Planning and Control for Simulated Robotic Sandia Hand for the DARPA Robotic Challenge

by

Cecilia G. Cantu

Submitted to the
Department of Mechanical Engineering
in Partial Fulfillment of the Requirements for the Degree of

Bachelor of Science in Mechanical Engineering

at the

Massachusetts Institute of Technology

June 2013

Signature of Author: ______________________________
Cecilia G. Cantu
Department of Mechanical Engineering
May 17, 2013

Certified by: ______________________________
Russell L. Tedrake
Associate Professor in the Department of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by: ______________________________
Anette Hosoi
Professor of Mechanical Engineering
Undergraduate Officer

Planning and Control for Simulated Robotic Sandia Hand for the DARPA Robotic Challenge

by

Cecilia G. Cantu

Submitted to the Department of Mechanical Engineering
on May 17, 2013 in Partial Fulfillment of the
Requirements for the Degree of

Bachelor of Science in Mechanical Engineering

ABSTRACT

The DARPA Robotic Challenge (DRC) required the development of user interface, perception, and planning and control modules for a robotic humanoid. This paper focuses on the planning and control component for the manipulation qualification task of the virtual section of the DRC. Nonlinear algorithms were employed for the planning systems, such as the grasp optimization system and the robot state trajectory computation system. However, for closed-loop control, a linear proportional-derivative (PD) joint position controller was used. The nonlinear algorithms used for the planning systems may be improved, but their current functionality allows the successful completion of the manipulation qualification task. Also, even though PD controllers seem appropriate for the closed-loop control, PID controllers might yield a higher level of accuracy if tuned properly. In conclusion, a linear controller appears sufficient for certain control of the highly nonlinear ATLAS humanoid robot and Sandia hand as long as accurate optimization and planning systems complement such control.

Thesis Supervisor: Russell L. Tedrake
Title: Associate Professor in the Department of Electrical Engineering and Computer Science

Acknowledgements

First and foremost, I would like to thank Prof. Russell Tedrake for his invaluable support not just this semester but for this past year in which I have worked in his group. He has provided me with opportunities to learn fascinating material and to grow as a student and as a person. Thanks to Mark Pearrow for his help every time I had build errors, bugs in my MATLAB Drake code, for troubleshooting every single computer/software catastrophe I was so apt at encountering, for teaching me how to become more adept with the LINUX terminal, and for becoming such a good friend to me in the process. Thanks to Sisir Karumanchi and Scott Kuindersama for taking me under their wing and being my to-go people for any project troubles I encountered. Thanks to Claudia Perez D'Arpino for being my partner in learning. Thanks to Michael Posa for sitting with me many times over the course of the previous semester and helping me understand how to apply linear complementarity problems to various robotic applications including manipulation and planar walking, and for teaching me some of the structure of Drake.

I would also like to thank my parents, Mario Cantu and Cecilia Maldonado, for always believing in me and for supporting me throughout my MIT career. Thank you to my sister, Andrea Cantu, for being my soul mate, as we share everything including likes, dislikes, and even thoughts. Thanks for understanding me always. Thanks to every single member of the MIT DARPA Robotic Challenge Team, especially Seth Teller, for making it possible for me to work in such a fascinating project and with such amazing and capable people.

Table of Contents

List of Figures

List of Tables

1. Introduction

1.1 DARPA Robotic Challenge

In April 16, 2012, the Defense Advanced Research Projects Agency (DARPA) announced the beginning of a large-scale robotic competition in which different groups, both academic and industrial, would compete in the DARPA Robotic Challenge (DRC). The DRC was introduced as a way to improve or develop control algorithms, user interfaces, and other major component aspects to allow robotic humanoids to be sent to a disaster area as first responders in place of humans to guarantee human protection while performing efficient disaster control. Figure 1-1 represents an artist rendition of the concept of the DRC competition [1], [6]. The DRC proposers emphasized the need for a human-robot interaction that grants a human user the ability to make high-level decisions and communicate them to a humanoid robot via a specific group-developed interface and requires the humanoid robot to execute accordingly without the need for the human user to specify low-level commands.

Figure 1-1: Artist rendition of two humanoid robots competing in the DRC. There is a clear emphasis on the humanoid robots' ability to interact with an environment that would be dangerous for humans [1], [6].

Although there are many tasks that compose the entirety of the DRC, this paper specializes in the aspect of robotic hand manipulation, the task of controlling the robot such that it can form power grasps and optimized grasps on environment objects. Performing appropriate manipulation is essential for the challenge as it allows the humanoid robot to interact with the controls of a terrain vehicle, with power tools that might be needed to overcome obstacles, and with valves for closing or opening a pipeline.[1]

1.1.1 Autonomous Robotic Manipulation (ARM) Program

As a research project agency, DARPA has introduced a variety of research challenges, of which the Autonomous Robotic Manipulation (ARM) program is extremely relevant. The ARM program was directed at developing "software and hardware that enables a robot to autonomously manipulate, grasp, and perform complicated tasks with humans providing only high-level supervision" [5]. The standard human-robot interface for robotic manipulation applications requires the user to employ a high level of control of the robot. This can prove cumbersome and time consuming for the average user, even after training. To stimulate the development of a human-robot interface, which provides the robot a higher degree of autonomy thus simplifying the control tasks of the user, DARPA introduce the ARM program. Thus far, it appears that many research agencies, primarily in the academic field, have been experiencing success in the development of this interface [5].

Deducing from the DRC proposers' hope for the DRC results and from the required task specifications, the DRC also requires the development, if at least partially, of a similar manipulation interface as the one demanded for the ARM program. The ARM-S track, or ARM Software Track, promotes the development of algorithms that allows a robot with robotic manipulators to execute a grasping task without requiring a human in the loop.[2] The chosen tactic by one of the top three ARM-S qualifiers, the team from Carnegie Mellon University (CMU), involves utilizing visual feedback from cameras and 3D sensors along with a mesh model matching software to analyze the environment. This analysis then provides the necessary information for trajectory planning to obtain the "best grasping angles" while efficiently avoiding collisions with obstacles. CMU's ARM software allows for execution of the planned trajectory while allowing re-planning if necessary, such as when there is an unforeseen slip in the grasp. For executing, position and force feedback from the manipulator is used to determine its orientation and grasp status, and a behavior tree complements the optimization and planning systems [2]. Figure 1-2 shows the steps the ARM robot takes to pick up a hang up a telephone as executed using CMU's software [2].

