationship between the initial Error Ratio and the Error Gain for each of the tests performed. The Error Gain vs. Error Ratio plots have been split into the following two figures, Figure 9.13 and Figure 9.14, representing either the approach gradient or departure gradient trends respectively.

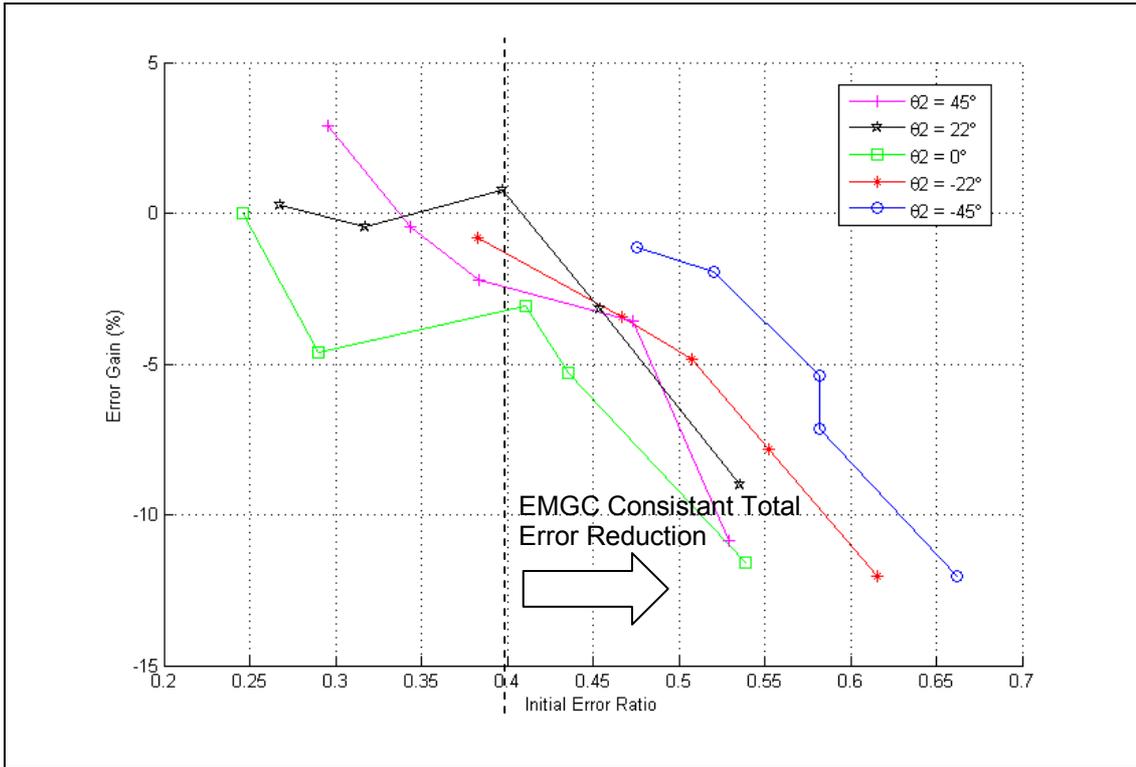


Figure 9.13 – Error Gain vs. Error Ratio for Various Approach Gradients (Test 73 to 122)

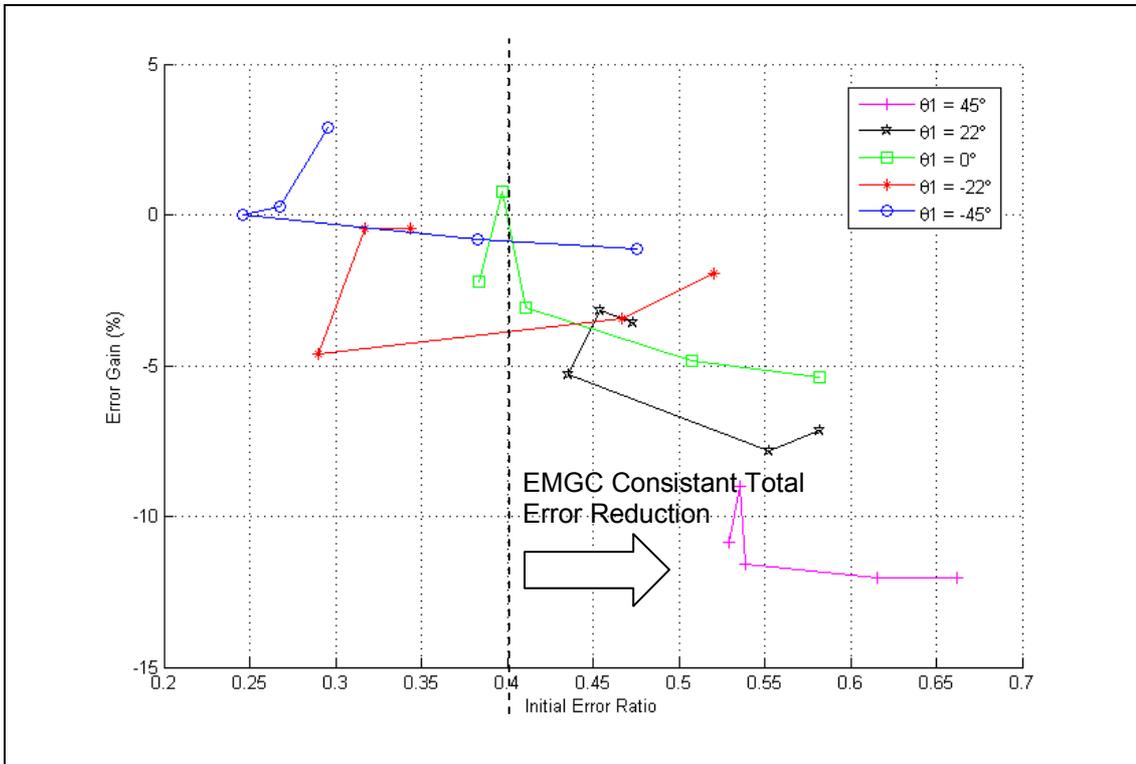


Figure 9.14 – Error Gain vs. Error Ratio for Various Departure Gradients (Test 73 to 122)

Figure 9.13 and Figure 9.14 illustrate the Error Ratio vs. Error Gain summary for Tests 73 to 122. Negative Error Gains mean that the EMGC has successfully reduced

a situation's Total Error. Thus for all but four tests (those lying on or above the 0% Error Gain line), the EMGC successfully reduced a situation's Total Error.

Comparing Figure 9.13 to Figure 9.14, there is a definite difference between the trends that the approach and departure gradients have on both the Error Gain and the Error Ratio. In Figure 9.13, there is a relatively linear-like overall trend between the Error Gain and the initial Error Ratio regardless of the approach gradient. This means that a variety of Error Gains exist for all approach gradients. A minimal relationship can be seen between the approach gradients and the initial Error Ratio, with larger initial Error Ratios existing for negative approach gradients as opposed to positive approach gradients. Thus the approach gradient is not useful for predicting the Error Gain of a situation, though it is somewhat useful for predicting the size of the initial Error Ratio with respect to other approach gradients.

The departure gradient allows a far greater approximation to be made about both the Error Gain and the initial Error Ratio. The most obvious relationship in Figure 9.14 is that the Error Gain increases negatively as the departure gradient moves from $\theta_1 = -45^\circ$ to $\theta_1 = 45^\circ$. This relationship is strong, and provides a useful means of approximating how beneficial the EMGC will be when handling a situation with a given departure gradient.

It can also be noticed that generally larger positive initial Error Ratios exist for larger positive departure gradients, regardless of what the approach gradient of a situation will be. The left end shape of the traces seen in Figure 9.14, some curving back around for example, are the result of the shape of the traces in Figure 9.6, producing both decreasing and increasing Error Ratios for a given departure gradient.

Without taking the approach and departure gradients into account, a strong relationship exists between the initial Error Ratio and the Error Gain. The EMGC consistently produces negative Error Gains when the initial Error Ratio is above 0.4. This threshold value was also determined empirically by the test data presented in Section 8.2.3, and here too produces only negative Error Gains for initial Error Ratios greater than 0.4.

This trend, with larger initial Error Ratios producing negatively larger Error Gains, confirms the reasoning discussed in Section 8.2.3, where the EMGC is more effective dealing with situations with large initial Error Ratios. Thus regardless of a situation's approach and departure gradient, the EMGC provides reliable error reducing performance for situations with initial Error Ratios greater than 0.4.

While Figure 9.14 illustrates that almost all departure gradients greater than and including $\theta_1 = -22^\circ$ result in a negative Error Gain, using the departure gradient solely as a metric to decide whether or not the EMGC is suitable for a situation would not produce an accurate prediction of EMGC effectiveness. The results in Section 8.2.3 were obtained from tests using only square wave desired trajectories, and illustrated that both positive and negative Error Gains exist for a single approach and departure gradient (both at 0°) with varying situation step sizes. Figure 9.14 also illustrates that as the departure gradient moves from $\theta_1 = -45^\circ$ to $\theta_1 = 45^\circ$, the initial Error Ratio is generally increased. Thus from the tests in this chapter and those discussed in Section 8.2.3, regardless of the departure gradient, the initial Error Ratio provides more effective information as to how beneficial the EMGC will be in reducing a situation's Total Error.

Thus the initial Error Ratio threshold of 0.4 can be used by the EMGC to discriminate between situations that should or should not have an EMG overlay applied to the desired trajectory situation. It is possible that some situations may benefit by use of the EMGC with initial Error Ratios below 0.4, but this is not guaranteed as it depends on a situation's approach and departure gradient. However if such a discrimination tool is used, the situations that would benefit by use of the EMGC which are excluded from EMGC use, would only have their Total Errors decreased slightly. Figure 9.13 and Figure 9.14 show that for situations with Error Ratios below 0.4, the maximum negative Error Gain was slightly less than 5%. Thus the benefit of using the EMGC on these situations would be only minimal.

The results in Figure 9.13 and Figure 9.14 also suggest that the EMGC could be always used for all situations regardless of a situation's initial Error Ratio. The reason for this is that the number of situations and their magnitude for situations with Error Ratios below 0.4, are weighted greater on the negative Error Gain side. This

means that for a random number of situations, situations with Error Ratios below 0.4 could produce either negative or positive Error Gains. However, the Total Error gains for these situations with Error Gains less than 0.4 would in effect cancel out the negative and positive Error Gains overall, producing a final trajectory with no benefit from using the EMGC. However, for situations with Error Ratios greater than 0.4, the EMGC will reduce a situation's Total Error. Thus the overall effect of the EMGC with no discrimination would be a trajectory with an overall reduced Total Error.

9.3 CONCLUSIONS

The purpose of the tests in this chapter was to prove that the EMGC algorithm was working in practice for a range of approach and departure gradients for situations produced from the joining of concatenated trajectories.

In conclusion of the tests undertaken throughout this chapter, the following points can be made for trajectories with various approach and departure gradients from -45° to 45° .

1. As the departure gradient increases from -45° to 45° , so too does the initial Error Ratio. This trend occurs regardless of the approach gradient, and the relationship between the initial Error Ratio and departure gradient is approximately linear. Increasing the departure gradient directly increases the magnitude of the first error region, and by Equation 9.1, causes an increase in the Error Ratio.
2. The approach gradient does not have a linear relationship with the initial Error Ratio of a situation, as the relationship changes depending on whether the approach gradient is positive or negative. For negative approach gradients, the greater the magnitude of the angle, the greater the initial Error Ratio. This is due to both the decrease in second error region magnitude and the increase in first error region magnitude, producing larger initial Error Ratios as the approach gradient moves from 0° to -45° .
3. For positive approach gradients, the time for the actuator to decelerate and change its direction of motion, means the second error region magnitude is reduced as the approach angle increases (for most departure gradients). Not only does this effect the size of the second error region, but the size of the first error region is effected too. Thus as the approach gradient moves from 0° to 45° , the

initial Error Ratio will vary only slightly in either the positive or negative direction.

4. As the departure gradient increases from -45° to 45° , there is a general increasing trend for the Error Gain to increase in negative magnitude. The close relationship between the initial Error Ratio and the departure gradient explains this trend.
5. The approach gradient has little effect on the Error Gain, with a subtle positive increase in the Error Gain as the approach gradient increases from 0° to 45° . This relationship is much weaker and far less obvious than the relationship between the departure gradient and the Error Gain.
6. A general trend can be seen between the initial Error Ratio and the Error Gain. As the initial Error Ratio increases positively, the Error Gain increases negatively. For all tests with initial Error Ratios greater than 0.4, a negative Error Gain was produced, resulting in a reduced Total Error with use of the EMGC. This initial Error Ratio threshold of 0.4 corresponds to the same threshold value obtained for the square wave tests in Section 8.2.3.
7. For the range of tests performed in this chapter, 4 out of 25 tests resulted in positive Error Gains, and 9 out of 25 tests had initial Error Ratios below 0.4. For the 9 tests with initial Error Ratios below 0.4, the Total Error reduction of those tests with negative Error Gains cancels out the gain in Total Error in the tests with positive Error Gains. Thus the results suggest that using the EMGC over a range of situations with random approach and departure gradients would have an overall Total Error reducing effect on the overall trajectory.
8. Using the initial ratio threshold of 0.4 as a situation discriminator value would allow the EMGC to only apply EMGs to situations with initial Error Ratios greater than 0.4. This would result in either no Total Error reduction or a reduced Total Error for all situations throughout a trajectory, resulting in no situations having an increased Total Error with use of the EMGC.

This finalises the testing performed on the EMGC with CMAC+PID controller. Additional tests, Test 37 to Test 72, were carried out on a variety of approach and departure gradients, using a 0° approach gradient with various departure gradients, and a 0° departure gradient with various approach gradients. Larger situation magnitudes were used with all tests resulting in initial Error Ratios greater than 0.4. These results are plotted and discussed in Appendix H for further interest.

10.0 CONCLUSIONS

The previous chapters have discussed the development of an adaptive controller, designed to minimise the error associated with the joining of separate trajectory profiles. This controller was developed to be suitable for use in learning and mimicking deafblind hand sign trajectories. The controller was shown to be able to adequately reduce a situation's Total Error on real hardware, when presented with a simulated series of discontinuous trajectories whereby the discontinuity exceeds the physical velocity limitations of the actuator. The thesis challenge stated in Section 2.10 has been met.

Thesis Challenge

To develop an adaptive method of training CMACs to control situations where the command trajectory exceeds the capabilities of the actuator, such that the combined sum of error magnitudes before and after a situation is minimised.

10.1 THESIS CONCLUSIONS

The Thesis Challenge was met by satisfying the sub-challenges outlined at the end of the Literature Review in Section 2.10.

A Cerebellar Model Articulation Controller (CMAC) (in the CMAC+PID controller) was implemented on a real-time system to adaptively control an air-muscle, which is a non-linear actuator exhibiting hysteresis. Providing the CMAC with enough input vector parameters allowed the CMAC to produce different outputs to control the air-muscle depending on the air-muscle's desired position, desired velocity, actual position and time.

All CMACs used were provided with a dynamic memory handling system, using Binary Search Trees to store the data dynamically over the previous static method of hashing. The binary search trees were created using random keys, producing close to balanced trees initially, and could be balanced offline for future search and building

time reductions. Using binary search trees provides a clean and dynamic way to store CMAC weight data, with search and build speeds comparing to hash table performance for small to medium sized CMAC weight tables.

A CMAC was successfully combined with a PID controller in the main control loop, producing a feedforward / feedback controller. The PID controller was added to the CMAC to handle newly encountered states so that an output was always produced in the required direction. The PID controller also helped the CMAC achieve quicker convergence, as it moves the system in the required direction while the CMAC output adapts to produce an error minimised output control signal. For learnt trajectories, the trained CMAC eventually eliminated the majority of the error between the desired and actual system response, and thus reduced the PID output close to zero. Systems were also put in place to allow the CMAC and PID controller to work seamlessly together, preventing residual PID error drift, etc.

While the CMAC+PID controller minimised the error for a particular sampling period, and provided a slight degree of Situation Pre-empting, an additional system was added into the CMAC training loop to also reduce the error associated with situations over time. The Error Minimising Gradient Controller (EMGC) was added to the training loop to detect situations, observe and record system data before and after a situation occurrence, and then alter the CMAC training signal to reduce the size of the error magnitude summation caused by the situation. The EMGC was able to evenly distribute and reduce a situation's Total Error equally before and after a situation. The benefit this has in terms of deafblind trajectory concatenation is that communicated information will be further preserved over CMAC control alone.

The EMGC was tested over a variety of situation magnitudes and on situations with varying approach and departure gradients. Conclusions were drawn as to when the EMGC produces a reduction in a situation's Total Error and when it does not, and why the EMGC either reduces or increases a situation's Total Error. An initial Error Ratio value threshold was discovered when using a fixed EMG Offset percentage, to be the lowest initial Error Ratio a situation can have if the EMGC is to successfully reduce a situation's Total Error. This threshold provided a switch for the EMGC to turn on only when the EMGC was able to guarantee no degradation in performance.

Testing of the EMGC was performed in such a way as to simulate the concatenation of random gesture trajectories to be able to be performed on an independent array of non-linear actuators, making up a possible deafblind fingerspelling hand.

Using the CMAC as the adaptive controller centrepiece of the entire control system provided a means for mimicking pre-sampled trajectories on a non-linear actuator. For systems where repetition of a trajectory can be reproduced numerous times, the CMAC has the ability to learn, adapt and reproduce the desired trajectory. In the case of learning deafblind fingerspelling gestures, the learning-from-repetition approach will work, because fingerspelling gestures can be pre-captured, learnt and then reproduced when needed to a high level of controlled accuracy.

The EMGC was tested against a CMAC+PID controller for benchmarking purposes. The CMAC+PID controller was able to learn efficiently and converge stably, producing an excellent trajectory imitation of the desired trajectory on the chosen actuator, providing a strong benchmark to test the EMGC against.

The integrated EMGC with CMAC+PID controller was implemented on a real-time operating system, interfacing to custom built pneumatic hardware. Using Xenomai with Linux on a standard desktop PC provided a solid, powerful and stable system to test the adaptive controller on, with real-time interfacing capabilities to both the external hardware and a custom developed GUI. This allowed intensive real-time monitoring of all system parameters to be recorded and presented both graphically and numerically.

The EMGC is a method of training CMACs to provide an additional level of intelligent error minimisation associated with situations. While the inherent level of Situation Pre-empting can already start to move the system output in the direction of a situation before the situation occurs, the addition of an EMGC to a CMAC+PID controller allows the error associated with situations to not only be reduced, but also minimised to a level where the error before and after the situation are equalised.

10.2 FUTURE WORK

A few improvements to the EMGC model would increase the robustness and efficacy of the controller in practical applications. Such improvements include the following.

1. Obtaining trajectory data from a data glove and using this data to train an array of EMGCs with CMAC+PID controllers for controlling an array of air-muscles connected to a robotic humanoid hand, to reproduce deafblind fingerspelling gestures. This was the original idea from which the work in this thesis developed, and thus applying the developed controller to this purpose would be the most obvious 'next step' in the project's evolution.

2. The EMG offset percentage of 50% was based on a single test. Increasing the EMG by a percentage increases the aggressiveness of the EMGC output. This could be greatly improved by first using an EMG with an offset percentage of 0% (meaning the EMG gradient would be equal to the maximum gradient of the actuator's response) and then once the Error Ratio is equal to zero, the EMG offset percentage could be increased gradually until the Total Error is further minimised. Thus the Error Ratio would converge first, and then the EMG offset percentage increased to an optimal level, further reducing a situation's Total Error.

3. A more robust way to classify situations would be beneficial. The EMGC model in this thesis samples two vector parameters and uses these to classify a situation so that the next time it occurs, the existing data for that situation can be further trained. An improved classification method would increase the robustness of the EMGC model by classifying similar situations, thus allowing better generalisation to occur, and separating situations which are different yet map to the same classification vectors used here.

4. Eliminating the EMGC 'delay' providing the horizon time. Training of the CMAC+PID CMAC can be done after a situation has occurred without using the Delay for the current system. Otherwise the delay can be used to a greater effect with the EMGC producing a more universal output to the CMAC+PID controller, not only being used in the training loop of the CMAC+PID CMAC but also as command

input vectors to the CMAC+PID controller. This would allow the command input vectors to change if a situation was approaching, providing an immediate change in the system output, rather than only changing the CMAC training signal the next time the situation occurs.

10.3 CONTRIBUTIONS OF THE THESIS

The main contributions of the thesis include the following.

1. An improved method of training CMACs was developed, reducing the error in terms of both position and time for situations created when the desired trajectory velocity requirements exceeded that of the driven actuator's capabilities. The Error Minimising Gradient Controller (EMGC) is able to replace desired trajectory situations with EMGs in the CMAC's training signal, and thus adaptively minimise the sum of the magnitudes of the position error over time.

2. The EMGC model was tested in real-time on hardware connected to a non-linear actuator for a variety of situations. The test results illustrated that depending on the inherent level of Situation Pre-empting achieved by the base CMAC controller, the EMGC can either reduce or increase a situation's Total Error. From the tests, metrics were obtained to allow the EMGC to decide whether a situation's Total Error was able to be reduced by use of the EMGC, allowing the EMGC to guarantee no degradation in CMAC controller performance.

3. A dynamic and clean method was devised for storing data in a CMAC, using binary search trees. This method was compared to the commonly used CMAC memory storage method of hashing, with test results illustrating that BSTs are a suitably fast alternative to hashing for small to medium sized CMAC weight tables.

In summary, the addition of an EMGC to a CMAC produces an improved approximation of reproduced motion trajectories when situations occur, not only minimising position error for a single sampling instance, but also over time for periodic signals.

REFERENCES

- 5DT. (2005). 5DT Data Glove 14 Ultra, Available from: <http://www.5dt.com/products/pdataglove14.html>, Accessed: 7/05/06
- Abdelhameed, M. M., Pinson, U., et al. (2002). Adaptive Learning Algorithm for Cerebellar Model Articulation Controller. *Mechatronics*, Vol. 12, pp.859-873.
- Albus, J. S. (1975a). Data Storage in the Cerebellar Model Articulation Controller (CMAC). *Transactions of the ASME: Journal of Dynamic Systems, Measurement, and Control*, pp.228-233. September
- Albus, J. S. (1975b). A new approach to manipulator control: The cerebellar model articulation controller (CMAC). *Transactions of the ASME: Journal of Dynamic Systems, Measurement, and Control*, pp.220-227. September
- Albus, J. S. (1981). *Brains, Behavior and Robotics*, Byte Publications, 0-07-000975-9, Massachusetts.
- Albus, J. S. and Meystel, A. M. (2001). *Engineering of Mind: An Introduction to the Science of Intelligent Systems*, John Wiley & Sons, Inc., New York.
- Alexandridis, A. P., Siettos, C. I., et al. (2002). Modelling of nonlinear process dynamics using Kohonen's neural networks, fuzzy systems and Chebyshev series. *Computers and Chemical Engineering*, Vol. 26 (4-5), pp.479-486.
- Allen, J. M., Asselin, P. K., et al. (2003). American Sign Language finger spelling recognition system. *IEEE 29th Annual Bioengineering Conference*, pp.285-286, 22-23 March.
- Almeida, P. E. M. and Simoes, M. G. (2003). Parametric CMAC Networks: Fundamentals and Applications of a Fast Convergence Neural Structure. *IEEE Transactions on Industry Applications*, Vol. 39 (5). Sept/Oct
- Almeida, P. E. M. and Simoes, M. G. (2005). Neural Optimal Control of PEM Fuel Cells with Parametric CMAC Networks. *IEEE Transactions on Industry Applications*.
- Animazoo. (2004a). The Future of Motion Capture FAQ's - How do the various motion capture technologies differ?, Available from: <http://www.animazoo.com/products/faqs.htm#mocaptech>, Accessed: 7/05/06
- Animazoo. (2004b). GypsyGyro-18 Technical Specs, Available from: http://www.animazoo.com/products/gypsyGyro_techSpecs.htm, Accessed: 7/05/06
- Arena, P., Fortuna, L., et al. (1998). *Neural Networks in Multidimensional Domains - Fundamentals and New Trends in Modelling and Control*, Springer-Verlag London Ltd, London.
- Astrom, K. J. (2002). Control System Design (Lecture Notes). Department of Mechanical and Environmental Engineering, University of California, Santa Barbara.
- Banerjee, S. and Majumdar, D. D. (2000). Shape matching in multimodal medical images using point landmarks with Hopfield net. *Neurocomputing*, Vol. 30 (1-4), pp.103-116.
- Beccai, L. (2001). Cyberhand: Development of a Cybernetic Hand Prosthesis, Available from: <http://www.cyberhand.org>, Accessed: 23/05/05

- Bekey, G. A., Tomovic, R., et al. (1990). Control Architecture for the Belgrade/USC Hand. *Dexterous Robot Hands*, Springer-Verlag New York, Inc., New York, pp.136-149.
- Beltzer, A. I. and Sato, T. (2003). Neural classification of finite elements. *Computers and Structures*, Vol. 81 (24-25), pp.2331-2335.
- Bentley, J. and Sedgewick, R. (2001). Ternary Search Trees. *Dr. Dobb's Journal*, Available from: <http://www.ddj.com/184410528>, Accessed: 13/08/06
- Bernardin, K., Ogawara, K., et al. (2003). A Hidden Markov Model Based Sensor Fusion Approach for Recognizing Continuous Human Grasping Sequences. *IEEE 3rd International Conference on Humanoid Robots*.
- Biagiotti, L., Lotti, F., et al. (2005). Development of UB Hand 3: Early Results. *IEEE International Conference on Robotics and Automation (ICRA)*, Barcelona, Spain, April 18-22.
- Black, P. E. (2006). Dictionary of Algorithms and Data Structures. US National Institute of Standards and Technology (NIST), Available from: <http://www.nist.gov/dads/HTML/redblack.html>, Accessed: 28/10/06
- Bonsor, K. (2001). *Space Robots may Help Astronauts*. USA Today - Science.
- Brown, M., Harris, C. J., et al. (1993). The Interpolation Capability of the Binary CMAC. *Neural Networks*, Vol. 6, pp.429-440.
- Butterfab, J., Fischer, M., et al. (2004). Design and Experiences with DLR Hand II. *World Automation Congress, 10th International Symposium on Robotics with Applications*, Seville, Spain, June 28th - July 1st.
- Cannata, G. and Maggiali, M. (2005). An Embedded Tactile and Force Sensor for Robotic Manipulation and Grasping. *IEEE-RAS International Conference on Humanoid Robots*, 5-7 December.
- Cembrano, G., Wells, G., et al. (1997). Dynamic Control of a Robot Arm Using CMAC Neural Networks. *Control Engineering in Practice*, Vol. 5 (4), pp.485-492.
- Chen, C.-M., Hong, C.-M., et al. (2004). A pruning structure of self-organizing HCMAC neural network classifier. *IEEE International Joint Conference on Neural Networks*, pp.861-866, July 25-29.
- Collins, D. and Wyeth, G. (1999). Cerebellar Control of a Line Following Robot. *The Australian Conference on Robotics and Automation*, pp.74-79, Brisbane, March 30 - April 1.
- Cornforth, D. (2001). The Kernel Addition Training Algorithm: Faster Training for CMAC based Neural Networks. *Conference of Artificial Neural Networks and Expert Systems*, Otago.
- Dario, P. and Carrozza, M. C. (2000). Design and Experiments on a Novel Biomechatronic Hand. *Seventh International Symposium on Experimental Robotics*, Honolulu, Hawaii, December 10-13.
- DBI.org. (2005). Frequently Asked Questions about DeafBlindness, Available from: <http://www.deafblindinfo.org/FAQ.asp>, Accessed: 27/02/06
- DBNZ. (2006). Manual Alphabet for Deafblind, Available from: www.deafblind.org.nz, Accessed: 1/03/06
- DeafBlind_UK. (2005). Deafblind manual communication tips, Available from: <http://deafblind.org.uk/deafblindness/commtips.html>, Accessed: 27/02/06
- Dean, T., Basye, K., et al. (1992). Reinforcement Learning for Planning and Control. *Machine Learning Methods for Planning and Scheduling*, Minton, S., ed., Morgan Kaufmann, San Mateo, CA, pp.67-92.

- Diftler, M. A., Platt., R., et al. (2003). Evolution of the NASA/DARPA Robonaut Control System. *IEEE Conference on Robotics and Automation (ICRA)*, Taipei, Taiwan, May.
- Dipietro, L., Sabatini, A. M., et al. (2003). Evaluation of an instrumented glove for hand-movement acquisition. *Journal of Rehabilitation Research and Development*, Vol. 40 (2), pp.179-190. March/April
- Dorvlo, A., Jervase, J., et al. (2002). Solar radiation estimation using artificial neural networks. *Applied Energy*, Vol. 71 (4), pp.307-319.
- Dwinnel, W. (1998). *Alternative Neural Architectures*. PC AI.
- Ekvall, S. and Kragic, D. (2005). Grasp Recognition for Programming by Demonstration. *IEEE International Conference on Robotics and Automation*, Barcelona, Spain, April.
- Eldracher, M., Staller, A., et al. (1994). Function Approximation with Continuous-Valued Activation Functions in CMAC. Technical University Munich.
- Festo. (2005). *Pressure and Vacuum Sensors, Type SDE1 (Produce Short Information)*. Festo Corporation.
- Fonseca, A. M., Biscaya, J. L., et al. (2006). Geographical classification of crude oils by Kohonen self-organizing maps. *Analytica Chimica Acta*, Vol. 556 (2), pp.374-382.
- Fukaya, N., Toyama, S., et al. (2000). Design of the TUAT/Karlsruhe Humanoid Hand. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Takamatsu, Japan.
- Goza, S. M., Ambrose, R. O., et al. (2004). Telepresence control of the NASA/DARPA robonaut on a mobility platform. *SIGCHI Conference on Human factors in computing systems*, pp.623-629, Vienna, Austria.
- Harmon, F. G., Frank, A. A., et al. (2005). The Control of a Parallel Hybrid-Electric Propulsion System for a Small Unmanned Aerial Vehicle using a CMAC Neural Network. *Neural Networks*, Vol. 18, pp.772-780.
- Hino, T. and Maeno, T. (2004). Development of a Miniature Robot Finger with a Variable Stiffness Mechanism using Shape Memory Alloy. *International Symposium on Robotics and Automation*, Queretaro, Mexico, August 25 - 27.
- Hontoria, L., Aguilera, J., et al. (2005). An Application of the Multilayer Perceptron - Solar Radiation Maps in Spain. *Solar Energy*, Vol. 79 (5), pp.523-530.
- Hopfield, J. J. (1982). Neural Networks and Physical Systems with Emergent Collective Computational Abilities. *Proceedings of the National Academy of Sciences*, Vol. 79, pp.2554-2558.
- Hopfield, J. J. and Tank, D. W. (1985). Neural Computation of Decisions in Optimization Problems. *Biological Cybernetics*, Vol. 52, pp.141-152.
- Horvath, G. (2003). CMAC: Reconsidering an Old Neural Network. *The Intelligent Control Systems and Signal Processing (ICONS)*, pp.173-178, Faro, Portugal, April.
- Horvath, G. (2004). Kernel CMAC with Improved Capability. *The International Joint Conference on Neural Networks (IJCNN)*, pp.681-686, Budapest, Hungary.
- Horvath, G. (2006). Kernel CMAC: an Efficient Neural Network for Classification and Regression. *Acta Polytechnica Hungarica*, Vol. 3 (1), pp.5-20.
- Houk, J. C., Buckingham, J. T., et al. (1996). Models of the Cerebellum and Motor Learning. *Behavioral and Brain Sciences*, Vol. 19 (3), pp.368-383.
- Hsu, A. L., Tang, S.-L., et al. (2003). An Unsupervised Hierarchical Dynamic Self-Organising Approach to Cancer Class Discovery and Marker Gene

- Identification in Microarray Data. *Bioinformatics*, Vol. 19 (16), pp.2131-2140.
- Hsu, Y.-P., Hwang, K.-S., et al. (2000). A new CMAC neural network architecture and its ASIC realization. *The Asia and South Pacific Design Automation Conference*, pp.481-484, Yokohama, Japan, January 25-28.
- Hsu, Y.-P., Hwang, K.-S., et al. (2002). An Associative Architecture of CMAC for Mobile Robot Motion Control. *Journal of Information Science and Engineering*, Vol. 18, pp.145-161.
- Hu, H., Gao, X., et al. (2004). Calibrating Human Hand for Teleoperating the HIT/DLR Hand. *The 2004 IEEE International Conference on Robotics and Automation*, New Orleans, LA, April.
- Huang, K.-L., Hsieh, S.-C., et al. (1997). Cascade-CMAC neural network applications on the color scanner to printer calibration. *International Conference on Neural Networks*, pp.10-15, Houston, TX.
- Huang, T. (2006). Neural Network Modeling and Feedback Error Learning Control for Automotive Fuel-Injection Systems. University of Illinois, Chicago.
- IBRS. (1997). Promoting Literacy for the Blind in the 21st Century: Braille Facts. International Braille Research Center, Available from: <http://www.braille.org/#WHAT>, Accessed: 1/03/06
- Immersion. (2005). CyberGlove, Available from: http://www.virtex.com/3d/products/cyber_glove.php, Accessed: 7/07/05
- IntervalZero. (2008). Deterministic, hard real-time performance – plus the Windows® user experience, Available from: <http://www.intervalzero.com/rtx.htm>, Accessed: 02/01/08
- Islam, T. and Saha, H. (2005). Hysteresis compensation of a porous silicon relative humidity sensor using ANN technique. *Sensors and Actuators B*.
- Jacobs, R., Jordan, M., et al. (1990). Task decomposition through competition in a modular connectionist architecture: the 'what' and 'where' vision tasks., University of Massachusetts, Amherst.
- Jacobsen, S. C. (1986). Design of the Utah/Mit Dexterous Hand. *IEEE International Conference on Robotics and Automation*, pp.1520-1532, Los Alamitos, California, April, IEEE Computer Soc. Press.
- Jaffe, D. L. (1989). Dexter II - The Next Generation Mechanical Fingerspelling Hand for Deaf-Blind Persons. *The 12th Annual RESNA Conference*, New Orleans, June.
- Jaffe, D. L. (1994). Evolution of Mechanical Fingerspelling Hands for People who are Deaf-Blind. *Journal of Rehabilitation Research and Development*, Vol. 31 (3), pp.236-244. August
- Jan, J. C. and Hung, S.-L. (2001). High-order MS CMAC neural network. *IEEE Transactions on Neural Networks*, Vol. 12 (3), pp.598-603. May
- Jaycar. (2003). SS0901 Solenoid DataSheet. Jaycar Electronics, Available from: http://www.jaycar.com.au/products_uploaded/SS0901.pdf, Accessed: 7/04/08
- Jian, Q., Jun, M., et al. (2005). The retrieval of the medical tongue-coating images using fuzzy CMAC neural network. *IEEE International Conference on Industrial Technology*, pp.465-468.
- Kadous, M. W. (1995). Recognition of Australian Sign Language using Instrumented Gloves. The University of New South Wales, Sydney.
- Ker, J.-S., Kuo, Y.-H., et al. (1997). Hardware Implementation of CMAC Neural Network with Reduce Storage Requirement. *IEEE Transactions on Neural Networks*, Vol. 8 (6), pp.1545-1556. November

- Kessler, G. D., Hodges, L. F., et al. (1995). Evaluation of the CyberGlove as a whole-hand input device. *ACM Transactions on Computer-Human Interaction*, Vol. 2 (4), pp.263-283. December
- Kim, J.-S., Jang, W., et al. (1996). A Dynamic Gesture Recognition System for the Korean Sign Language (KSL). *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics*, Vol. 26 (2), pp.354-359. April
- Kohonen, T. (1990). The Self-Organizing Map. *Proceedings of IEEE*, Vol. 78, pp.1464-1480.
- Kriete, F. (2004). Sensor Gloves, Available from: <http://www.tufts.edu/programs/mma/emid/projectreportsS04/kriete.html>, Accessed: 6/06/06
- Laenger, C. J. and Peel, H. H. (1978). Further Development and Tests of an Artificial Hand for Communication with Deaf-Blind People. South-West Research Institute (SWRI).
- Lane, S. H., Handelman, D. A., et al. (1992). *Theory and Development of Higher-Order CMAC Neural Networks*. IEEE Control Systems Magazine, pp.23-30.
- Lee, H.-M., Chen, C.-M., et al. (2003). A Self-Organizing HCMAC Neural-Network Classifier. *IEEE Transactions on Neural Networks*, Vol. 14 (1), pp.15-27. January
- Lee, K. and Shimoyama, I. (1999). Skeletal Framework Artificial Hand Actuated by Pneumatic Artificial Muscles. *IEEE International Conference on Robotics and Automation*, pp.926-931, Detroit, Michigan, May.
- Lee, S., Kim, J., et al. (2005). Newton-Type Algorithms for Dynamics-Based Robot Movement Optimization. *IEEE Transactions on Robotics*, Vol. 21 (4), pp.657-667. August
- Li, Z. (2003). Using Robotic Hand Technology for the Rehabilitation of Recovering Stroke Patients with Loss of Hand Power. Masters Thesis, North Carolina State University, Carolina.
- Liang, R.-H. and Ouhyoung, M. (1995). A Real-time Continuous Alphabetic Sign Language to Speech Conversion VR System. *Computer Graphics Forum*, Vol. 14 (3), pp.C67-C76.
- Lim, B., Ra, S., et al. (2005). Movement Primitives, Principal Component Analysis and the Efficient Generation of Natural Motions. *The International Conference on Robotics and Automation (ICRA2005)*, pp.4641-4646, Barcelona, Spain, April 18-22.
- Lin, C.-S. and Li, C.-K. (1996). A low-dimensional-CMAC-based neural network. *IEEE International Conference on Systems, Man and Cybernetics*, pp.1297-1302, Beijing, China, October 14-17.
- Lin, Y. and Song, S.-M. (1992). A CMAC Neural-Network-Based Algorithm for the Kinematic Control of a Walking Machine. *Engineering Applications of Artificial Intelligence*, Vol. 5 (6), pp.539-551.
- Lisboa, P. G. J. (1992). *Neural Networks Current Applications*, Chapman & Hall, London.
- Liu, J. and Gader, P. (2002). Neural Networks with Enhanced outlier rejection ability for off-line handwritten word recognition. *Pattern Recognition*, Vol. 35 (10), pp.2061-2071.
- Martin, T. B., Ambrose, R. O., et al. (2004). Tactile Gloves for Autonomous Grasping with the NASA/DARPA Robonaut. *IEEE International Conference on Robotics and Automation*, pp.1713-1718, New Orleans, LA.

- McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, Vol. 5, pp.115-133.
- Mehrabian, A. (1972). *Nonverbal Communication*, Aldine Atherton, Chicago.
- Menozzi, A. B. and Chow, M.-Y. (1997). On the training of a multi-resolution CMAC neural network. *International Conference on Industrial Electronics, Control and Instrumentation*, pp.1130-1135, New Orleans, LA, USA, November, 9-14.
- Mese, E. (2003). A rotor position estimator for switched reluctance motors using CMAC. *Energy Conversion and Management*, Vol. 44, pp.1229-1245.
- Miller, W. T., Glanz, F., et al. (1990a). CMAC: An associative neural network alternative to backpropagation. *Proceedings of the IEEE*, Vol. 78 (10).
- Miller, W. T. and Glanz, F. H. (1996). The University of New Hampshire Implementation of the Cerebellar Model Arithmetic Computer - CMAC. Robotics Laboratory, University of New Hampshire, Durham, New Hampshire.
- Miller, W. T., III, Hewes, R. P., et al. (1990b). Real-time dynamic control of an industrial manipulator using a neural network-based learning controller. *IEEE Transactions on Robotics and Automation*, Vol. 6 (1), pp.1-9, 1042-296X.
- Moller, C. (2003). *Deafblindness: Living with sensory deprivation*. The Lancet, pp.s46-s47.
- Moody, J. (1989). Fast Learning in Multi-Resolution Hierarchies. *Advances in Neural Information Processing*, Touretsky, D., ed., Morgan Kaufmann, Los Altos, California, pp.29-39.
- Moody, J. and Darken, C. J. (1989). Fast Learning in networks of locally-tuned processing units. *Neural Computation*, Vol. 1, pp.281-294.
- Morita, H., Hashimoto, S., et al. (1991). *A Computer Music System that Follows a Human Conductor*. Computer, pp. 44-53.
- Murakami, K. and Taguchi, H. (1991). Gesture recognition using recurrent neural networks. *Conference on Human Factors in Computing Systems*, pp.237-242, New Orleans, Louisiana.
- NASA. (2004). Robonaut Hands, Available from: <http://robonaut.jsc.nasa.gov/hands.htm>, Accessed: 27/03/06
- Negnevitsky, M. (2002). *Artificial Intelligence: A Guide to Intelligent Systems*, Pearson Education Limited, Essex.
- Ogawara, K., Iba, S., et al. (2001). Acquiring hand-action models by attention point analysis. *IEEE International Conference on Robotics and Automation*, Seoul, Korea, 21-26 May.
- Ogawara, K., Iba, S., et al. (2000). Recognition of human behaviour using stereo vision and data gloves. *Seisan Kenkyu*, Vol. 52 (5), pp.225-230.
- Okada, T. (1982). Computer Control of Multi-Jointed Finger System for Precise Object Handling. *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-12 (3), pp.289-299. May/June
- O'Malley, D. (2005). Deprivation of Information, Available from: http://www.deafblindinternational.org/standard/publications_deprivation.html, Accessed: 27/02/06
- Pak-Cheung, E. (1991). An Improved Multi-Dimensional CMAC Neural Network: Receptive Field Function and Placement. PhD Dissertation, University of New Hampshire, New Hampshire.

- Parks, P. C. and Militzer, J. (1991). Improved allocation of weights for associative memory storage in learning control systems. *IFAC Design Methods of Control Systems*, pp.507-512, Zurich, Switzerland.
- Pavlovic, V. I., Sharma, R., et al. (1997). Visual Interpretation of Hand Gestures for Human-Computer Interaction: A Review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 19 (7), pp.677-694. July
- Pfaff, B. (2004). *An Introduction to Binary Search Trees and Balanced Trees*, Free Software Foundation, Inc., Boston, MA.
- Picton, P. (2000). *Neural Networks*, Palgrave, New York.
- Rosenblatt, F. (1958). The Perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, Vol. 65, pp.386-408.
- RTAI. (2008). RTAI - the RealTime Application Interface for Linux, Available from: <https://www.rtai.org/>, Accessed: 15/01/08
- Rumelhart, D. E. and McClelland, J. L. (1986). *Parallel Distributed Processing*, MIT Press, Cambridge, MA.
- Sabourin, C. and Bruneau, O. (2005). Robustness of the Dynamic Walk of a Biped Robot Subjected to Disturbing External Forces by using CMAC Neural Networks. *Robotics and Autonomous Systems*, Vol. 51, pp.81-99.
- Sadighi, M., Setayeshi, S., et al. (2002). PWR fuel management optimization using neural networks. *Annals of Nuclear Energy*, Vol. 29 (1), pp.41-51.
- Salisbury, K. S. and Roth, B. (1982). Kinematics and Force Analysis of Articulated Mechanical Hands. PhD Thesis, Stanford University, Stanford, CA.
- Sarle, W. (2002). comp.ai.neural-nets FAQ, Available from: <http://www.faqs.org/faqs/ai-faq/neural-nets/>, Accessed: 4/10/07
- Scarfe, P. (2004). Pneumatic Air-Muscle Controlled Robotic Hand. Curtin University of Technology, Perth.
- Scott, J. (1998). Communicating with a deafblind person in an emergency. *Accident and Emergency Nursing*, Vol. 6 (3), pp.164-166.
- Sedgewick, R. (1990). *Algorithms in C*, Addison-Wesley Publishing Company, Inc., USA.
- SENSE. (2005). How do people who are deafblind communicate?, Available from: <http://www.sense.org.uk/publications/allpubs/communication/C01.htm>, Accessed: 27/02/06
- Shadow. (2005a). Shadow Dexterous Hand. Shadow Robot Company Ltd., Available from: <http://www.shadow.org.uk/products/newhand.shtml>, Accessed: 2005-05-08
- Shadow. (2005b). Shadow Dexterous Hand. Shadow Robot Company Ltd., Available from: <http://www.shadow.org.uk/products/newhand.shtml>, Accessed: 5/08/05
- Shen, Z. and Guo, C. (2005). Study on fuzzified CMAC control for ship steering based on eligibility. *Proceedings of the 2005 American Control Conference*, pp.4345-4350, 0743-1619, Portland, OR, USA, June 8-10.
- Shen, Z., Guo, C., et al. (2004). Reinforcement learning control for ship steering based on general fuzzified CMAC. *5th Asian Control Conference.*, pp.1552-1557 Vol.3.
- Shen, Z., Guo, C., et al. (2005). General Fuzzified CMAC Based Model Reference Adaptive Control for Ship Steering. *IEEE International Symposium on Automation and Intelligent Control.*, pp.1257-1262.
- Shewchuk, J. and Dean, T. (1990). Toward learning time-varying functions with high input dimensionality. *5th IEEE International Symposium on Intelligent Control.*, pp.383-388.

- Shim, J. K. (2004). Hand and Finger Model. The Pennsylvania State University, Available from:
<http://www.personal.psu.edu/users/j/u/jus149/handfinger/handandfinermodel/handfingermodel.htm>, Accessed: 15/05/05
- Smith, R. L. (1998). Intelligent Motion Control with an Artificial Cerebellum. University of Auckland, Auckland.
- Southern, N. and Drescher, L. (2005). *Technology and the needs of Deafblind people*. International Congress Series, pp. 997-1001.
- SparkFun. (2005). Single Axis MEMs Gyroscope - ADXRS Series, Available from:
http://www.sparkfun.com/commerce/product_info.php?products_id=649#,
 Accessed: 6/06/06
- Sturman, D. J., Zeltzer, D., et al. (1994). A survey of glove-based input. *IEEE Computer Graphics and Applications*, Vol. 14 (1), pp.30-39. January
- Szabo, T. and Horvath, G. (1999). CMAC and its Extensions for Efficient System Modeling and Diagnosis. *International Journal of Applied Mathematics and Computer Science*, Vol. 9 (3), pp.571-598.
- Tarassenko, L. (1998). *A Guide to Neural Computing Applications*, Arnold Publishers, London.
- TELEO. (2004). BendSensor: How to use a bendsensor, Available from:
<http://www.makingthings.com/teleo/cookbook/bendsensor.htm#ideas>,
 Accessed: 6/06/06
- TenAsys. (2008). INtime RTOS for Windows, Available from:
<http://www.tenasys.com/products/intime.php>, Accessed: 02/01/08
- Tham, C. K. (1994). Modular On-Line Function Approximation for Scaling up Reinforcement Learning. PhD, University of Cambridge, Cambridge.
- Tham, C. K. (1995). Reinforcement Learning of Multiple Tasks Using a Hierarchical CMAC Architecture. *Robotics and Autonomous Systems*, Vol. 15, pp.247-274.
- Tolba, A. S. (1999). GloveSignature: A Virtual-Reality-Based System for Dynamic Signature Verification. *Digital Signal Processing*, Vol. 9 (4), pp.241-266. October
- TRI. (2003). National Deaf-Blind Child Count Summary. The National Technical Assistance Consortium.
- Tubiana, R. T. (1981). *Architecture and Functions of the Hand*, W.B. Saunders, Philadelphia.
- Tuffield, P. and Elias, H. (2003). The Shadow Robot Mimics Human Actions. *The Industrial Robot: An International Journal*, Vol. 30 (1), pp.56-60.
- Turner, M. L., Findley, R. P., et al. (2000). Development and Testing of a Telemanipulation System with Arm and Hand Motion. *ASME IMECE 2000 Symposium on Haptic Interfaces for Virtual Environments and Teleoperator Systems*.
- Venkataram, P., Ghosal, S., et al. (2002). Neural network based optimal routing algorithm for communication networks. *Neural Networks*, Vol. 15 (10), pp.1289-1298.
- Virtual Realities. (2006). P5 Glove, Available from:
<http://www.vrealities.com/P5.html>, Accessed: 7/04/07
- VPL_Research_Inc. (1992). *Data-Glove Model 2+ Operation Manual*, VPL Research Inc., CA.

- Wang, S. (2005). Classification with incomplete survey data: a Hopfield neural network approach. *Computers and Operations Research*, Vol. 32 (10), pp.2583-2594.
- Wang, S. and Jiang, Z. (2004). Valve Fault Detection and Diagnosis Based on CMAC Neural Networks. *Energy and Buildings*, Vol. 36, pp.599-610.
- Wang, Z.-Q., Schiano, J. L., et al. (1996). Hash-coding in CMAC neural networks. *IEEE International Conference on Neural Networks*, pp.1698-1703, Washington, DC, USA, June 3-6.
- Wong, M. L. D., Jack, L. B., et al. (2006). Modified self-organising map for automated novelty detection applied to vibration signal monitoring. *Mechanical Systems and Signal Processing*, Vol. 20 (3), pp.593-610.
- wxWidgets. (2008). wxWidgets Cross-Platform GUI Library, Available from: <http://www.wxwidgets.org/>, Accessed: 25/01/08
- Xenomai. (2008). Xenomai: Real-Time Framework for Linux, Available from: <http://www.xenomai.org/>, Accessed: 15/01/08
- Yamano, I. and Maeno, T. (2005). Five-Fingered Robot Hand using Ultrasonic Motors and Elastic Elements. *IEEE International Conference on Robotics and Automation*, Barcelona, Spain, April.
- Yan, H., Jiang, Y., et al. (2006). A multilayer perceptron based medical decision support system for heart disease diagnosis. *Expert Systems with Applications*, Vol. 30 (2), pp.272-281.
- You, S., Wang, T., et al. (2001). A low-cost internet-based telerobotics system for access to remote laboratories. *Artificial Intelligence in Engineering*, Vol. 15 (3), pp.265-279. July
- Zheng, Y., Luo, S., et al. (2006). Control Double Inverted Pendulum by Reinforcement Learning with Double CMAC Network. *18th International Conference on Pattern Recognition.*, pp.639-642.

“Every reasonable effort has been made to acknowledge the owners of copyright material. I would be pleased to hear from any copyright owner who has been omitted or incorrectly acknowledged.”

A.0 HUMANOID HANDS

Several attempts have been made by various institutes and companies to create a humanoid hand which as close as possible mimics all the functions of the human hand. To date, compromises have always been necessary in the design process. A humanoid hand containing the large number of degrees of freedom, with the capability to provide both strength and precision grasps at variable speeds, together with the integration of a high number of sensors for the measurement of force, position, velocity, temperature, etc., all built into the size and weight of a human hand and arm, has as yet to be built. Undertaking such a project requires an immense effort to be undertaken across various technological and even medical disciplines such as mechanics, electronic sensory feedback, control systems and biomechanics. Depending on the choice of actuation, pneumatics or hydraulics may also have to be integrated. Often large teams are therefore required to design and assemble all of the above systems with specialists in each of the above areas integrating their knowledge. For this reason, such projects are often costly and extremely complex.

To justify such an effort and the high costs that are associated with a humanoid hand's development, hands created to date have always leaned towards particular functions, weighing certain human replicated attributes over others. For example, for prosthetic purposes size and lightness is of utmost importance. Where high precision, sensory feedback and numerous degrees of freedom are the design preference, the weight and size are often far greater than that of a human's arm and hand. Thus the decision for integrating particular attributes over others depends on what the hand is being designed to achieve. However, since the human hand has proved to be a technologically impressive challenge to replicate itself, for this reason alone, many research groups have simply built a hand for the sake of the challenge it presents. Several different major designs of humanoid hands will now be discussed.

A.1 HUMANOID HAND USES

One purpose for designing and creating humanoid hands is for prosthesis. The prosthetic hand industry is one in which there is a very clear and defined purpose of why the design of a humanoid hand would be beneficial. However, several difficulties arise in the development of designing a prosthetic hand which are not

found in other robotic humanoid hands. Reasons include the extreme lightness and small area in which to house the actuation devices, together with nerve control interfacing.

One such project that is attempting to build the world's first biologically integrated prosthetic hand is the Cyberhand. The Cyberhand incorporates 4 fingers and a thumb, and is to be controlled by the amputee in a very natural way by processing the different neural signals coming from the patient's nervous system (Beccai, 2001). Thus the prosthesis will be felt by the amputee as the lost natural limb, since a natural sensory feedback will be delivered to him/her by means of the stimulation of nerves. This will in effect produce the 'life-like' perception of the natural hand. This project has covered research in the area of bio-mechanical nerve integration, amongst the mechanical and electrical end-effector design processes (Dario & Carrozza, 2000).

In addition to humanoid hands built for prosthesis applications, hands are also built to be used in place of a human in situations such as hazardous environments. Often a human is required to be in a situation that puts him/her in significant danger. Situations like these include extreme thermal situations, radiation exposed areas, or outer space. In situations like this, it would be beneficial for a robot with human like mechanical abilities to be present. For this reason, the purpose has arisen to design robotic humanoid hands and arms capable of performing certain tasks in the physical place of a human.

One robotic humanoid with two hands, a head and a torso is the Robonaut, designed by NASA (Diftler et al., 2003). Traditionally astronauts have needed to wear space suits costing approximately \$12 million each to protect them from exposure to the suns radiation and temperature swings from -100 to 120 degrees Celsius (Bonsor, 2001). Therefore, Robonaut has been designed to perform activities in outer space to limit the exposure of humans to such dangerous environments. Robonaut is of equivalent size to an adult human in a space suit, and both of its hands incorporate four fingers and a thumb. Each hand possesses 14 degrees of freedom (NASA, 2004).

Another reason to replicate the human hand is for the sake of the challenge of incorporating many different technologies into the complex design of a humanoid hand. Such hands are often used as research tools into the understanding of mechanical and biological dexterity (Jacobsen, 1986). Many different hands have been built over the past three decades. Hands have evolved designs with three fingers and no thumb (Okada, 1982), all the way to an almost exact replication of a human hand (Shadow, 2005a), comprising of 4 multi-jointed fingers and a thumb. One of the most advanced hand created to date is the Shadow Hand, comprising of 25 individual degrees of freedom, actuated entirely by compliant air-muscles (Shadow, 2005a).

Several forms of actuation have been utilised in these hands such as electrical revolute motors (Okada, 1982; Salisbury & Roth, 1982), DC servo motors (Bekey et al., 1990), ultrasonic motors (Yamano & Maeno, 2005), pneumatic actuators (Jacobsen, 1986), Shape Memory Alloy (SMA) (Hino & Maeno, 2004) and McKibben air-muscles (Lee & Shimoyama, 1999; Scarfe, 2004; Shadow, 2005a). The two major choices for humanoid hand actuation however are air-muscles and DC servo motors. Both have their advantages and disadvantages. The placement of the actuators and torque transfer mechanism is also of importance. For these reasons, a brief discussion of these factors will now be discussed.

A.2 HUMANOID HAND ACTUATION

In choosing an actuation system, the weight and force produced are both very important factors, together with the ease of control. Two of the major forms of actuation chosen to be used in humanoid hands are DC servos and McKibben air-muscles. Table A.1 compares the properties of both DC servos and air-muscles.

Actuator	Advantages	Disadvantages
Air-muscles	Cost effective Simple to construct (low cost) Very high strength to weight ratio Compliant High Speed Long and Thin in design	Complex Control Requires Control Valves Requires Air hosing Requires compressed pneumatic air Maintenance can be troublesome Position Feedback is external
DC Servos	Simple Control Requires only electricity Simple maintenance Position feedback is internal	Complex to construct (high cost) Low strength to weight ratio Non Compliant Moderate Speed

Table A.1 – Air-muscles vs. DC Servos for Humanoid Hand Actuation

For the reasons presented in Table A.1, air-muscles provide an alternative in place of biological muscles when situated in the forearm of a humanoid hand. However, the requirement of a supply of regulated compressed air induces problems involving space (air hoses, valves and compressor). DC servos on the other hand only require electricity and are simple to control, with the feedback already integrated. In terms of strength to weight ratio, air-muscles are by far superior, unless the servo is very highly geared which then compromises its speed. Thus depending on what application the hand is being designed for, the choice of actuator will be important.

The placement of actuators also varies from hand to hand. Some designs incorporate the actuators in the palm (Dario & Carrozza, 2000; Yamano & Maeno, 2005) but most commonly they are housed in the forearm. Several different methods have been used to transfer the force generated by the actuators to the joints in the fingers. Pulleys (Jaffe, 1989; Yamano & Maeno, 2005), link-rods (Fukaya et al., 2000), sheathed cables (Shadow, 2005b) and centre joint channelling (Biagiotti et al., 2005; Scarfe, 2004) are just a few of the mechanical methods used to transfer actuator torque to the fingers. Pulleys require precision joint machining and link rods are suitable when a particular degree of freedom is directly linked mechanically to a pre-defining degree of freedom, allowing a hand to have numerous yet non-independent joint functionality. Methods such as centre joint channelling uses less physical space for each tendon, however perfect centre alignment is required for this method to be used effectively. Sheathed cables instead require more physical space per tendon, however their channelling through joints in the hand remains independent of the centre of the joint axis.

Apart from the actuation methods used, many of these hands also incorporate several different forms of feedback such as touch sensitivity sensors. However the addition of more and more sensors produces problems in regards to the level of available physical space in the fingers and weight. Therefore to date, it has not been possible to incorporate the level of sensory feedback that the human hand possesses. Nonetheless, sensory feedback remains vitally important when replicating the human hand mechanically, and thus a brief presentation of the sensory feedback used to date on humanoid hands will now be presented.

A.3 HUMANOID HAND SENSORY FEEDBACK

Humanoid hands built to date have all incorporated some level of basic feedback. Most commonly, this has been in the form of position feedback for the finger joints via the use of hall effect sensors (Tuffield & Elias, 2003) or potentiometers (Scarfe, 2004). In addition, more advanced hands have also incorporated tactile force sensors in the fingertips. However the more advanced the hands' capabilities become, the more complex the hardware required to be designed to fit into the size of a human adult's hand. Together with the large force capabilities desired at the fingertips, the feedback must be robust and provide a high resolution of feedback. While several humanoid hands created have used feedback of some sort, the major ones include the DLR Hand II, the Robonaut hands and the MAC-HAND.

The DLR Hand II uses strain gauge joint torque sensors together with potentiometers in each joint, together with two linear Hall effect sensors in the DC motors to calculate velocity (Butterfab et al., 2004). Temperature sensors are also integrated for monitoring and compensation of the other sensors. In addition, force torque tactile sensors are integrated in each fingertip to provide feedback for the required grip strength. The Robonaut hand (Martin et al., 2004) and the MAC-HAND (Cannata & Maggiali, 2005) incorporate similar sensory designs to the DRL Hand II in terms of the capabilities provided, which represent the majority of complex sensory and feedback systems in humanoid hands to date. An increased number of sensors enhances the hands feedback ability, however cost, size and weight are obvious reasons as to why particular technologies for sensory feedback are used over others. Sensors are also chosen based on the particular application that the hand is intended for. The number of sensors and accuracy and resolution of each are also taken into account for the reasons above.

B.0 PID GAINS

The feedback control section of the controller uses a standard PID controller as illustrated in Equation 5.1. The gains have been set as such:

B.1 PROPORTIONAL GAIN:

The proportional gain is required to drive the system in the required direction at all times, even when the system has never encountered the set of input vectors before. Based solely on the error at a particular point in time, the proportional gain multiplies the error by a factor, becoming the proportional driving force.

The proportional gain was chosen so that minimal overshoot occurs at any large change in desired position, yet not enough so that hunting does not occur with any small change in desired position. In any case, it is more important for the system to achieve stability at the desired set point, and thus hunting should be eliminated at all costs. For this reason an overly damped response from a low proportional gain is preferred over a rapid response.

The proportional gain should not be too small whereby it takes a significant amount of time to finally reach the required set point. A proportional gain that starts to cause hunting should be lessened slightly to obtain this gain value. If this gain causes overshoot to occur at large changes in desired position, then it is best that the derivative gain is used to minimise this.

B.2 DERIVATIVE GAIN:

The derivative gain is used here to slow the velocity of the actuator down as the velocity increased. Much like a viscous damper, the faster the actuator moves, the more resistance it receives against the force of motion.

The derivative gain is not as critical as the proportional gain, however improves the speed of response by eliminating the overshoot associated with high proportional gains. If large overshoots occur with chosen proportional gains, the derivative gain can help stabilise the system faster.

Note that the derivate gain has little effect on small changes in desired position as the velocity associated with such changes are very small.

B.3 INTEGRAL GAIN:

The integral gain is normally used to drive the system to the desired state when steady state error exists. A summation of the error over time is usually used to drive the system to eventually achieve zero error.

The air-muscle physical system used here exhibits steady state error properties, however the PID equation adds itself to the control value, and in effect always drives the system out of steady state error in much the same way.

This continual addition of the PID driving force (based on the steady state error) to the current control value pushes the control value in the required direction until zero error is reached. At this point the PID driving force equals zero, and the control value remains constant.

Using this method, there is no need to incorporate an integral gain in the PID controller.

C.0 VELOCITY INPUT VECTOR PARAMETER

The $\tan(x)$ function is one such function that can easily map a zero to infinite value to a finite value while maintaining the CMAC input vector space boundaries.

For the physical system in use, as in anything, an infinite velocity will never occur. The upper limit can be calculated however. Using a control sampling rate of 100Hz (see Appendix D) means that the smallest velocity that can be calculated will have a time base of 10msec. Knowing that the maximum range of motion of the actuator possible is its 8bit feedback resolution range of 256, means that the greatest physical attainable velocity of the system possibly available is 25600units/second. This value should map to the maximum CMAC input vector space limit of 2048. This is set to the upper limit of the required velocity. Since actuator motion is required in both directions, the lower limit will map -25600 units/second to the minimum CMAC input vector of 0. Knowing these bounds, and using $\tan(x)$ as the mapping function, the shape and limits of the $\tan(x)$ function must be calculated. y is the desired unbounded velocity. x is the desired velocity CMAC input vector.

Starting with a standard $\tan(x)$ function with an x offset of half the CMAC input vector space:

$$y = a \tan\left(\frac{x-1024}{b}\right) \quad \text{Equation C.1}$$

Where a is an amplitude multiplier and b defines the curve of the function over the required range.

Using the upper limit of $y = 25600$ and mapping it to $x = 2048$, and then rearranging to obtain a in terms of b gives:

$$a = \frac{25600}{\tan\left(\frac{1024}{b}\right)} \quad \text{Equation C.2}$$

Plotting Equation C.2, we obtain Figure C.1.

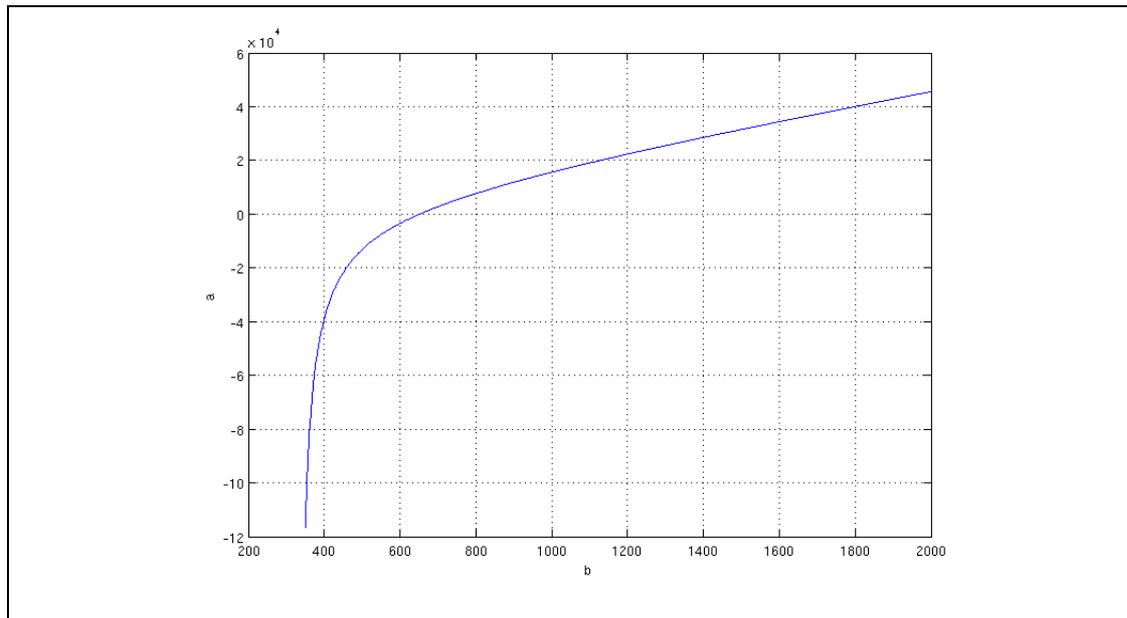


Figure C.1 – a vs. b from Equation C.2

We want positive values of y for positive values of a , so from Figure C.1, $b > 651.90$. To obtain the final bounded mapping function, substituting Equation C.2 into Equation C.1 and rearranging x in terms of y produces:

$$x = b \tan^{-1} \left(\frac{y \tan\left(\frac{1024}{b}\right)}{25600} \right) + 1024 \quad \text{Equation C.3}$$

Thus, for values of $b > 651.90$, we obtain the following plot, constrained to the required boundaries.

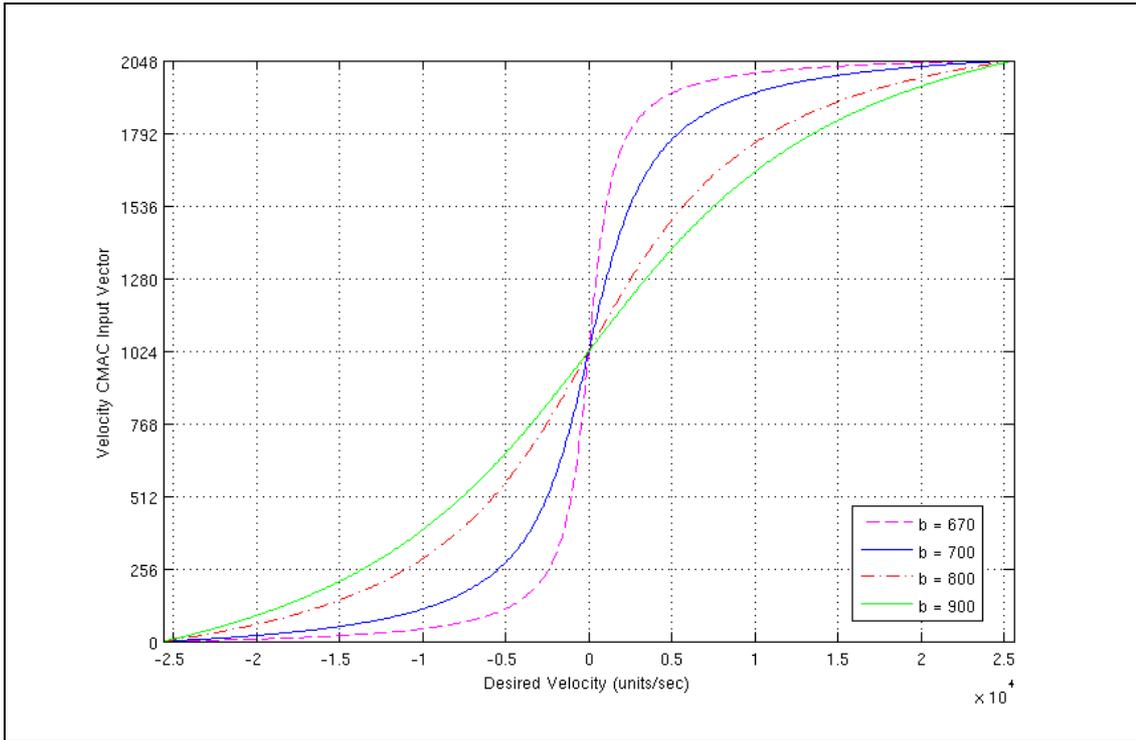


Figure C.2 – Desired Velocity to CMAC Input Space Mapping Relationship

In Figure C.2, by changing the value of b as shown, the shape of the mapping function varies. As mentioned, a finer resolution at small velocities and a coarser resolution at higher velocities is required. Values of b greater than 700, as illustrated in Figure C.2, produce such mapping functionality. Thus the $\tan(x)$ function bounded as in Equation C.3 works well to provide this velocity mapping.

D.0 INTERFACE HARDWARE

The interface hardware, namely the electronic hardware and the pneumatic valves, were created originally for use in Peter Scarfe's Bachelor Project (Scarfe, 2004). A number of modifications have been implemented on the original hardware specifically for use in the testing of the control algorithm designed in this paper. Both new hardware modifications and hardware specific functionality to this project will be presented in this section. Supporting background material for this section is contained in "Pneumatic Air-Muscle Controlled Robotic Hand" by Scarfe (2004) included on the Thesis CD, and it is recommended that it be read to best understand the remainder of this section.

D.1 ELECTRONICS HARDWARE – PWM GENERATION

Electronic simulation results of the Electronic Hardware output system (Scarfe, 2004) showed that a 100Hz PWM signal was sufficient for obtaining a consistent DC analog voltage with little voltage ripple and suitable lag times. Higher frequency PWM signals would result in a smaller DC ripple though the Pressure Balancing Valve (PBV) solenoids didn't show a change in performance. Similarly, a lower frequency PWM signal would result in a larger DC ripple, however this would cause the PBV solenoids to buzz. 100Hz was chosen as a suitable frequency, low enough for the PC to handle and fast enough to obtain smooth PBV solenoid performance.

PWM frequency generation and resolution was generated via the use of a software generated timer interrupt in Xenomai at 20kHz. To generate a PWM signal at 100Hz meant that with a 20kHz interrupt, a PWM signal with a resolution of 200 could be generated.

With a feedback resolution of 256 bits, a control output of 200 bits is insufficient to obtain control of every 256 feedback bits. An output resolution equal to at least the feedback resolution is required to obtain 1 bit movement.

Instead of increasing the timer interrupt speed on the computer which Xenomai would most probably struggle with, another method was designed to increase the output resolution while maintaining the timer interrupt at 20kHz and the PWM signal

at 100Hz. This method, termed ‘Enhanced Resolution’ was designed to increase the output resolution five times that of the original resolution, and proved to be successful.

The concept of Enhanced Resolution is quite simple. Basically, a secondary PWM signal at a lower frequency than the primary frequency is applied to the primary PWM signal. This concept is best explained with the use of an illustration, such as that in Figure D.1.

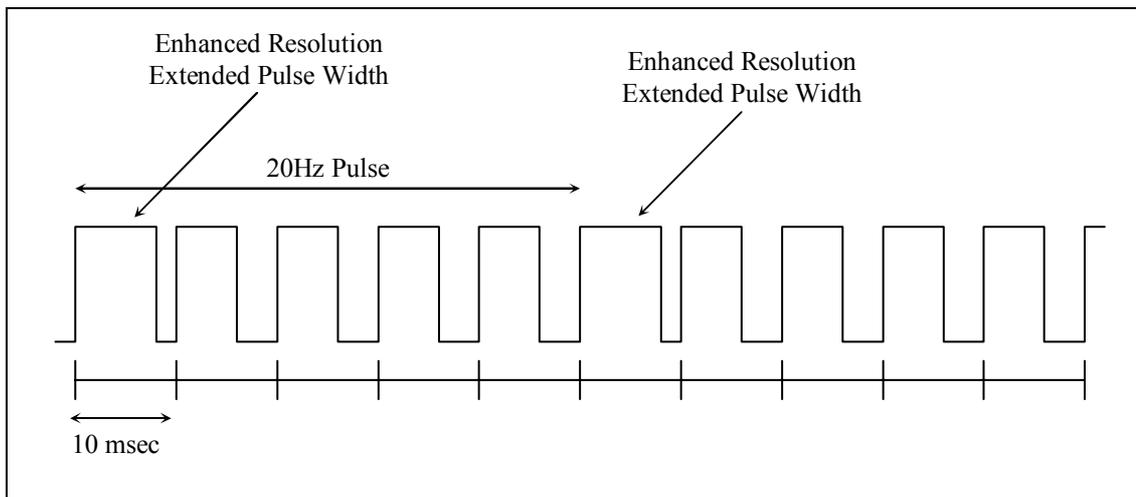


Figure D.1 – 20Hz Enhanced Resolution Timing Diagram

The primary PWM frequency is set here at 100Hz, the secondary is at 20Hz. In Figure D.1, every first pulse of the secondary signal is increased to one primary resolution unit greater than the remaining four pulses in the secondary signal. When averaged out, this additional pulse width at the secondary frequency alters the filtered DC output voltage by a slight margin.

For example, lets say that the primary frequency in Figure D.1 has a duty cycle of 60.0%. The primary PWM signal has a resolution of 200, so a 1 bit increase in duty cycle would result in a 60.5% duty cycle. With the enhanced resolution method applied, the additional pulse width on every fifth primary pulse will result in the DC filter seeing an average duty cycle of 60.1%. If every two out of five primary pulses are increased with a 1 bit increase in duty cycle, the DC filter will see an averaged duty cycle of 60.2%.

Using this method, the final obtainable resolution is equal to the primary PWM resolution of 200, multiplied by the Enhanced Resolution factor of 5, resulting in a 1000 bit output resolution. This value is much greater than the 256 bits of position feedback resolution, and thus much finer levels of motion can be controlled this way, without the need to impose additional burden on the real-time software timer interrupt.

D.2 AIR-MUSCLE UPGRADES

The most important upgraded feature to the air-muscles used in 2004 involves the use of silicon rubber bladders. Silicon rubber provides longer lasting durability over the natural rubber latex bladders used previously, as silicon rubber does not deteriorate with age as latex does. This is a significant advantage over the 2004 air-muscles as the problems associated with the bladder failure and low life expectancy of the latex are now overcome. This results in a more durable and longer lasting air-muscle.

While the new silicon rubber bladders provide significant improvements, there are a few downfalls from the choice of using silicon over latex. Silicon rubber is less elastic than latex, and thus requires more force to inflate, with air pressure forces lost stretching the silicon rubber itself. Latex on the other hand is very elastic and requires little force to stretch. In addition, the silicon rubber bladders have thicker walls than did the latex bladders. Thus even more force is required to inflate the bladders to the same volume than with the latex bladders.

Overall however the benefits outweigh the shortfalls of using the silicon bladders over latex bladders, with the longer lasting and more durable properties being the most significant advantages.

D.3 PBV BOARD UPGRADES

The several upgrades were made to the PBV board from the 2004 model.

D.3.1 PBV BOARD MANIFOLD REGULATOR/FILTER

The original unbranded regulator/filter was replaced with a quality SMC AW3000 regulator/filter. The pressure gauge on this regulator has been tested against the

previous regulator pressure gauge using a Festo SDE1 pressure sensor. It was discovered that the original pressure gauge was producing inaccurate readings up to +0.5Bar. The SMC pressure gauge however appears by sight (from the analog dial) to be within 0.05Bar accurate to the Festo SDE1 digital pressure sensor used for calibration purposes. Thus more accurate and reliable readings can be made about the actual PBV manifold pressure at any given time.

