An Integrated System for Autonomous Robotics Manipulation

J. Andrew Bagnell, Felipe Cavalcanti, Lei Cui, Thomas Galluzzo, Martial Hebert,
Moslem Kazemi, Matthew Klingensmith, Jacqueline Libby, Tian Yu Liu,
Nancy Pollard, Mihail Pivtoraiko, Jean-Sebastien Valois and Ranqi Zhu[‡]

***Abstract*— We describe the software components of a robotics system designed to autonomously grasp objects and perform dexterous manipulation tasks with only high-level supervision. The system is centered on the tight integration of several core functionalities, including perception, planning and control, with the logical structuring of tasks driven by a *Behavior Tree* architecture. The advantage of the implementation is to reduce the execution time while integrating advanced algorithms for autonomous manipulation. We describe our approach to 3-D perception, real-time planning, force compliant motions, and audio processing. Performance results for object grasping and complex manipulation tasks of in-house tests and of an independent evaluation team are presented.**

I. Introduction

Our work focuses on developing software that enables a robot to autonomously manipulate, grasp, and perform complicated everyday tasks with humans providing only high-level supervision. This work was performed as part of a DARPA Autonomous Robotic Manipulation–Software track (ARM-S). Six teams, each with identical hardware, participated in a fast-pased initial phase centered around grasping and manipulation challenges. Grasping centered around detecting, localizing and grasping everyday objects like screwdrivers, hammers, shovels, that are typically small relative to the hand. The actual objects during testing are often novel variants of the class of object, e.g., testing on a new screwdriver never seen before. In addition, objects with small size relative the hand were deliberately selected to showcase the dexterity of our approach.

More advanced manipulation challenges included drilling into a block of wood at spot indicated by a red dot, unlocking and opening a door, hanging up a phone, and turning a flashlight on. The performance of each team was measured by the successful task completion percentage, as well as by the average execution time over multiple trials. We attributed our successes to design decisions that enabled integration of state-of-the-art perception and decision making algorithms and an architecture supporting easy composition of primitive behaviors into complex and robust manipulation programs. In this paper, we present the details of our design and implementation.

Our choice of algorithms and overall design is influenced by two critical elements. First, the *execution speed* of the software affects the system's ability to react to the progression of events. Hence, a speed up allows errors to be caught faster, minimizing plan changes and resulting actions. Second, the *quality and performance* of the algorithms affect the feedback information that is propagated through the system, and in turn, the sophistication of subsequent data consumers to deal with error sources. The robotics manipulation architecture proposed here focuses on both accuracy and speed. This comes from both the selection of algorithms and organization of the software architecture.

[‡]All the researchers are with the Robotics Institute at Carnegie Mellon University, Pittsburgh, PA, USA.

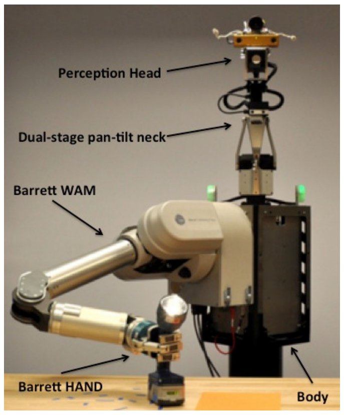


Fig. 1: Robotics manipulation platform.

The robotics platform provided for development is shown in Fig. 1. Mounted on top of the perception head is a high resolution (5 mega pixel) monocular color camera with a fixed focal length lens. A Bumblebee2® stereo pair and an SR4000® time-of-flight camera provide 3-D sensing capability. Two Audio-Technica cardioid condenser microphones form a stereo pair, which act as ears for the perception head. The head is mounted on a 4-axis neck capable of pan, tilt and forward motions for sensor positioning. Attached to the fixed torso is a 7-DOF Barrett WAM with a 3-fingered robotic hand. The hand is attached to the WAM through a 3-axis force-torque sensor. The palm and each finger is equipped with a 24-cell pressure sensor.

Related Work: The field of robotic manipulation is comprised of numerous fully integrated (mobile and non-mobile) systems designed for performing grasping and manipulation tasks. A detailed review of all those platforms and related research is beyond the scope of this paper, and here we only mention some of the efforts towards the development of fully autonomous systems that are closely related to our work.

Although the existing manipulation platforms share similarities and differences in hardware design, they are mainly distinguished by their software architecture, the variety of manipulation tasks they can accomplish, and their level of autonomy. A force sensing and compliant humanoid, called Domo [1] was developed in the Humanoid Robotics Group at MIT as a research platform for exploring general dexterous manipulation, visual perception, and learning tasks. As a main distinction, Domo employs Series Elastic Actuators in its arms, providing natural compliance for safe interactions with the environment. Researchers at UMass Amherst have developed a variety of platforms for robotic manipulation, e.g., UMan [2] and Dexter [3], that integrate perception and manipulation capabilities. Similarly to our system, both UMan and Dexter use the combination of Barret WAM arms and hands as their manipulators which provide compliance through cable driven joints, requiring active control of compliance using accurate force sensing. Our system implements a number of active force compliant motions towards grasping and manipulation of objects[4]. HERB, developed at the Personal Robotics laboratory at Carnegie Mellon University [5], is a mobile manipulation platform developed for performing manipulation tasks in human environments. HERB has been employed to perform daily tasks such as fetching objects by integrating perception and manipulation planning, and open-loop execution of planned motions. Finally, the PR2 platform developed at Willow Garage [6] provides a research and development platform based on the Robotic Operation System (ROS) for robotic manipulation. Each of the existing platforms try incorporate some aspects of autonomous robotic manipulation to accomplish specific tasks with less focus on full integration of all required components including: perception, planning, force compliant motions, and closed-loop execution of planned motion. In our design we paid careful attention to the choice of these techniques as well as their integration to achieve a high level of robustness and repeatability for a variety of tasks as demanded by the competitive nature of the DARPA ARM-S program.