---

[1] These are some of the required tasks for successful completion of the DRC.

[2] The DRC differs from the ARM program in that it focuses on full body control rather than on grasping, however grasping still plays an important role. The DRC also requires a human to be involved in the decision-making process whereas the ARM program eliminates human elements and leaves all the planning and execution decisions to be computed by algorithms.

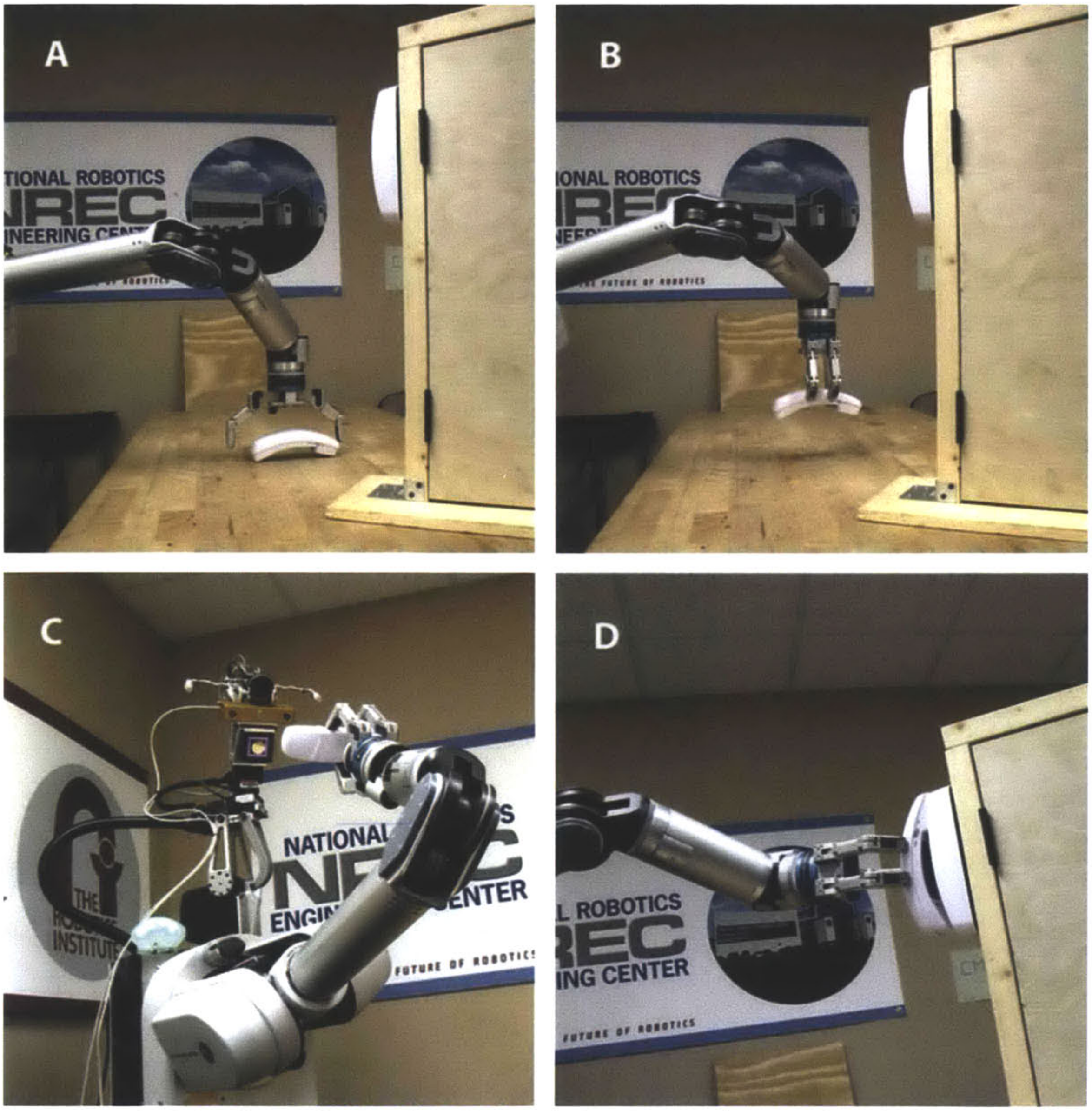


Figure 1-2: (A) After evaluating the object and planning a trajectory, the robot executes the reaching trajectory. (B) The robot executes the planned grasp and picks up the telephone. (C) The robot uses sensors to determine the orientation of the telephone in the grasp. (D) The robot then proceeds to planning and executing the hanging up task [2].

1.1.2 Boston Dynamics' ATLAS Robot and the Robotic Sandia Hand

The DRC competitors were given an option to participate in one of three tracks: Track A competitors are in charge of developing both software and hardware whereas track B and track C competitors focus purely on software development and, if successful, would acquire a "Government Furnished Equipment hardware platform." The chosen hardware platform is the ATLAS humanoid robot developed by Boston Dynamics. Figure 1-3 depicts an image of the PETMAN humanoid robot, a robotic cousin to the ATLAS. The PETMAN and ATLAS humanoid robots differ in that they were developed for different applications, namely PETMAN is used to test military apparel whereas ATLAS focuses of "rough terrain mobility" [4].