D.3.2 PBV AIRFLOW TAPS

The airflow taps which regulate the airflow from the PBV board manifold to each PBV have been replaced with inline barbed-fitting taps over the previous inline screw-fitting taps. At pressures above 2.5Bar it was discovered that the screw-fitting taps were leaking, with the seal between the supply pipes from the manifold and the screw-fitting being insufficient to withstand the air pressure. Due to the successful use from the other barbed-fitting components throughout the PBV board, the airflow taps were also upgraded to barbed-fitting taps. Tested at up to 4.0Bar, these taps have not been found to leak.

D.3.3 HIGHER USABLE PRESSURE

With the addition of the new airflow taps and the silicon bladder air-muscles, the standard working pressure of the PBV board can be increased from that of 2.5Bar in 2004. The new standard working pressure of the PBV board has been increased to 3.6Bar, and has been tested continuously at this pressure without failure for all the tests in this thesis. The PBV board and silicon air-muscles have been tested with up to a maximum pressure of 4.0Bar without failure, though tests at this pressure have only been short term. It is not recommended that greater pressures than 4.0Bar be used in this system without further testing as component failure may result.

D.3.4 ACCURATE PBV CALIBRATION METHOD

Previously, the PBVs were calibrated using a combination of trial & error and good judgment. Suitable working solenoid air-gap heights were adjusted along with the airflow tap rates and outlet pipe gap heights. Satisfactory combinations were found by a series ad hoc of tests until a suitable combination was reached. This was then duplicated with a low level of accuracy to each of the other PBVs. Results proved

sufficient, however stability and performance suffered when switching amongst PBVs.

The addition of using the Festo SDE1 Pressure Sensor makes the PBV calibration process far more accurate and repeatable. Producing an absolute accuracy of $\pm 2\%$ and a repeatability of $\pm 0.3\%$, with a two decimal-point digital LCD display, allows for a finer level of calibration precision (Festo, 2005). Thus based on a number of different tests (see Appendix I) to better understand the properties of each PBV variable, a procedure was devised to calibrate PBVs (see Section D.5).

D.3.5 POSSIBLE IMPROVEMENTS

The next set would be to increase the size of the PBV board manifold to pipe of larger diameter, providing a larger air reservoir to produce a more stable pressure source to all the PBVs. This allows faster air-pressure replenishing whilst the manifold whilst the PBVs are changing states.

D.4 PBV BOARD VARIABLES

The following lists and explains the variables associated with the performance of the PBV board.

D.4.1 PNEUMATIC SUPPLY PRESSURE

The pneumatic supply pressure to the PBV board regulator needs to be as high as possible, yet must also remain consistent throughout PBV board usage over time. The higher this pressure, the higher the airflow rate available to the PBV manifold regulator, meaning faster replenishing to the manifold regulator for times when there are high PBV venting rates. 6.0Bar was the highest 'stable' airflow rate available with the given equipment available.

D.4.2 PBV BOARD MANIFOLD PRESSURE

This pressure is set by the SMC PBV Board Manifold Regulator/Filter. The higher the pressure here, the faster the inflation speed of the air-muscles, and the greater the final pressure attainable within the air-muscles. In addition, the greater the pressure in each PBV, the higher the required current needed to fully seal the PBV outlet air-pipe by the solenoid. The overall stress to the PBV manifold board components is

increased as the PBV manifold pressure increases. Finally the greater the PBV manifold pressure, the faster the PBV manifold regulator can replenish the air pressure within the manifold at times of high PBV venting rates.

There is no optimal pressure to run the air-muscles at. Too great a pressure and component failure will result to either the PBV board or the air-muscles. Too low a pressure and the air-muscles will inflate slowly and produce less axial force. A compromise is therefore needed. The PBV board and the silicon air-muscles were tested with a maximum pressure of 4.0Bar. This test was not continuous however but proved that for short periods of time the system could withstand this pressure. Therefore at slightly less than 4.0Bar, a pressure of 3.6Bar was tested and run continuously on the system without failure. Thus this pressure has been chosen as the working pressure for the PBV/air-muscle setup.

D.4.3 AIR-MUSCLE VOLUME

Larger air-muscles, either in length or diameter require more air to fully inflate to the PBV manifold pressure. This produces a slower response than would an air-muscle of a lower volume. Again there is no specific air-muscle length/volume required, however this fact must be remembered when designing test rigs and analysing test data.

D.4.4 MAXIMUM CONTROL VOLTAGES

The electronic output board has been designed to run at standard 12V, producing a maximum of 9.47V at each PBV solenoid. This voltage produces enough current sufficient to fully seal a PBV at a PBV manifold pressure up to 4.0Bar.

D.4.5 PBV SPECIFIC VARIABLES

In addition to the PBV board variables mentioned above, there are 3 variables which must be carefully calibrated to produce suitable control from each PBV. A procedure for PBV calibration will follow the explanation of the calibration of the PBV variables.

D.4.6 SOLENOID AIR-GAP HEIGHT

The solenoid air-gap distance determines how sensitive the PBV is to input current. The larger the gap, the less sensitive the PBV is. On the contrary, the smaller the gap, the more sensitive the PBV is. Figure D.2 shows the relationship between force with respect to the solenoid air-gap between the plunger and the cap. While efficient, an air-gap between 0 to 1mm is highly exponential. Thus using the portion of the air-gap between 1 to 2mm provides an approximately linear relationship between force and air-gap. As the air-gap is directly and linearly proportional to current through the solenoid, hence controlling the current through the solenoid coil directly controls the air-gap distance. (Voltage is directly proportional to current since the solenoid coil resistance is fixed.)

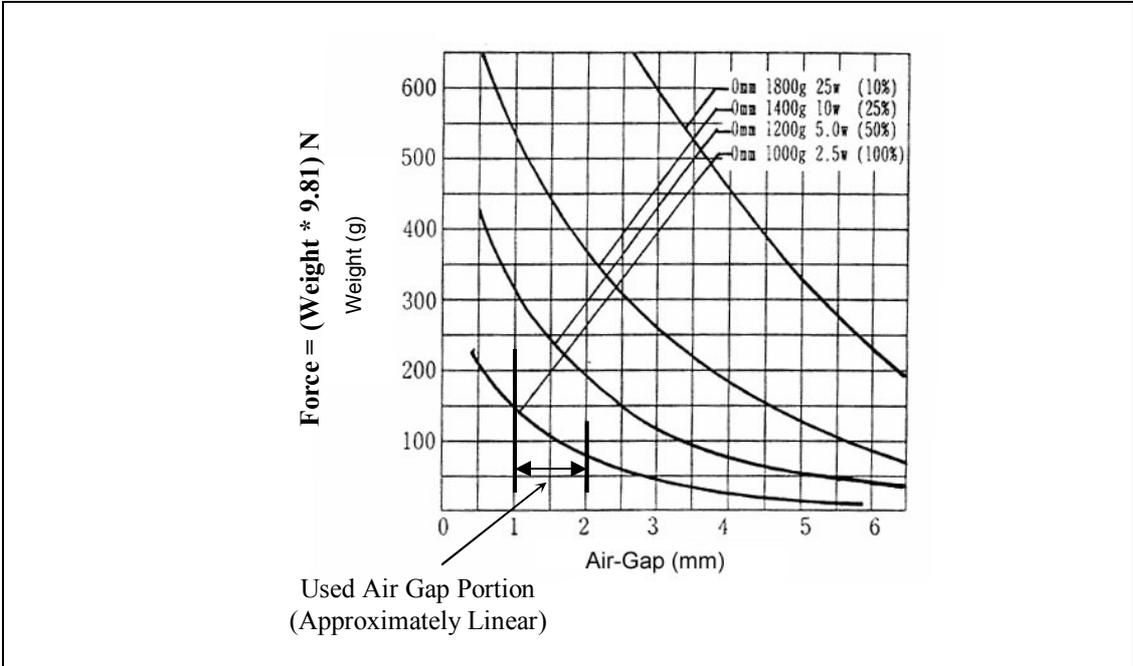


Figure D.2 – Solenoid air-gap / force relationship (Jaycar, 2003)

Tests shown in Appendix I proved that a maximum air-gap of 1.5mm achieved a very useful response, proving enough maximum force to fully seal the PBV while maintaining a decent control voltage range. The range of movement the solenoid needs to achieve is based on the outlet pipe air-gap, and maintains the solenoid plunger movement between the 1 to 2mm solenoid air-gap range.

The relationship result between solenoid voltage and air-muscle position of a PBV is shown below in Figure D.3, at 3.6Bar pressure using the PBV calibration outlined in

Section D.5. The air-muscle displacement is represented in bits, measured on a linear scale between 0 to 255 based on the feedback resolution. Using the controllable range of motion (between 3 to 8V), accurate control can be obtained. Note however the hysteresis that this system produces, a standard air-muscle attribute known to plague air-muscle control.

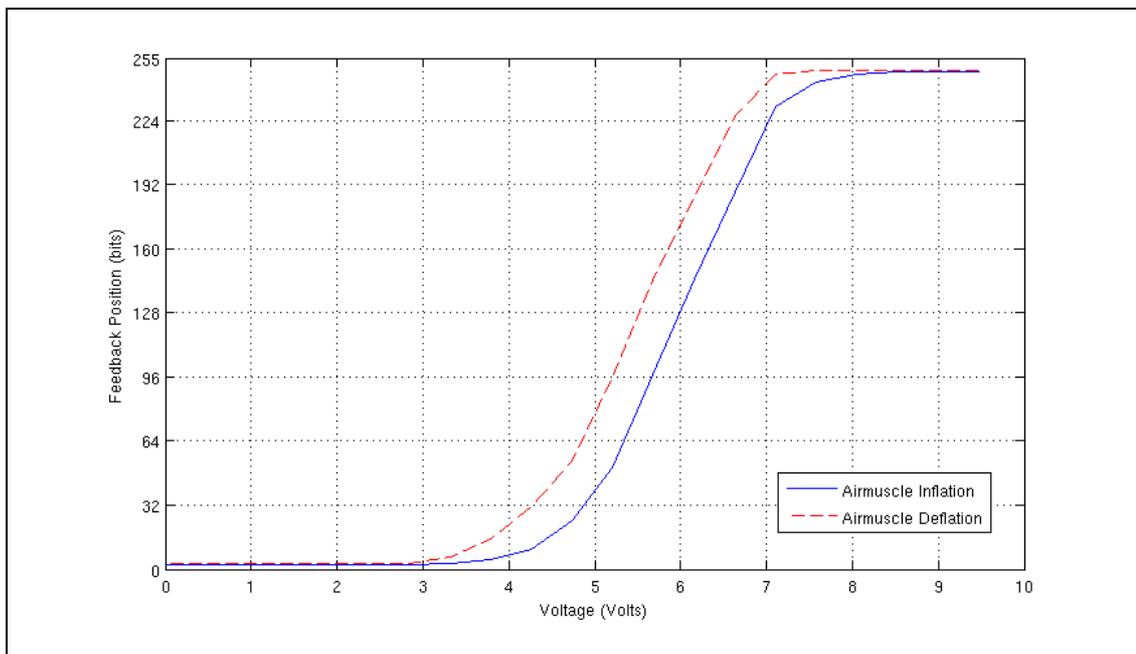


Figure D.3 – Relationship between Solenoid PBV Voltage and Air-Muscle position.

It is important to point out that the reason for the lower non-responsive segment of the graph, where the voltage needs to reach approximately 3.0 volts before there is any change in displacement, is due to the fact that a minimum force is required to neutralise the vertical forces acting downwards on the solenoid plunger. At the point where the air-muscle position starts to change, the magnetic force produced from the current through the solenoid coil starts to be greater than the sum of the gravitational acceleration and the air pressure force venting the outlet pipe. Thus at above approximately 3V when this minimum force is overcome, the plunger basically 'floats' to a given height to produce the required pressure in the PBV as shown in Figure D.4.

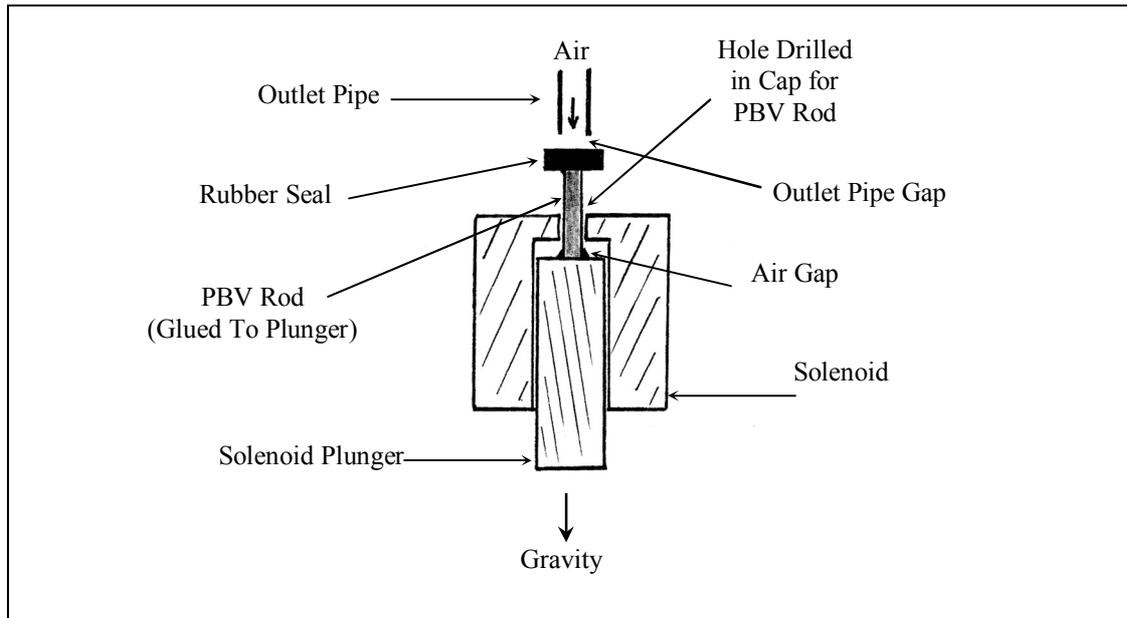


Figure D.4 – Modified Solenoid for PBV Usage (Scarfe, 2004)

D.4.7 AIRFLOW TAP ADJUSTMENT

The airflow tap provides airflow control to the PBV from the supply manifold. It allows adjustment of the rate at which the air-muscle fills with air, directly proportional to the rate at which the air-muscle moves from a full extension to a full contraction.

The airflow tap along with the outlet pipe gap also allows the minimal PBV pressure to be set. So long as the PBV board manifold is pressurised above atmospheric pressure and the PBV at 0V is venting, the PBV pressure will never be equal to the atmospheric. This also means that the pressure in the air-muscle will never be equal to the atmospheric pressure. While it may seem that the air-muscle is totally deflated, some pressure still remains inside due to the fact that the PBV is venting and thus pressurised to a greater pressure than the atmospheric pressure. This means that based on the airflow tap setting and the maximum outlet pipe gap, a minimal pressure can be set for the PBV.

D.4.8 OUTLET PIPE AIR-GAP

The outlet-pipe gap distance determines how far the solenoid needs to move for the air-muscle to be fully extended and contracted at the air-gap's limits. There is a close relationship between the outlet pipe air-gap and the airflow tap. For example, if the

air-gap is too small, the air-muscle will not be able to fully extend back to its resting state if the minimum airflow rate produces a high minimal PBV pressure.

If the gap is too large, a control dead-band will be created, where the controllable solenoid movement will be wasted whilst not producing a change in PBV pressure. In addition, the solenoid runs the risk of falling out of the controllable magnetic field range into the dead-band when large current changes instantly occur, producing undesirable control problems.

The outlet pipe gap should be adjusted carefully to prevent excessive PBV manifold venting through the PBV when the PBV is in its resting state (0V). A few combinations together with the airflow tap rate will now be presented to demonstrate the relationship these two control parameters have on each other.

1. Outlet Pipe Air-Gap Large, Minimal PBV Pressure High:

With a large outlet pipe air-gap and high minimal PBV pressure (meaning the airflow tap is opened by a large amount to generate this large airflow rate), there is a lot of wasted air from the PBV manifold. If each of the 22 PBVs on the PBV board is venting at this rate in their resting state (0V), large fluctuations will exist across the length of the PBV manifold, in addition to when PBVs open and close. Inconsistencies in maximum PBV pressures will occur; large amounts of air will be wasted resulting in excessive power and air compressor usage. The air-muscle will inflate at a rapid rate, producing a fast response in the contraction direction of motion. However since the minimum PBV pressure is high, the minimal air-muscle will be high also, resulting in an air-muscle that cannot fully deflate, limiting the range of controllable motion. Depending on the size of the air-gap, there may or may not be a control dead-band.

2. Outlet Pipe Air-Gap Large, Minimal PBV Pressure Low:

With a large outlet pipe air-gap and low minimal PBV pressure (meaning the airflow tap is opened by a large amount to generate this large airflow rate), there will still be a lot of wasted air. To keep the minimal air pressure low, due to the fact that the outlet pipe gap is large, a lot of air will still need to vent whilst the PBV is at rest (0V). The air-muscles will inflate at a relatively fast rate and will

be able to fully deflate as well. There will also be a control dead-band, as the solenoid will have to move a distance before any noticeable change in PBV pressure occurs.

3. Outlet Pipe Air-Gap Small, Minimal PBV Pressure High:

With a small outlet pipe air-gap and high minimal PBV pressure, chances are that minimal air will be wasted due to the overall low airflow rate venting the PBV due to the small outlet pipe gap. The most noticeable effect here is that the minimum deflation state of the air-muscle will be poor, resulting in a somewhat inflated air-muscle even when the PBV is in its resting state (0V). Thus the control range will be unnecessarily limited. The inflation speed of the air-muscle will be fast (though not as fast as when the outlet pipe air-gap is large and the minimal PBV pressure high).

4. Outlet Pipe Air-Gap Small, Minimal PBV Pressure Small:

With a small outlet pipe air-gap and small minimal PBV pressure, there will be no wasted air apart from necessary PBV venting. However inflation speed will be poor. This will be the most noticeable effect here. Deflation speed of the air-muscle should be satisfactory with full deflation possible. Whilst the controllable range of the air-muscle will be fully available and there won't be any wasted airflow, speed of response will be poor, especially when inflating the air-muscle. There is also a slight chance that a small control dead-band will be present.

Based on the above combinations and resulting factors, a combination needs to be reached to achieve good control range, good control performance, while at the same time not wasting any unnecessary air.

After several trials using the above combinations (see Appendix I), satisfactory air-muscle response performance was achieved using a measured outlet pipe air-gap of approximately 0.25mm. This distance was measured after calibration (using the calibration procedure in Section D.5), using a digital calliper. (I.e. the distance the solenoid plunger moves from the outlet pipe gap's fully open state (0.00V) to its fully closed state (9.47V) when pressurised.) It should be noted that all control of the air-

muscle, to each of the feedback's 8bit resolution positions, must be controlled over this tiny 0.25mm range.

D.5 PBV CALIBRATION PROCEDURE

After the properties of the PBV variables were studied independently and together, a procedure was put together for their calibration. Appendix I shows the results from the different tests that were run to arrive at the control settings used in this PBV calibration procedure.

This procedure configures the maximum solenoid air-gap height, the airflow rate into the air-muscle and outlet pipe gap height, with PBV parameters to be reproducible on any PBV valve on the PBV Board. Using the PBV Board setup as per the supply pressures and voltages below, each PBV was setup as follows. The procedure is written as a guide to allow others to reproduce the same setup:

Air-muscle Type: McKibben(Silicon 6mmID/8mmOD bladder)

Air-muscle Max Length (in test at 3.60Bar): 143mm (inflatable area only)

Air-muscle Min Length (in test at 0.20Bar): 110mm (inflatable area only)

Max Control Voltage: 9.47V DC

Max Control Voltage: 0.00V DC

Control Board Supply Voltage = 12.00V

Pneumatic Supply Pressure: 6.0Bar (read from Supply Regulator)

PBV Board Manifold Pressure: 3.6Bar (read from PBV Board SMC Regulator)

Steps:

1. Before you do anything make sure that you adjust the PBV at a supply manifold pressure of 3.6Bar. With the addition of more open PBVs, the pressure will drop slightly; so make sure that the manifold is re-pressurised to 3.6Bar before each PBV calibration.
2. Make sure that the solenoid plunger (look under the solenoid) has its black marker dot aligned with the black dot on the front of the solenoid. This is to

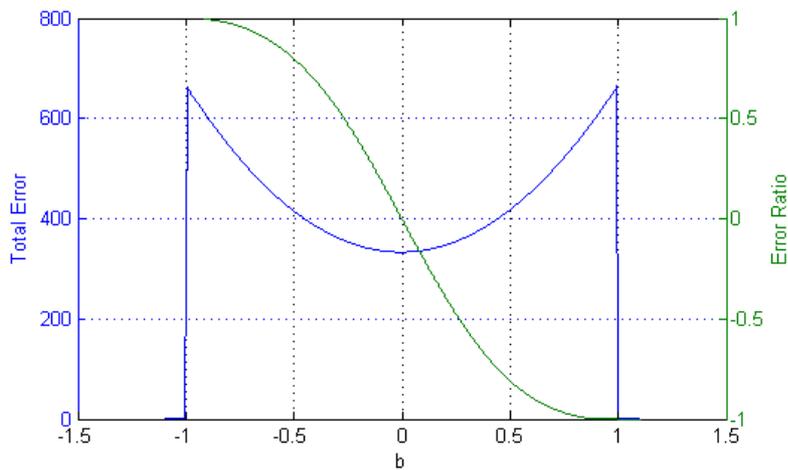
make sure that there will be no friction between the top of the solenoid and the brass rod, as some of the solenoid's brass rods are not aligned on the plunger directly vertically.

3. This step adjusts the solenoid air-gap height.
 - a. Unscrew the solenoid mounting bracket so that the solenoid moves down to its lowest position possible. At this point the solenoid air-gap inside the solenoid will be equal to 0mm as the solenoid cap and plunger will be touching.
 - b. Measure the distance from the top of the solenoid to the bottom surface of the PBV board. Use the digital calliper to take an accurate measurement.
 - c. Move the solenoid up 1.5mm from this distance and secure it in place by fastening the solenoid bracket screws until solenoid is snug in place.
 - d. Using the same practice as that of step 3.b, measure the height of the solenoid. If the height is not $1.5\text{mm} \pm 0.1\text{mm}$ greater than the height measured in step 3.b, start the procedure from step 3.a again.
4. Connect the PBV to an air-muscle and pressure sensor.
5. Move the outlet pipe away from the rubber stopper so that any exiting airflow from the pipe will be unrestricted. Do not bend the pipe out anymore than you have to, to prevent the adjustable aluminium bracket from warping and or fatiguing.
6. Supply the PBV board manifold with 3.6Bar pressure.
7. Adjust the airflow tap on the PBV you are calibrating until 0.21Bar is read on the pressure sensor. This is the current PBV pressure, stabilised in the outlet pipe, the air-muscle and the pressure sensor.
8. Move the outlet pipe back in place so that the outlet is sitting vertically over the rubber stopper connected to the solenoid plunger. Undo the screw in the centre of the aluminium bracket by a few turns. This screw holds the outlet pipe in place when fastened.
9. By adjusting the height of the outlet pipe opening from the rubber stopper the pressure will change. With the solenoid at 0.00V, adjust the outlet pipe gap height until the pressure is between 0.6 to 0.7Bar. This in effect sets up the outlet pipe gap to a very precise yet reproducible height.
10. Fasten the screw in the centre of the aluminium bracket so that the vertical height of the outlet pipe is fixed in place.

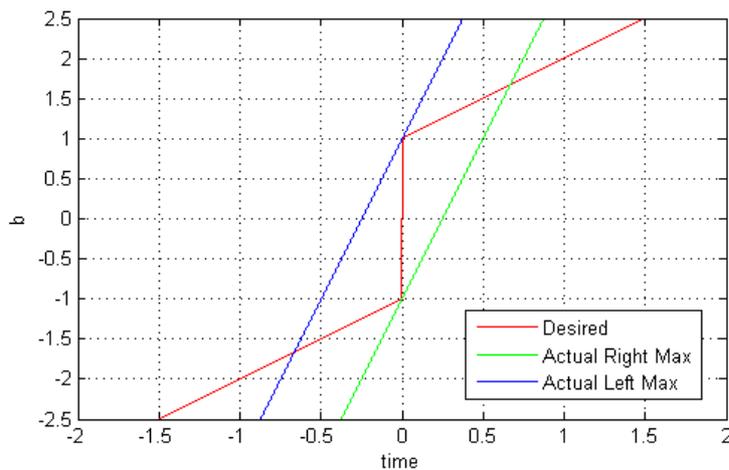
11. Adjust the airflow tap until a pressure of 0.20Bar is achieved on the pressure sensor.
12. Power the PBV to the maximum voltage of 9.47V to fully seal the PBV. With the PBV fully sealed, make sure that there are no leaks throughout the air pipes leading to the air-muscle and the pressure sensor. If no leaks are present, the pressure sensor should be reading the same pressure as the PBV manifold regulator. If a leak is found, the calibration process will have to be repeated after the leak is resolved.
13. This completes the PBV calibration process.

E.0 TOTAL ERROR REDUCTION METHOD

The EMGC uses the Error Ratio convergence approach to reduce the Total Error of a situation to a minimum value. The approach assumes that when the first and second error regions are equal, the minimum Total Error for a situation is minimised. The EMG is moved backwards and forwards in time until the system produces a response with the Error Ratio converged to zero. The theory behind this approach works for trajectories which are symmetrical about the middle position point of the situation. Figure E.1 illustrates this concept.



(a) Total Error Profile between Max Left and Right Actual Trajectory Gradients, with Error Ratio



(b) Desired Trajectory with Max Left and Right Actual Trajectory Response Gradients

Figure E.1 – Minimum Total Error and Error Ratio Response for Symmetrical Situation

The Figure E.1(b) illustrates a symmetrical desired response trajectory profile. It is symmetrical about the mid desired position point at the situation occurrence in that the profiles are equal but of opposite signs. This point occurs in Figure E.1(b) at (0,0) on the plot axis. The approach gradient is $\theta_2=-45^\circ$ and the departure gradient is $\theta_1=45^\circ$.

The actual response profile, when using the EMGC, will lie between the Actual Right and Left maximum limits. The gradients of these two aforementioned lines represent the actual response gradient profiles of an actuator's situation response, being of relatively consistent slope. The actual response must be bounded by a minimum and maximum intersection with the desired response trajectory. The actual response's maximum left response has a 'b' value of +1. The actual response's maximum right response has a 'b' value of -1. The actual response thus exists for $-1 < b < 1$.

Shifting the actual response from $b = -1$ to $b = 1$ will move shift the Error Ratio of a situation from -1 to 1, and will cause the situation's Total Error to vary throughout, with some value of 'b' resulting in a minimum Total Error value for the given actual response gradient. Figure E.1(a) illustrates the Total Error function created by moving the actual response gradient from $b = -1$ to $b = 1$. The Error Ratio for the same change in 'b' is also illustrated on the same figure.

Note that for this symmetrical desired trajectory profile, the minimum Total Error exists at $b = 0$. This is also the point where the Error Ratio is equal to zero. Thus when the Error Ratio is equal to zero, the Total Error will be minimised.

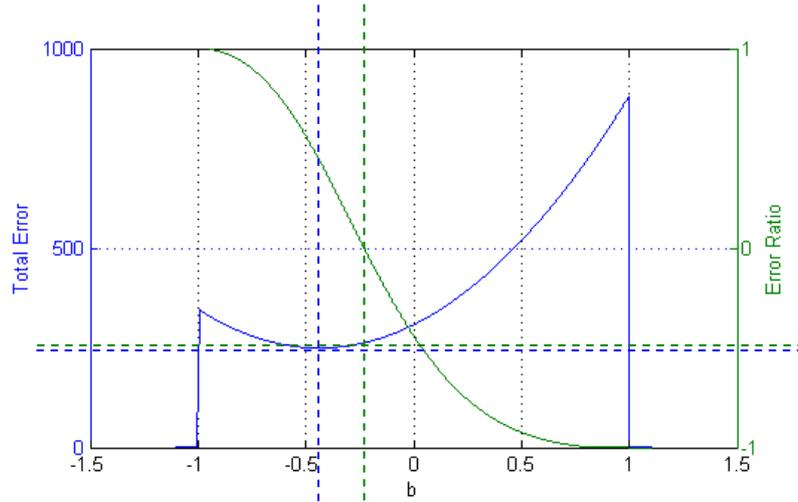
While true for symmetrical desired trajectory functions, the same error minimising approach was also used for non-symmetrical functions. Non symmetrical function testing was performed in Chapter 9.0. The feasibility of using the Error Ratio convergence approach for minimising the Total Error of non-symmetrical functions is illustrated in Figure E.2.

Figure E.2(b) illustrates a non-symmetrical desired trajectory. Compared to Figure E.1(b), the situation now has an approach gradient of $\theta_2=-60^\circ$ and a departure

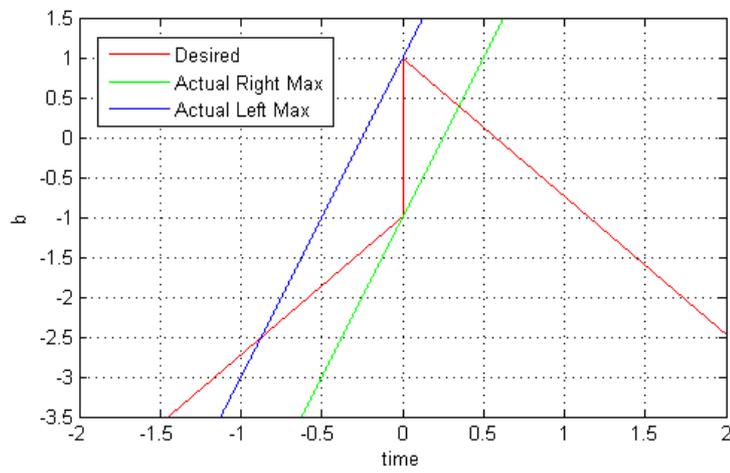
gradient of $\theta_1 = -60^\circ$. This creates a non-symmetrical desired trajectory representing the approach and departure gradient limits used in the tests in Appendix H.

Figure E.2(a) illustrates that the minimum Total Error no longer exists when the Error Ratio is equal to zero. The minimum Total Error exists at $b = -0.45$. The converged Error Ratio exists at $b = -0.26$.

Using the converged Error Ratio value will thus not result in a minimised Total Error. However, the Total Error value it does produce is very close to the minimised Total Error value compared to the maximum Total Error deviation, illustrated by the very close horizontal dashed lines in Figure E.2(a). Thus while not minimised, using the converged Error Ratio approach to minimise a situation's Total Error will in most practical cases (especially for the situations tested throughout this thesis) result in a substantially reduced Total Error.



(a) Total Error Profile between Max Left and Right Actual Trajectory Gradients, with Error Ratio



(b) Desired Trajectory with Max Left and Right Actual Trajectory Response Gradients

Figure E.2 – Minimum Total Error and Error Ratio Response for Non-Symmetrical Situation

F.0 EMGC TEST LOG

The following tests refer to those completed throughout the final test stages of the EMGC, as documented in Peter Scarfe's "PhD Raw Note Book 2". Please also refer to Figure 9.2 and Figure 9.3. Log Files are located on the Thesis CD.

Test 1		
Training Time: 1800 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 200$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 75$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080728-133513	Other:	
Test 2		
Training Time: 1800 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 200$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 75$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080728-141037	Other:	
Test 3		
Training Time: 1800 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 200$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 75$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080728-225637	Other:	
Test 4		
Training Time: 1800 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 200$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 75$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080728-232951	Other:	
Test 5		
Training Time: 1400 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 200$ bits	$T_1 = 90$ samples	$\theta_1 = 0^\circ$
$y_{min} = 75$ bits	$T_2 = 90$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080731-134024	Other:	

Test 6		
Training Time: 1400 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 200$ bits	$T_1 = 90$ samples	$\theta_1 = 0^\circ$
$y_{min} = 75$ bits	$T_2 = 90$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080731-140858	Other:	
Test 6a		
Training Time: 1400 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 200$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 70$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080731-144156	Other: 60% EMG, 0 EMG Offset	
Test 7		
Training Time: 1400 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 200$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 70$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080731-150639	Other: 60% EMG, 0 EMG Offset	
Test 7a		
Training Time: 1400 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 200$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 70$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080731-153134	Other: 100% EMG, 0 EMG Offset	
Test 7b		
Training Time: 1400 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 200$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 70$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080731-155550	Other: 0% EMG, 0 EMG Offset	
Test 7c		
Training Time: 1400 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 200$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 70$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080731-163101	Other: 50% EMG, 0 EMG Offset	

Test 7d		
Training Time: 1400 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 200$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 70$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080731-170502	Other: 25% EMG, 0 EMG Offset	
Test 7e		
Training Time: 1400 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 200$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 70$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080731-172933	Other: 75% EMG, 0 EMG Offset	
Test 8		
Training Time: 1300 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 128 + 30$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 128 - 30$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080803-205017	Other:	
Test 9		
Training Time: 1300 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 128 + 30$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 128 - 30$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080803-212127	Other:	
Test 10		
Training Time: 1300 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 128 + 50$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 128 - 50$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080803-214408	Other:	
Test 11		
Training Time: 1300 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 128 + 50$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 128 - 50$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080803-220641	Other:	

Test 12		
Training Time: 1300 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 128 + 70$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 128 - 70$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080803-222850	Other:	
Test 13		
Training Time: 1300 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 128 + 70$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 128 - 70$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080803-225110	Other:	
Test 14		
Training Time: 1300 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 128 + 80$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 128 - 90$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080803-232022	Other:	
Test 15		
Training Time: 1300 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 128 + 80$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 128 - 90$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080803-234228	Other:	
Test 16		
Training Time: 1800 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 128 + 70$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 128 - 70$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080807-203736	Other: Situation Total Error Selection Criteria Off	
Test 17		
Training Time: 1800 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 128 + 70$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 128 - 70$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080807-225458	Other: Situation Total Error Selection Criteria Off	

Test 18		
Training Time: 1800 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 128 + 70$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 128 - 70$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080808-001121	Other: Situation Total Error Selection Criteria On	
Test 19		
Training Time: 1200 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 200$ bits	$T_1 = 300$ samples	$\theta_1 = -0.5$ bits per iteration
$y_{min} =$ Not Applicable	$T_2 =$ Not Applicable	$\theta_2 =$ Not Applicable
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080810-141619	Other:	
Test 20		
Training Time: 1200 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 200$ bits	$T_1 = 300$ samples	$\theta_1 = -0.5$ bits per iteration
$y_{min} =$ Not Applicable	$T_2 =$ Not Applicable	$\theta_2 =$ Not Applicable
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080810-144046	Other:	
Test 21		
Training Time: 1300 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 128 + 80$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 128 - 90$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 912N/m	
Log File: 080824-135140	Other:	
Test 22		
Training Time: 1300 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 128 + 80$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 128 - 90$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 912N/m	
Log File: 080824-141500	Other:	
Test 23		
Training Time: 1300 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 128 + 70$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 128 - 70$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 912N/m	
Log File: 080824-144345	Other:	

Test 24		
Training Time: 1300 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 128 + 70$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 128 - 70$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 912N/m	
Log File: 080824-150832	Other:	
Test 25		
Training Time: 1300 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 128 + 50$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 128 - 50$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 912N/m	
Log File: 080824-164841	Other:	
Test 26		
Training Time: 1300 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 128 + 50$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 128 - 50$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 912N/m	
Log File: 080824-162640	Other:	
Test 27		
Training Time: 1300 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 128 + 30$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 128 - 30$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 912N/m	
Log File: 080824-203133	Other:	
Test 28		
Training Time: 1300 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 128 + 30$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 128 - 30$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 912N/m	
Log File: 080824-205450	Other:	
Test 29		
Training Time: 1300 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 128 + 80$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 128 - 90$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 304N/m	
Log File: 080824-212818	Other:	

Test 30		
Training Time: 1300 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 128 + 80$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 128 - 90$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 304N/m	
Log File: 080824-215115	Other:	
Test 31		
Training Time: 1300 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 128 + 70$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 128 - 70$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 304N/m	
Log File: 080824-221410	Other:	
Test 32		
Training Time: 1800 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 128 + 70$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 128 - 70$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 304N/m	
Log File: 080824-224434	Other:	
Test 33		
Training Time: 1300 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 128 + 50$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 128 - 50$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 304N/m	
Log File: 080824-230642	Other:	
Test 34		
Training Time: 1900 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 128 + 50$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 128 - 50$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 304N/m	
Log File: 080824-233908	Other:	
Test 35		
Training Time: 1300 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 128 + 30$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 128 - 30$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 304N/m	
Log File: 080825-000136	Other:	

Test 36		
Training Time: 1300 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 128 + 30$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 128 - 30$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 304N/m	
Log File: 080825-002340	Other:	
Test 37		
Training Time: 1300 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 224$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 96$ bits	$T_2 = 150$ samples	$\theta_2 = -60^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080827-144726	Other:	
Test 38		
Training Time: 1300 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 224$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 96$ bits	$T_2 = 150$ samples	$\theta_2 = -60^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080827-150927	Other:	
Test 39		
Training Time: 1300 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 224$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 96$ bits	$T_2 = 150$ samples	$\theta_2 = -45^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080827-153444	Other:	
Test 40		
Training Time: 1300 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 224$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 96$ bits	$T_2 = 150$ samples	$\theta_2 = -45^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080827-155800	Other:	
Test 41		
Training Time: 1300 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 224$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 96$ bits	$T_2 = 150$ samples	$\theta_2 = -30^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080827-164058	Other:	