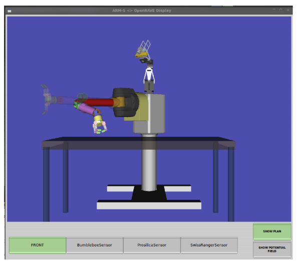
In Section II we give an overview of the system design and the main components related to perception, planning, control and behavior execution. Section III covers statistical performance of our system at various tasks. Finally, the conclusion is found in Section IV where a summary of our work, together with future improvements are described.

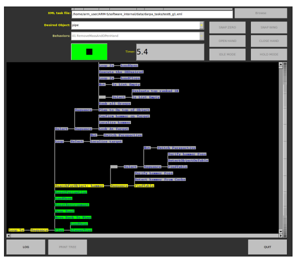
II. Software Design

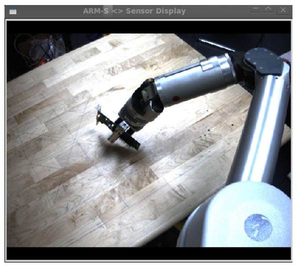
Our software relies on several open-source packages that are ubiquitous in robotics. We use the Robot Operating System (ROS) for inter-process communication between the main application, all the sensor interface applications and our robot controller. The Open Robotics Automation Virtual Environment (OpenRAVE) is primarily used for storing the environment representation of the manipulators, sensor positions, and other scene models. We use the OpenWAM [1] software to control both the WAM arm and Barrett.

The design tenets of our software are rapid perception/decision making feedback loops, simple compositional architecture, and globally accessible state containers.

[1]The software and documentation for OpenWAM can be found at http://personalrobotics.ri.cmu.edu/intel-pkg/owd/html


(a)


(b)


(c)

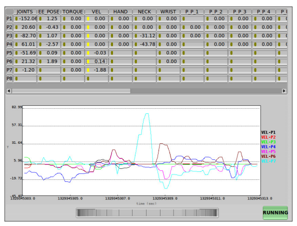

(d)

Fig. 3: Windows from the system user interface. a) 3D graphical renderer of the OpenRAVE environment. b) Behavior visualization and execution. c) Image renderer d) Sensor data plotter.

Through the use of ROS, we implemented data filtering algorithms to automatically process data coming from, or sent to, various sensors and actuators. For example, images from the stereo camera are rectified and placed into an image cache for later consumption by the vision algorithms. Complex tasks are built upon grouping simple and rapid-execution action primitives into behavior tree structures. Finally, the use of globally accessible data "blackboards" simplifies access to previously saved states without passing data pointers between functions.

The software diagram as implemented is shown in Fig. 2. At the top level, a behavior tree orchestrates the execution of tasks through direct access to the data processing classes. The OpenRAVE data structure stores the environment states, which are constantly updated from telemetry, pose filters, and from perception algorithms. The robot interface class abstracts connections to various perception sensors, neck control, and to the OpenWAM controller. The sensor class stores a list of images for easy access by the perception algorithms. Finally, a graphical user interface provides behavior execution control, sensor visualization and data plotting, together with rendering of the OpenRAVE environment. Typical windows from the user interface are found in Fig. 3.

The following subsections contain the details of the behavior architecture for tasks execution, and also the planning, control and perception modules implemented.

A. The Behaviors Architecture for Robotic Tasks (BART)

Our system development required a method of task orchestration and execution that facilitated re-use and enabled a team with broad skill sets to contribute easily. Given the task complexity and the need to adjust plans, we implemented a *behavior tree* [7] based architecture to help meet these

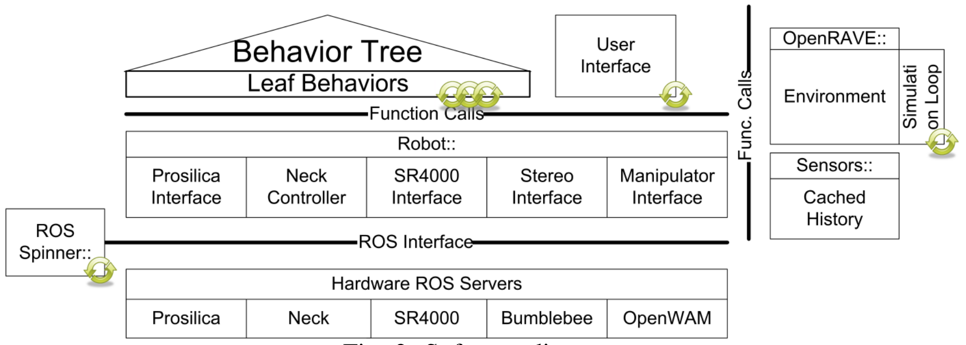


Fig. 2: Software diagram.

challenges. Behavior tree are a graphical modeling language primarily used in software engineering. They facilitate the representation of organizational elements in a large-scale system; an ideal tool for building complex manipulation tasks from primitives with limited capabilities. This concept form the basis for our task execution architecture library called BART [2] , Behavior Architecture for Robotic Tasks.