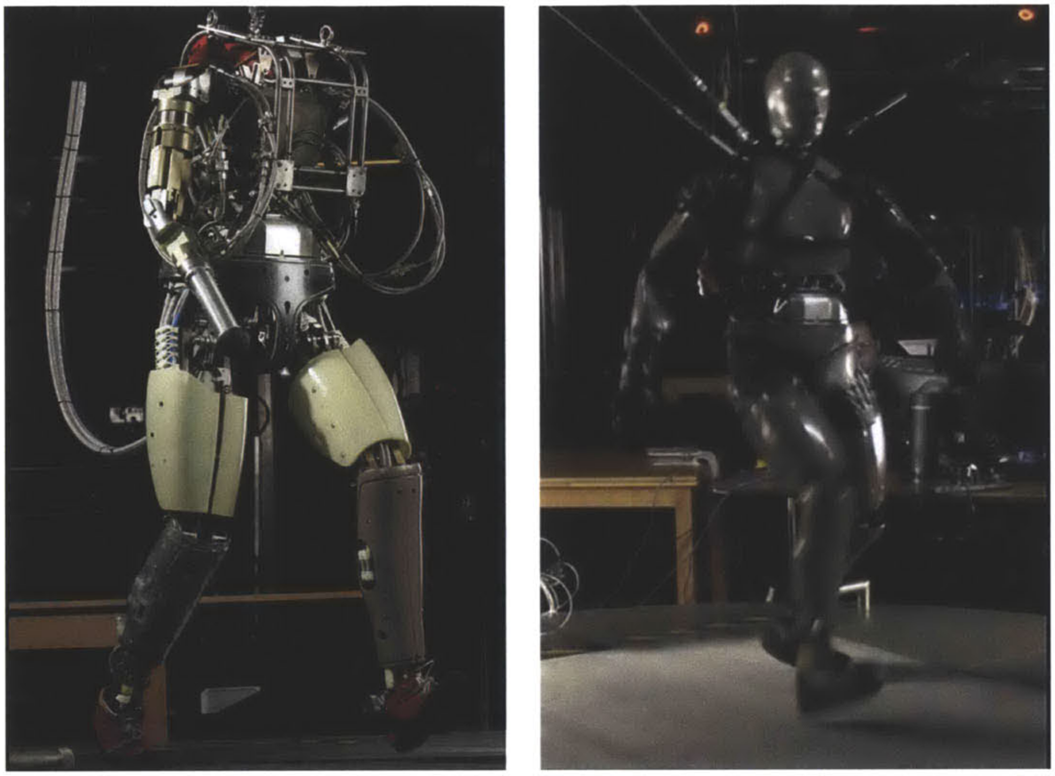


Figure 1-3: Left: Image of the PETMAN humanoid robot developed by Boston Dynamics (BDI) without a head or anthropomorphic upper limbs. Right: Image of the PETMAN humanoid robot during testing of military apparel [4].

The robotic manipulator for the DRC was chosen to be the robotic Sandia hand. This robotic manipulator, developed by the Sandia laboratories, consists of four modular robotic phalanges, one of which is opposable. Figure 1-4 depicts the robotic Sandia hand manipulating a fruit and also with the modular robotic phalanges removed [19]. Each of the robotic phalanges had three degrees of freedom (DOFs) in two joint locations. Figure 1-5 illustrates the DOFs of a single robotic finger. However since the hand has four fingers, it has a total of 12 DOFs.

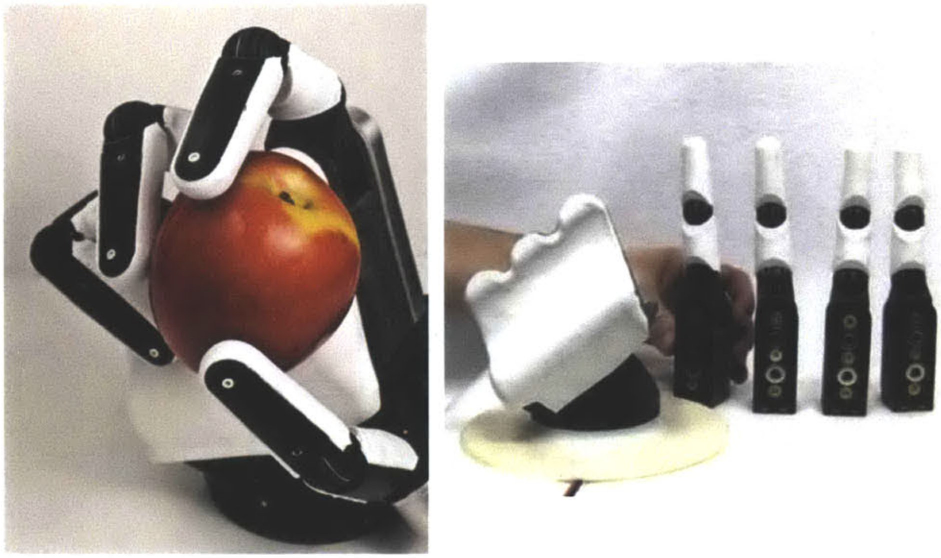


Figure 1-4: Left: Robotic Sandia hand manipulating an apple. Right: Sandia hand with the modular robotic phalanges removed [19].

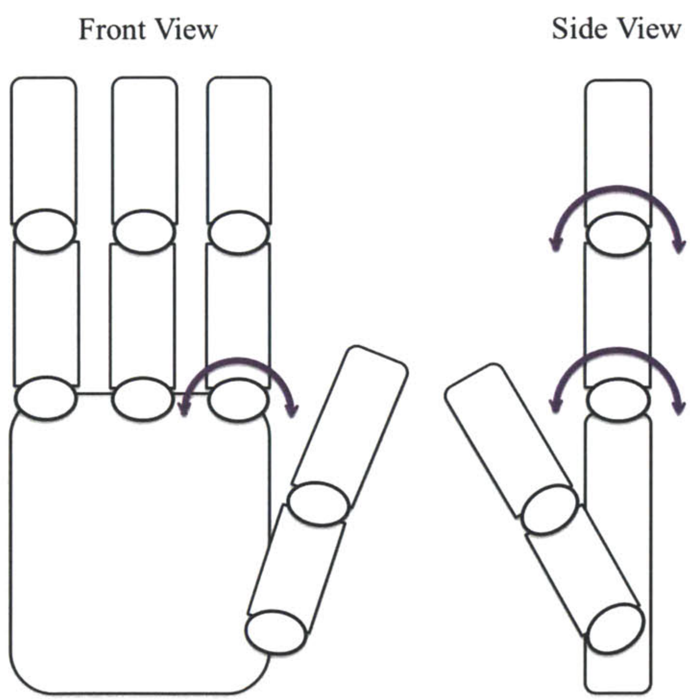


Figure 1-5: Abstract representation of the front and side views of the robotic Sandia hand illustrating the three DOFs of each robotic finger. The Sandia hand has a total of 12 DOFs.