Test 42		
Training Time: 1300 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 224$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 96$ bits	$T_2 = 150$ samples	$\theta_2 = -30^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080827-170306	Other:	
Test 43		
Training Time: 1300 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 224$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 96$ bits	$T_2 = 150$ samples	$\theta_2 = -15^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080827-172610	Other:	
Test 44		
Training Time: 1300 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 224$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 96$ bits	$T_2 = 150$ samples	$\theta_2 = -15^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080827-174812	Other:	
Test 45		
Training Time: 1300 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 224$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 96$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080828-122654	Other:	
Test 46		
Training Time: 1300 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 224$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 96$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080828-124858	Other:	
Test 47		
Training Time: 1300 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 224$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 64$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080828-131138	Other:	

Test 48		
Training Time: 1300 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 224$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 64$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080828-133342	Other:	
Test 49		
Training Time: 1300 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 224$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 64$ bits	$T_2 = 150$ samples	$\theta_2 = 15^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080828-135656	Other:	
Test 50		
Training Time: 1300 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 224$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 64$ bits	$T_2 = 150$ samples	$\theta_2 = 15^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080828-141921	Other:	
Test 51		
Training Time: 1300 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 224$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 64$ bits	$T_2 = 150$ samples	$\theta_2 = 30^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080828-144242	Other:	
Test 52		
Training Time: 1300 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 224$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 64$ bits	$T_2 = 150$ samples	$\theta_2 = 30^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080828-150656	Other:	
Test 53		
Training Time: 1300 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 224$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 64$ bits	$T_2 = 150$ samples	$\theta_2 = 45^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080828-153352	Other:	

Test 54		
Training Time: 1300 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 224$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 64$ bits	$T_2 = 150$ samples	$\theta_2 = 45^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080828-155550	Other:	
Test 55		
Training Time: 1300 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 224$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 64$ bits	$T_2 = 150$ samples	$\theta_2 = 60^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080828-161752	Other:	
Test 56		
Training Time: 1300 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 224$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 64$ bits	$T_2 = 150$ samples	$\theta_2 = 60^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080828-163955	Other:	
Test 57		
Training Time: 1300 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 224$ bits	$T_1 = 150$ samples	$\theta_1 = -15^\circ$
$y_{min} = 64$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080830-135418	Other:	
Test 58		
Training Time: 1300 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 224$ bits	$T_1 = 150$ samples	$\theta_1 = -15^\circ$
$y_{min} = 64$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080830-141653	Other:	
Test 59		
Training Time: 1300 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 224$ bits	$T_1 = 150$ samples	$\theta_1 = -30^\circ$
$y_{min} = 64$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080830-144949	Other:	

Test 60		
Training Time: 1300 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 224$ bits	$T_1 = 150$ samples	$\theta_1 = -30^\circ$
$y_{min} = 64$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080830-151614	Other:	
Test 61		
Training Time: 1300 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 224$ bits	$T_1 = 150$ samples	$\theta_1 = -45^\circ$
$y_{min} = 64$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080830-153923	Other:	
Test 62		
Training Time: 1300 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 224$ bits	$T_1 = 150$ samples	$\theta_1 = -45^\circ$
$y_{min} = 64$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080830-160133	Other:	
Test 63		
Training Time: 1300 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 224$ bits	$T_1 = 150$ samples	$\theta_1 = -60^\circ$
$y_{min} = 64$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080830-162733	Other:	
Test 64		
Training Time: 1300 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 224$ bits	$T_1 = 150$ samples	$\theta_1 = -60^\circ$
$y_{min} = 64$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080830-164935	Other:	
Test 65		
Training Time: 1300 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 160$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 32$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080901-145918	Other:	

Test 66		
Training Time: 1300 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 160$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 32$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080901-152139	Other:	
Test 67		
Training Time: 1300 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 160$ bits	$T_1 = 150$ samples	$\theta_1 = 15^\circ$
$y_{min} = 32$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080901-154753	Other:	
Test 68		
Training Time: 1300 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 160$ bits	$T_1 = 150$ samples	$\theta_1 = 15^\circ$
$y_{min} = 32$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080901-160952	Other:	
Test 69		
Training Time: 1300 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 160$ bits	$T_1 = 150$ samples	$\theta_1 = 30^\circ$
$y_{min} = 32$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080901-163254	Other:	
Test 70		
Training Time: 1300 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 160$ bits	$T_1 = 150$ samples	$\theta_1 = 30^\circ$
$y_{min} = 32$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080901-165459	Other:	
Test 71		
Training Time: 1300 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 160$ bits	$T_1 = 150$ samples	$\theta_1 = 45^\circ$
$y_{min} = 32$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080901-171922	Other:	

Test 72		
Training Time: 1300 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 160$ bits	$T_1 = 150$ samples	$\theta_1 = 45^\circ$
$y_{min} = 32$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 080901-174120	Other:	
Test 73		
Training Time: 1000 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 176$ bits	$T_1 = 150$ samples	$\theta_1 = 45^\circ$
$y_{min} = 80$ bits	$T_2 = 150$ samples	$\theta_2 = -45^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 090202-194946	Other:	
Test 74		
Training Time: 1000 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 176$ bits	$T_1 = 150$ samples	$\theta_1 = 45^\circ$
$y_{min} = 80$ bits	$T_2 = 150$ samples	$\theta_2 = -45^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 090202-200707	Other:	
Test 75		
Training Time: 1000 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 176$ bits	$T_1 = 150$ samples	$\theta_1 = 22^\circ$
$y_{min} = 80$ bits	$T_2 = 150$ samples	$\theta_2 = -45^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 090202-202549	Other:	
Test 76		
Training Time: 1000 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 176$ bits	$T_1 = 150$ samples	$\theta_1 = 22^\circ$
$y_{min} = 80$ bits	$T_2 = 150$ samples	$\theta_2 = -45^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 090202-204301	Other:	
Test 77		
Training Time: 1000 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 176$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 80$ bits	$T_2 = 150$ samples	$\theta_2 = -45^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 090202-210059	Other:	

Test 78		
Training Time: 1000 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 176$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 80$ bits	$T_2 = 150$ samples	$\theta_2 = -45^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 090202-220251	Other:	
Test 79		
Training Time: 1000 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 176$ bits	$T_1 = 150$ samples	$\theta_1 = -22^\circ$
$y_{min} = 80$ bits	$T_2 = 150$ samples	$\theta_2 = -45^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 090202-234342	Other:	
Test 80		
Training Time: 1000 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 176$ bits	$T_1 = 150$ samples	$\theta_1 = -22^\circ$
$y_{min} = 80$ bits	$T_2 = 150$ samples	$\theta_2 = -45^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 090203-000152	Other:	
Test 81		
Training Time: 1000 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 176$ bits	$T_1 = 150$ samples	$\theta_1 = -45^\circ$
$y_{min} = 80$ bits	$T_2 = 150$ samples	$\theta_2 = -45^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 090202-230734	Other:	
Test 82		
Training Time: 1000 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 176$ bits	$T_1 = 150$ samples	$\theta_1 = -45^\circ$
$y_{min} = 80$ bits	$T_2 = 150$ samples	$\theta_2 = -45^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 090202-232453	Other:	
Test 83		
Training Time: 1000 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 176$ bits	$T_1 = 150$ samples	$\theta_1 = 45^\circ$
$y_{min} = 80$ bits	$T_2 = 150$ samples	$\theta_2 = -22^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 090203-155847	Other:	

Test 84		
Training Time: 1000 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 176$ bits	$T_1 = 150$ samples	$\theta_1 = 22^\circ$
$y_{min} = 80$ bits	$T_2 = 150$ samples	$\theta_2 = -22^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 090203-154105	Other:	
Test 85		
Training Time: 1000 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 176$ bits	$T_1 = 150$ samples	$\theta_1 = 22^\circ$
$y_{min} = 80$ bits	$T_2 = 150$ samples	$\theta_2 = -22^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 090203-161611	Other:	
Test 86		
Training Time: 1000 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 176$ bits	$T_1 = 150$ samples	$\theta_1 = 22^\circ$
$y_{min} = 80$ bits	$T_2 = 150$ samples	$\theta_2 = -22^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 090203-163315	Other:	
Test 87		
Training Time: 1000 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 176$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 80$ bits	$T_2 = 150$ samples	$\theta_2 = -22^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 090203-165253	Other:	
Test 88		
Training Time: 1000 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 176$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 80$ bits	$T_2 = 150$ samples	$\theta_2 = -22^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 090203-170949	Other:	
Test 89		
Training Time: 1000 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 176$ bits	$T_1 = 150$ samples	$\theta_1 = -22^\circ$
$y_{min} = 80$ bits	$T_2 = 150$ samples	$\theta_2 = -22^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 090203-172658	Other:	

Test 90		
Training Time: 1000 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 176$ bits	$T_1 = 150$ samples	$\theta_1 = -22^\circ$
$y_{min} = 80$ bits	$T_2 = 150$ samples	$\theta_2 = -22^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 090203-174355	Other:	
Test 91		
Training Time: 1000 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 176$ bits	$T_1 = 150$ samples	$\theta_1 = -45^\circ$
$y_{min} = 80$ bits	$T_2 = 150$ samples	$\theta_2 = -22^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 090203-180130	Other:	
Test 92		
Training Time: 1000 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 176$ bits	$T_1 = 150$ samples	$\theta_1 = -45^\circ$
$y_{min} = 80$ bits	$T_2 = 150$ samples	$\theta_2 = -22^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 090203-181829	Other:	
Test 93		
Training Time: 1000 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 176$ bits	$T_1 = 150$ samples	$\theta_1 = 45^\circ$
$y_{min} = 80$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 090204-142120	Other:	
Test 94		
Training Time: 1000 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 176$ bits	$T_1 = 150$ samples	$\theta_1 = 45^\circ$
$y_{min} = 80$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 090204-143821	Other:	
Test 95		
Training Time: 1000 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 176$ bits	$T_1 = 150$ samples	$\theta_1 = 22^\circ$
$y_{min} = 80$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 090204-150021	Other:	

Test 96		
Training Time: 1000 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 176$ bits	$T_1 = 150$ samples	$\theta_1 = 22^\circ$
$y_{min} = 80$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 090204-151730	Other:	
Test 97		
Training Time: 1000 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 176$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 80$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 090204-153446	Other:	
Test 98		
Training Time: 1000 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 176$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 80$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 090204-155152	Other:	
Test 99		
Training Time: 1000 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 176$ bits	$T_1 = 150$ samples	$\theta_1 = -22^\circ$
$y_{min} = 80$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 090204-160911	Other:	
Test 100		
Training Time: 1000 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 176$ bits	$T_1 = 150$ samples	$\theta_1 = -22^\circ$
$y_{min} = 80$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 090204-162620	Other:	
Test 101		
Training Time: 1000 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 176$ bits	$T_1 = 150$ samples	$\theta_1 = -45^\circ$
$y_{min} = 80$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 090204-164422	Other:	

Test 102		
Training Time: 1000 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 176$ bits	$T_1 = 150$ samples	$\theta_1 = -45^\circ$
$y_{min} = 80$ bits	$T_2 = 150$ samples	$\theta_2 = 0^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 090204-170128	Other:	
Test 103		
Training Time: 1000 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 176$ bits	$T_1 = 150$ samples	$\theta_1 = 45^\circ$
$y_{min} = 80$ bits	$T_2 = 150$ samples	$\theta_2 = 22^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 090205-133013	Other:	
Test 104		
Training Time: 1000 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 176$ bits	$T_1 = 150$ samples	$\theta_1 = 45^\circ$
$y_{min} = 80$ bits	$T_2 = 150$ samples	$\theta_2 = 22^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 090205-134712	Other:	
Test 105		
Training Time: 1000 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 176$ bits	$T_1 = 150$ samples	$\theta_1 = 22^\circ$
$y_{min} = 80$ bits	$T_2 = 150$ samples	$\theta_2 = 22^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 090205-140412	Other:	
Test 106		
Training Time: 1000 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 176$ bits	$T_1 = 150$ samples	$\theta_1 = 22^\circ$
$y_{min} = 80$ bits	$T_2 = 150$ samples	$\theta_2 = 22^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 090205-142120	Other:	
Test 107		
Training Time: 1000 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 176$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 80$ bits	$T_2 = 150$ samples	$\theta_2 = 22^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 090205-143827	Other:	

Test 108		
Training Time: 1000 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 176$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 80$ bits	$T_2 = 150$ samples	$\theta_2 = 22^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 090205-145527	Other:	
Test 109		
Training Time: 1000 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 176$ bits	$T_1 = 150$ samples	$\theta_1 = -22^\circ$
$y_{min} = 80$ bits	$T_2 = 150$ samples	$\theta_2 = 22^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 090205-151230	Other:	
Test 110		
Training Time: 1000 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 176$ bits	$T_1 = 150$ samples	$\theta_1 = -22^\circ$
$y_{min} = 80$ bits	$T_2 = 150$ samples	$\theta_2 = 22^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 090205-152927	Other:	
Test 111		
Training Time: 1000 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 176$ bits	$T_1 = 150$ samples	$\theta_1 = -45^\circ$
$y_{min} = 80$ bits	$T_2 = 150$ samples	$\theta_2 = 22^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 090205-154629 and 090210-175458	Other:	
Test 112		
Training Time: 1000 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 176$ bits	$T_1 = 150$ samples	$\theta_1 = -45^\circ$
$y_{min} = 80$ bits	$T_2 = 150$ samples	$\theta_2 = 22^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 090205-160335 and 090210-181247	Other:	
Test 113		
Training Time: 1000 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 176$ bits	$T_1 = 150$ samples	$\theta_1 = 45^\circ$
$y_{min} = 80$ bits	$T_2 = 150$ samples	$\theta_2 = 45^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 090205-162110	Other:	

Test 114		
Training Time: 1000 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 176$ bits	$T_1 = 150$ samples	$\theta_1 = 45^\circ$
$y_{min} = 80$ bits	$T_2 = 150$ samples	$\theta_2 = 45^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 090205-163811	Other:	
Test 115		
Training Time: 1000 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 176$ bits	$T_1 = 150$ samples	$\theta_1 = 22^\circ$
$y_{min} = 80$ bits	$T_2 = 150$ samples	$\theta_2 = 45^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 090205-165514	Other:	
Test 116		
Training Time: 1000 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 176$ bits	$T_1 = 150$ samples	$\theta_1 = 22^\circ$
$y_{min} = 80$ bits	$T_2 = 150$ samples	$\theta_2 = 45^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 090205-171235	Other:	
Test 117		
Training Time: 1000 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 176$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 80$ bits	$T_2 = 150$ samples	$\theta_2 = 45^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 090205-173036 and 090205-193202	Other:	
Test 118		
Training Time: 1000 seconds	EMGC Status: OFF	
Training Function Parameters:		
$y_{max} = 176$ bits	$T_1 = 150$ samples	$\theta_1 = 0^\circ$
$y_{min} = 80$ bits	$T_2 = 150$ samples	$\theta_2 = 45^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 090205-174733 and 090205-191454	Other:	
Test 119		
Training Time: 1000 seconds	EMGC Status: ON	
Training Function Parameters:		
$y_{max} = 176$ bits	$T_1 = 150$ samples	$\theta_1 = -22^\circ$
$y_{min} = 80$ bits	$T_2 = 150$ samples	$\theta_2 = 45^\circ$
Test Apparatus: Single air-muscle	Spring Return Constant (k) = 608N/m	
Log File: 090205-180638 and 090210-163409	Other:	

Test 120

Training Time: 1000 seconds

EMGC Status: OFF

Training Function Parameters:

$y_{max} = 176 \text{ bits}$

$T_1 = 150 \text{ samples}$

$\theta_1 = -22^\circ$

$y_{min} = 80 \text{ bits}$

$T_2 = 150 \text{ samples}$

$\theta_2 = 45^\circ$

Test Apparatus: Single air-muscle

Spring Return Constant (k) = 608N/m

Log File: 090205-182338 and 090210-165113

Other:

Test 121

Training Time: 1000 seconds

EMGC Status: ON

Training Function Parameters:

$y_{max} = 176 \text{ bits}$

$T_1 = 150 \text{ samples}$

$\theta_1 = -45^\circ$

$y_{min} = 80 \text{ bits}$

$T_2 = 150 \text{ samples}$

$\theta_2 = 45^\circ$

Test Apparatus: Single air-muscle

Spring Return Constant (k) = 608N/m

Log File: 090205-184036

Other:

Test 122

Training Time: 1000 seconds

EMGC Status: OFF

Training Function Parameters:

$y_{max} = 176 \text{ bits}$

$T_1 = 150 \text{ samples}$

$\theta_1 = -45^\circ$

$y_{min} = 80 \text{ bits}$

$T_2 = 150 \text{ samples}$

$\theta_2 = 45^\circ$

Test Apparatus: Single air-muscle

Spring Return Constant (k) = 608N/m

Log File: 090205-185730

Other:

G.0 RESULTS FROM TEST 73 TO TEST 122

The results presented here this section use an average Error Ratio and average Error Gain, obtained from the converged portion of the Total Errors and Error Ratios obtained for each test.

Test	θ_1 (departure)	θ_2 (approach)	Error Ratio	Error Gain
073	45	-45	0.66	-12.06
075	45	-22	0.58	-7.15
077	45	0	0.58	-5.38
079	45	22	0.52	-1.96
081	45	45	0.47	-1.15
083	22	-45	0.62	-12.12
085	22	-22	0.55	-7.84
087	22	0	0.51	-4.85
089	22	22	0.47	-3.56
091	22	45	0.38	-0.81
093	0	-45	0.54	-11.61
095	0	-22	0.44	-5.30
097	0	0	0.41	-3.09
099	0	22	0.29	-4.60
101	0	45	0.25	0.01
103	-22	-45	0.53	-9.02
105	-22	-22	0.45	-3.15
107	-22	0	0.40	0.75
109	-22	22	0.32	-0.46
111	-22	45	0.27	0.28
113	-45	-45	0.52	-10.86
115	-45	-22	0.47	-3.59
117	-45	0	0.38	-2.20
119	-45	22	0.34	-0.47
121	-45	45	0.30	2.91

Table G.1 – Average Error Ratio and Error Gains for Tests 73 to 122

H.0 ADDITIONAL APPROACH AND DEPARTURE GRADIENT TESTS

This Appendix provides an additional study in situation approach and departure gradient tests. These tests were performed with either the approach or departure gradient fixed at 0° , independently testing approach and departure gradients separately. All initial Error Ratios for these tests exist above 0.4, thus the EMGC reduces the Total Error for all tests conducted in this Appendix.

Contained on Thesis CD.

I.0 PRELIMINARY PBV CALIBRATION TESTS

These tests were used to better understand the properties of the PBVs and were used to help design the calibration procedure in Section D.5.

Contained on Thesis CD.

H.0 ADDITIONAL APPROACH AND DEPARTURE GRADIENT TESTS

This Appendix provides an additional look addition situation approach and departure gradient tests. These tests were performed with either the approach or departure gradient fixed at 0° , independently testing approach and departure gradients separately. All initial Error Ratios for these tests exist above 0.4, thus the EMGC reduces the Total Error for all tests conducted in this Appendix.

H.1 SITUATION DEPARTURE GRADIENTS

As illustrated in Figure 9.2, the departure gradients for a situation are defined by a situation's θ_1 value. A variety of gradients for situation departure trajectories were used to test the performance of the EMGC not only in terms of successfully producing EMGs through the situation, but also in terms of the error minimising capabilities of the EMGC for situations with varying departure velocities.

The series of tests used in testing the EMGC's performance throughout different departure gradients, are separated into both positive and negative departure gradients. The reason for this separation is that the situation step size used in the following tests change for the negative and positive gradient tests. The reason for this is due to the limited range of motion available from the air-muscle actuation stroke. To produce a situation with a larger and clearer Total Error, a larger situation step size was chosen to adequately test the range of gradients from 0 to 60° in both the positive and negative directions of motion (for the departure gradients). The negative gradient tests will first be presented followed by the positive gradient tests.

To start off with, the equalisation of a situation's errors by the EMGC convergence will be presented, as illustrated by the error-ratio vs. EMGC iteration plot in Figure H.1.

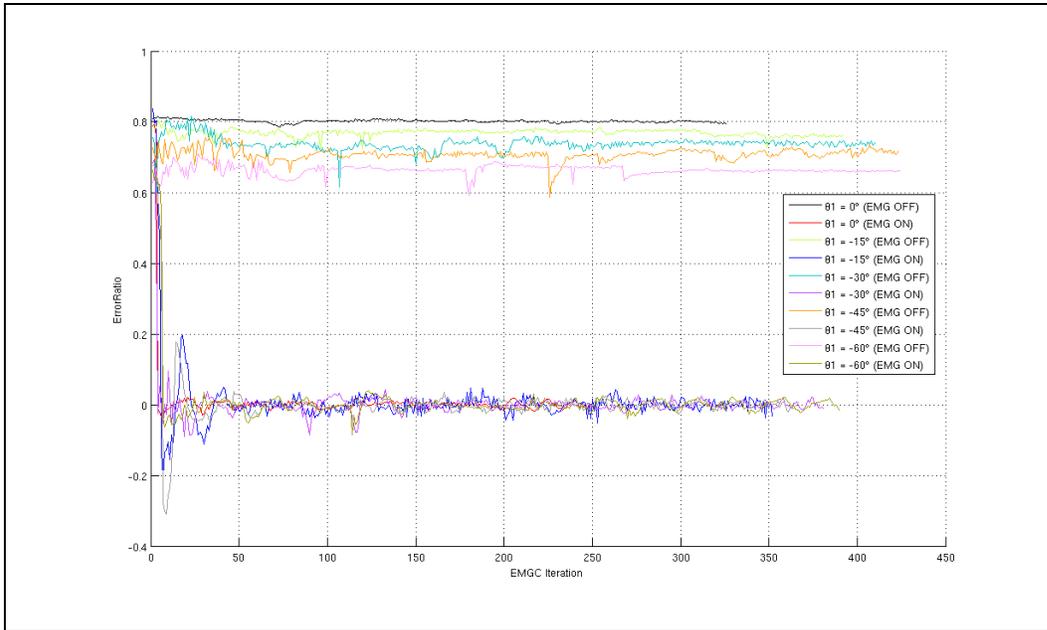


Figure H.1 – Situation Departure Gradient ($-\theta_1$) Tests, ErrorRatio vs. Iteration (Test 47 to Test 64)

In Figure H.1 we see that for each situation, the EMGC successfully converges the Error Ratio to zero from the controllers inherent initial Error Ratio, which in these tests is between 0.6 to 0.8. It is interesting to note that for the traces produced when the EMGC is On, when the departure gradient equals -15° and -45° , there is a clear initial oscillatory trend for the Error Ratio before it stabilises to zero. This oscillatory pattern is directly related to the EMG shift rate value used in the EMGC algorithm (see Section 6.2.2.7). Acting in much the same way as the proportional gain term in a PID controller, the EMG shift rate value controls the degree of oscillation seen here. It is important to note that even though this oscillation does occur, the EMGC does successfully converge all Error Ratios to zero.

For the same tests, the relationship between the situation's Total Error against the EMGC's training iteration is illustrated in Figure H.2.

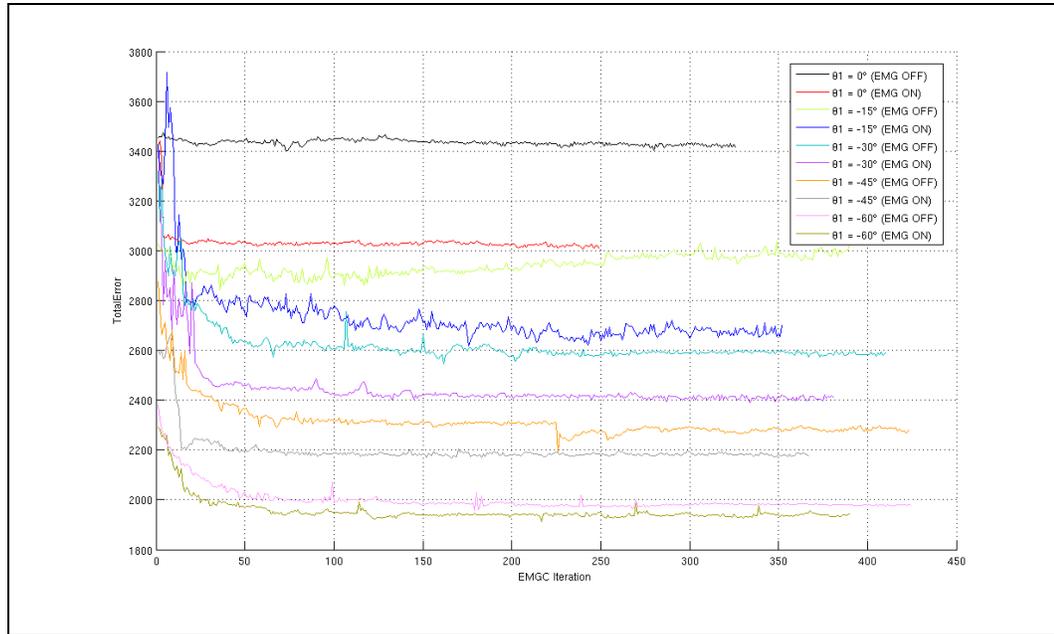


Figure H.2 – Situation Departure Gradient ($-\theta_1$) Tests, TotalError vs. Iteration (Test 47 to Test 64)

Figure H.2 illustrates the Total Error at each EMGC training interval for a situation with negative departure gradients. For each test, regardless of whether the EMGC is either on or off, convergence of the error always occurs. This is more evident for some traces than others, however compared to the size of the Total Error, a continual slight fluctuation in the Total Error over time appears insignificant overall. The dynamic nature of the physical system, together with a slight degree of noise experienced in the feedback sensor are possible reasons for any convergent fluctuations experienced. The consistently dynamic online learning capabilities of both the EMGC CMAC and the CMAC+PID CMAC, the resolution of the feedback sensor and the generalisation parameters in the CMACs, provide a number of possible reasons for the bumps and spikes evident in the converged errors in Figure H.2. However all reasons aside, the Total Errors for each trace still converge satisfactorily.

From Figure H.1 and Figure H.2 it is evident that the EMGC successfully converges both the Error Ratio and the Total Error's associated with a situation with negative departure gradients. However to best illustrate the effect the changing negative departure gradients have on a situations Total Error, a 3-D representation of Figure H.2 against the departure gradients is provided in Figure H.3.

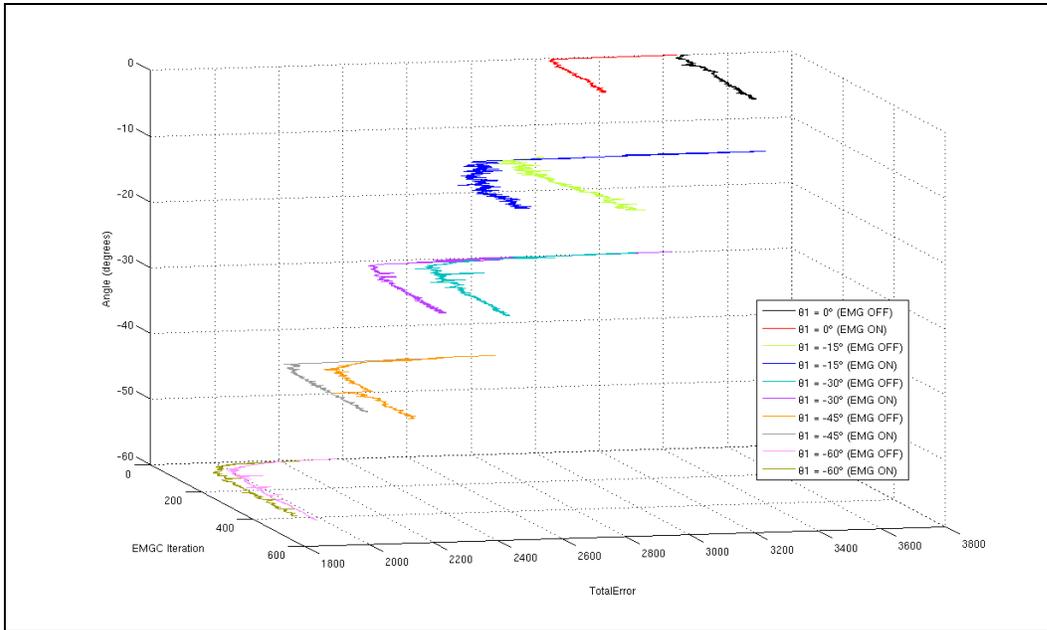


Figure H.3 – Situation Departure Gradient ($-\theta_1$) Tests, TotalError vs. Iteration vs. θ_1 (Test 47 to Test 64)

Figure H.3 illustrates the relationship that the converged Total Errors against the EMGC iteration have with respect to the associated departure gradient. Most evident from Figure H.3 is the decreasing Total Error pattern as the departure gradient increases from 0° to -60° . Also evident is the decreasing gap between the Total Error for when the EMGC is off and on for any particular departure gradient. This relationship trend is best compared using the Error Gain for each of the Total Error traces for each departure gradient by use of Equation 8.1. Figure H.4 illustrates the comparison between the Error Gain, the departure gradient and the EMGC training iteration.

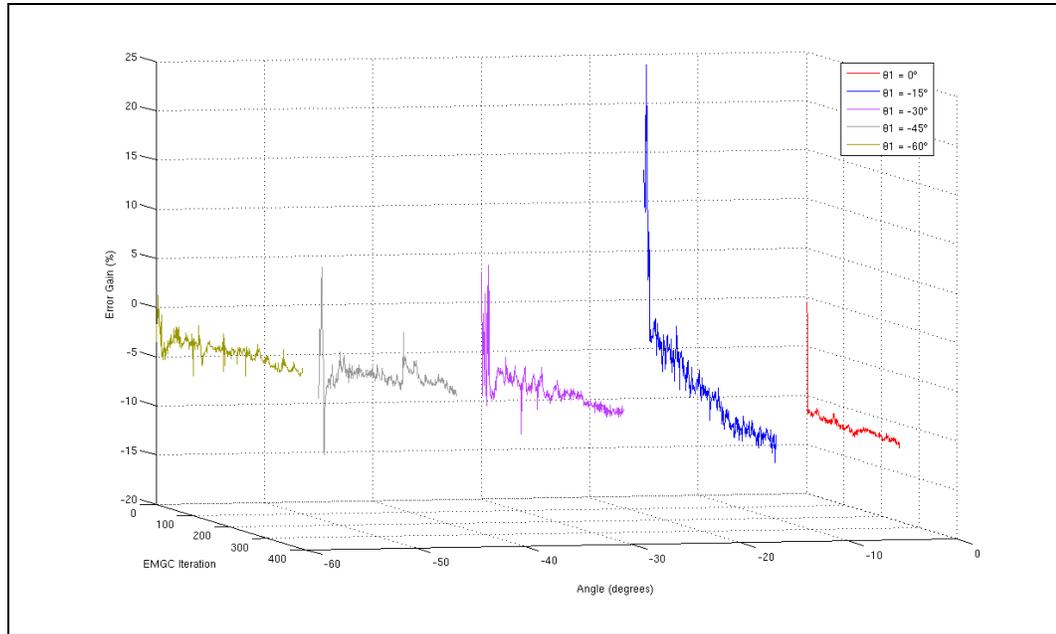


Figure H.4 – Situation Departure Gradient ($-\theta_1$) Tests, ErrorGain vs. Iteration vs. θ_1 (Test 47 to Test 64)

Figure H.4 illustrates the relationship that the Error Gains for the Total Errors in Figure H.3 have against the EMGC iteration with respect to the associated departure gradient. Evident here is the continual increase in Error Gain (in terms of magnitude) as the departure gradient increases from -60° to 0° . The largest Error Gain produced from the tests is approximately -12% for when the departure gradient is equal to 0° . The smallest Error Gain is approximately -4% for when the departure gradient is equal to -60° . The use of the EMGC reduces each situation's Total Error for all negative departure gradients, evident in Figure H.4 where all Error Gains are below zero.

The positive gradient tests for the departure gradients will now be presented, starting off with the equalisation of a situation's errors by the EMGC convergence, as illustrated by the error-ratio vs. EMGC iteration plot in Figure H.5.

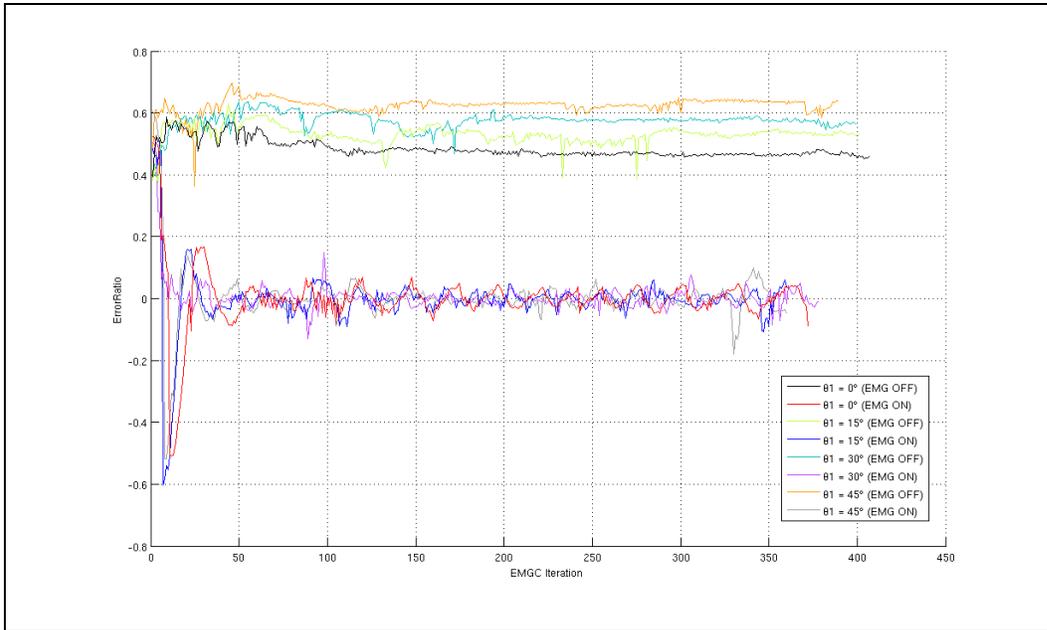


Figure H.5 – Situation Departure Gradient ($+\theta_1$) Tests, ErrorRatio vs. Iteration (Test 65 to Test 72)

Figure H.5 shows that the EMGC converges the Error Ratio successfully to zero, with a slight oscillation occurring for most of the EMGC ON tests. In comparison to their equivalent EMGC OFF tests however, it is evident that the EMGC greatly reduces the error from its initial value, sitting at an Error Ratio of approximately 0.5 for all situations when the EMGC is not being used. Each trace stabilises at a roughly consistent Error Ratio value, ignoring the noise related spikes present in the system. Note that only departure gradients from 0° to 45° were used in these tests, as a departure gradient of 60° resulted in a desired trajectory whose limits were beyond the position limits of the actuator.

For the same test, the relationship between the situation's Total Error against the EMGC's training iteration is illustrated in Figure H.6.

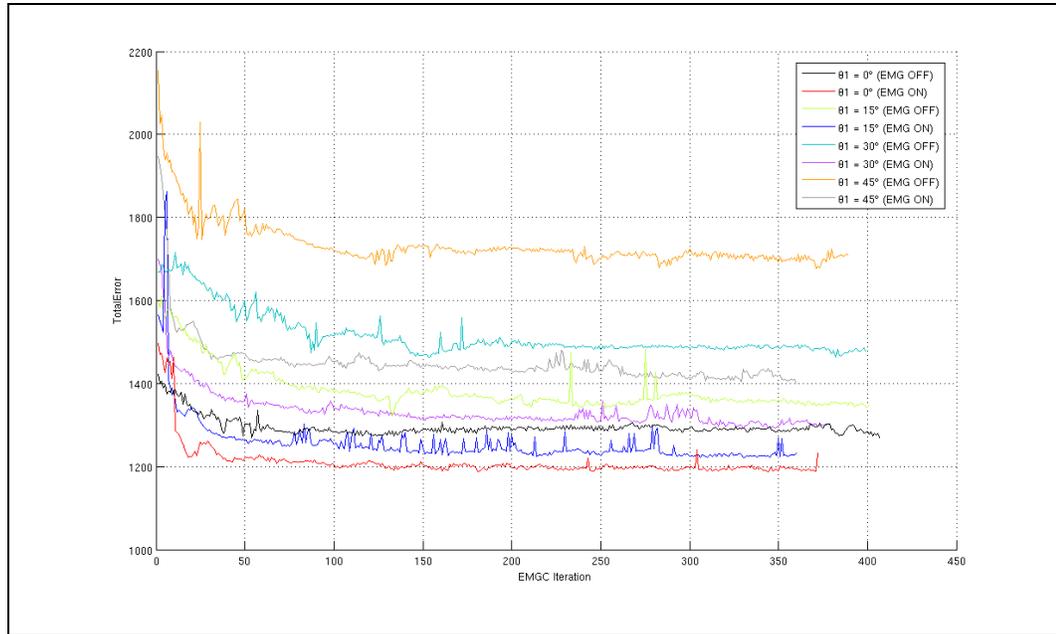


Figure H.6 – Situation Departure Gradient ($+\theta_1$) Tests, TotalError vs. Iteration (Test 65 to Test 72)

Figure H.6 illustrates the Total Error at each EMGC training interval for a situation with negative departure gradients. For each test, whether the EMGC is either on or off, stable convergence of the error always occurs. Fluctuations in the traces do occur due to sensor noise, especially evident for $\theta_1=15^\circ$ (EMG ON). The average pattern of each trace however indicates that the Total Error for each situation, whether the EMGC is on or off, converges to a stabilised value.

From Figure H.5 and Figure H.6 it is evident that the EMGC successfully converges both the Error Ratio and the Total Error's associated with a situation with positive departure gradients. To best illustrate the effect the changing positive departure gradients have on a situation's Total Error, a 3-D representation of Figure H.6 against the departure gradients is provided in Figure H.7.