A behavior tree is a hierarchical structure of individual behavior nodes, or simply behaviors. A behavior is a simple first class function object that accepts nothing as input and returns a boolean success value. Each behavior is responsible for carrying out some action or checking some condition for the system. The lowest level of the tree is made up of leaf behaviors. These behaviors connect the tree to the robotic system. They provide a direct interface mechanism that allows feedback information to flow into the tree and data to be output from the tree. Composite behaviors form the higher levels of the tree. They contain both leaf behaviors and sub-composite behaviors. The role of composite behaviors is to structure the way in which sub-behaviors are executed. For example, some composite behaviors can execute sub-behaviors in sequence, and others can execute sub-behaviors in parallel for concurrency.

Behavior trees provide a scalable way to organize the logic and execution of any process. The main difference between a typical programming language and a behavior tree is a dynamic object rather than static code. Therefore it is possible to construct and restructure program execution at run-time rather than at compile time, in ways similar to embedded dynamic scripted languages (e.g. Python, PHP, Ruby). The addition of real-time introspection facilitates the development of behavior trees by providing a way to observe what the system is executing at any time. By seeing which behaviors are running, which have succeeded and which have failed, one can quickly determine how the overall task is performing.

Another benefit is the possibility to implement algorithms that modify the tree structure on-line in order to change how the system performs. For example, the system can automatically plan or learn sequences of behaviors that improve or optimize a certain task [8].

Other architecture libraries such as SMACH [9] employ hierarchical state machines to organize complex robotic tasks. However, there are clear advantages to using behavior trees over state machines. The main advantage is that individual behaviors can easily be reused in the context of another higher-level behavior without needing to specify how they relate to subsequent behaviors. Behavior execution is defined by the context of the parent behavior. In contrast, a state in a hierarchical state machines (HSM) requires the definition of transitions to subsequent states and therefore must be redefined with new transitions when used in a different context. Ease of reuse was the primary reason we chose behavior trees rather than an HSMs for task execution.

Behavior trees have also become popular in the game development industry [10]. Primarily they have been used as an alternative to existing game AI systems, which were previously implemented by finite state machines (FSMs) and in particular HSMs[11].

Types of behaviors: In BART, the behavior tree is implemented as a *binary* decision tree. Each composite behavior may contain only two child behaviors that are executed according to a binary logical operator that is specified by the parent composite. Typically, there are two types of composite behaviors, sequences and selectors. Sequences execute a set of child behaviors one-after-the-other. If one of the child behaviors fails in the sequence, no further child behaviors are executed and the entire sequence behavior fails. This sequence behavior is equivalent to the short-circuiting C++ logic operator ”&&”. Therefore, in BART, sequences are implemented by the ”&&” syntax, which has been overloaded to combine behavior objects together.

With selectors, each child behavior is executed one-after-the-other until one of them succeeds. As soon as a child behavior succeeds, the selector immediately returns success. If no child behavior succeeds then the selector returns failure. This is equivalent to the short-circuiting C++ logic operator ”||”. Like the sequence, we overloaded the ”||” syntax to implement selectors.

BART also implements several other composite behaviors that enable more flexibility. For example, BART has a parallel composite that can execute multiple child behaviors simultaneously in separate threads. We have also implemented the non-short-circuiting versions of the and and or logic operators, which enables a composite to execute all child behaviors and then return the complete logic result to

[2]The C++ software library can be found at https://code.google.com/p/bart/

indicate success or failure.

In addition to composite behaviors, BART also supports behavior *decorators*. These behavior decorators can wrap any existing behavior with additional functionality. For example, one can apply a Loop decorator to a behavior to repeat it a certain number of times or until it succeeds. Other decorators such as *logging* or *announcement* are used to store or print out additional debugging information as the behavior tree is executed.

To illustrate its simplicity, we include three lines of code from the shovel grasping task. The first line uses the *Announce* decorator to warn the operator that the system has started an object search routine, while concurrently moving the arm to an initial configuration using the $*$ operator. On the second line, the *Loop* operator will repeat the routine until successful and then it will sequentially (through the && operator) plan for grasp and then execute the planned motion. Finally, the third line shows how composite behaviors are launched using the *execute* method. Note that the non-bold names are leaf behavior pointers called by the composite behaviors.

1) **Announce**(**Behavior**(findHammer), "Search for Hammer") * **Behavior**(moveToWing);
2) **Loop**(findHammerAndReadyArm) && **Behavior**(planGrasp) && **Announce**(**Behavior**(graspHammer), "Caution, moving arm");
3) findAndGraspHammer.**execute**();

B. Perception Subsystem

Autonomous manipulation requires identifying and localizing relevent workspace objects. This information is needed for motion planning, to determime goal locations for the manipulator end-effector, and for obstacle avoidance. This section explains the algorithms and techniques employed for these perception requirements.

1) Object Identification and Localization: Our perception software is comprised of several behavior layers that break down the process into three steps: detection, verification and localization.

In the detection step, the neck actuators are used to position the sensors to survey the workspace. Individual 3-D point clouds are stitched together and analyzed to segment the background table from potential candidate objects. A plane detection and fitting algorithm using a RANSAC [12] filtering method is used to distinguish points belonging to the table from others belonging to object candidates. Object candidate points are then grouped according to image connected components. Groups having an area that is close to known objects are kept for further analysis. The average point location of filtered clusters is stored into a list to be processed in the following step.