2. Background

2.1 Lightweight Communications Marshalling (LCM)

The MIT DARPA Urban Challenge team[3] developed a communications system that was employed in the DRC. As software development overemphasizes modularity, this system, known as LCM (Lightweight Communications and Marshalling), serves as the primary means of communication between the multitudes of systems required for end-to-end operation. LCM is characterized by low-latency, single-message, subscribe-publish communication using UDP Multicast [14]. LCM messages are carried in LCM channels, the actual objects that are subscribed to and published. LCM requires that the LCM message structure, or type, be specified *a priori*; both the publisher and the subscriber to a specific message must know its structure to decode and encode the information being exchanged. The structure specifications of LCM messages, or LCM message types, come from a simple description written in C language, which includes the name of the message type and the contents of the message. Figure 2-1 provides two examples of LCM message types [14]. As seen in Figure 2-1, an LCM message type can be used as an attribute for another LCM message type [14]. This increases efficiency as it avoids the sole use of primitives in every LCM message type. Furthermore, the LCM system allows for multiple subscribers to listen to a single message. Figure 2-2 provides an abstract representation of LCM in which there are multiple messages, channels, subscribers, and publishers.

```
struct waypoint_t {
string id;
float position[2];
}

struct path_t {
int64_t utime;
int32_t num_waypoints;
waypoint_t waypoints[num_waypoints];
}
```

Figure 2-1: Example of two LCM message types. The first LCM type contains primitives as attributes. The second LCM type contains both primitives as well as another LCM type [14].

---

[3] The DARPA Urban Challenge was the previous challenge proposed by DARPA, which culminated in a competition in 2007. This challenge consisted of developing an " autonomous ground vehicle [capable of] maneuvering in a mock city environment, executing simulated military supply missions while merging into moving traffic, navigating traffic circles, negotiating busy intersections, and avoiding obstacles" [7].

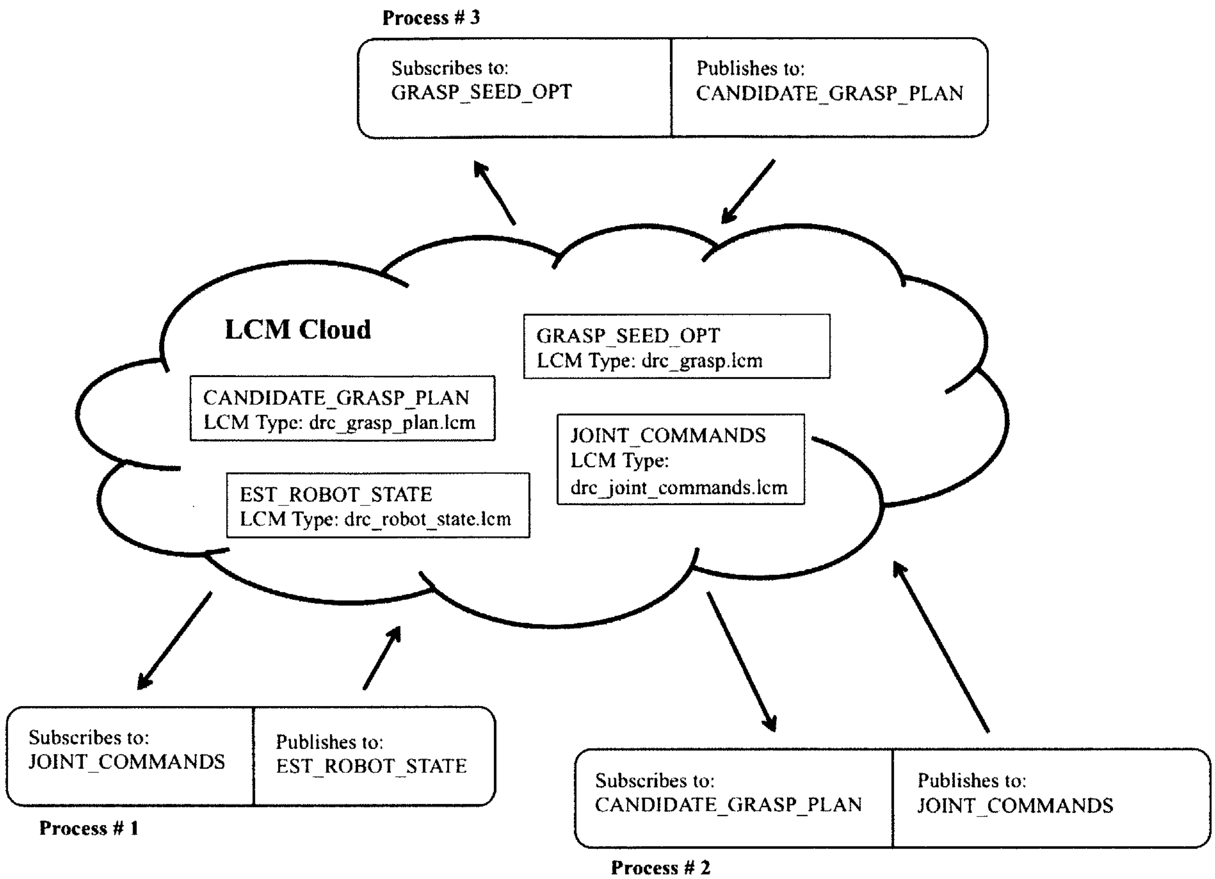


Figure 2-2: Abstract representation of the functioning of the LCM system. This particular example illustrates the interaction between three different processes. Note that each process can publish and subscribe to more than one channel, however each channel only has one LCM type.

2.2 Drake

The MIT DRC team uses Drake,[4] a MATLAB toolbox, for control design, stability analysis, trajectory planning, and optimization. Drake inherently functions by simulating dynamical systems, such as plants and controllers, using a Simulink engine [20]. These systems, or blocks, can be arranged in many well-known configurations including feedback and cascade. Furthermore, Drake provides tools that use the block system structure for analysis and controller design [20]. Drake operates as a hierarchy of MATLAB classes that employ an input-output structure [20].