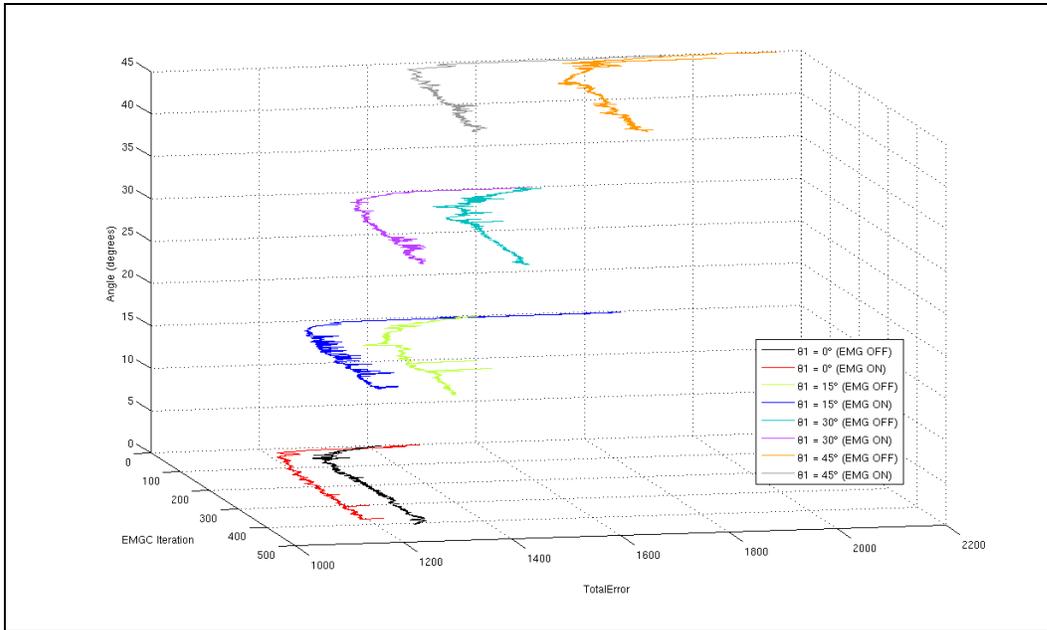


Figure H.7 – Situation Departure Gradient (+ θ_1) Tests, TotalError vs. Iteration vs. θ_1 (Test 65 to Test 72)

Figure H.7 illustrates the relationship that the converged Total Errors against the EMGC iteration have with respect to the associated departure gradient. Most evident from Figure H.7 is the decreasing Total Error pattern as the departure gradient decreases from 45° to 0° . Also evident is the decreasing gap between the Total Error for when the EMGC is off and on for any particular departure gradient. This relationship trend is best compared using the Error Gain for each of the Total Error traces for each departure gradient by use of Equation 8.1. Figure H.8 illustrates the comparison between the Error Gain, the departure gradient and the EMGC training iteration.

Figure H.8 illustrates the relationship that the Error Gains for the Total Errors in Figure H.7 have against the EMGC iteration with respect to the associated departure gradient. Evident here is the continual increase in Error Gain magnitude as the departure gradient increases from 0° to 45° . The largest Error Gain produced from the tests is approximately -17% for when the departure gradient is equal to 45° . The smallest Error Gain is approximately -7% for when the departure gradient is equal to 0° . However the use of the EMGC reduces each situation's Total Error for all positive departure gradients, evident in Figure H.8 where all Error Gains are below zero.

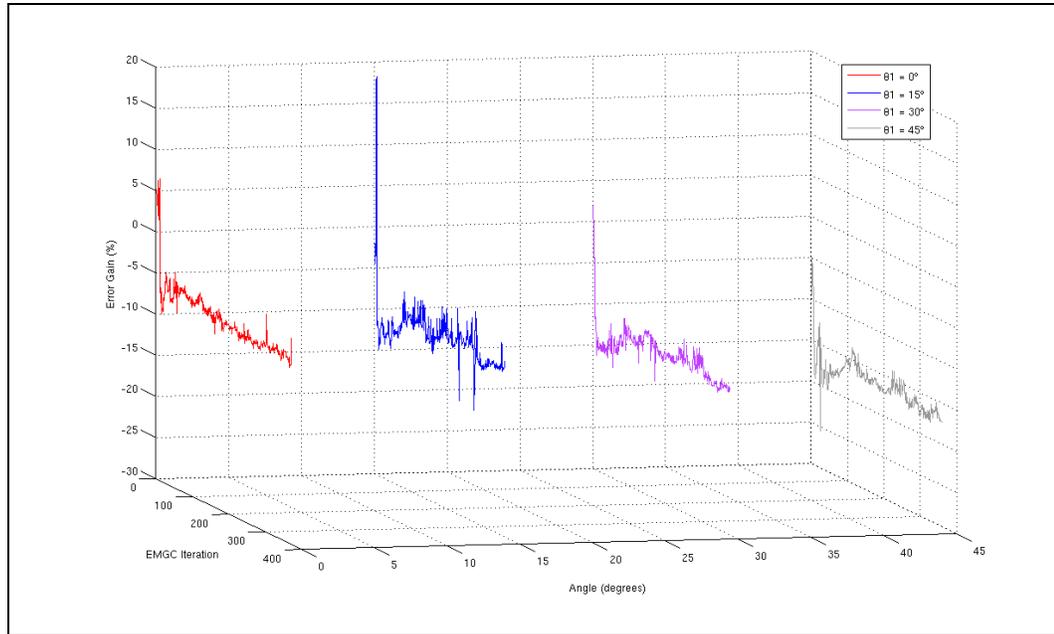


Figure H.8 – Departure Gradient (+ θ_1) Tests, ErrorGain vs. Iteration vs. θ_1 (Test 65 to Test 72)

The combination of tests presented above for both positive and negative departure gradients illustrate that the EMGC does indeed reduce the Total Error regardless of a situation's departure gradient. The EMGC's performance on a variety of approach gradients for situations will now be tested in a similar manner.

H.2 SITUATION APPROACH GRADIENTS

As illustrated in Figure 9.2, the approach gradient for a situation is defined by a situation's θ_2 value. A variety of gradients for situation approach trajectories were used to test the performance of the EMGC not only in terms of successfully producing EMGs throughout the situation, but also in terms of the error minimising capabilities of situations with varying approach gradients.

The series of tests for testing the EMGC's performance throughout different approach gradients are separated into both positive and negative approach gradients. The reason for this separation again is that the situation step size used in the following tests change for the negative and positive gradient tests. The reason for this is due to the limited range of motion available from the air-muscle actuation stroke length. To produce a situation with a larger and clearer error, a larger situation step size was chosen to adequately test the range of gradients from 0 to 60° in both the

positive and negative directions of motion (for the approach gradients). The negative gradient tests will first be presented followed by the positive gradient tests.

To start off with, the equalisation of a situation's errors by the EMGC convergence will be presented, as illustrated by the error-ratio vs. EMGC iteration plot in Figure H.9.

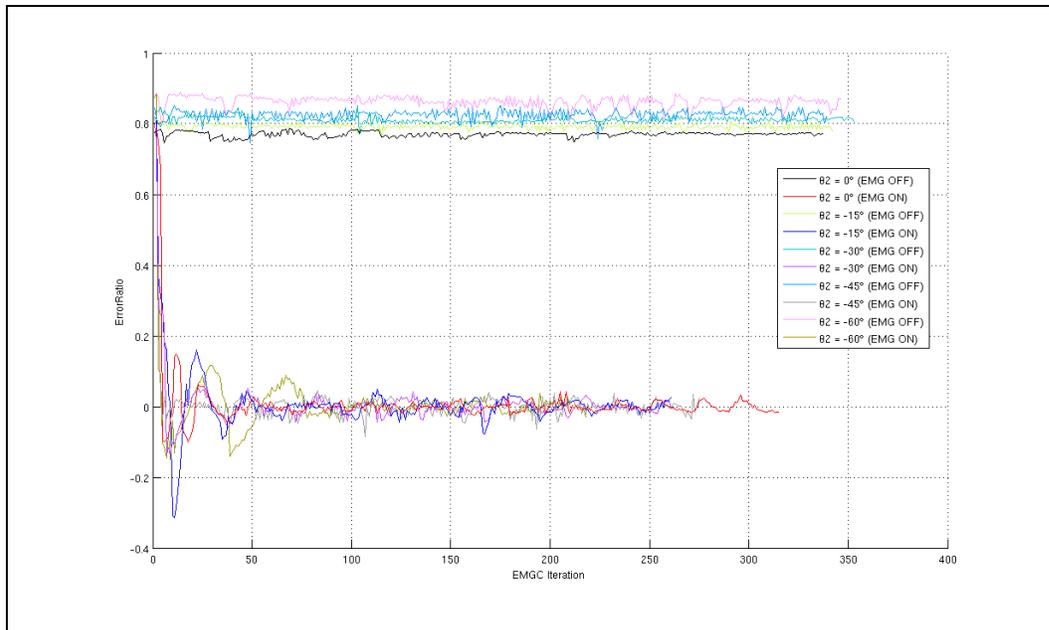


Figure H.9 – Situation Approach Gradient ($-\theta_2$) Tests, ErrorRatio vs. Iteration (Test 37 to Test 46)

In Figure H.9 we see that for each situation, the EMGC successfully converges the Error Ratio to zero. A slight decaying oscillatory response is observed for most of the traces produced when the EMGC is on, with the results converging to an Error Ratio of zero. This again is a result of the EMG shift rate value used to converge the Error Ratio to zero. Again the results show that unlike when the EMGC is off, the use of the EMGC results in a final Error Ratio of approximately zero.

From here on, unlike the results in Section H.1, the 3-D comparison showing the relationship between a situation's Total Error, the EMGC iteration and the approach angle, will be presented without analysing the Total Error vs. the EMGC iteration initially. The basic result pattern again is similar, and thus for conciseness, we will skip directly to the 3-D representation of the results, illustrated in Figure H.10.

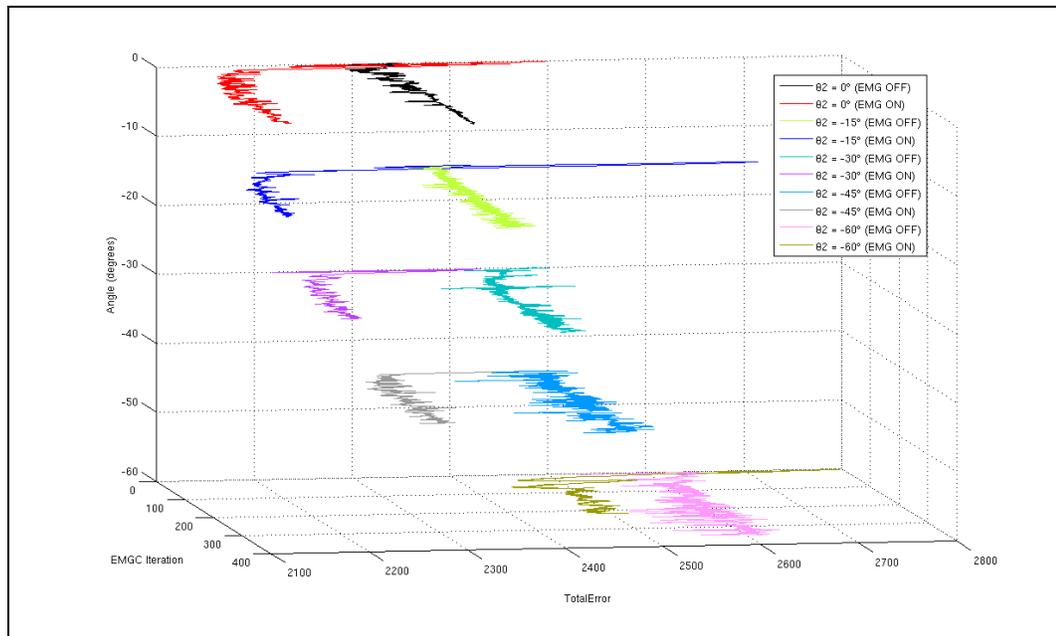


Figure H.10 – Situation Approach Gradient ($-\theta_2$) Tests, TotalError vs. Iteration vs. θ_2 (Test 37 to Test 46)

Figure H.10 illustrates the relationship that the converged Total Errors against the EMGC iteration have with respect to the associated approach gradient. An overall increasing trend in the Total Error as the approach angle decreases can be seen, however the trend is not as uniform as for the equivalent departure gradients plots. There is also an overall decreasing gap between the Total Error for when the EMGC is off and on for any particular approach gradient, but again is not as uniform as for the equivalent departure gradient plots. This relationship trend is best compared using the Error Gain for each of the Total Error traces for each departure gradient by use of Equation 8.1. Figure H.11 illustrates the comparison between the Error Gain, the departure gradient and the EMGC training iteration.

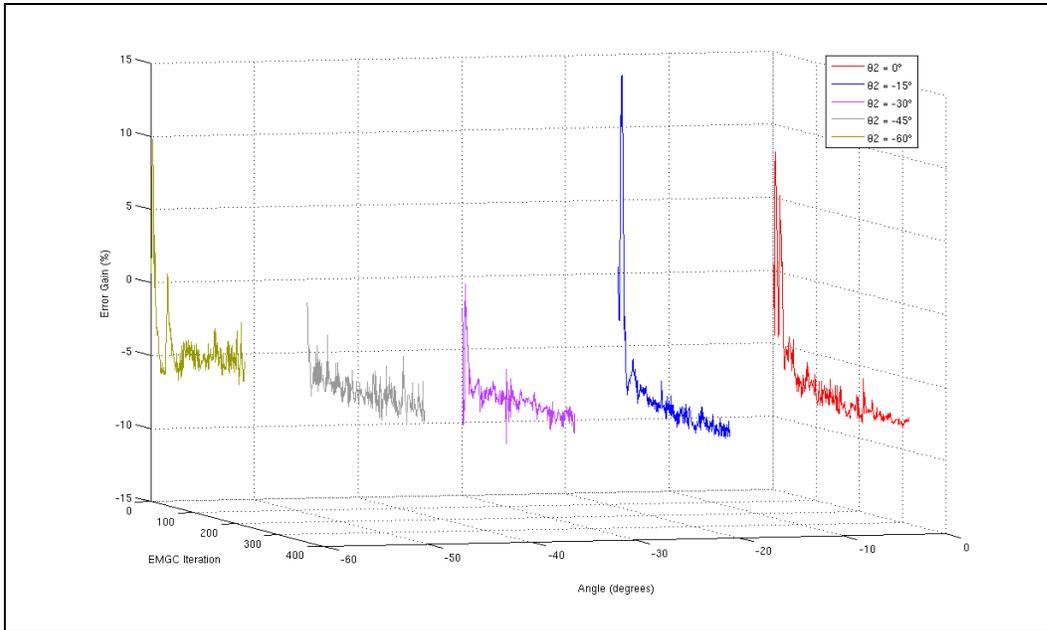


Figure H.11 – Situation Approach Gradient ($-\theta_2$) Tests, ErrorGain vs. Iteration vs. θ_2 (Test 37 to Test 46)

Figure H.11 illustrates the relationship that the Error Gains for the Total Errors in Figure H.10 have against the EMGC iteration with respect to the associated approach gradient. Evident here is the mostly continual increase in Error Gain magnitude as the departure gradient increases from -60° to 0° . The largest Error Gain produced from the tests is approximately -9% for when the departure gradient is equal to -15° . The smallest Error Gain is approximately -5% for when the departure gradient is equal to -60° . Again the use of the EMGC reduces each situation's Total Error for all negative departure gradients, evident in Figure H.11 where all Error Gains are below zero.

The positive gradient tests for the approach gradients will now be presented, starting off with the equalisation of a situation's errors by the EMGC convergence, as illustrated by the error-ratio vs. EMGC iteration plot in Figure H.12.

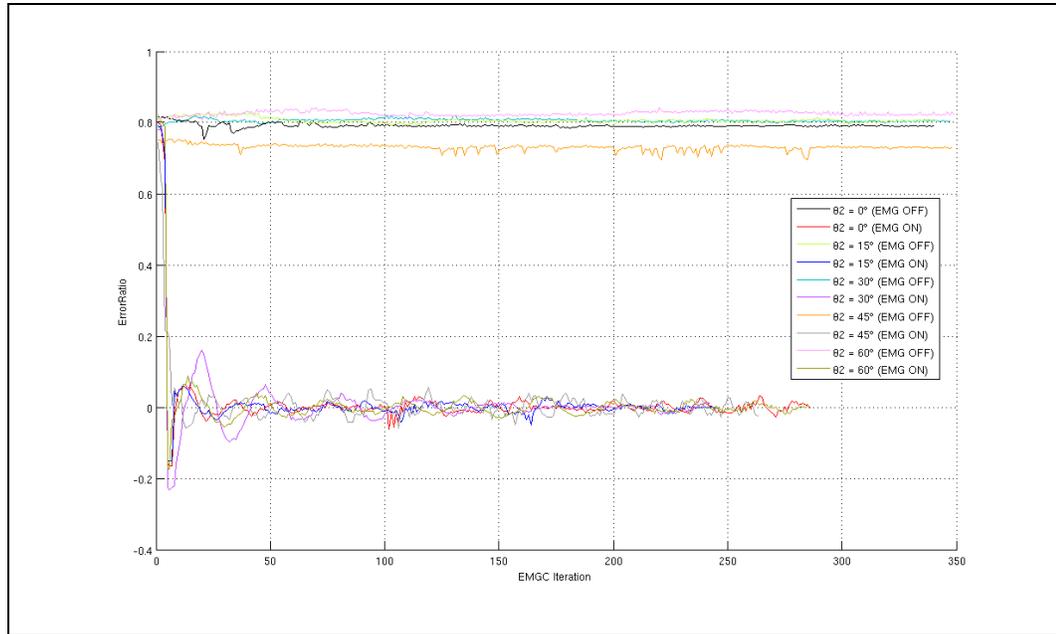


Figure H.12 – Situation Approach Gradient (+θ₂) Tests, ErrorRatio vs. Iteration (Test 47 to Test 56)

Figure H.12 shows again that the EMGC converges the Error Ratio successfully to zero with slight end fluctuations apparent. However in comparison to the Error Ratio values when the EMGC is not used, the use of the EMGC greatly minimises the Error Ratio for all positive approach gradients tested. A gradual oscillatory decay can be seen for a few of the approach gradients, especially evident when the approach gradient is equal to 60°. Sufficient stabilisation of the Error Ratio finally occurs however, stabilising approximately at zero as intended.

Again, the direct relationship comparison between the Total Error and the EMGC iteration like that presented in Section H.1 will be omitted, as the basic result pattern is similar. Thus for conciseness, we will skip directly to the 3-D representation of the results, illustrated in Figure H.13.

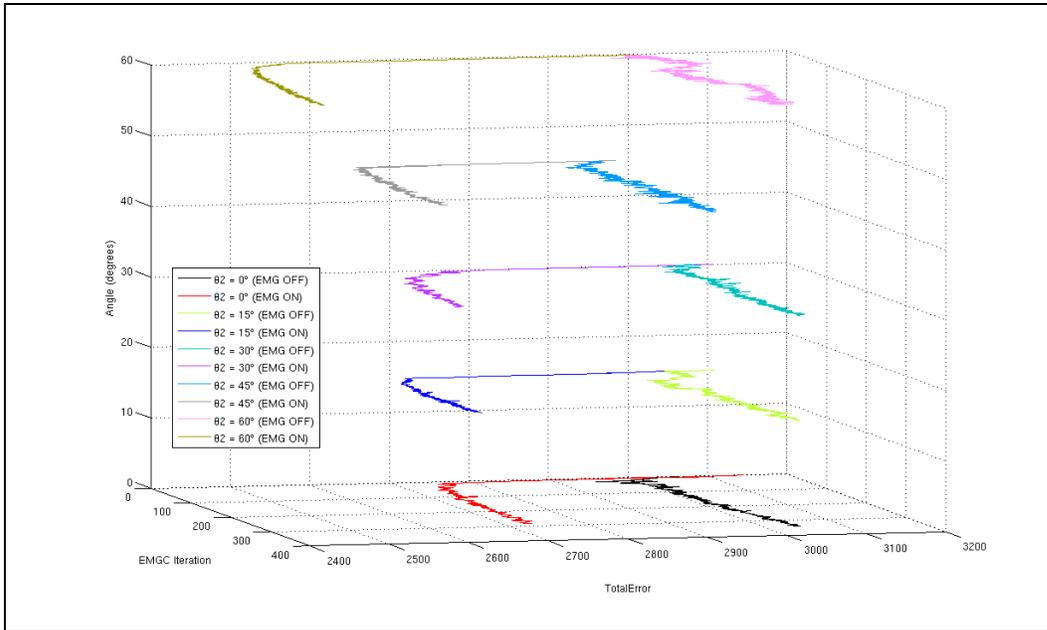


Figure H.13 – Situation Approach Gradient (+ θ_2) Tests, TotalError vs. Iteration vs. θ_2 (Test 47 to Test 56)

Figure H.13 illustrates the relationship that the converged Total Errors against the EMGC iteration have with respect to the associated approach gradient. An overall increasing trend in the Total Error as the approach gradient decreases can be seen, however again the trend is not as uniform as for the equivalent departure gradient plots. The traces representing the response of the system when the EMGC is ON produce a rather random distribution in terms of Total Error. Convergence is apparent for each plot, with some difficulty or confusion seen in the convergence of when the approach gradient equals 60° with the EMGC off. However the stabilised error value is within ± 100 Total Error bits, which when compared to the mean Total Error of approximately 3000 equates to only a 3% deviation. However the main point of these tests is to prove that the EMGC minimises the error for situations with various approach gradients, and thus the Error Gains relating to Figure H.13 are illustrated in Figure H.14.

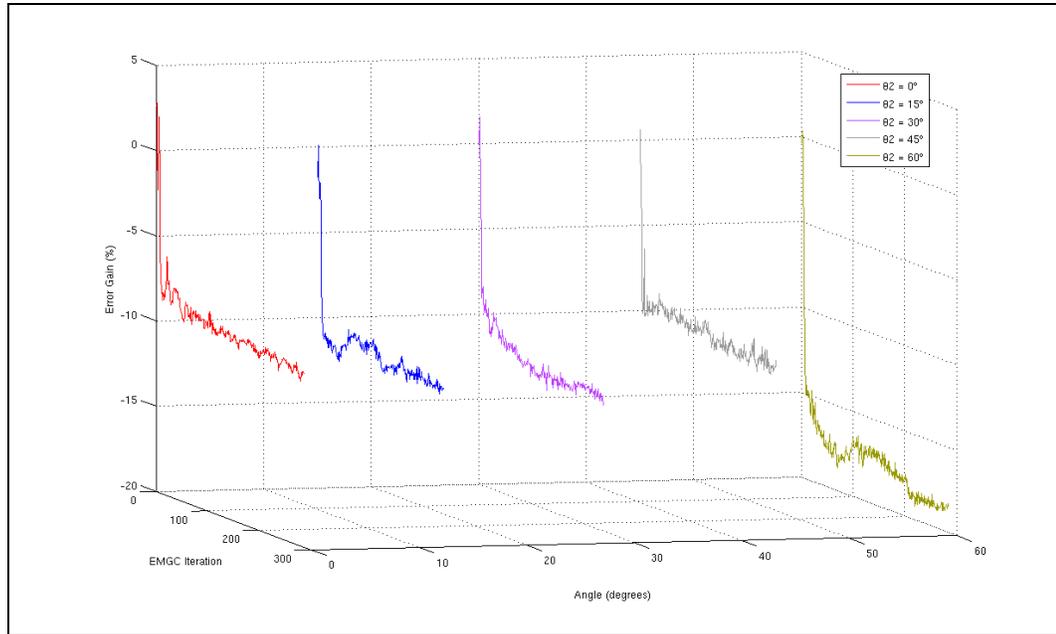


Figure H.14 – Situation Approach Gradient (+ θ_2) Tests, ErrorGain vs. Iteration vs. θ_2 (Test 47 to Test 56)

Figure H.14 illustrates the relationship that the Error Gains for the Total Errors in Figure H.13 have against the EMGC iteration with respect to the associated approach gradient. There is no clear relationship here showing a steadily decreasing Error Gain as the approach gradient increases as there is for the equivalent plots for the departure gradients. However importantly all Error Gains are stable and are below zero, meaning that the use of the EMGC for all situations for all positive approach gradients produces a smaller situation Total Error.

0.5mm Maximum AirGap Height Test

Tap & Outlet Pipe Gap Setup: Standard Config

Air muscle Type: McKibben(Silicon 6mmID/8mmOD bladder)

Air muscle Max Length (in test): 143mm (inflatable area only)

Air muscle Max Length (in test): 110mm (inflatable area only)

Max Control Voltage: 9.46V DC

Max Control Voltage: 0.00V DC

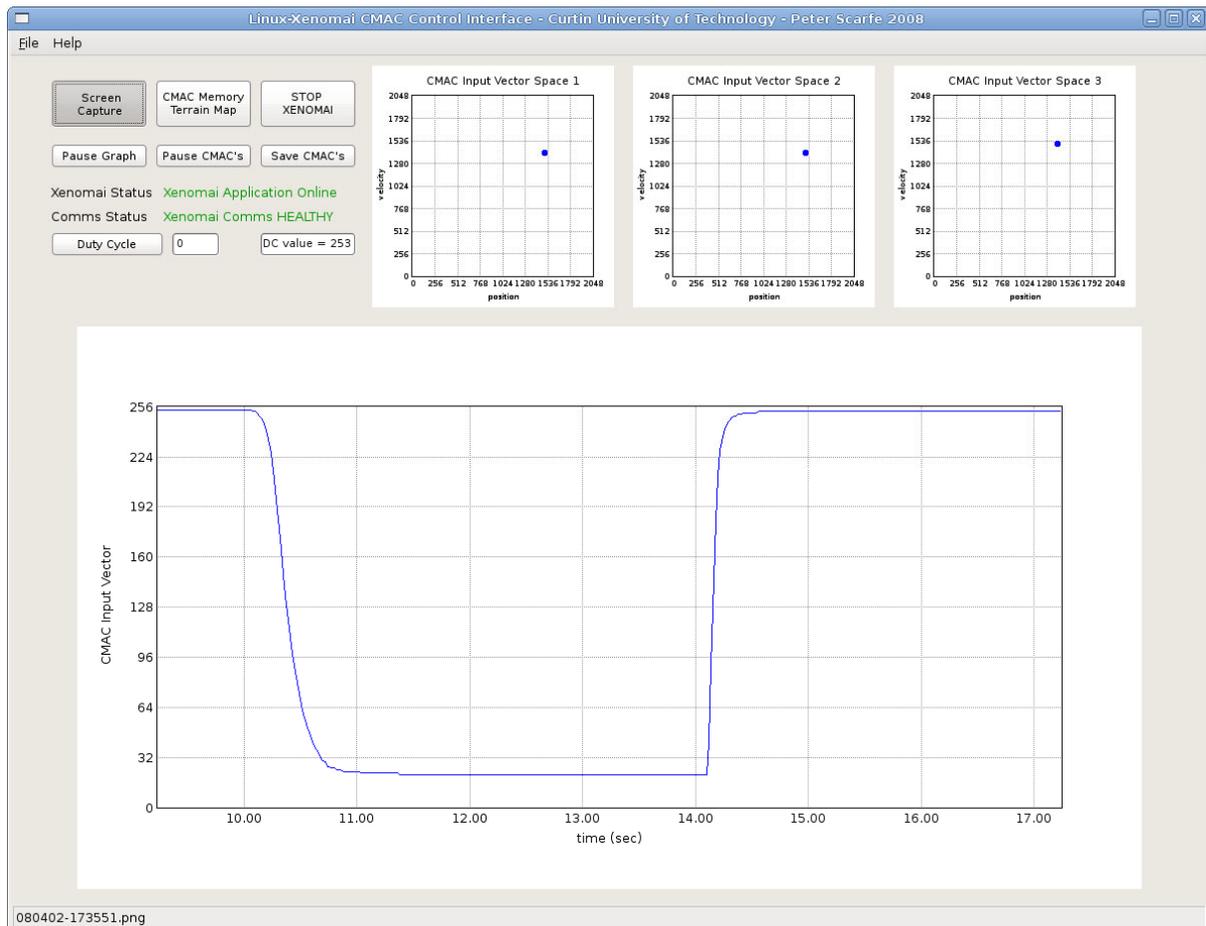
Control Board SupplyVoltage = 11.98V

Pneumatic Supply Pressure: 600kPa (read from Supply Regulator)

PBV Board Manifold Pressure: 320kPa (read from PBV Board SMC Regulator)

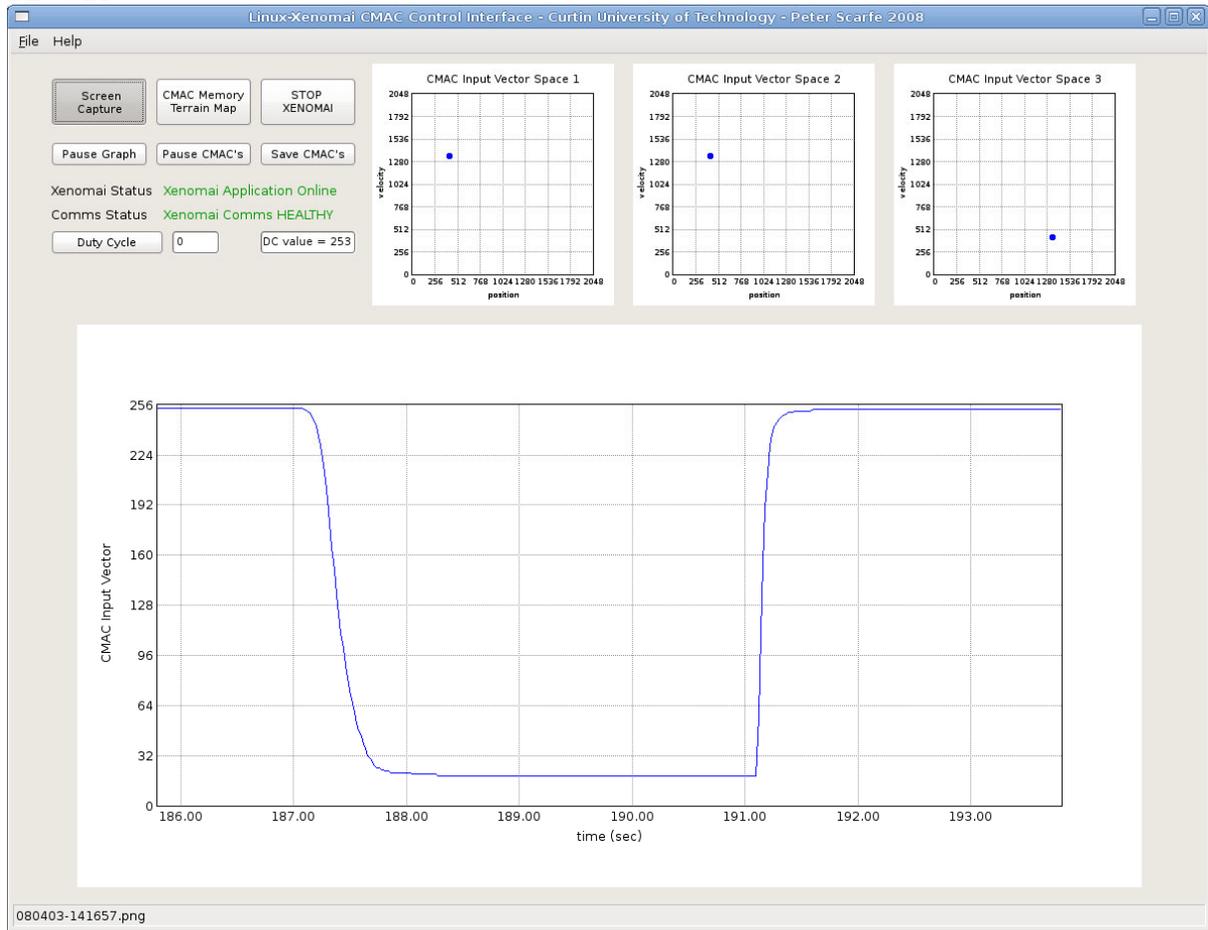
Sequence of Events:

1. 0Bits produces 0.00V, producing 0.20,0.21,0.26Bar at air muscle
2. 255Bits produces 9.47V, producing 2.96,2.98,2.94Bar at air muscle
3. 0Bits produces 0.00V, producing 0.20,0.21,0.23Bar at air muscle