During the verification step, each candidate is re-sampled by centering the sensor view on its estimated location, and repeating previous detection process. The calibrated camera center line is aligned with the candidate location by computing thecompletely neck pan and tilt angle inverse kinematics. This alignment minimizes sensor distortion effects and helps ensure that the object will be completely contained in the field of view. After alignment, the background subtraction process is repeated and if the remaining cluster size is still within the appropriate size range, the object candidate is considered "verified" and the next step, localization, is executed.

For localizing known objects we match the observed data with coarse 3-D geometric templates that are generated *a priori*. The templates cover expected orientations of an object at a 5 to 10 degree resolution. Observation-to-template similarity is correlated by segmenting and comparing depth information. The result is an estimated object pose and match "quality" score that is used to verify identification of the desired object and its location. After the initial estimation, the localized pose is refined through a viewpoint constrained Iterative Closest Point (ICP) algorithm that corresponds 3-D points from the object model to the observed 3-D data. The model point set is reduced such that only points visible from the estimated camera viewpoint are included in the refinement. This eliminates the effect of model and sample outliers degrading the pose regression. If there are multiple object candidates, the one with the best match "quality" score is taken as the solution. Fig. 4 presents the flow sequence of this localization process.

Additional 2-D vision information– including expected colors and edge/texture statistics– are used to improve the match score, particularly on object classes that are difficult to detect purely geometrically. Such data reduces the generality of the approach, but improves accuracy on objects for which stereo and time-of-flight data is sparse or inaccurate.

C. Hand Tracking

Hand tracking is a necessary element to reduce errors introduced by inaccuracies in the WAM joint positions. Our approach is to track the arm using an edge detection method that is based on the RAPiD[13] algorithm. The robustness of the method was improved through several modifications. First, we augmented the algorithm to use both images from our stereo camera such that only consistent edges from the left and right images are preserved. Second, we extended the algorithm with multiple re-projection iterations in a tracking fashion to reduce edge location uncertainty. Third, the algorithm is run multiple times for each frame with a set of candidate starting position seeds in order to avoid local minima. From those seeds, we pick best match according to a linear regression classifier that was trained on labeled convergences. Currently, our hand tracking system runs at 4fps and achieves pose estimation precision of 5 mm and 3 degrees. In Fig. 5 we summarize this process in a block diagram.

D. Sound Classification of Drill Speed

The stereo pair of microphones on the perception head is used to classify sounds that the arm or hand might induce while interacting with objects during various tasks. This knowledge provides feedback to aid in successful task execution. For example, sound analysis can be used to detect whether the trigger of an electric drill has been pressed fully or not. Hence, the speed of the drill is directly correlated

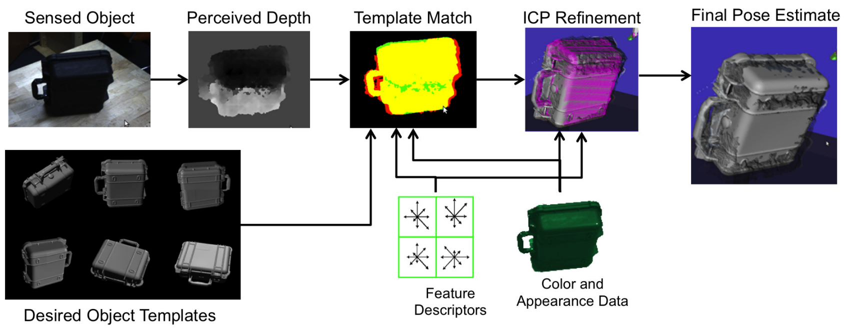


Fig. 4: Perception localization pipeline.

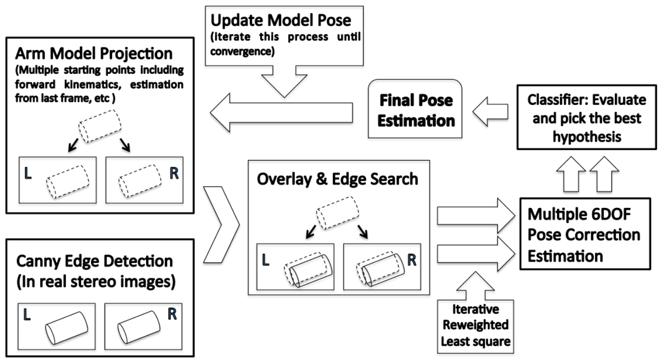

Update Model Pose
(Iterate this process until convergence)
Arm Model Projection
(Multiple starting points including forward kinematics, estimation from last frame, etc)
L
R
Final Pose Estimation
Classifier: Evaluate and pick the best hypothesis
Overlay & Edge Search
Multiple 6DOF Pose Correction Estimation
Canny Edge Detection
(In real stereo images)
Iterative Reweighted Least square

Fig. 5: Robot arm tracking method chart.

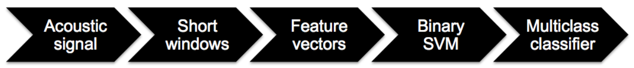


Fig. 6: The processing pipeline for sound classification.

to how far the drill lever is depressed by the fingers. It is easier to hear the drill speed, rather than try to see the finger position with the cameras or feel the finger position with pressure and torque sensors.