Because the DRC requires for messages between systems to be structure specific, the inputs, states, and outputs of Drake systems need to be user-specified. Thus a MATLAB class, `CoordinateFrame`, was developed to represent structures in Drake. Each state, input, and output is specified through its `CoordinateFrame` instance. However, for Drake to

[4] Drake was developed by the Robot Locomotion Group of the MIT Computer Science and Artificial Intelligence Laboratory with Prof. Russell Tedrake as the major contributor.

communicate with non-MATLAB systems, special sub-classes of the `CoordinateFrame` were also developed, namely the `LCMCoordinateFrame` and the `LCMCoordinateFrameWCoder`. The `LCMCoordinateFrame` allows the information communicated through the coordinate frame structure to be published, or read, as an LCM message. The `LCMCoordinateFrameWCoder` is more specific in that it utilizes a "coder" written in Java language to create the structure of the coordinate frame and sets the method for encoding and decoding information. Generally, because the state of a Drake system is internal, it's uncommon for it to be associated with an `LCMCoordinateFrame` or an `LCMCoordinateFrameWCoder`. However, inputs and outputs of systems should be LCM capable for modularity and improved efficiency.

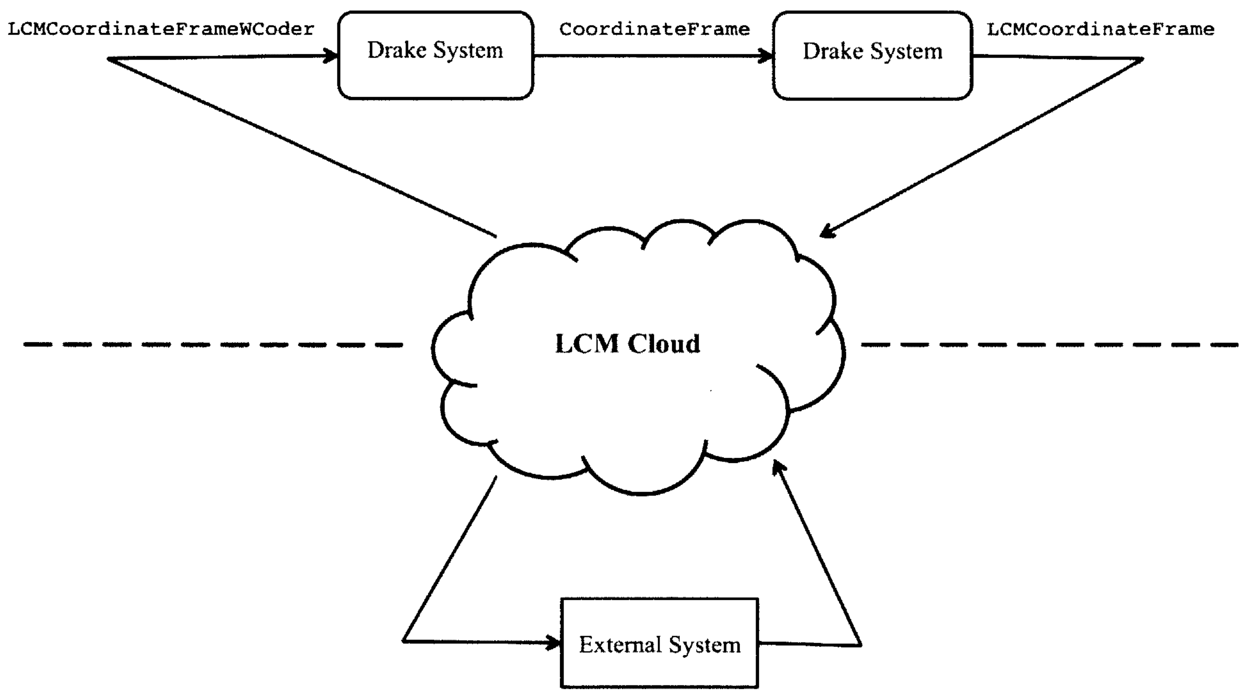


Figure 2-3: Block diagram representation of the interaction between Drake systems with the LCM cloud and with each other. If the output of a Drake system is connected to the input of another, the input-output pair doesn't need to be LCM capable as long as it is running in the same instance of MATLAB. However, Drake systems can communicate with each other via the LCM cloud as well, especially if they are running in different instances of MATLAB or in different computers. The external system in the figure could or could not be another Drake system.

2.3 Proportional-Integral-Derivative (PID) and Proportional-Derivative (PD) Controllers

Proportional-Integral-Derivative (PID) controllers are controllers that utilize proportional, integral, and derivative control. PID controllers are linear, allowing analysis via Laplace transforms. These controllers are used to both decrease, or eliminate, the steady state error, and manipulate the settling time of a system. The transfer function of a PID controller is

$$C_{PID} = K_p + K_d s + \frac{K_i}{s} = \frac{K_d s^2 + K_p s + K_i}{s} \quad (1)$$

If a PID controller is employed to control a linear plant, the entire feedback system can be analyzed mathematically via Laplace transforms because it would also be linear. Figure 2-4 provides an example of a block diagram of a PID controller in a feedback loop controlling a linear plant.

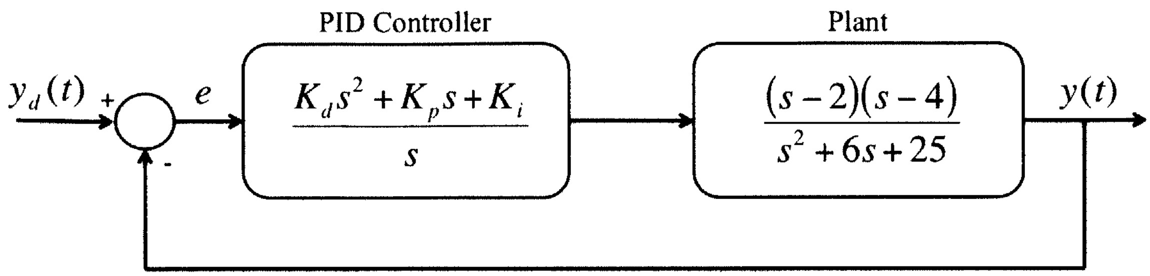


Figure 2-4: Example block diagram of a PID controller used in a negative feedback loop configuration to control a plant represented by a Laplace transform. $y_d(t)$ represents the desired trajectory, $y(t)$ represents the actual trajectory of the system, and e represents the error, or the difference between $y_d(t)$ and $y(t)$.