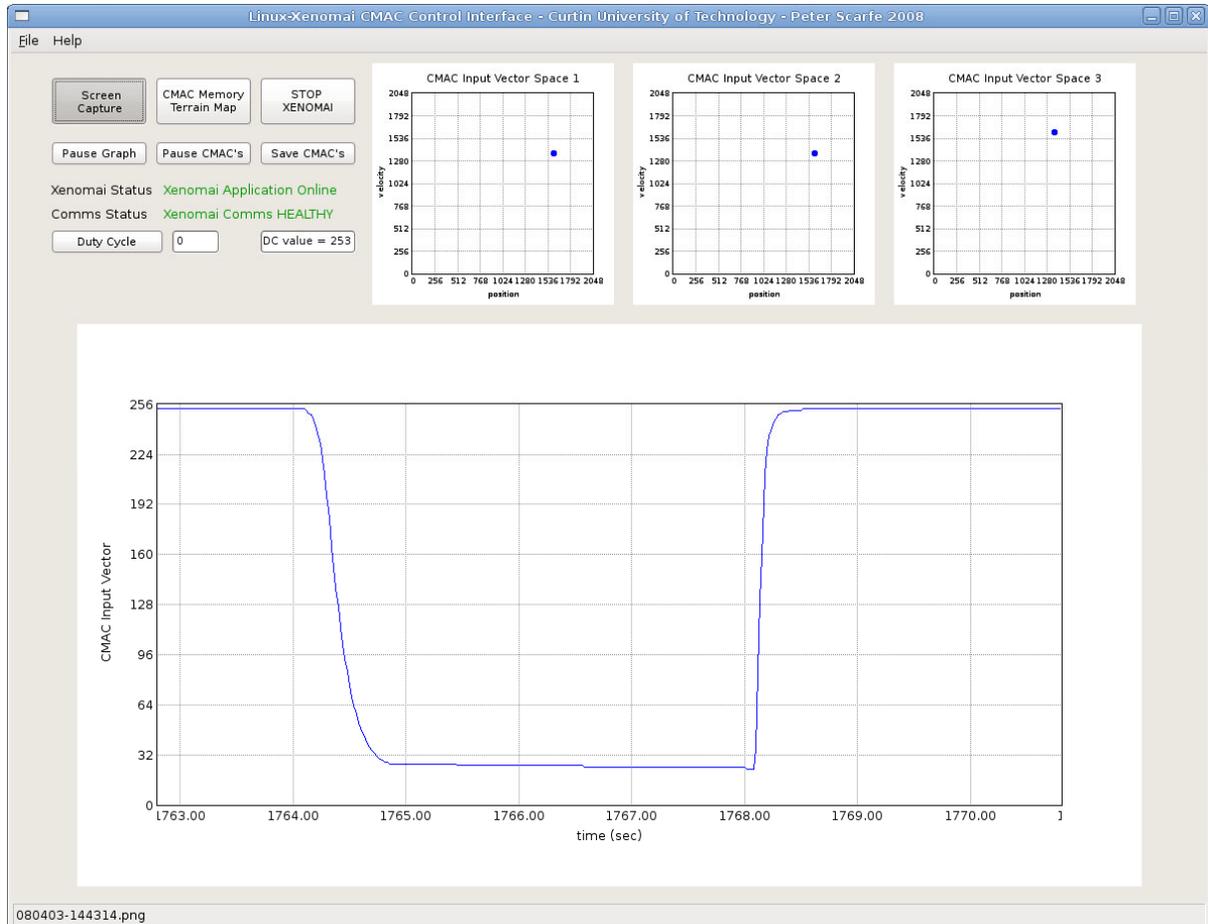


PBV#22

PBV#21



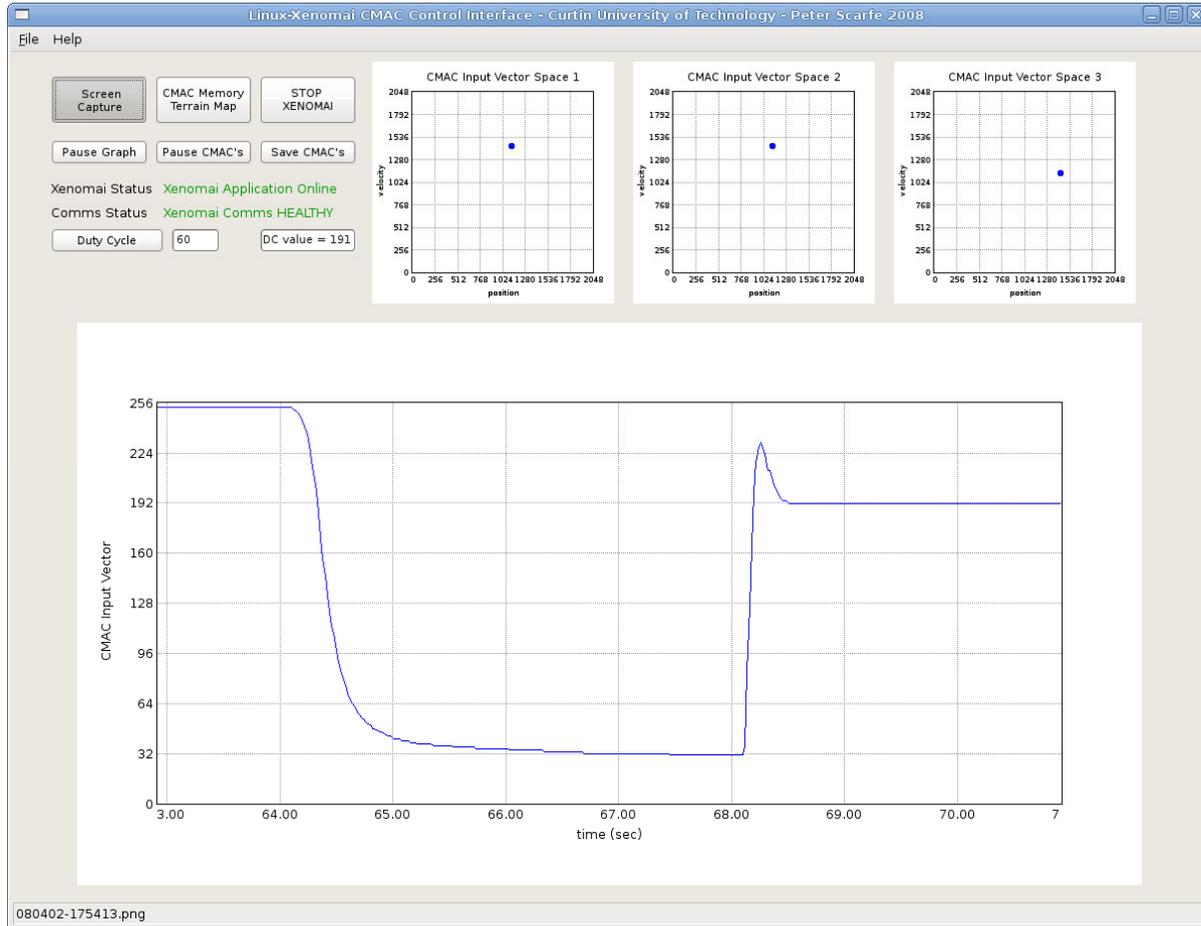
PBV#20



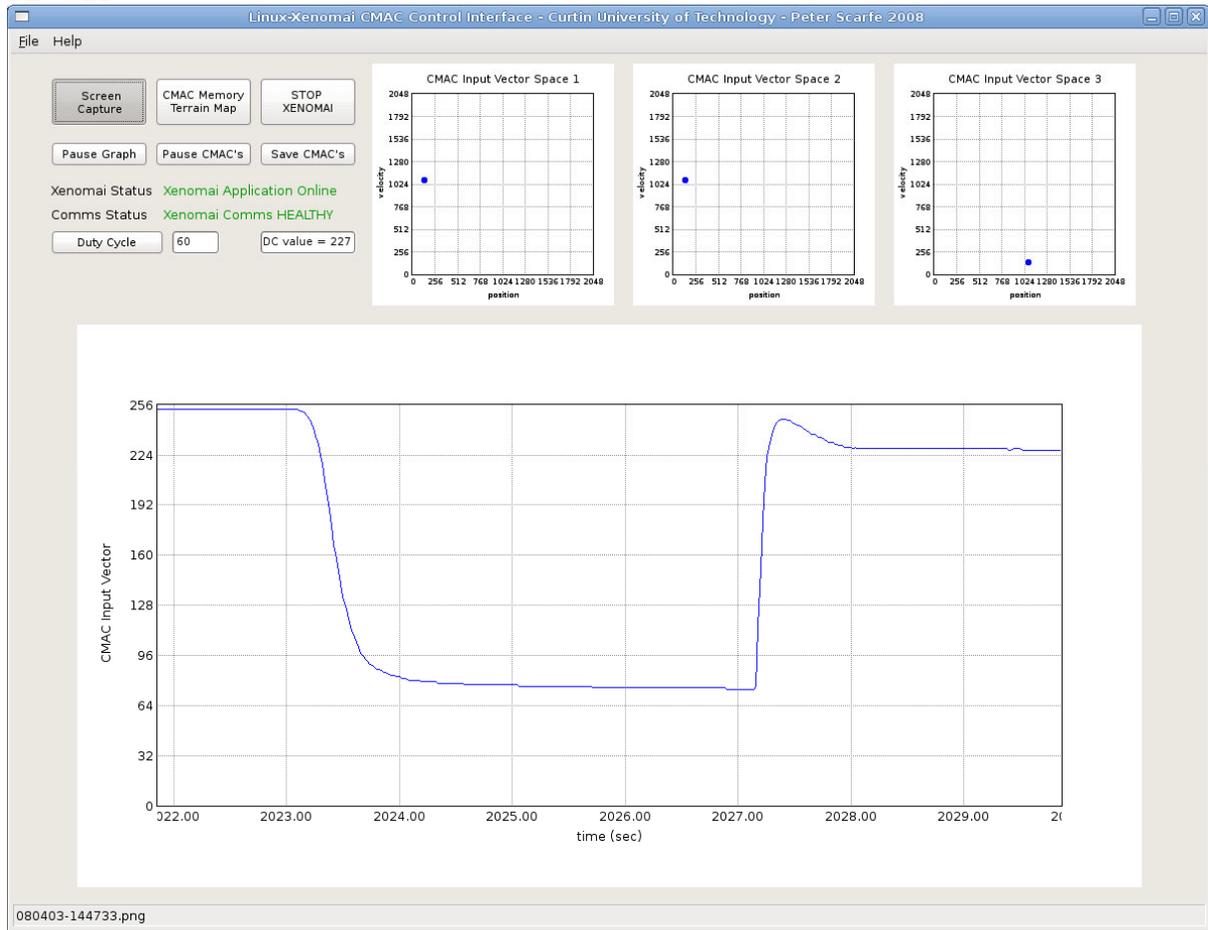
Sequence of Events:

1. 0Bits produces 0.00V, producing 0.21,0.22,Bar at air muscle
2. 100Bits produces 4.46V, producing 2.87,2.59,2.37Bar at air muscle
3. 60Bits produces 2.45V, producing 1.53,1.38,1.27Bar at air muscle

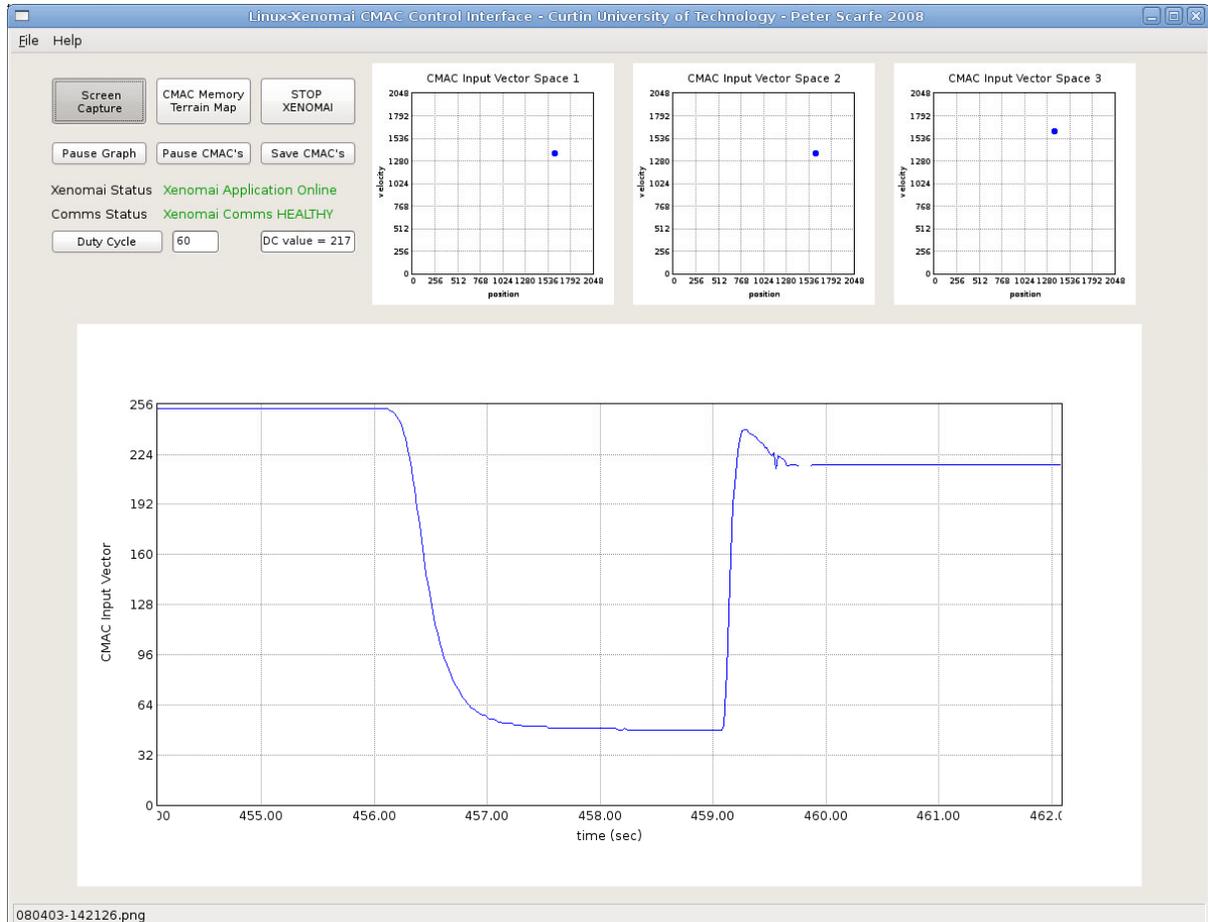
PBV#22



PBV#21



PBV#20



1.0mm Maximum AirGap Height Test

Tap & Outlet Pipe Gap Setup: Standard Config

Air muscle Type: McKibben(Silicon 6mmID/8mmOD bladder)

Air muscle Max Length (in test): 143mm (inflatable area only)

Air muscle Max Length (in test): 110mm (inflatable area only)

Max Control Voltage: 9.46V DC

Max Control Voltage: 0.00V DC

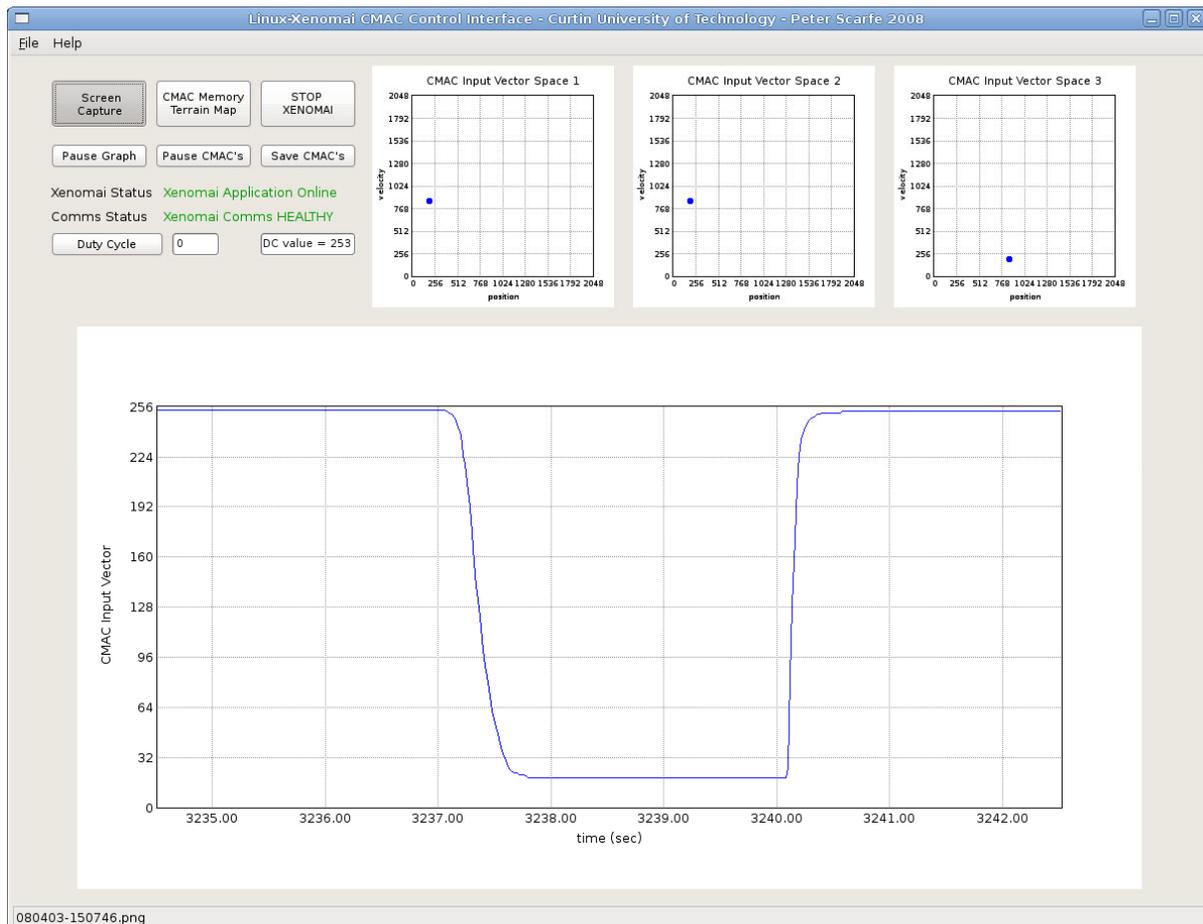
Control Board SupplyVoltage = 11.98V

Pneumatic Supply Pressure: 6.0Bar (read from Supply Regulator)

PBV Board Manifold Pressure: 3.2Bar (read from PBV Board SMC Regulator)

Sequence of Events:

2. 0Bits produces 0.00V, producing 0.20,0.21,0.26Bar at air muscle
2. 255Bits produces 9.47V, producing 3.05,2.98,2.94Bar at air muscle
3. 0Bits produces 0.00V, producing 0.20,0.21,0.23Bar at air muscle

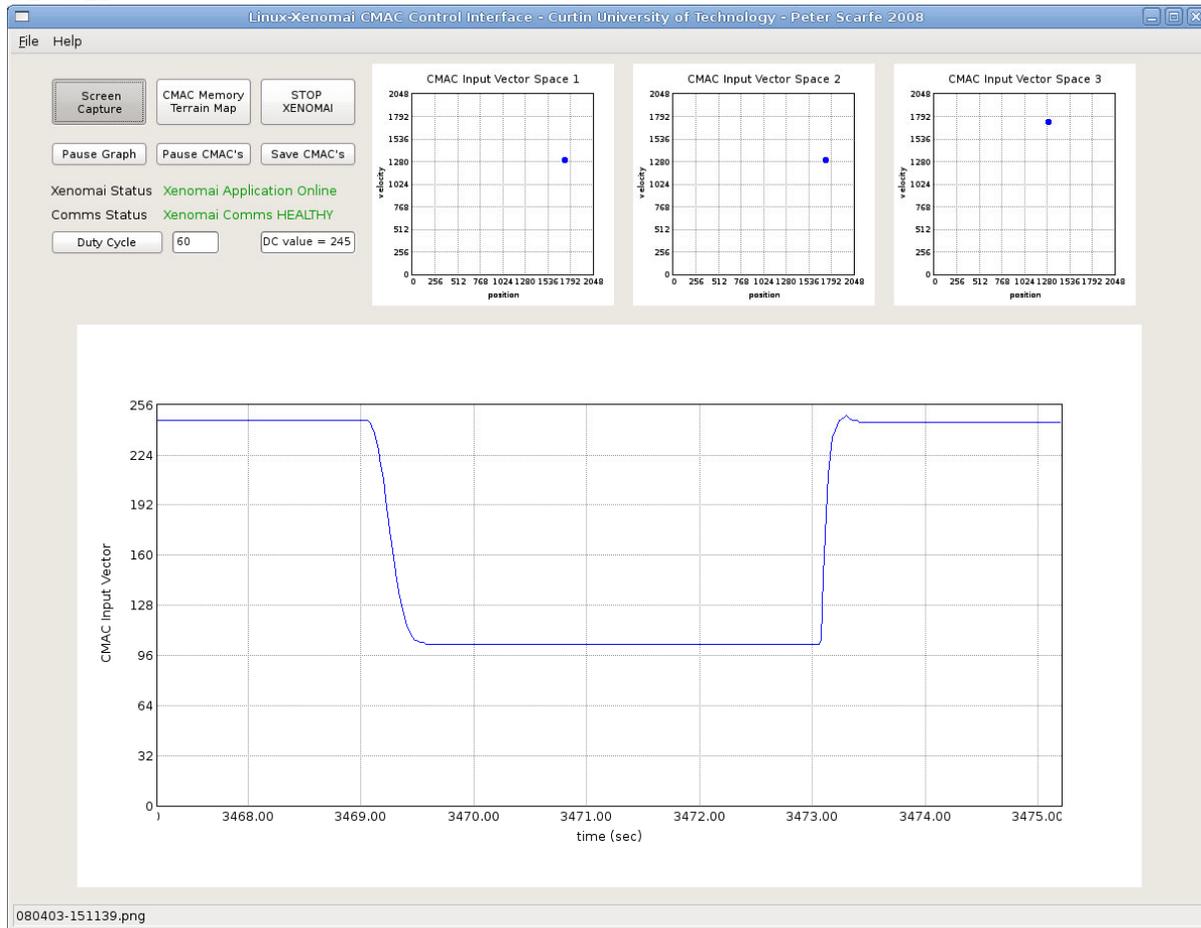


PBV#22

Sequence of Events:

- 0Bits produces 0.00V, producing 0.21,0.22,Bar at air muscle
- 100Bits produces 4.46V, producing 2.20,2.59,2.37Bar at air muscle
- 60Bits produces 2.45V, producing 0.78,1.38,1.27Bar at air muscle

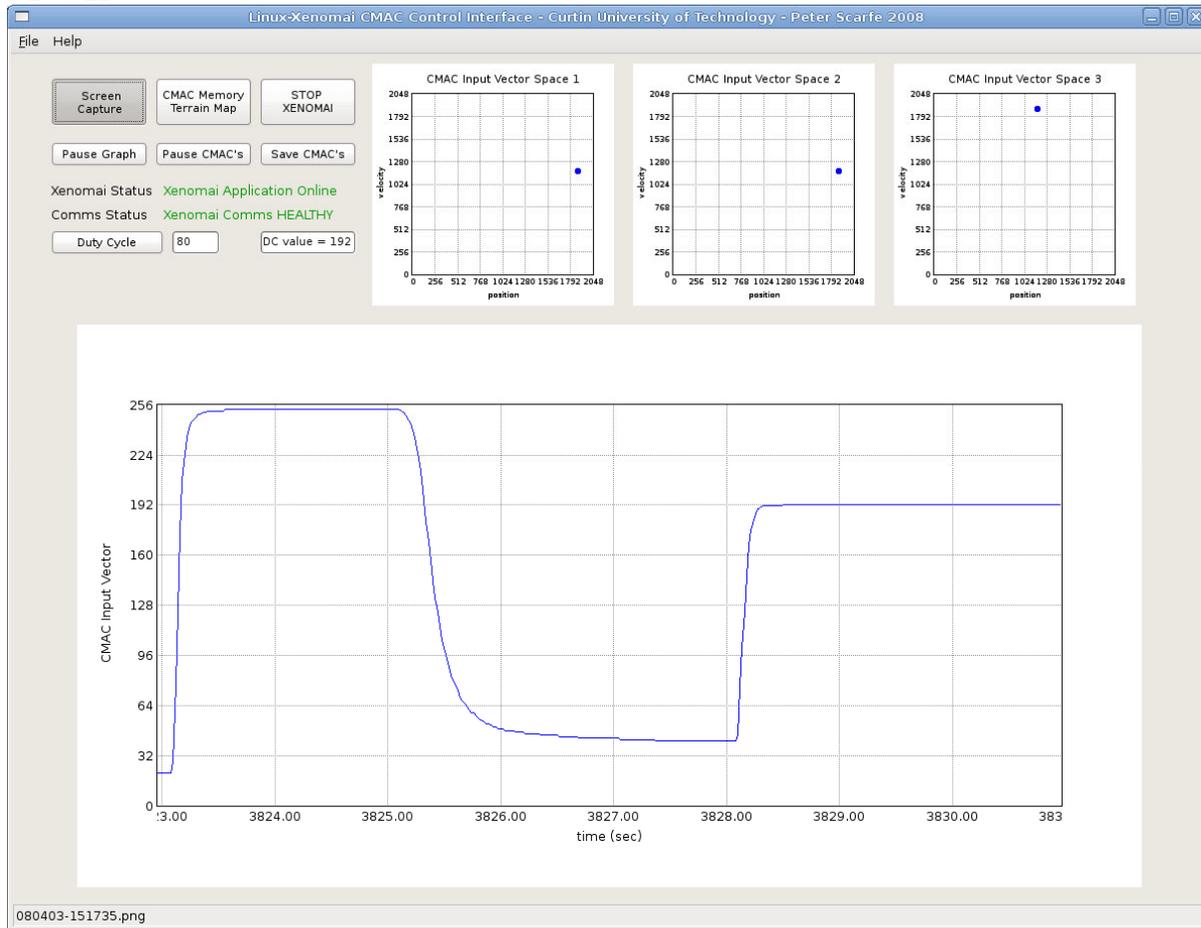
PBV#22



Sequence of Events:

- 0Bits produces 0.00V, producing 0.21,0.22,Bar at air muscle
- 120Bits produces 5.46V, producing 2.73,2.59,2.37Bar at air muscle
- 80Bits produces 3.49V, producing 1.42,1.38,1.27Bar at air muscle

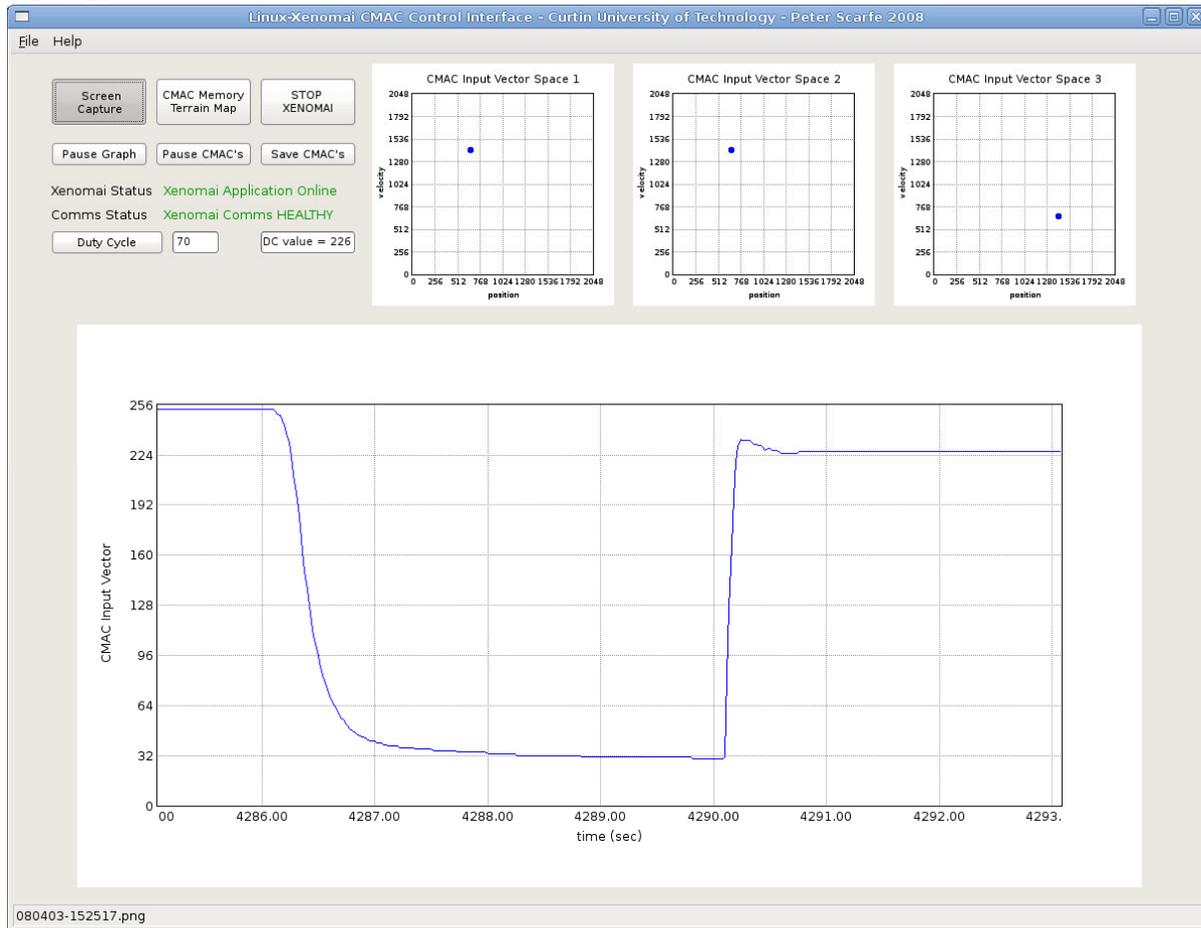
PBV#22



Sequence of Events:

- 0Bits produces 0.00V, producing 0.21,0.22,Bar at air muscle
- 130Bits produces 5.96V, producing 2.85,2.59,2.37Bar at air muscle
- 70Bits produces 2.98V, producing 1.20,1.38,1.27Bar at air muscle

PBV#22



1.5mm Maximum AirGap Height Test

Tap & Outlet Pipe Gap Setup: Standard Config

Air muscle Type: McKibben(Silicon 6mmID/8mmOD bladder)

Air muscle Max Length (in test): 143mm (inflatable area only)

Air muscle Max Length (in test): 110mm (inflatable area only)

Max Control Voltage: 9.46V DC

Max Control Voltage: 0.00V DC

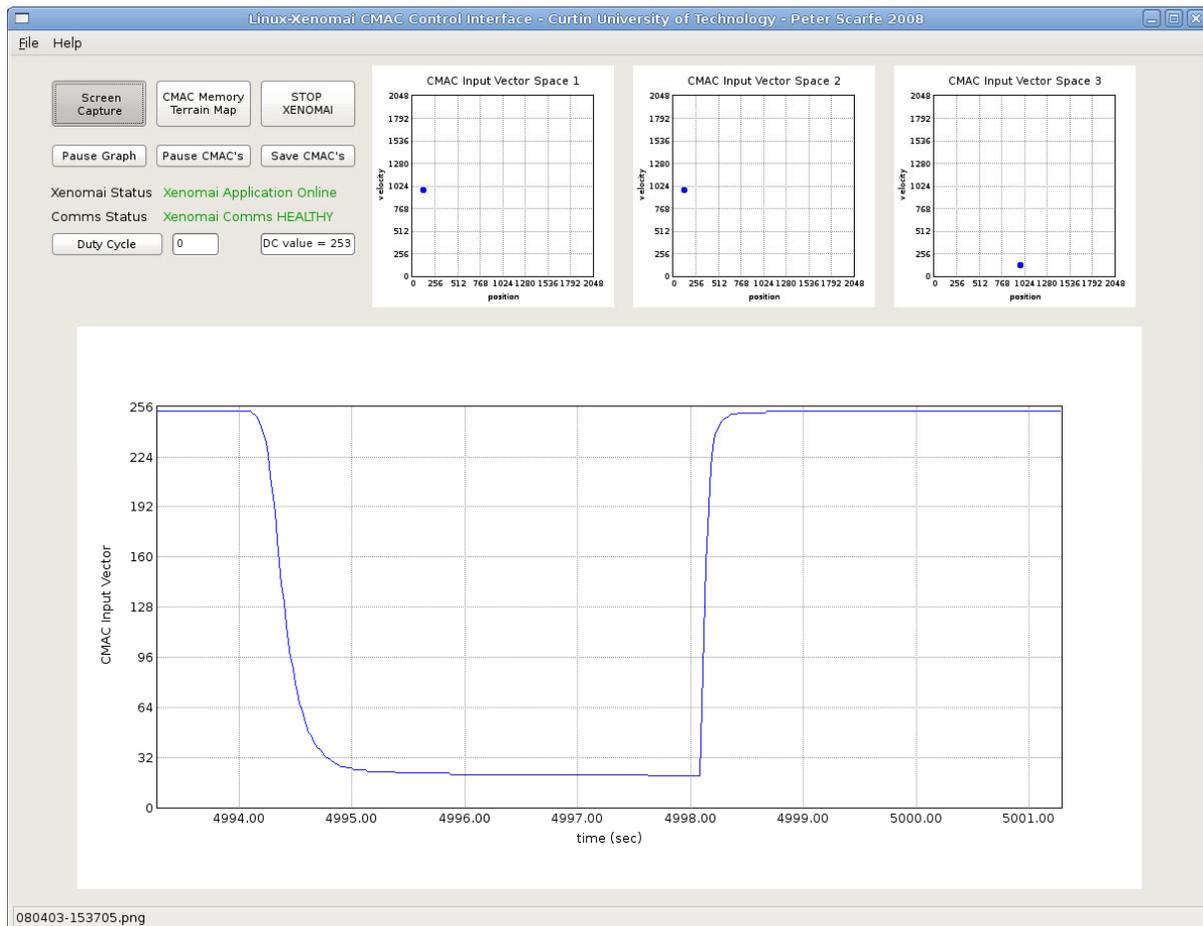
Control Board SupplyVoltage = 11.98V

Pneumatic Supply Pressure: 600kPa (read from Supply Regulator)

PBV Board Manifold Pressure: 320kPa (read from PBV Board SMC Regulator)

Sequence of Events:

3. 0Bits produces 0.00V, producing 0.20,0.21,0.26Bar at air muscle
2. 255Bits produces 9.47V, producing 3.02,2.98,2.94Bar at air muscle
3. 0Bits produces 0.00V, producing 0.20,0.21,0.23Bar at air muscle

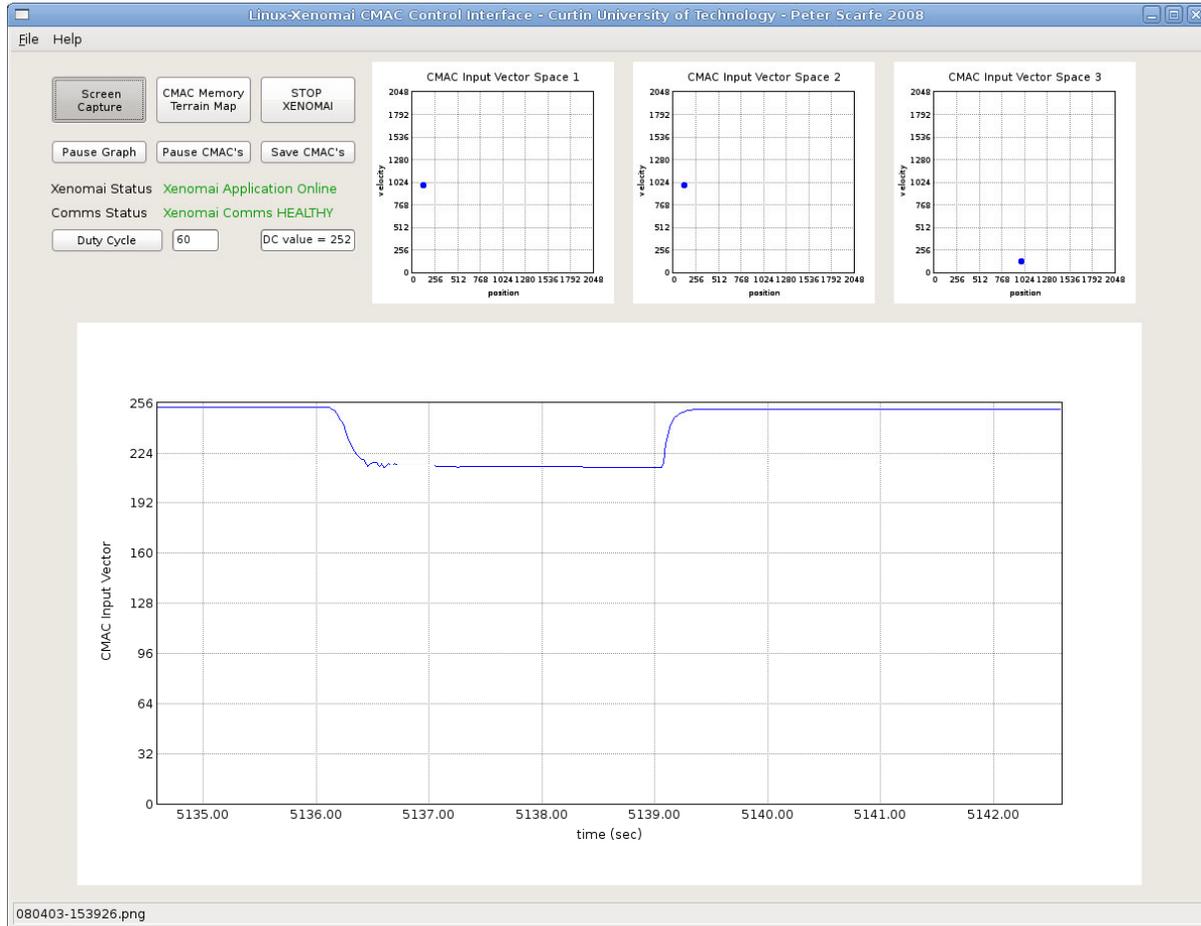


PBV#22

Sequence of Events:

- 0Bits produces 0.00V, producing 0.20,0.21,0.26Bar at air muscle
- 100Bits produces 4.46V, producing 1.42,2.98,2.94Bar at air muscle
- 60Bits produces 2.45V, producing ,0.28,0.23Bar at air muscle

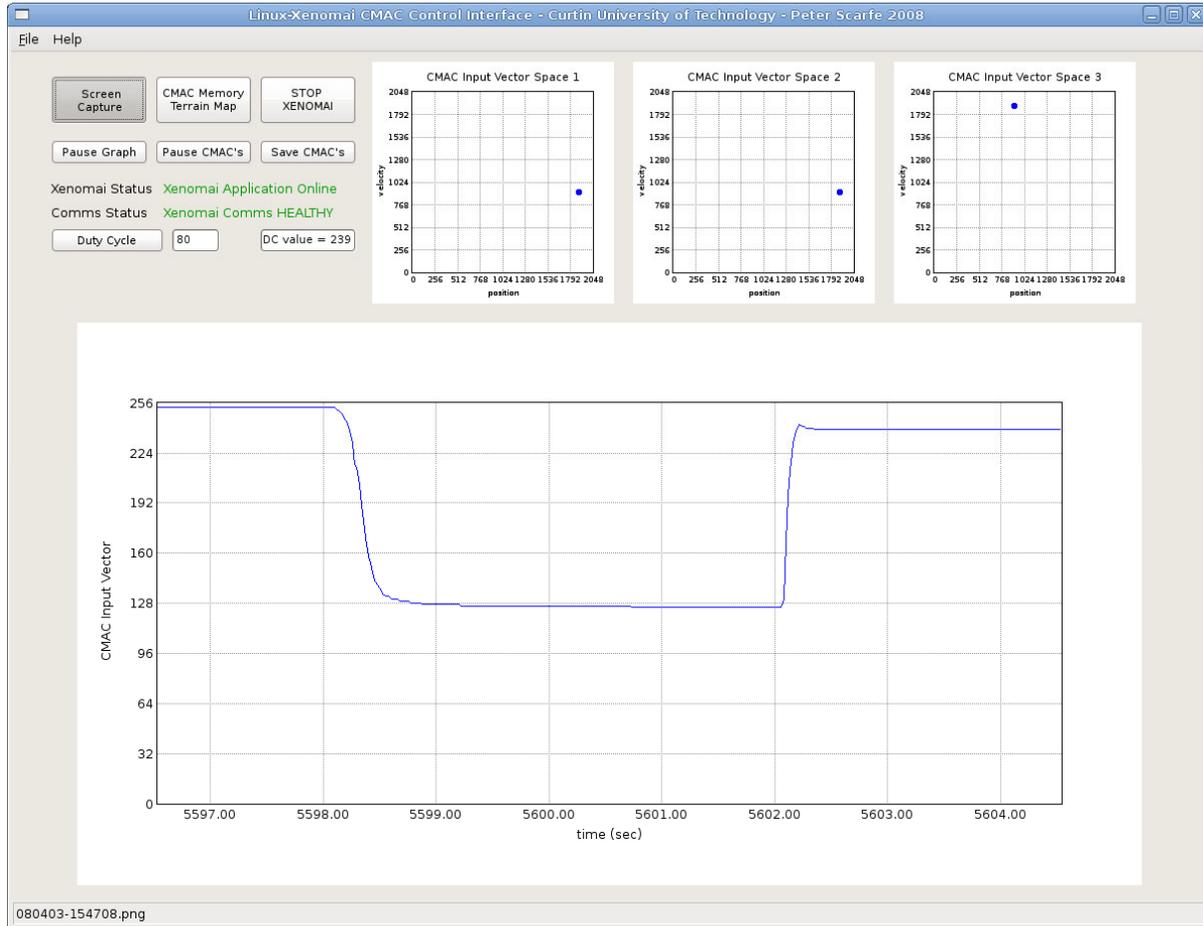
PBV#22



Sequence of Events:

- 0Bits produces 0.00V, producing 0.21,0.22,Bar at air muscle
- 120Bits produces 5.46V, producing 2.06,2.59,2.37Bar at air muscle
- 80Bits produces 3.49V, producing 1.02,1.38,1.27Bar at air muscle

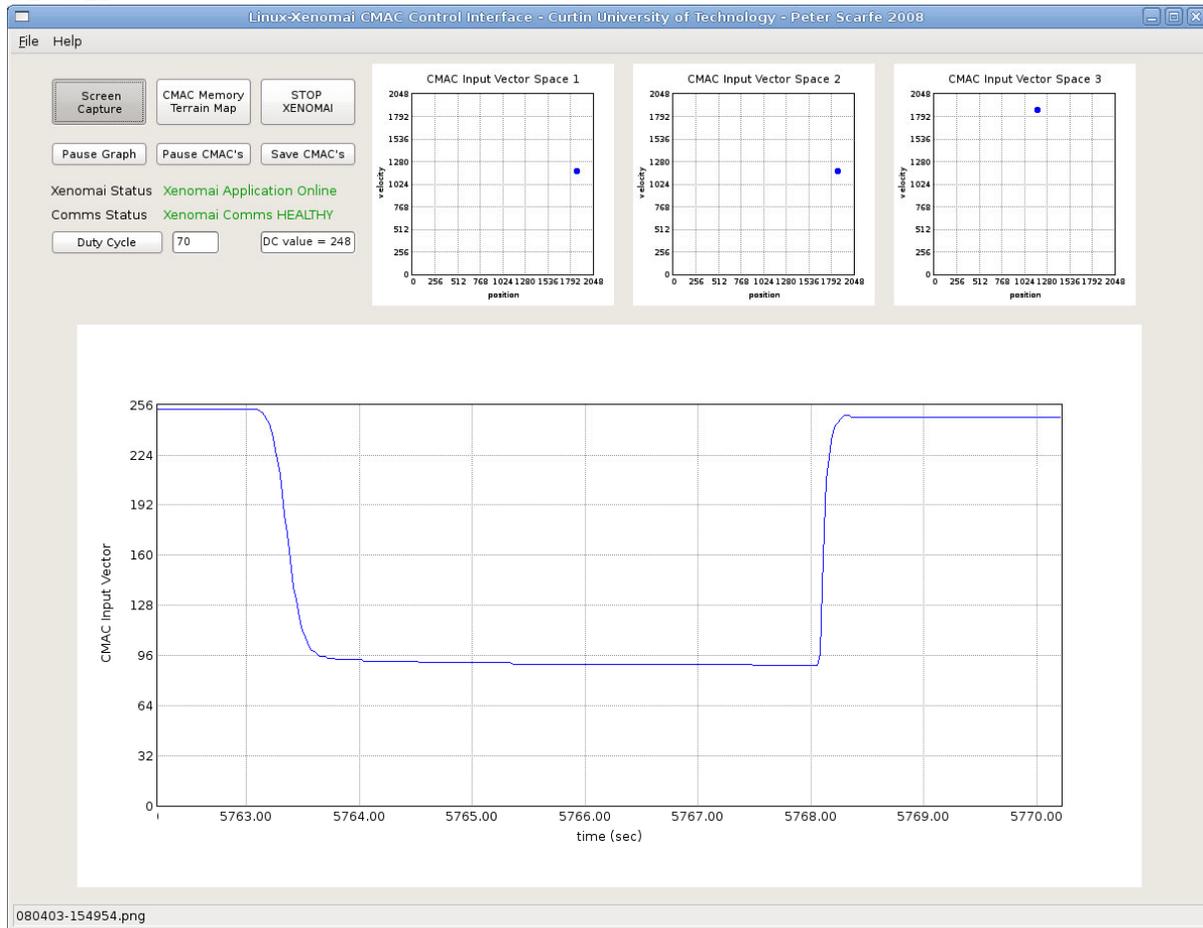
PBV#22



Sequence of Events:

- 0Bits produces 0.00V, producing 0.21,0.22,Bar at air muscle
- 130Bits produces 5.96V, producing 2.32,2.59,2.37Bar at air muscle
- 70Bits produces 2.98V, producing 0.72,1.38,1.27Bar at air muscle

PBV#22



2.0mm Maximum AirGap Height Test

NOTE: NEW AIR-MUSCLE Used here, the one used in the 0.5 to 1.5mm tests popped!