We discretized the speed of the drill into 3 states: low speed, medium speed, and high speed. This allows us to classify the sound of the drill into one of three classes. The task of turning on the drill is only successful if it reaches the high speed class. The processing pipeline for this classification is depicted in Fig. 6.

The raw signal is sampled at 44.1 kHz with a 16 bit word. The digitized signal is then split into short windows, around 0.5 second in length. An FFT is used to convert each window into the frequency domain, and then various feature vectors are extracted from both the frequency domain and the original time domain. The feature vectors from the right and left stereo pair are averaged together. These vectors are then used as inputs into multiple binary Support Vector Machines (SVMs), one for each pair of classes. The binary SVM outputs are then combined into a multi-class classifier using a Decision Directed Acyclic Graph. The SVMs were trained offline using hand labeled data, by running trials with the drill at each of the three speeds. The classifier can then run online using the trained models. Refer to [14] for more details on the technical approach.

E. Motion Planning

The motion planning component of the system is responsible for computing trajectories for the arm to complete grasping and manipulation tasks. For planning algorithms, we specify the tasks in the robot operational space [15]. This space is more suitable for collision detection and for computing the desired hand motion. However, the mapping between the operational into a configuration ($\mathcal{C}$) spaces of the manipulator is required for arm control, where joint angle trajectories are used.

We approached this problem in a hierarchy consisting of three layers, where the lowest level planner is executed first, and subsequent higher level, i.e more complex and slower, planners are only executed if the previous layer fails. At the root of the hierarchy is the fastest operational space planner based on inverse kinematics for direct linear hand motions. This planner is capable of detecting trajectories that are in collision along its path, although it is not able to avoid them. The second planning layer is a variant of the CHOMP algorithm [16]. It is capable of modifying trajectories so to avoid collisions using a covariant gradient descent. Finally, the third layer uses a global planner based on the Bi-RRT [17] algorithm, enabled with an extension to satisfy end-effector pose constraints [18] in order to accomplish certain tasks, such as carrying objects without changing their orientation during the motion. The details of all the layers of planning are presented in the following subsections.

1) Inverse Kinematics: The inverse kinematic planner is the first to be executed. It computes a set of configurations that correspond to a given workspace pose of the end-effector using the *IK-fast* method [19]. Each element q_{IK} in the resulting set of configurations is then tested for collision. The configurations that are free from collision (satisfy the free-space constraint) are further sorted based on a task-dependent *utility* function, $\mathcal{F}_{IK} : \mathcal{C} \rightarrow \mathbb{R}$, that attempts to assess the quality of a configuration with respect to the task being accomplished. For example, this assessment may be arm manipulability at q_{IK} [20] or proximity to obstacles. For some tasks, the initial (e.g. stowed) arm configuration, q_I, may be factored into $\mathcal{F}_{IK}$: for example, it may be preferred to minimize the $\mathcal{C}$-space distance between q_I and q_{IK} or obstacle proximity along the geodesic connecting the two configurations, computed using operational space control in Section II-F.1. The feasible q_{IK} are stored in a priority queue based on their $\mathcal{F}_{IK}$ value. Using the highest ranking q_{IK}, the trajectory from q_I to q_{IK} is sampled at a regular intervals for collision, joint limits and singularities. If this operation is successful, the trajectory is used, otherwise the next planner in the queue is attempted.

2) Covariant Optimization-based Planning: This approach utilizes numerical optimization in order to compute a trajectory that is both free of collision and smooth. An initial trajectory is first chosen based on a heuristic; for example a geodesic in $\mathcal{C}$-space, or a straight line in the robot operational space. Suppose the endpoint configurations of this trajectory are q_I and q_F. Then the problem of computing a feasible trajectory can be stated as minimization of a certain cost function $\mathcal{F}$ under the constraint that the trajectory is free of collision. We assume a time-independent *cost function* $\mathcal{F} : Q \times \mathcal{U} \to \mathbb{R}$ assigns cost to steering the system with a control u while it is at configuration $q \in \mathcal{C}$. In practice, cost may be related to time duration of the associated control, energy consumed by the system during motion, risk experienced, etc. The objective function is typically designed such that the robot's perceptual information is mapped into a scalar value that is consistent with the desired notion of motion cost. We define the cost function as a weighted sum of component functions, representing a number of desired performance criteria:

$$\mathcal{F}(q_I, u) = \sum_{i}^{k} w_i f_i(q_I, u) \tag{1}$$

for a weight vector $\mathbf{w} \in \mathbb{R}^k$. Several components of the cost are most relevant in this setting, including obstacle, smoothness and end-effector motion constraint costs. These are given below in this sequence.

The *smoothness cost*, $f_s(\tau_u(t))$, is a function of the trajectory. This function is a sum of squared derivatives. For each derivative $d = 1, \ldots, D$, we assume a finite differencing matrix K_d and represent smoothness cost as a sum:

$$f_s(\tau_u) = \frac{1}{2} \sum_{d=1}^{D} w_d \|K_d \tau_u + e_d\|^2 \tag{2}$$

where constants e_d represent the boundary values q_I and q_F, and w_d are weights for each of the cost component. The smoothness cost can be re-written as quadratic form:

$$f_s(\tau_u) = \frac{1}{2} \tau_u^T A \tau_u + \tau_u^T b + c \tag{3}$$

for the appropriate constants, including a positive definite A. The latter will also be used as the metric for covariant gradient descent.