Proportional-Derivative (PD) controllers are similar to PID controllers, with the difference lying in that PD controllers lack an integral part in their transfer function. The transfer function of a PD control is

$$C_{PD} = K_p + K_d s. \quad (2)$$

This lack of an integral component prevents the PD controller from being able to control steady state error, which is why PID controllers are more widely used as they grant a higher degree of control.

2.4 Gazebo Simulation Environment

The DRC proposers chose Gazebo as the simulation environment for the virtual component of the challenge. Gazebo is open source software with the capability to simulate multiple robots, sensors, and objects in a three-dimensional environment. Currently, Bullet, also open source software, is the physics engine employed by Gazebo, allowing for simulation of rolling friction, collisions, and rigid body dynamics [9]. Figure 2-5 depicts a Gazebo simulation environment with a humanoid robot and several objects. To spawn different robots into the Gazebo simulation environment, they must be described using the Universal Robot Description Format (URDF). URDF files, or URDFs, are written in XML format and contain the description of the robot, including the moment of inertias, mass, and visual and collision shapes of each of the robot's link, and specify the types of joints that connect these robotic links. Refer to Appendix B for a more information regarding URDFs.

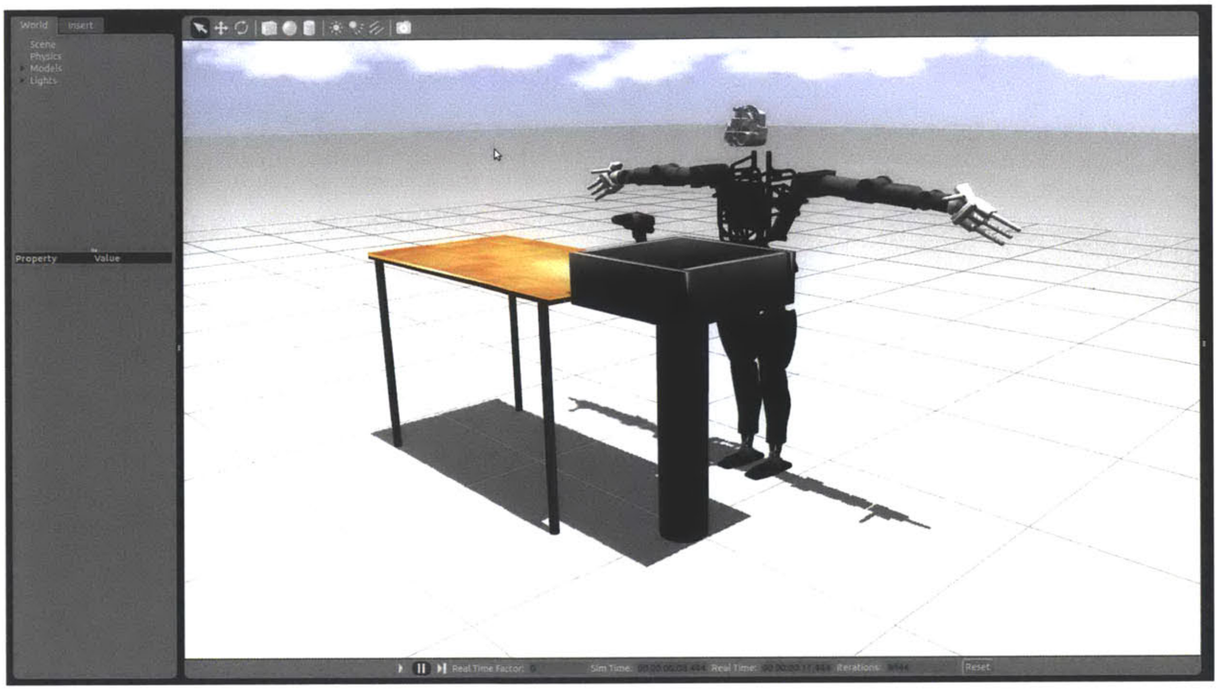


Figure 2-5: Visual of the ATLAS humanoid robot and several objects for manipulation spawned in Gazebo. Several important features and interface parameters of Gazebo include the real time factor, simulation (sim) time, and real time, play and pause physics buttons located at the bottom of the screen. The sidebar informs the user about the spawned models and the lighting.

2.5 Grasping

Grasping plays an important role in robotics due to the emphasis on the interaction between robots and their environments; robots are increasingly being seen as tools that allow interaction with certain objects that would prove either cumbersome or dangerous for humans. Grasps are determined by optimizing several different grasp aspects, including grasp wrench spaces, contact detection and determination, and friction cone approximations. Generally, grasp planning and execution involves the use of mathematical models of the robotic manipulator, including velocity kinematics, system dynamics, contact modeling, restraint analysis, etc. [10]. However, for real-life robotic applications, these tools must be combined with perception modules so that the robot is aware of the objects in its environment, whether for manipulation or to avoid collisions.

3. Simulation Setup

This paper focuses on the DRC qualification task for manipulation in which the simulated humanoid robot is pinned to the world at the hip and is allowed to move the upper body to complete the task. This simplifies manipulation since the robot doesn't need a balancing controller to execute the necessary commands. In summary, the MIT DRC team chose to execute the task in steps as follows: The user selects an object of interest through the user interface, which is visualized in a viewer from perception information obtained from simulated sensor data. The user specifies a direction normal to the object for grasp optimization. The user-specified normal direction and object information are sent to a Drake grasp optimizer. The Drake grasp optimizer outputs an optimized grasp, which is visually represented in the viewer. The user then has the option to specify a command to be sent to a reaching planner. These options include reaching a pre-grasp configuration, meaning that the end effector of the robot will be sent to a location proximal to the optimized grasp, or executing a palm touch, meaning that the end effector will be send to the optimized grasp but without giving any command to the robotic fingers. Once the user has specified the desired command, a reaching planner that uses an inverse kinematic algorithm computes a joint trajectory for each of the joints in the upper body of the humanoid robot. This joint trajectory is then visualized in the viewer for user approval. If the user is content with the plan, he approves it for execution in the simulation environment.