Tap & Outlet Pipe Gap Setup: Standard Config

Air muscle Type: McKibben(Silicon 6mmID/8mmOD bladder)

Air muscle Max Length (in test): 143mm (inflatable area only)

Air muscle Max Length (in test): 110mm (inflatable area only)

Max Control Voltage: 9.46V DC

Max Control Voltage: 0.00V DC

Control Board SupplyVoltage = 11.98V

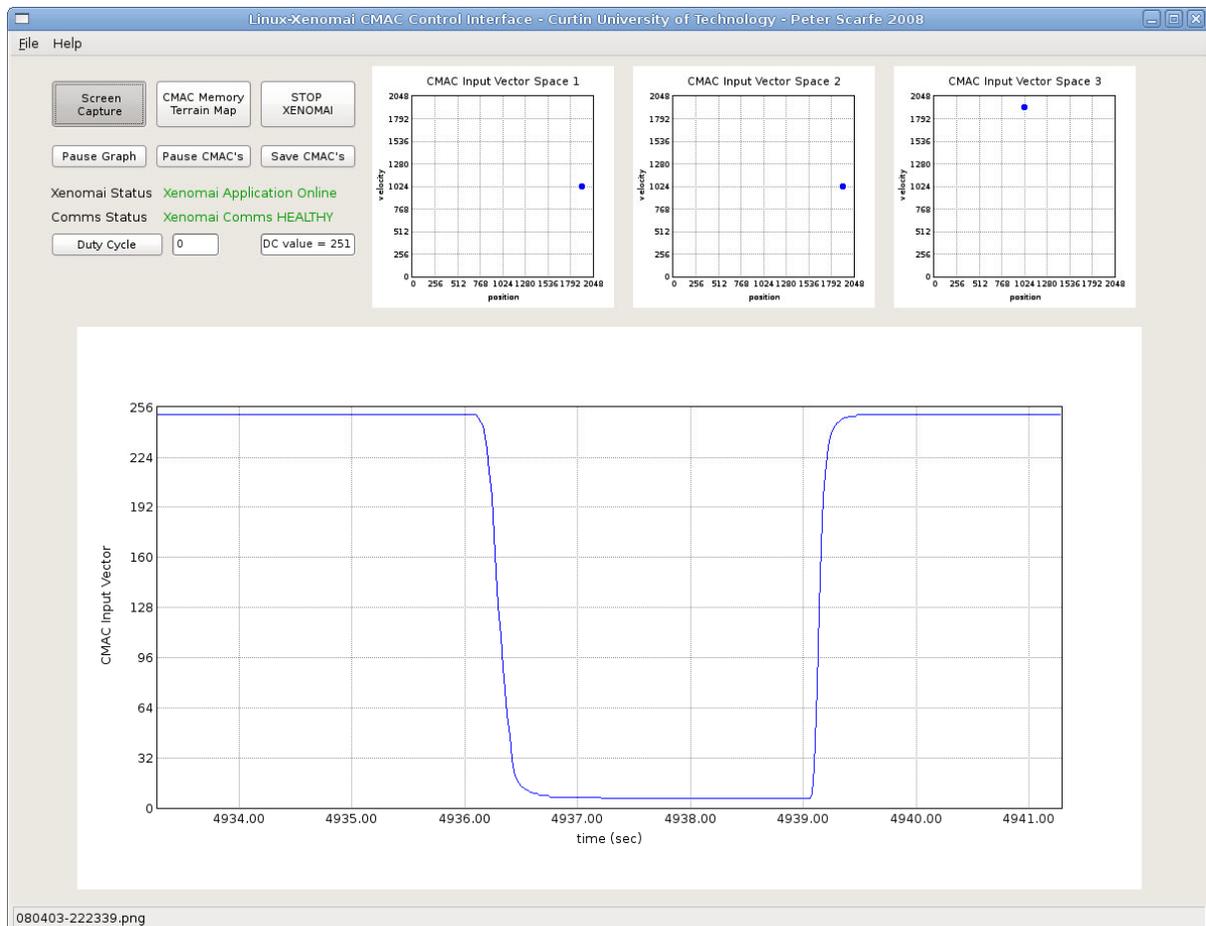
Pneumatic Supply Pressure: 600kPa (read from Supply Regulator)

PBV Board Manifold Pressure: 320kPa (read from PBV Board SMC Regulator)

Sequence of Events:

5. 0Bits produces 0.00V, producing 0.20Bar at air muscle
2. 255Bits produces 9.47V, producing 3.04Bar at air muscle
3. 0Bits produces 0.00V, producing 0.21Bar at air muscle

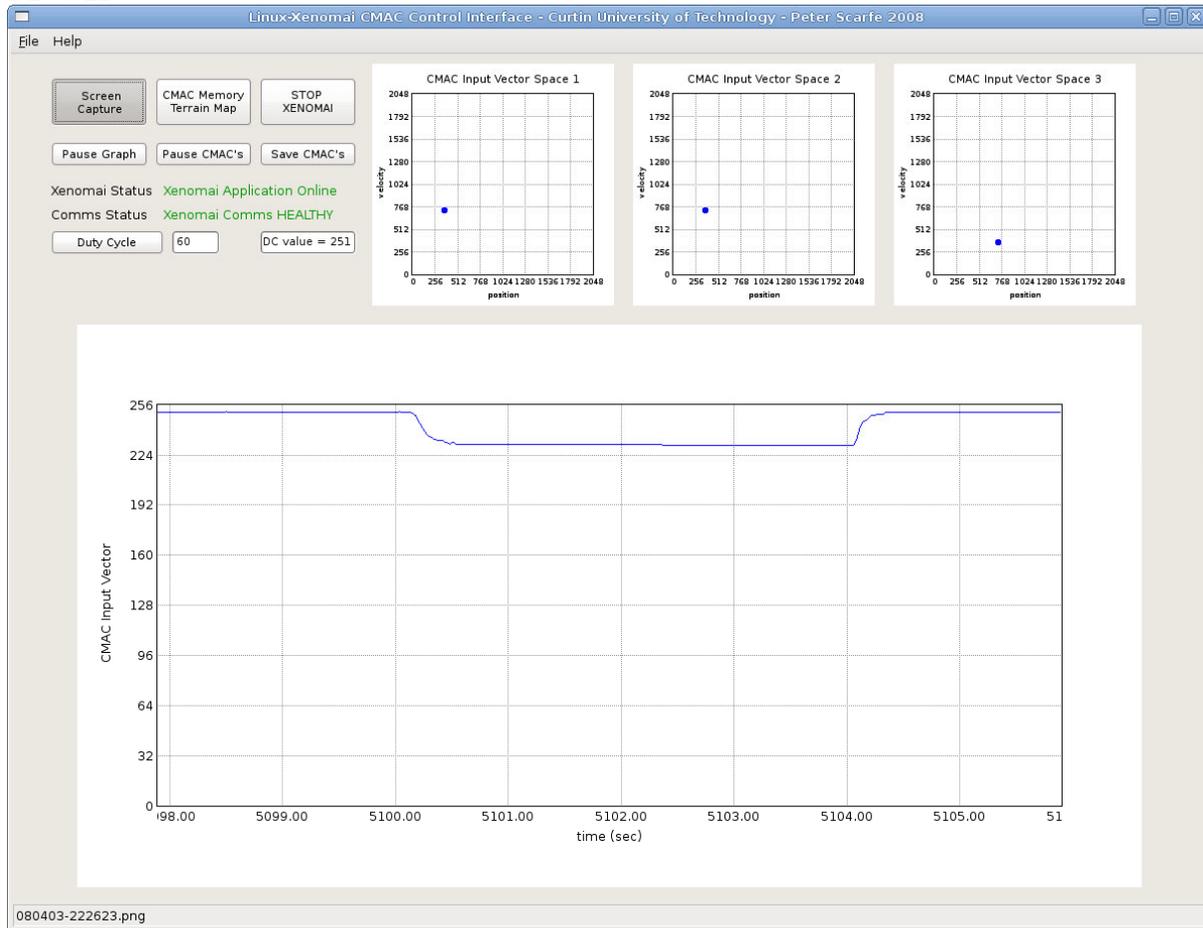
PBV#22



Sequence of Events:

- 0Bits produces 0.00V, producing 0.21Bar at air muscle
- 100Bits produces 4.46V, producing 1.13Bar at air muscle
- 60Bits produces 2.45V, producing ,0.23Bar at air muscle

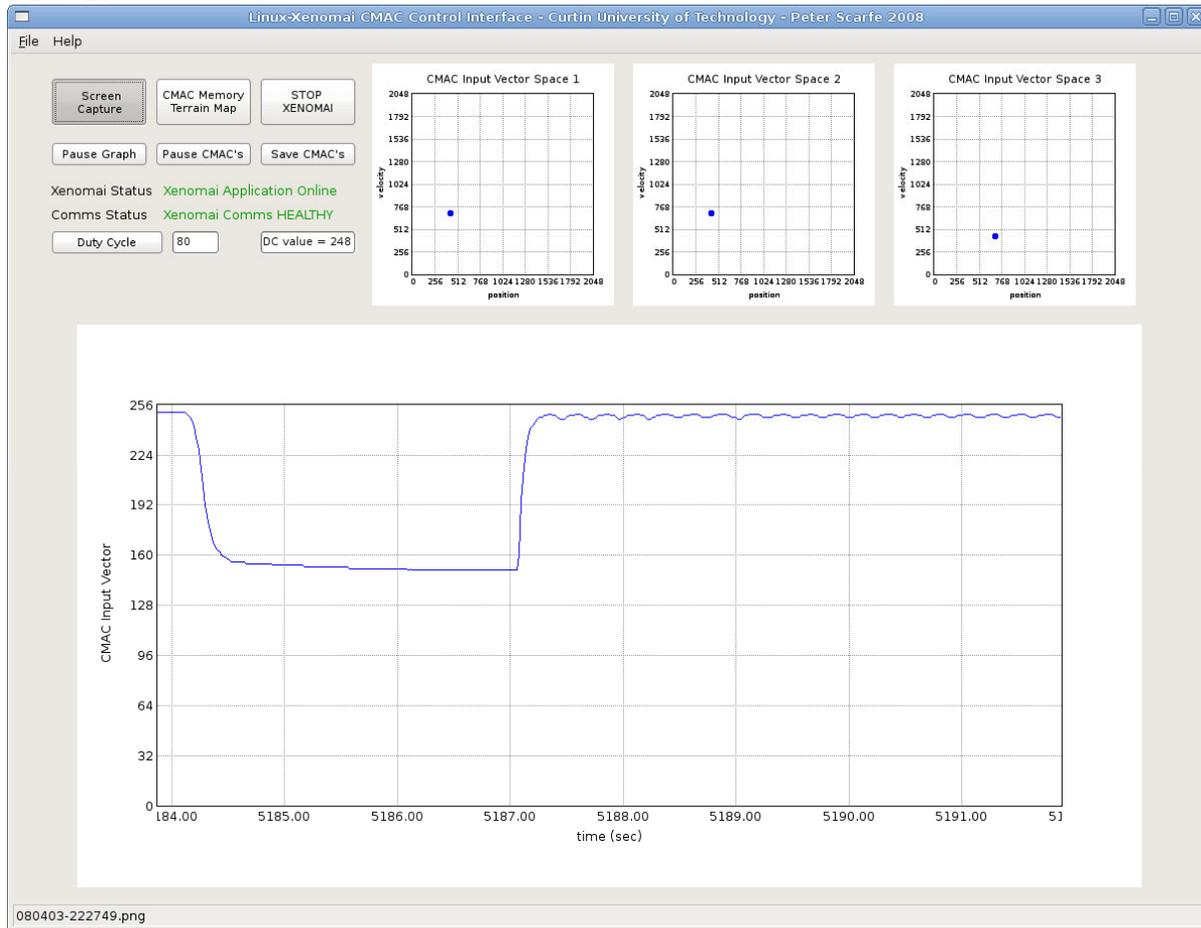
PBV#22



Sequence of Events:

1. 0Bits produces 0.00V, producing 0.21Bar at air muscle
2. 120Bits produces 5.46V, producing 1.66Bar at air muscle
3. 80Bits produces 3.49V, producing Flucuating 0.3-0.7Bar at air muscle

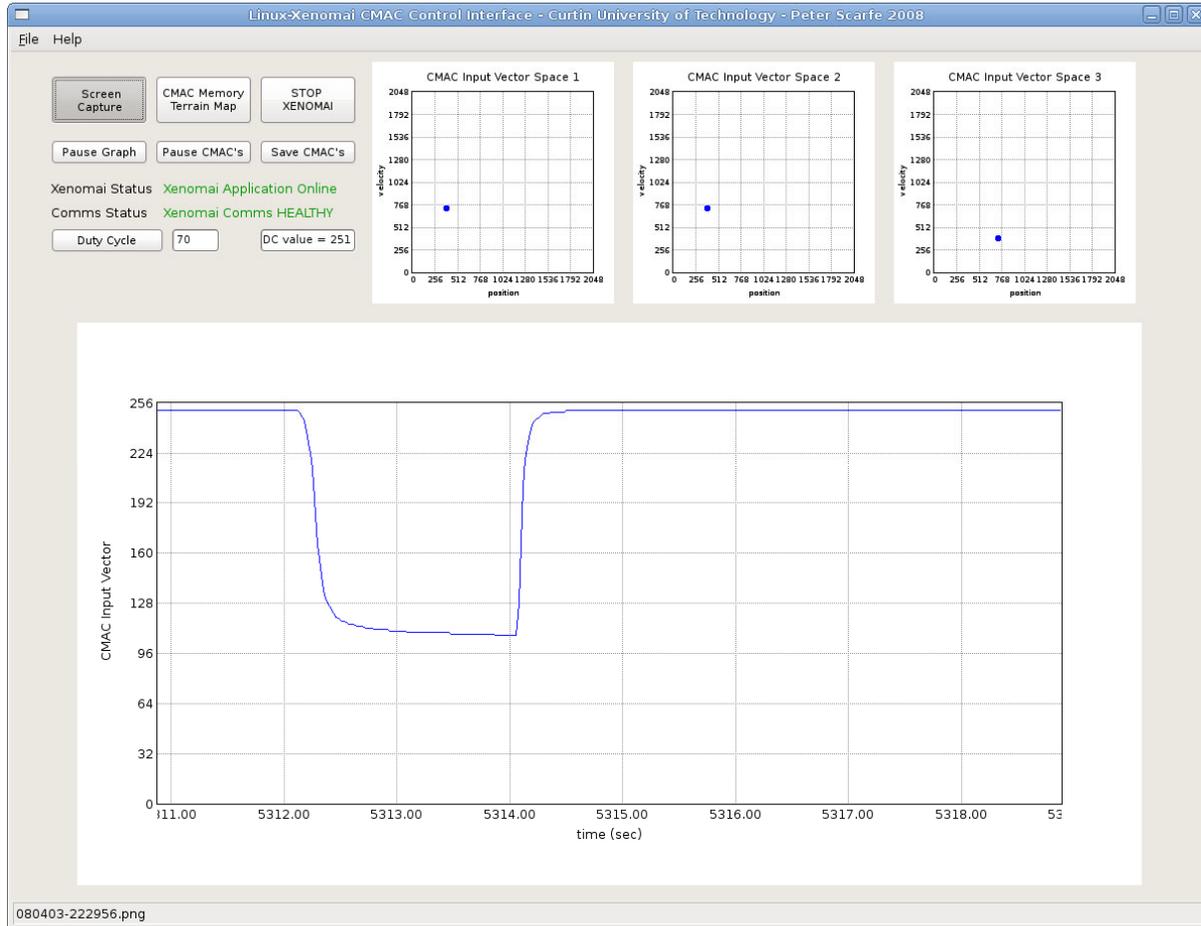
PBV#22



Sequence of Events:

- 0Bits produces 0.00V, producing 0.21Bar at air muscle
- 130Bits produces 5.96V, producing 1.92Bar at air muscle
- 70Bits produces 2.98V, producing 0.26Bar at air muscle

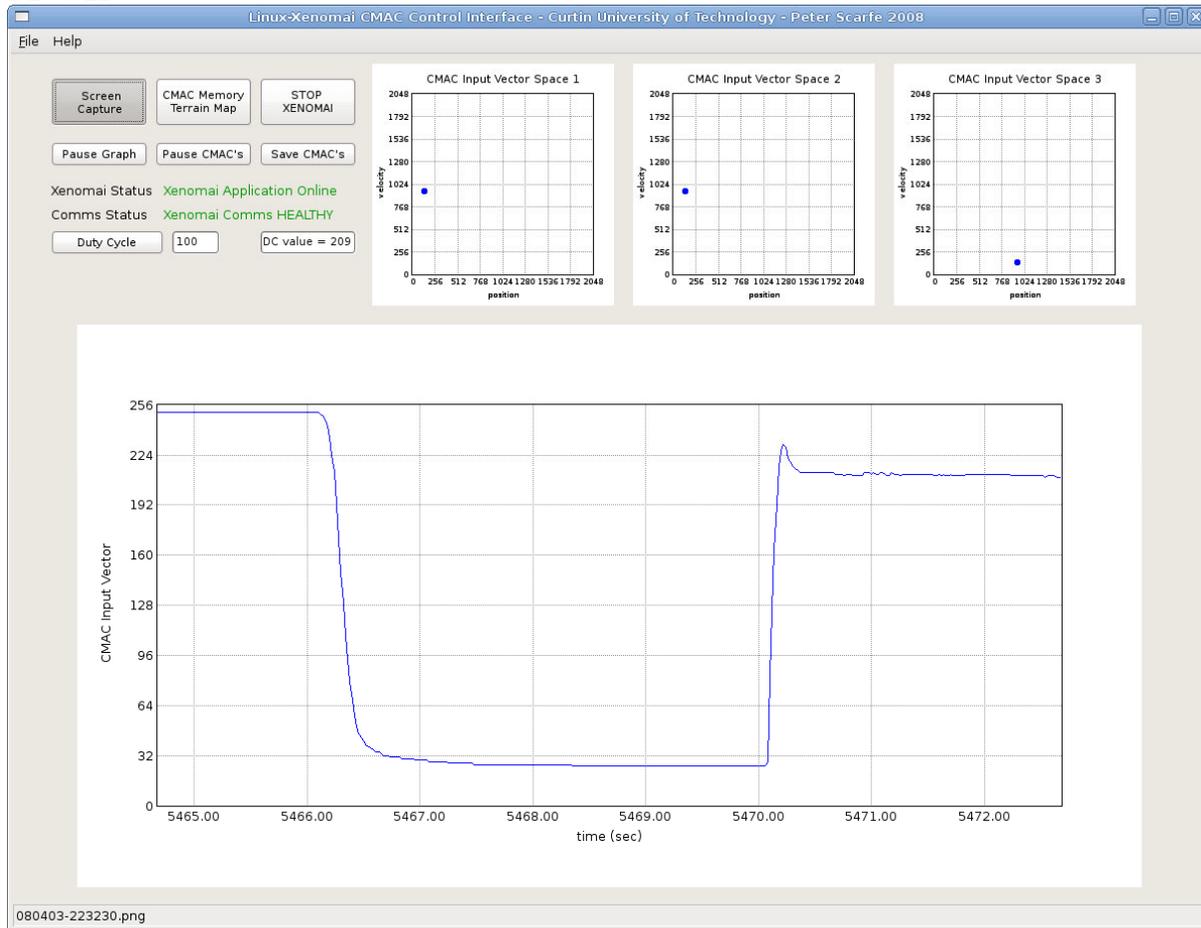
PBV#22



Sequence of Events:

3. 0Bits produces 0.00V, producing 0.21Bar at air muscle
2. 150Bits produces 6.97V, producing 2.47Bar at air muscle
3. 100Bits produces 4.47V, producing 1.24Bar at air muscle

PBV#22



Note: results independent from consistent outlet pipe gap.

The results show that the airgap shifts the response along the control voltage scale.

Ie. From PBV #22:

	At 0.5mm	At 1mm	At 1.5mm	At 2mm
100bits(4.46V)	2.87Bar	2.20Bar	1.42Bar	1.13Bar
60 bits(2.45V)	1.53Bar	0.78Bar	0.28Bar	0.23Bar
120bits(5.46V)	Not tested	2.74Bar	2.06Bar	1.66Bar
80 bits(3.49V)	Not tested	1.42Bar	1.02Bar	Unstable

PROBLEMS:

1. Outlet Pipe gap too large:

Demonstration as such:

Minimum AirGap Height: 1.5mm

Tap & Outlet Pipe Gap Setup:

1. Open (no stopper) Air Pressure: 0.20Bar
 2. Open (w/ stopper) Air Pressure: 0.28Bar
- Final Adjusted Open (w/ stopper) Air Pressure: 0.20Bar (@0.00V)

Air muscle Type: McKibben(Silicon 6mmID/8mmOD bladder)

Air muscle Max Length (in test): 143mm (inflatable area only)

Air muscle Max Length (in test): 110mm (inflatable area only)

Max Control Voltage: 9.46V DC

Max Control Voltage: 0.00V DC

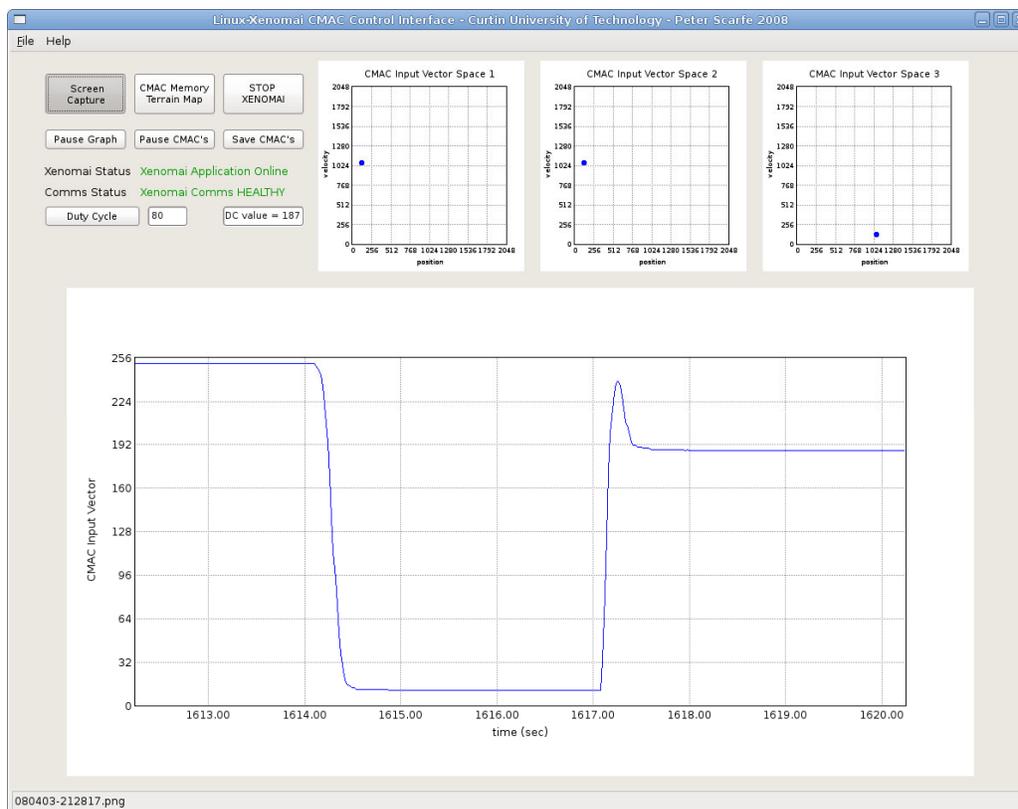
Control Board SupplyVoltage = 11.98V

Pneumatic Supply Pressure: 6.0Bar (read from Supply Regulator)

PBV Board Manifold Pressure: 3.5Bar (read from PBV Board SMC Regulator)

Sequence of Events:

7. 0Bits produces 0.00V, producing 0.21Bar at air muscle
2. 120Bits produces 5.47V, producing 2.79Bar at air muscle
3. 80Bits produces 3.48V, producing 1.41Bar at air muscle



2. Outlet Pipe gap too Small:

Maximum AirGap Height: 1.5mm

Tap & Outlet Pipe Gap Setup:

1. Open (no stopper) Air Pressure: 0.20Bar
 2. Open (w/ stopper) Air Pressure: 1.01Bar
- Final Adjusted Open (w/ stopper) Air Pressure: 0.20Bar (@0.00V)

Air muscle Type: McKibben(Silicon 6mmID/8mmOD bladder)

Air muscle Max Length (in test): 143mm (inflatable area only)

Air muscle Max Length (in test): 110mm (inflatable area only)

Max Control Voltage: 9.46V DC

Max Control Voltage: 0.00V DC

Control Board SupplyVoltage = 11.98V

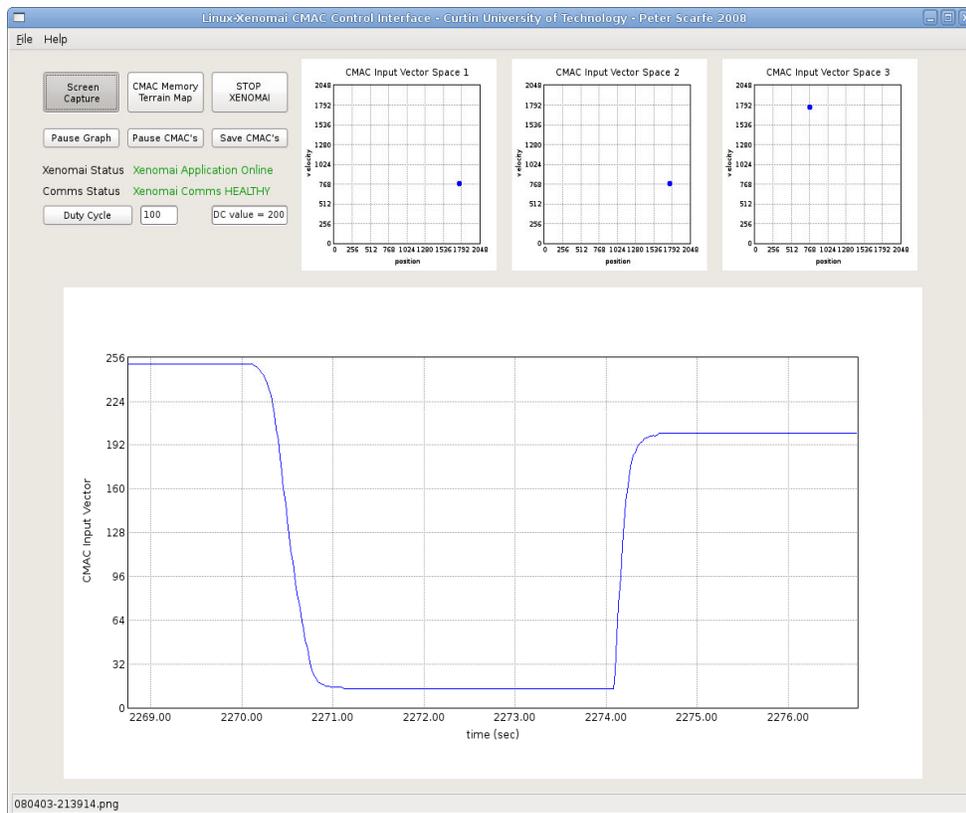
Pneumatic Supply Pressure: 6.0Bar (read from Supply Regulator)

PBV Board Manifold Pressure: 3.5Bar (read from PBV Board SMC Regulator)

Comments: Smaller the outlet pipe gap, the harder it is to physically configure, both the air-flow tap position to obtain the final adjusted open (w/ stopper) air pressure and to make the airgap just the right height to achieve an Open pressure (no stopper) air pressure of 0.2Bar.

Sequence of Events:

8. 0Bits produces 0.00V, producing 0.21Bar at air muscle
2. 150Bits produces 6.97V, producing 2.62Bar at air muscle
3. 100Bits produces 4.47V, producing 1.22Bar at air muscle



Discussion:

Firstly, notice the massive overshoot in Demo 1. This is totally unacceptable.
Secondly, this system did a faster response, both in the inflation and deflation.

Thus a compromise between the two should be reached due to the larger outlet pipe gap at the same final adjusted, meaning higher flow rate through the PBV thus into the airmuscle.

Also note that several trials were tested to achieve the worst performance for both systems, the graphs showing close to the worst case scenario found for demonstration purposes.

3. Outlet Pipe gap Intermediate:

Maxmum AirGap Height: 2.0mm

Tap & Outlet Pipe Gap Setup:

1. Open (no stopper) Air Pressure: 0.20Bar
 2. Open (w/ stopper) Air Pressure: 0.58Bar
- Final Adjusted Open (w/ stopper) Air Pressure: 0.21Bar (@0.00V)

Air muscle Type: McKibben(Silicon 6mmID/8mmOD bladder)

Air muscle Max Length (in test): 143mm (inflatable area only)

Air muscle Max Length (in test): 110mm (inflatable area only)

Max Control Voltage: 9.46V DC

Max Control Voltage: 0.00V DC

Control Board SupplyVoltage = 11.98V

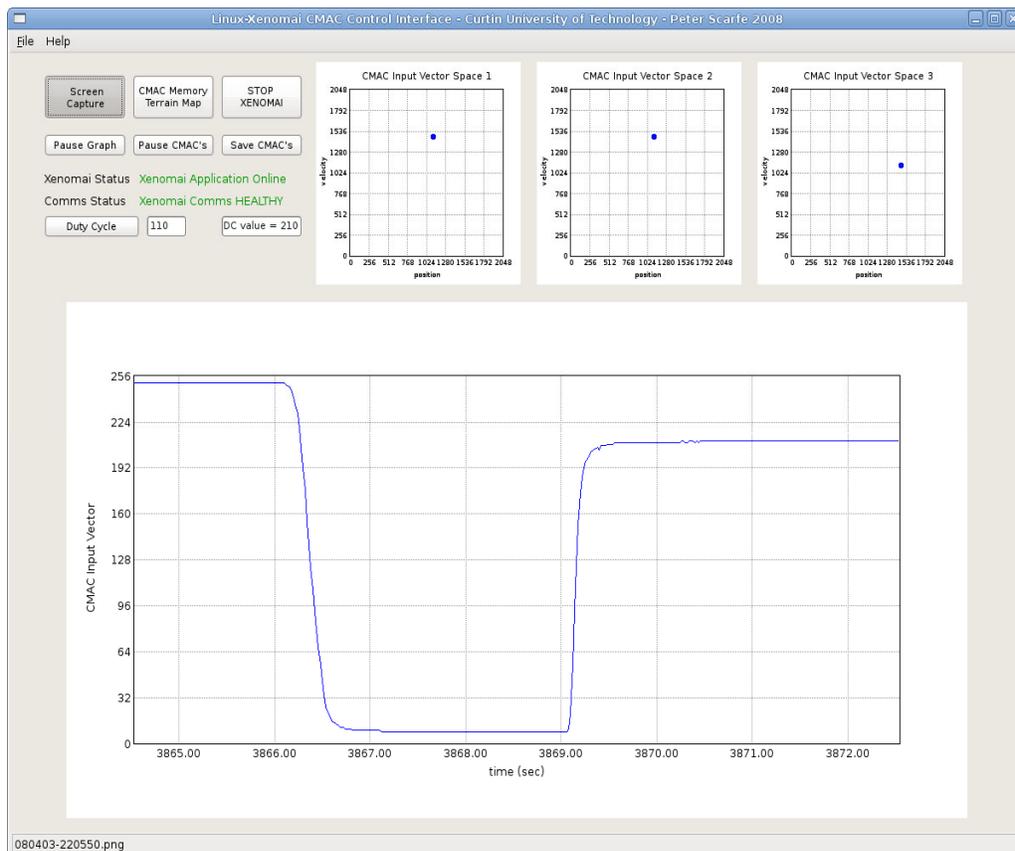
Pneumatic Supply Pressure: 6.0Bar (read from Supply Regulator)

PBV Board Manifold Pressure: 3.5Bar (read from PBV Board SMC Regulator)

Sequence of Events:

9. 0Bits produces 0.00V, producing 0.21Bar at air muscle
2. 180Bits produces 8.47V, producing 2.92Bar at air muscle
3. 110Bits produces 4.97V, producing 1.15Bar at air muscle

Note: Voltages adjusted to give same performance as 1.5mm solenoid height, as in previous 2 tests.