The *collision* cost function is assumed to be continuous and differentiable. One approach to obtaining such a cost function for representing environment obstacles is to use a signed distance field $d : \mathbb{R}^3 \to \mathbb{R}$ which gives the distance from a point $\mathbf{x} \in \mathbb{R}^3$ to the nearest boundary of an obstacle. Values of the distance field are negative inside obstacles, positive outside, and zero at the boundary.

Discrete variants of $d(\mathbf{x})$ can be derived that operate on a voxel grid representation of distance fields. A number of fast techniques to compute discrete distance transforms exist; for example, a method due to [21] has linear complexity in the number of voxels in the grid. Unfortunately, this computation still takes several seconds on modern hardware for typical manipulator reachability regions, even at the minimal admissible voxel resolution. Moreover, it is beneficial to increase this resolution as much as possible, as it improves convergence of the optimizer. Therefore, it is beneficial to utilize boundary representations of the objects with geometric primitives (such as spheres, boxes, etc.) for which distance computations have a closed form.

Finally, it is beneficial in many settings to impose a *workspace* cost on certain robot motion. In the context of manipulation, visual servoing and other applications, one might desire to constrain the end-effector of the manipulator to be in certain region in $SE(3)$. As stated, any continuous and differentiable cost function can be utilized in this approach. Given a certain element of the robot body, $b \in \mathcal{B}$ and its workspace pose, we can setup a per-dimension cost function that is 0 if b satisfies the desired constraint, otherwise the cost is quadratic with respect to distance from the constraint. Using the x- dimension as an example, suppose we desire to limit b's pose such that its x-coordinate is greater than $x_{\min}$, we can define a workspace cost potential:

$$c\,(x) = \begin{cases} (x - x_{\min})^2 & \text{if } x < x_{\min} \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

In practice, we found that the performance of the planner can be sensitive to the initial trajectory seed, and we generate a sequence of a few different initial trajectories based on the contextual information from the environment. To minimize the number of initial trajectories attempted, the order of the elements in the sequence is ranked by leveraging the submodular nature of these attempts. Each trajectory in the sequence is selected to maximize the marginal improvement in planning success of the sequence, assuming all previous elements have failed.[8]

F. Control and Grasping

1) Operational Space Control: We employ a velocity-based operational space formulation to generate the desired joint velocities $\dot{\mathbf{q}}_d$ for a given robot end-effector velocity $\dot{\mathbf{x}}_d$. We favor this approach due to its overall good performance compared to other variations of operational space control such as presented in [22].

The desired joint velocities for a given hand velocity $\dot{\mathbf{x}}_d$, are calculated using Liegeois' resolved motion rate control approach [23] as

$$\dot{\mathbf{q}}_d = \mathbf{J}^{+}\dot{\mathbf{x}}_d + \lambda(\mathbf{I} - \mathbf{J}^{+}\mathbf{J})\nabla\mathbf{H}(\mathbf{q}), \tag{5}$$

where $\mathbf{J}$ is the robot Jacobian with its pseudo-inverse denoted $\mathbf{J}^{+}$, λ is a gain value, and $\mathbf{H}(\mathbf{q})$ is a null-space cost/utility function. Different criteria can be used to define $\mathbf{H}(\mathbf{q})$ depending on the objective, e.g., staying away from joint limits or kinematic singularities.

The desired motors torque command is calculated using the computed torque control method with an added velocity feedback [22] to track the desired joint velocities in (5),

$$\boldsymbol{\tau} = \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}}_d + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q}) + \mathbf{K}_{q,d}(\dot{\mathbf{q}}_d - \dot{\mathbf{q}}) \quad (6)$$

where $\mathbf{M}(\mathbf{q})$ is the inertia matrix, $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ is the Coriolis/centrifugal vector, $\mathbf{g}(\mathbf{q})$ is the gravity vector, $\mathbf{K}_{q,d}$ is a gain matrix, and $\boldsymbol{\tau}$ is the joints torque vector. The desired joint acceleration $\ddot{\mathbf{q}}_d$ is obtained by differentiating $\dot{\mathbf{q}}_d$.

The end-effector velocity is determined on a task by task basis, but typically it performs position-based servoing of the end-effector to a desired pose $\mathbf{x}_d$ in task space until contact with an object. In such case, the desired hand velocity is calculated as $\dot{\mathbf{x}}_d = -\mathbf{K}_{x,p}(\mathbf{x}(t) - \mathbf{x}_d)$ where $\mathbf{K}_{x,p}$ is a positive proportional gain. We use the above operational space control formulation to achieve the desired end-effector velocities required by many of the manipulation behaviors.

2) Force Compliant Grasping Primitives: Although our perception system provides reasonable object pose estimation, the combination of localization and kinematic errors create a situation where the hand is positioned far enough from its intended target to affect the manipulation performance. Such inaccuracy can be dealt with by using additional sensing information from the tactile and force/torque sensors to determine when the hand is in contact with certain objects. Despite promising progress in geometric grasp planning and the efforts devoted to analyzing grasp properties, challenges still remain in real-world manipulation tasks because these method typically fails to consider opportunities presented by contacts with the environment. In our view, supported by our empirical studies, two factors must be considered for achieving robust grasps: 1) dealing with object-to-hand relative positioning uncertainties, and 2) handling contacts between the robot and objects in the environment using compliant control methods.