3.1 Systems for the Qualification Task for Manipulation

End-to-end operation for the qualification task for manipulation requires the use of several systems, including optimization systems, planners, and user interfaces. This section will present all of these different components in detail.

3.1.1 MIT DARPA Grand Challenge (DGC) Viewer

The MIT team that participated in the DGC developed a user interface for data visualization and logging. This interface was adapted for use in the DRC and is referred to as the MIT DRC Viewer, or simply Viewer. The Viewer developers focused on creating a modular platform to allow, and even foster, parallel development. As one of the Viewer's primarily uses is data visualization, individual Viewer users develop data renderers according to their specific tasks and needs. For example, the MIT DRC footstep-planning sub-team wrote code to allow for visualization of footstep plans in the Viewer.

The qualification task for manipulation for the virtual competition of the DRC, or Virtual Robotic Challenge (VRC), requires visualization of the state of the robotic humanoid, optimized grasps using the robotic Sandia hand, and state of manipulation and environmental objects for interaction planning and collision detection. Figure 3-1 represents the visualization of some of this information. Subsequent figures in section 3 will provide visualization for the rest of the information.

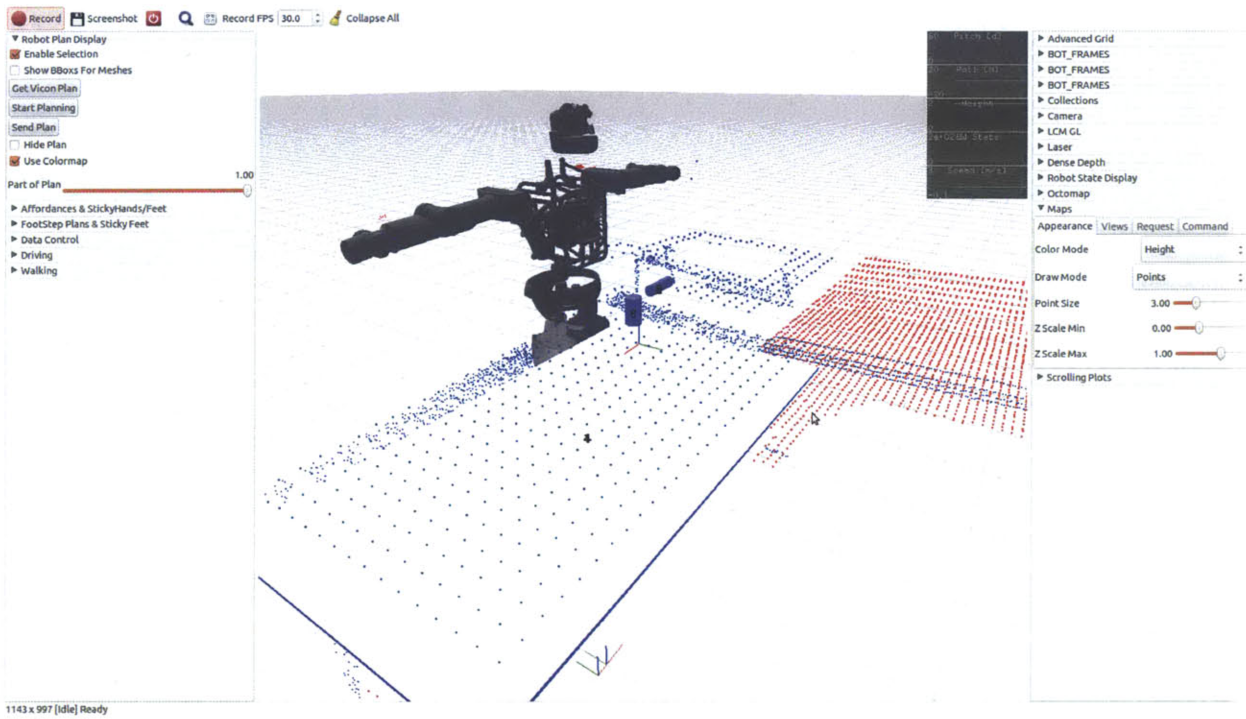


Figure 3-1: MIT Viewer rendering of the state of the humanoid robot and other relevant objects. The interactive objects are drawn in blue, and other relevant environmental objects can be seen either as lidar sensor data or as, in this case, white objects.

It is important to note that for the VRC, given that the Gazebo simulation environment is meant to mimic a real-life situation in which the humanoid robot is in a disaster zone, the user is not aware of the state of the robot as visualized in this simulation environment, and is only aware of the state of the robot as visualized in the Viewer. Thus for the actual VRC, the user will only have access to information portrayed in the Viewer. However for the qualification task, as currently progressed, it is possible, and often necessary, for the user to see the Gazebo simulation environment along with the Viewer.

Since the Viewer also serves as user interface, it is where plans are committed or rejected. The Viewer may thus serves as a pass-through system in which the user controls which information passes through and which doesn't.

3.1.2 Grasping-Specific Drake Systems

Drake systems were developed to perform optimization for grasping, for planning trajectories based on the output of the grasp optimizer and the desired action specified by the user through the Viewer, and for executing the plans. Overall, there are five main Drake systems that are used for the manipulation qualification task: Grasp Optimizer, Reaching Planner, Grasp Controller, Reach Plan Follower, and Qualification 2 State Machine. The Grasp Optimizer and Reaching Planner are planning systems while the Grasp Controller and the Reach Plan Follower are executing systems. The Qualification 2 State Machine determines the state of the robot and the controller that will be used. For the manipulation task, the robot is always pinned, thus a controller for a harnessed robot is used.