Conclusions: --> **Quarter-Final Test**

From the results of the above graphs... we will run another test based on:

Best Solenoid AirGap Height: 1.5mm

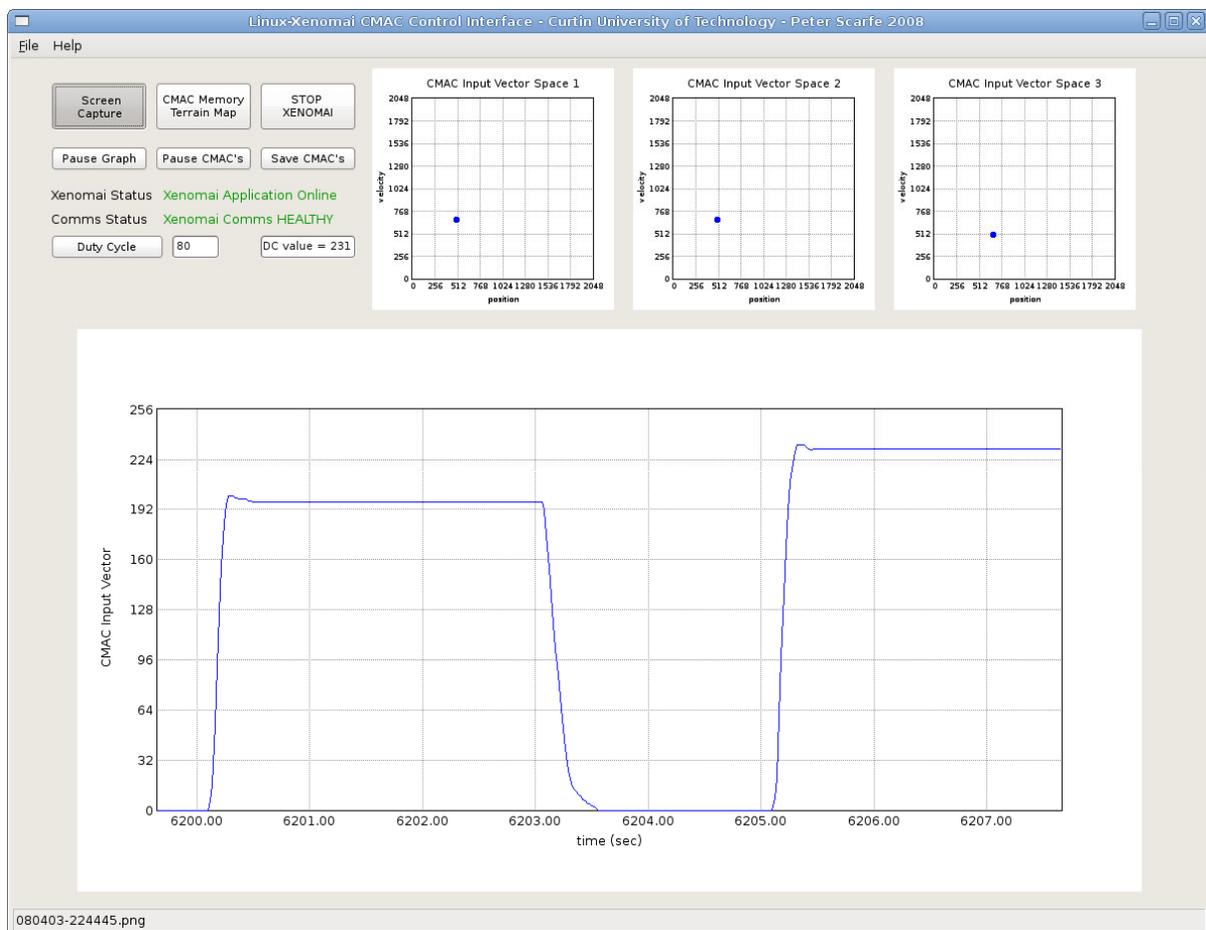
Best Outlet Pipe AirGap Height: Intermediate

Extras: PBV Board Manifold Pressure: 3.6Bar (read from PBV Board SMC Regulator)

Thus, testing with the BEST parameters produce the following responses:

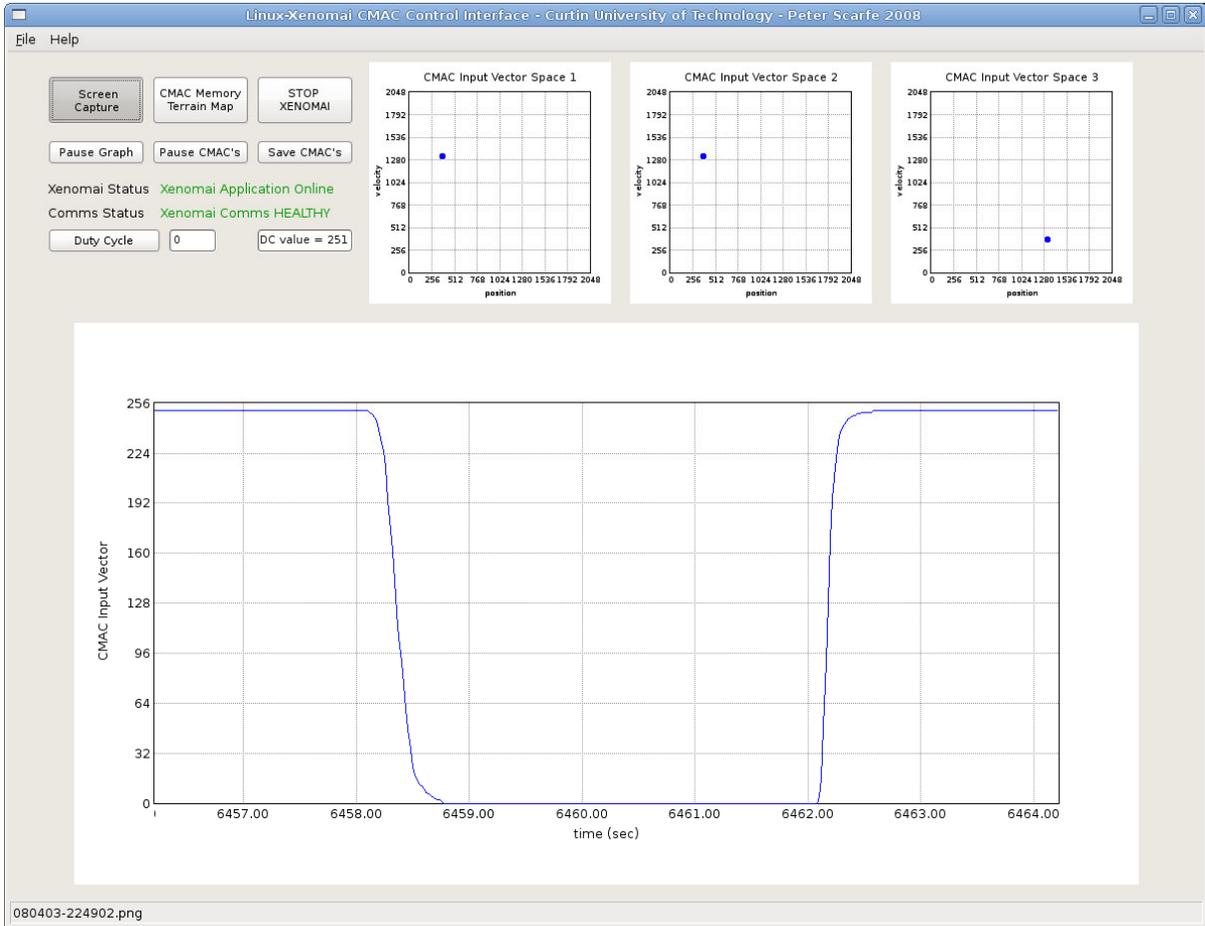
Sequence of Events:

1. 0Bits produces 0.00V, producing 0.21Bar at air muscle
2. 255Bits produces 9.48V, producing 3.62Bar at air muscle
3. 90Bits produces 3.97V, producing 1.22Bar at air muscle
4. 255Bits produces 9.97V, producing 3.62Bar at air muscle
5. 80Bits produces 3.48V, producing 0.95Bar at air muscle



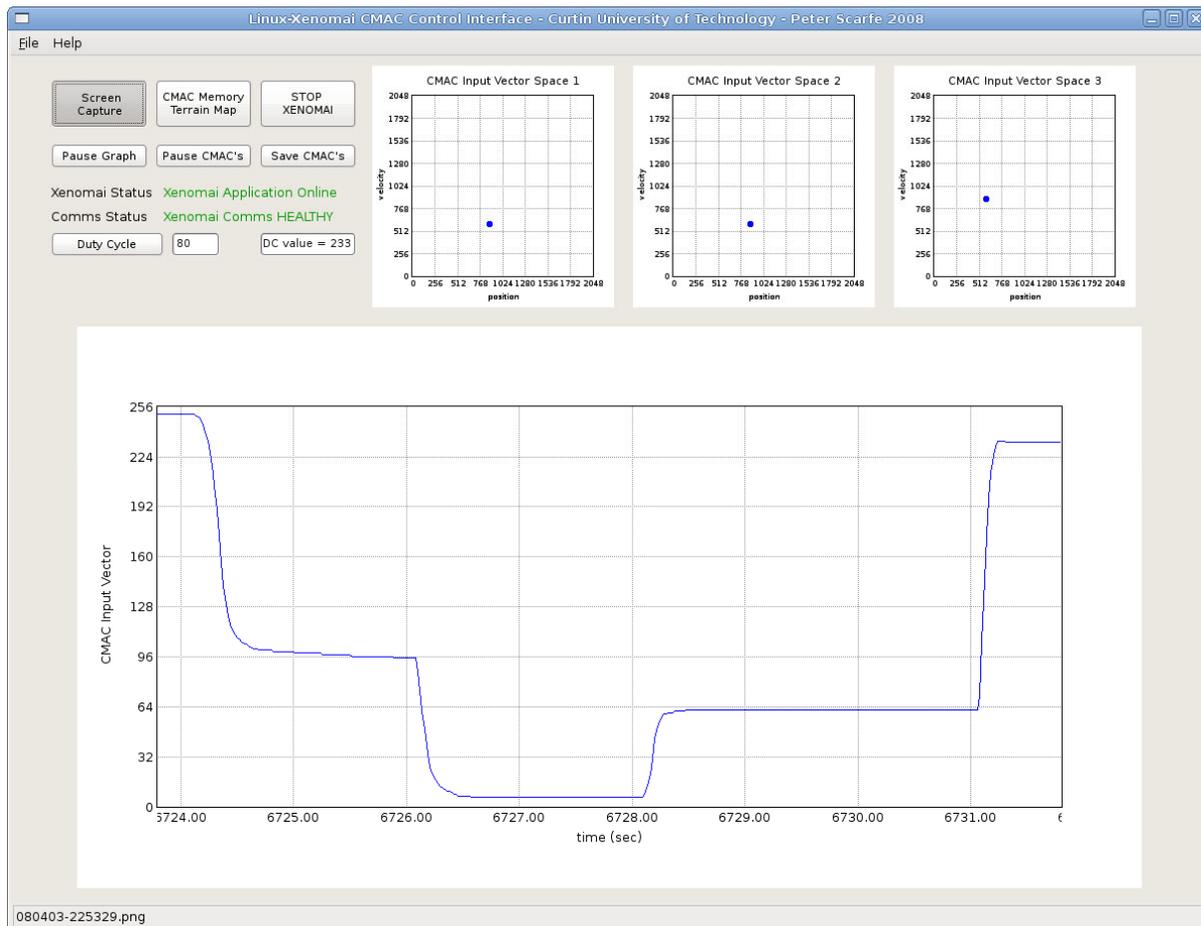
Sequence of Events:

1. 0Bits produces 0.00V, producing 0.21Bar at air muscle
2. 255Bits produces 6.97V, producing 3.62Bar at air muscle
3. 0Bits produces 0.00V, producing 0.21Bar at air muscle



Sequence of Events:

1. 0Bits produces 0.00V, producing 0.21Bar at air muscle
2. 120Bits produces 5.47V, producing 1.93Bar at air muscle
3. 150Bits produces 6.98V, producing 3.00Bar at air muscle
4. 120Bits produces 5.47V, producing 1.93Bar at air muscle
5. 80Bits produces 3.49V, producing 0.95Bar at air muscle



Note: Max Pressure obtained in PBV/AirMuscle as in PBV Board Manifold. Therefore at a solenoid airgap of 1.5mm, there is enough force available for the solenoid rubber stopper to fully seal the PBV outlet pipe. This is also true when tested at a manifold pressure of 4.0Bar (the maximum pressure this system has been tested at.)

Conclusion:

At these settings the Solenoid AirGap seems to be working well. Response is quick enough with minimal overshoot, between a wide range of input voltages, from 3.49 to 6.98V.

At these settings the Outlet Pipe AirGap seems to be also working well minimising the overshoot throughout the response range. The speed of inflation at this setting is suitably quick. It may be possible to eliminate these overshoots however at the expense of some inflation velocity. The next test will test this...

Due to the nature of the PBV setup narrowing the range of the control voltage, the Enhanced Resolution PWM generator will have to be implemented.

Semi-Final Test:

Maximum AirGap Height: 1.5mm

Tap & Outlet Pipe Gap Setup:

1. Open (no stopper) Air Pressure: 0.20Bar
 2. Open (w/ stopper) Air Pressure: 0.70Bar
- Final Adjusted Open (w/ stopper) Air Pressure: 0.21Bar (@0.00V)

Air muscle Type: McKibben(Silicon 6mmID/8mmOD bladder)

Air muscle Max Length (in test): 143mm (inflatable area only)

Air muscle Max Length (in test): 110mm (inflatable area only)

Max Control Voltage: 9.46V DC

Max Control Voltage: 0.00V DC

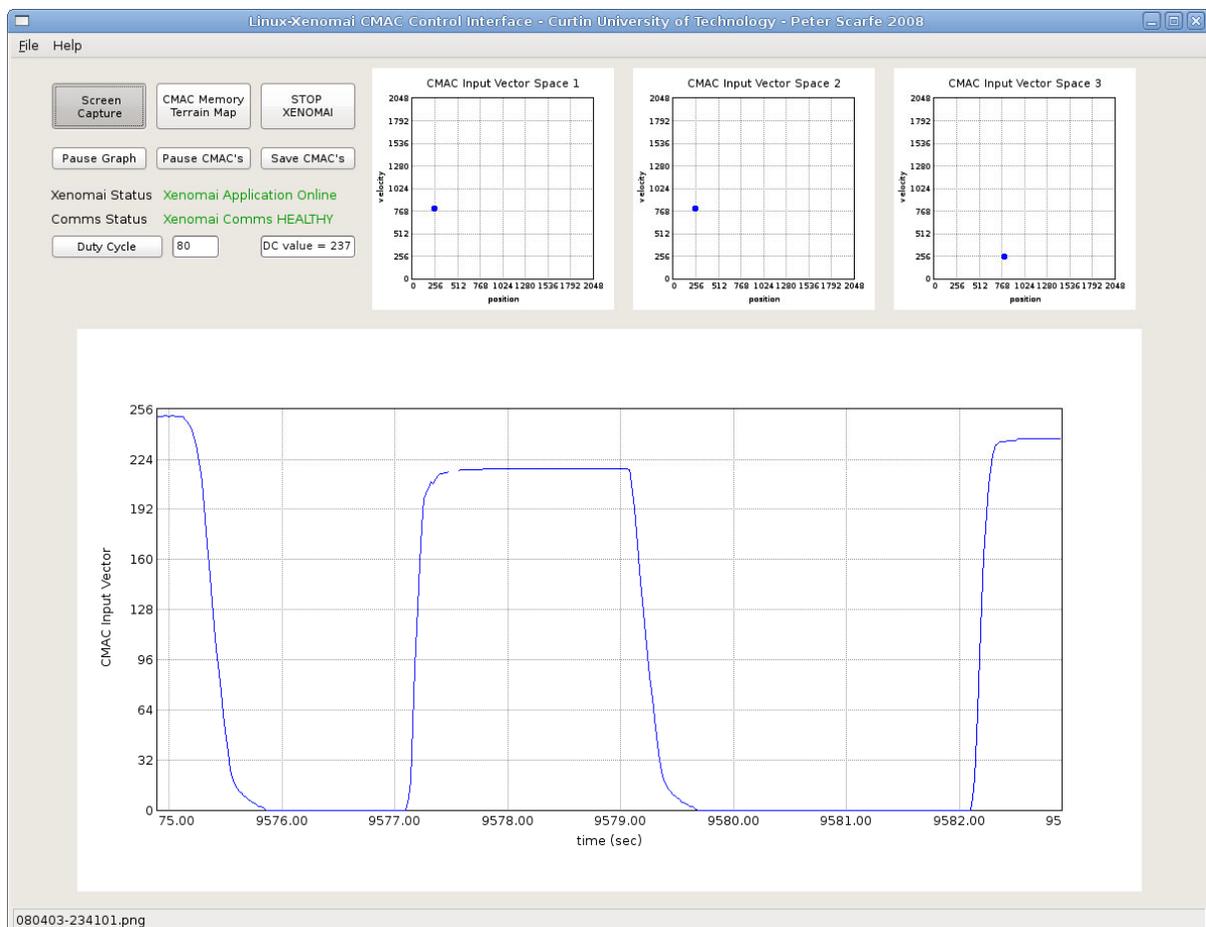
Control Board SupplyVoltage = 11.98V

Pneumatic Supply Pressure: 6.0Bar (read from Supply Regulator)

PBV Board Manifold Pressure: 3.6Bar (read from PBV Board SMC Regulator)

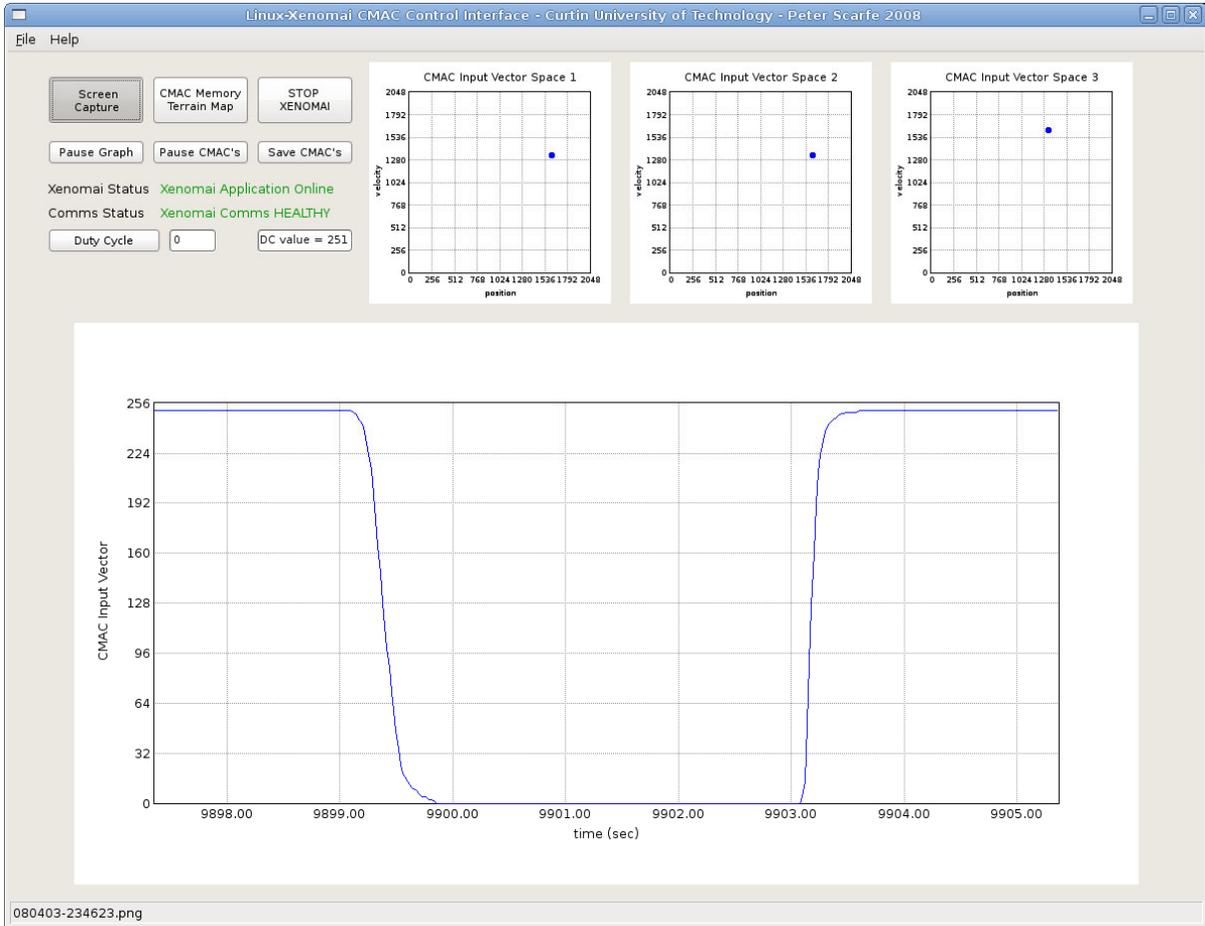
Sequence of Events:

2. 0Bits produces 0.00V, producing 0.21Bar at2 air muscle
2. 255Bits produces 9.48V, producing 3.61Bar at air muscle
3. 90Bits produces 3.97V, producing 1.13Bar at air muscle
4. 255Bits produces 9.97V, producing 3.62Bar at air muscle
5. 80Bits produces 3.48V, producing 0.84Bar at air muscle



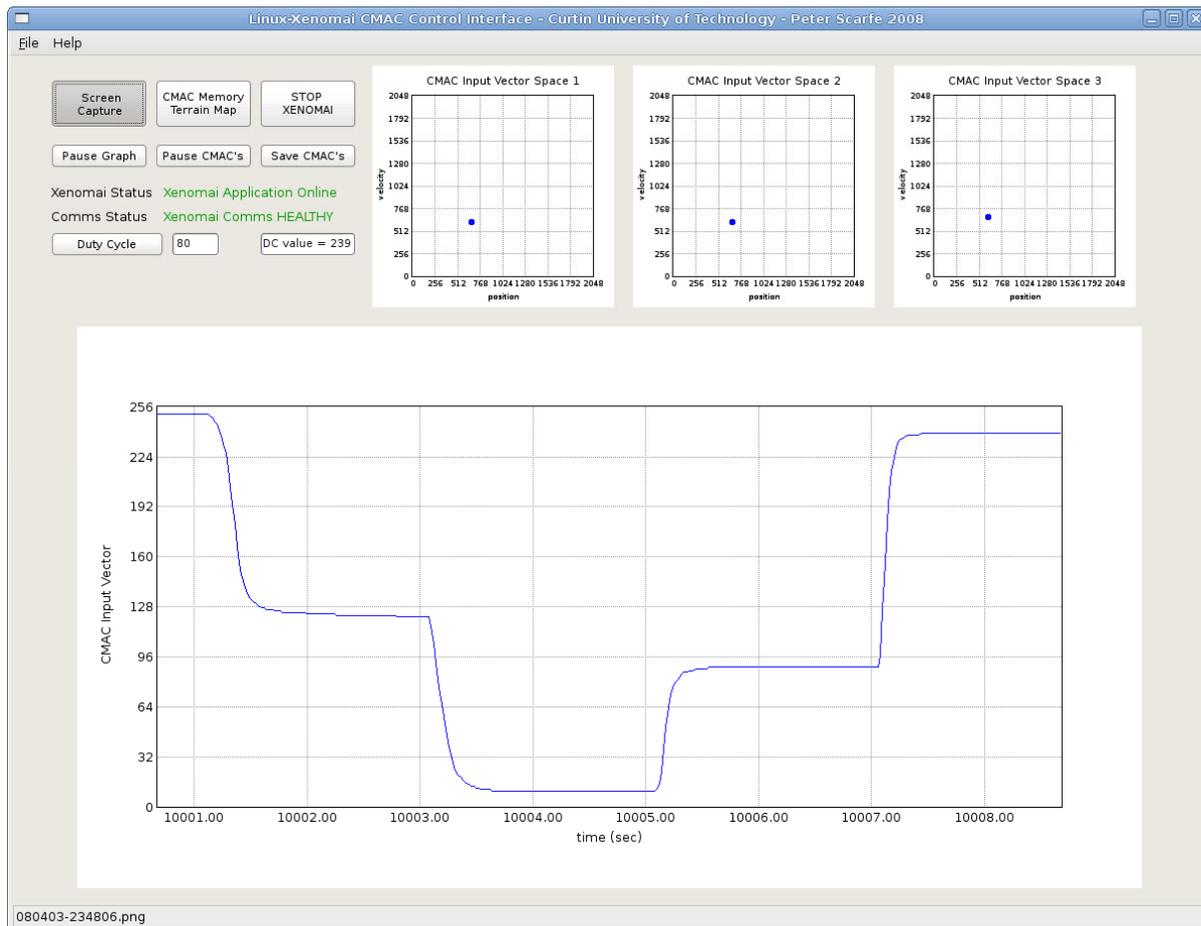
Sequence of Events:

1. 0Bits produces 0.00V, producing 0.21Bar at air muscle
2. 255Bits produces 6.97V, producing 3.61Bar at air muscle
3. 0Bits produces 0.00V, producing 0.21Bar at air muscle



Sequence of Events:

1. 0Bits produces 0.00V, producing 0.21Bar at air muscle
2. 120Bits produces 5.47V, producing 1.94Bar at air muscle
3. 150Bits produces 6.98V, producing 2.93Bar at air muscle
4. 120Bits produces 5.47V, producing 1.94Bar at air muscle
5. 80Bits produces 3.49V, producing 0.83Bar at air muscle



Conclusions:

The responses of all graphs are looking great! One last final tune-up to the Outlet Pipe Gap, inbetween the Intemediate Test and the Semi-Final Test, using an open (w/ stopper) Air Pressure of 0.65Bar.

Final Test:

Minimum AirGap Height: 1.5mm

Tap & Outlet Pipe Gap Setup:

1. Open (no stopper) Air Pressure: 0.21Bar
 2. Open (w/ stopper) Air Pressure: 0.66Bar
- Final Adjusted Open (w/ stopper) Air Pressure: 0.21Bar (@0.00V)

Air muscle Type: McKibben(Silicon 6mmID/8mmOD bladder)

Air muscle Max Length (in test): 143mm (inflatable area only)

Air muscle Max Length (in test): 110mm (inflatable area only)

Max Control Voltage: 9.46V DC

Max Control Voltage: 0.00V DC

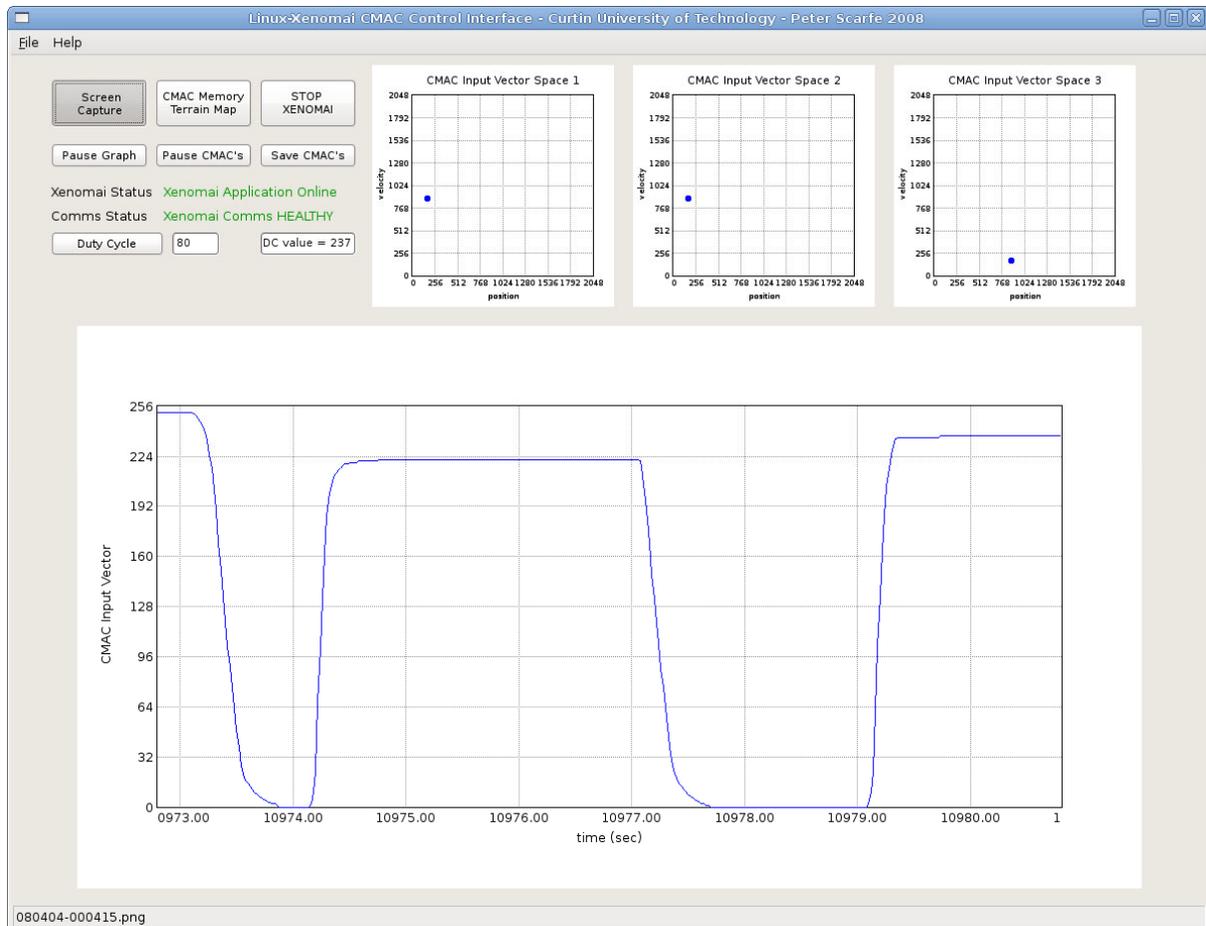
Control Board SupplyVoltage = 11.98V

Pneumatic Supply Pressure: 6.0Bar (read from Supply Regulator)

PBV Board Manifold Pressure: 3.6Bar (read from PBV Board SMC Regulator)

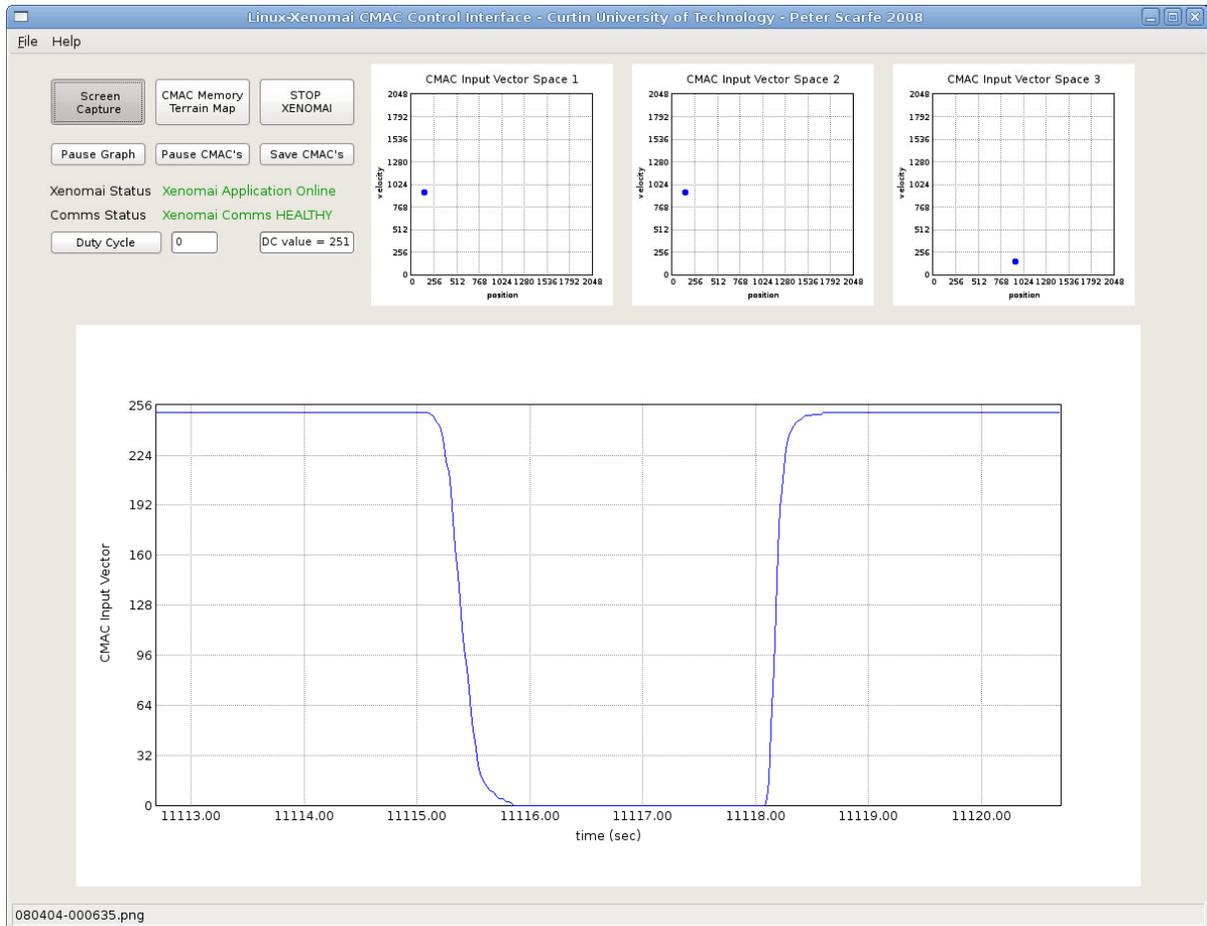
Sequence of Events:

3. 0Bits produces 0.00V, producing 0.23Bar at2 air muscle
2. 255Bits produces 9.48V, producing 3.61Bar at air muscle
3. 90Bits produces 3.97V, producing 1.09Bar at air muscle
4. 255Bits produces 9.97V, producing 3.61Bar at air muscle
5. 80Bits produces 3.48V, producing 0.81Bar at air muscle



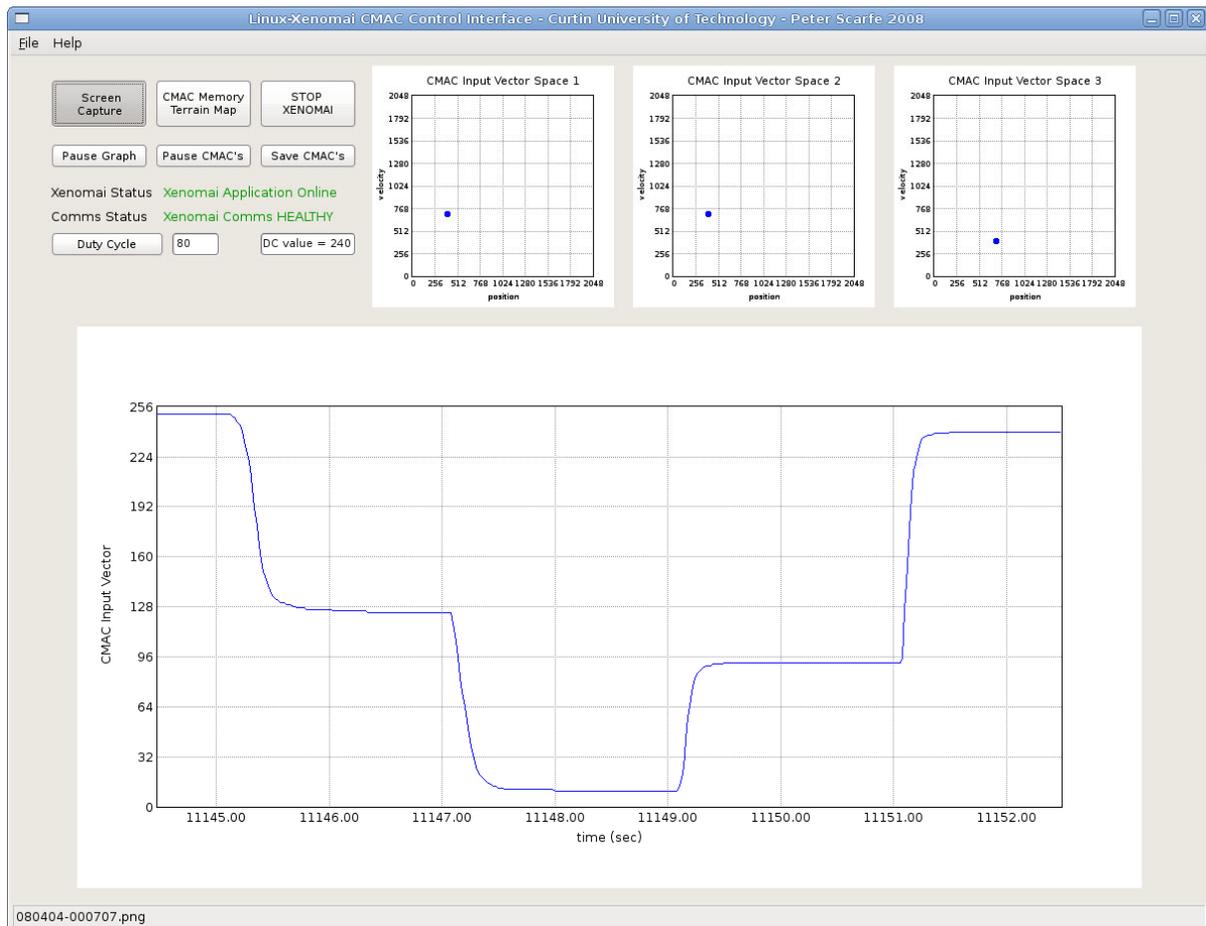
Sequence of Events:

1. 0Bits produces 0.00V, producing 0.21Bar at air muscle
2. 255Bits produces 6.97V, producing 3.61Bar at air muscle
3. 0Bits produces 0.00V, producing 0.21Bar at air muscle



Sequence of Events:

1. 0Bits produces 0.00V, producing 0.21Bar at air muscle
2. 120Bits produces 5.47V, producing 1.83Bar at air muscle
3. 150Bits produces 6.98V, producing 2.82Bar at air muscle
4. 120Bits produces 5.47V, producing 1.79Bar at air muscle
5. 80Bits produces 3.49V, producing 0.78Bar at air muscle



Conclusions:

The response is basically the same as the the Semi-Final test. The responses from the Quarter Final test and the Semi-Final and the Final tests all differ only marginally. Thus the results seem to show that using an open (w/ stopper) Air Pressure of between 0.7-0.6Bar is most appropriate. This range also helps setting up due to the trickyness of adjusting this VERY fine physical parameter by hand.

Thus based on all the testing, the following parameters should be used for optimal PBV value configuration: (see next page)

Optimal PBV Valve Parameters:

Maximum AirGap Height:	1.5mm
Open (no stopper) Air Pressure:	0.21Bar
Open (w/ stopper) Air Pressure:	0.6 – 0.7 Bar
Final Adjusted Open (w/ stopper) Air Pressure:	0.21Bar (@0.00V)

The above parameters have been found to be optimal using the following system:

Air muscle Type: McKibben(Silicon 6mmID/8mmOD bladder)

Air muscle Max Length (in test): 143mm (inflatable area only)

Air muscle Max Length (in test): 110mm (inflatable area only)

Max Control Voltage: 9.46V DC

Max Control Voltage: 0.00V DC

Control Board SupplyVoltage = 11.98V

Pneumatic Supply Pressure: 6.0Bar (read from Supply Regulator)

PBV Board Manifold Pressure: 3.6Bar (read from PBV Board SMC Regulator)