Most approaches to grasping attempt to model uncertainty and devise plans for it. In contrast, we believe that the effect of uncertainty can be ignored simply by *caging* the objects and using compliant interactions between the robot and the environment. Because we allow contact with the environment, compliant controllers are crucial to ensure safety of the robot. Our approach is motivated by anecdotal experiments with human subjects which demonstrate similar behavior in human grasping of small objects. The use of compliance for grasping small objects is one of the main contribution of our work.

We present three simple, yet effective, manipulation primitives for robust grasping (and releasing) of small objects from support surfaces [4]: (1) *Compliant Finger Placement* for bringing all fingers safely in contact with the support surface, (2) *Compliant Object Grasping* for maintaining the contact between the fingertips and the support surface during the finger closure, and (3) *Compliant Object Release* for releasing and placing the object gently on the support surface using a method similar to step 2. An example of landing and grasping primitives is shown in Fig.7. In these primitives the hand is controlled in compliant with the forces exerted

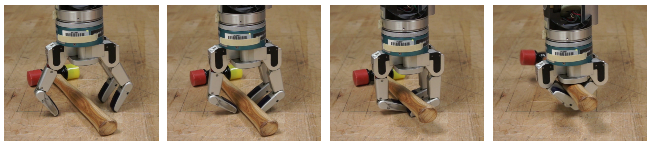


Fig. 7: Grasping of a hammer composed of three steps: compliant landing, compliant grasping, and compliant releasing

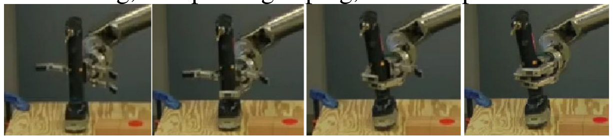


Fig. 8: Steps for drill grasping using compliant manipulation: touch shaft, touch base, capture shaft, and actuate the button.

to the fingertips while the fingers make contact with the support surface during landing, grasping, and releasing. We encourage the reader to study [4] for more details regarding the implementation of these primitives.

We conducted extensive grasping experiments on a variety of small objects with similar shape and size. The results demonstrates that our approach is robust to localization uncertainties and applies to many real-world objects.

3) Force Compliant Manipulation Primitives: The salient feature of the manipulation tasks is that position uncertainties caused by the perception system and robot positioning errors far exceed the accuracy required by the tasks,and visual assistance is not available to resolve the uncertainty due to low resolution and/or occlusion. Under this scenario, It is necessary to apply a localization strategy to dealing with the uncertainties using force sensing.

We developed hand coded behaviors to utilize guarded moves to effectively reduce pose uncertainties between the perception system outputs and the hand location. Guarded moves are known to be much more accurate than the absolute motion with respect to the world frame to place the end-effector to the desired configuration. A sample force compliant motion sequence is given in Fig. 8. It illustrates the steps used to capture a known drill, and later to pull on the drill button to active it. The sequence starts with the hand fully open and translating towards the drill shaft until a contact is made. This is used to reduce uncertainty in the plane of the table. Next, the bottom finger is closed, and the hand is moved down until the finger contacts the bottom battery. The relative height of the button is thus established. Finally, the remaining fingers are closed to firmly capture the drill and actuate the button.

III. Tasks and Performance

Throughout the development period, the system was tested on both object grasping, and on more complex, lengthier, manipulation activities. For grasping, success was measured by the robot's ability to lift an object off a table surface, and gently place it over a letter-sized paper target (without dragging the object). The typical object travel distance was between 40 to 50 cm. Test objects were selected for their diversity in shape, texture and mass. Common household objects that were used are shown in Fig. 9.

Grasping	Rock	Maglite	Ball	Radio	Driver	Hammer	Shovel	Pelican	Floodlight
Num. Trials	26	24	32	24	30	35	33	37	25
Success Rate	100%	92%	100%	88%	92%	97%	82%	97%	84%

Manipulation	Stapling	Flashlight On	Open Door	Unlock Door	Drill Hole	Hangup phone
Num. Trials	36	32	36	20	38	31
Success Rate	95%	91%	88%	95%	90%	89%

TABLE I: System performance for grasp and manipulation tasks

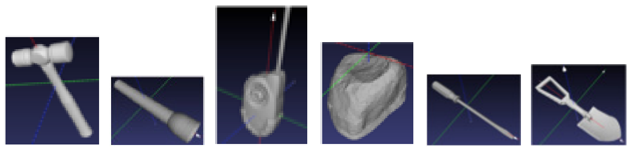


Fig. 9: Example of objects grasped by the system.

Manipulation tasks included stapling papers, turning a flashlight on, opening a door, unlocking a door using a key, and hanging up a princess phone. In addition, one of the tasks was to drill a 2cm deep hole in a piece of 4”x4” lumber using a standard drill. The desired hole location was indicated by a red dot on the lumber surface. Results from several test runs are presented in Tab. I. Notice for the grasping tasks that our success rates tend to be lower for small laying down objects. Most of those failures were due to a finger “break-away” hardware issue, where proximal finger joints locked when the system contacted the table top incorrectly. The other main mode of failure was from inaccurate object localization due to calibration error in both the perception system and the forward kinematics.

We have posted video clips for certain grasping and manipulation experiments:

- http://www.youtube.com/watch?v=sukjzddaCUc
- http://www.youtube.com/watch?v=SCO7Ws3jujQ

IV. CONCLUSIONS AND FUTURE WORK

The software design and techniques presented here were developed as part of a project for autonomous robotic manipulation where tasks were performed with minimal human intervention. We found that our approach was effective at localizing and grasping individual objects–including versions never seen before, as well as manipulating objects for simple tasks. On-going work focuses on both dual-arm manipulation and more intricate tasks. Improvements of the system will include: dual-hand tracking to minimize the need for extrinsic calibration accuracy, localization of objects in dynamic and cluttered workspaces, bi-manual motion planning, and increased speed of task execution.

V. ACKNOWLEDGMENTS

The authors gratefully acknowledge funding under the DARPA Autonomous Robotic Manipulation Software Track (ARM-S) program.

REFERENCES

[1] A. Edsinger-Gonzales and J. Weber, “Domo: a force sensing humanoid robot for manipulation research,” in *IEEE/RAS Intl Conf. on Humanoid Robots*, 2004, pp. 273 – 91.

[2] D. Katz, E. Horrell, O. Yang, B. Burns, T. Buckley, A. Grishkan, V. Zhylkovskyy, O. Brock, and E. Learned-miller, “The umass mobile manipulator uman: An experimental platform for autonomous mobile manipulation,” in *In Workshop on Manipulation in Human Environments at Robotics: Science and Systems*, 2006.

[3] S. Hart, S. Ou, J. Sweeney, and R. Grupe, “A framework for learning declarative structure,” in *Proc. RSS Workshop: Manipulation for Human Environments*, 2006.

[4] M. Kazemi, J.-S. Valois, J. A. Bagnell, and N. Pollard, “Robust object grasping using force compliant motion primitives,” in *Proceedings of Robotics: Science and Systems*, Sydney, Australia, July 2012.

[5] S. Srinivasa, D. Ferguson, C. Helfrich, D. Berenson, A. Collet, R. Diankov, G. Gallagher, G. Hollinger, J. Kuffner, and M. VandeWeghe, “Herb: A home exploring robotic butler,” *Autonomous Robots*, 2009.

[6] W. Garage. Pr2 platform. [Online]. Available: http://www.willowgarage.com/pages/pr2/overview

[7] R. G. Dromey, “From requirements to design: Formalizing the key steps,” in *Conference on Software Engineering and Formal Methods (SEFM 2003)*, 2003, pp. 43 – 53.

[8] D. Dey, T. Y. Liu, M. Hebert, and J. A. Bagnell, “Predicting contextual sequences via submodular function maximization,” Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-12-05, February 2012.

[9] J. Bohren and S. Cousins, “The smach high-level executive,” *IEEE Robotics & Automation Magazine*, vol. 17, no. 4, pp. 18 – 20, 2010.

[10] C. Lim, R. Baumgarten, and S. Colton, “Evolving behavior trees for the commercial game defcon,” in *EvoGAMES*, 2010.

[11] M. Cutumisu and D. Szafron, “An architecture for game behavior ai: Behavior multi-queues,” in *Fifth Artificial Intelligence for Interactive Digital Entertainment Conference*, 2009.

[12] M. Fischler and R. Bolles, “Random sample consensus: A paradigm for model fitting applications to image analysis and automated cartography,” vol. 24, no. 6, pp. 381 – 395, 1981.

[13] C. Harris, “Tracking with rigid objects,” *MIT Press*, 1992.

[14] J. Libby and A. J. Stentz, “Using Sound to Classify Vehicle-Terrain Interactions in Outdoor Environments,” in *IEEE International Conference on Robotics and Automation (ICRA)*, St. Paul, MN, May 2012.

[15] O. Khatib, “A unified approach for motion and force control of robot manipulators: The operational space formulation,” *IEEE Journal of Robotics and Automation*, vol. 3, no. 1, pp. 43–53, 1987.

[16] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, “CHOMP: Gradient optimization techniques for efficient motion planning,” 2009.

[17] J. Kuffner, J. J. and S. M. LaValle, “Rrt-connect: An efficient approach to single-query path planning,” in *Proc. IEEE Int. Conf. Robotics and Automation ICRA ’00*, vol. 2, 2000, pp. 995–1001.

[18] D. Berenson, S. S. Srinivasa, D. Ferguson, A. Collet, and J. J. Kuffner, “Manipulation planning with workspace goal regions,” in *ICRA*, 2009.

[19] R. Diankov, “Automated construction of robotics manipulation programs,” Ph.D. dissertation, Robotics Institute, Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA 15213, 10 2010.

[20] T. Yoshikawa, “Manipulability of robotic mechanisms,” *International Journal of Robotics Research*, vol. 4, pp. 3–9, 1985.

[21] P. Felzenszwalb and D. Huttenlocher, “Distance transforms of sampled functions,” Cornell University, Tech. Rep. TR2004-1963, 2004.

[22] J. Nakanishi, R. Cory, M. Mistry, J. Peters, and S. Schaal, “Operational space control: a theoretical and empirical comparison,” *Int. J. Robot. Res.*, vol. 27, no. 6, pp. 737 – 57, 2008.

[23] A. Liegeois, “Automatic supervisory control of the configuration and behavior of multibody mechanisms,” *IEEE Trans. Syst. Man Cybern.*, vol. SMC-7, no. 12, pp. 868 – 71, 1977.

[24] S. Javdani, M. Klingensmith, D. Bagnell, N. Pollard, and S. Srinivasa, “Efficient touch localization with adaptive submodularity,” in preperation.