3.1.2.1 Grasp Optimizer (GO)

For grasp optimization, the user must select an interactive object from the Viewer and specify an approach direction, "defining the direction of approach movements toward an object" [22]. The pose and contact information about the object and the approach direction are sent via LCM to the Grasp Optimizer. The Grasp Optimizer computes a grasp by maximizing the number of contact points between the robotic Sandia hand and the object of interest using this information. A MATLAB built-in function from the Optimization toolbox, `fmincon`, which finds the value of x which minimizes some function $f(x)$, subject to the constraints

$$c(x) \leq 0 \ , \tag{3}$$

$$c_{eq}(x) = 0, \tag{4}$$

$$Ax \leq b \ , \tag{5}$$

$$A_{eq}x = b_{eq}, \tag{6}$$

$$lb \leq x \leq ub, \tag{7}$$

where $f(x)$, $c(x)$, and $c_{eq}(x)$ can all be nonlinear functions, is used for this purpose [8]. The function $c(x)$, represents the distance between contact points in the robotic hand and surface contact points of the object, which are modeled for cylinders, rectangular prisms, or tori depending of the best fit to perception data.[5] Currently, only the constraint represented by equation (3) is being used for maximizing the number of contact points between the robotic hand and the object.

3.1.2.2 Reaching Planner (RP)

Once the grasp optimizer calculates a grasp, the user has the option to specify in the Viewer an action for the robot to perform. Currently, these options include touching the object or moving the end effector, or robotic hand, close to the object, or in actuality close to the pose of the optimized grasp. The Viewer then, depending on the user-specified command, outputs the desired pose for the end effector, which is received by the Reaching Planner. The planner then uses spline interpolation to compute waypoints between the current estimated state of the robot and a desired state determined by inverse kinematics (IK). These waypoints are then fed into an IK solver, which computes the state trajectory of the robot. [6]

---

[5] Object fitting is an essential component of the MIT DRC software, however, as it is the responsibility of the MIT DRC perception sub-team it is beyond the scope of this paper; functionality of the necessary perception systems to execute grasping are assumed.
[6] The IK Solver used by the Reaching Planner is a C++ adaptation of an IK code from Drake. Implementation in C++ allows the solver to execute faster.

To allow the user freedom to alter the trajectory, the plan computed by the Reaching Planner is sent back to the Viewer for visualization. Figure 3-2 shows the Viewer visualization of a reaching plan. If the user is satisfied with the plan, the plan can be committed for execution in the simulation environment. Otherwise, the user has the option of selecting two intermediary waypoints and adjusting the desired pose of the robot at those points. Once this has been done, the new waypoints are sent to the Reaching Planner for spline interpolation and trajectory computation to reoccur. The user also has the option to completely reject the plan.

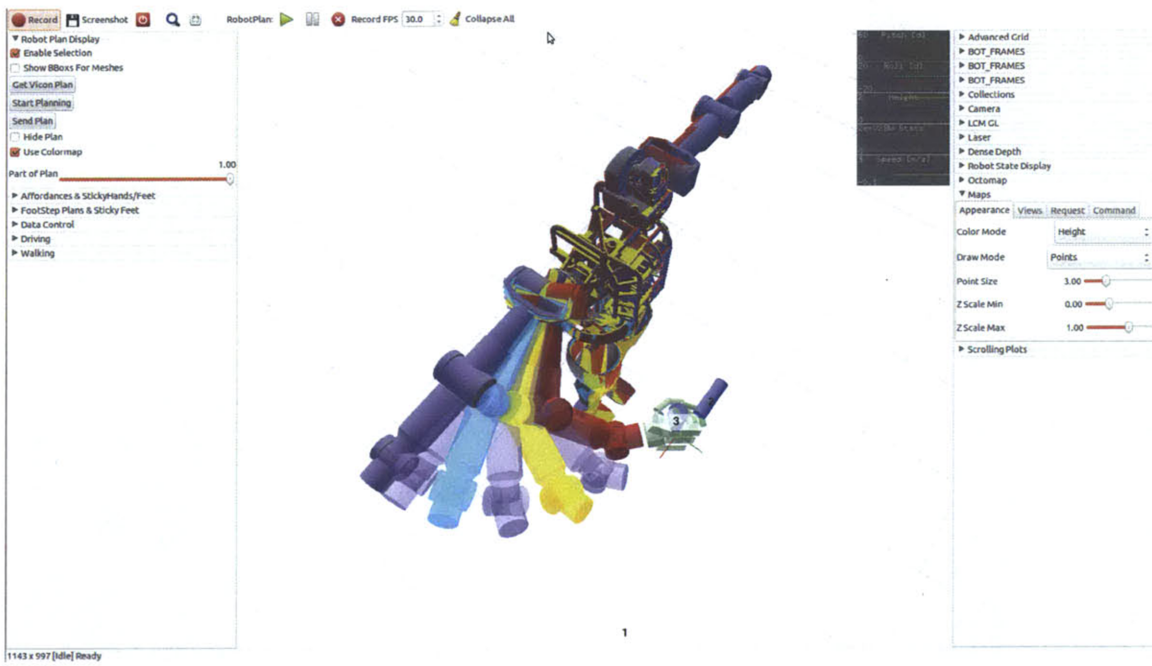


Figure 3-2: The reach plan computed by the reaching planner is shown in the Viewer using a color map. Two of the waypoints computed by the Reaching Planner are visualized in light blue and yellow. These waypoints can be reconfigured by the user for re-planning if the user is dissatisfied with the current plan. Alternatively, the user may completely reject the plan and re-plan.

3.1.2.3 Grasp Controller (GC) and Reach Plan Follower (RPF)

The Grasp Controller and the Reach Plan Follower are simple systems that convert plans to commands sent to the robot in the Gazebo simulation environment. Both the Grasp Controller (GC) and the Reach Plan Follower (RPF) trigger upon receiving a message from the Viewer indicating that the user has committed a candidate plan or grasp. The GC and RPF systems also have the option between publishing the gains used by the PID controller in Gazebo. Currently, the controller gains are non-dynamic, however, the code base allows the controller gain values to change during execution.

3.2 LCM System Connections and Debugging

All of the individual systems that have been described in the previous sections share information with an LCM cloud, thus allowing each system to interact with others. For grasping, the Viewer interacts with all of the Drake systems. The Grasp Optimizer and Reaching Planner require the Viewer to send initialization signals, and the Grasp Controller and Reach Plan Follower depend on the candidate plans' conversion to committed plans through the Viewer. Figure 3-3 represents a block diagram abstraction containing the names of the LCM channels to which each system subscribes and publishes. Figure 3-4 shows the connections between the individual Drake systems the locations in which Viewer is used to transform candidate plans to committed plans to be executed in the simulation environment.

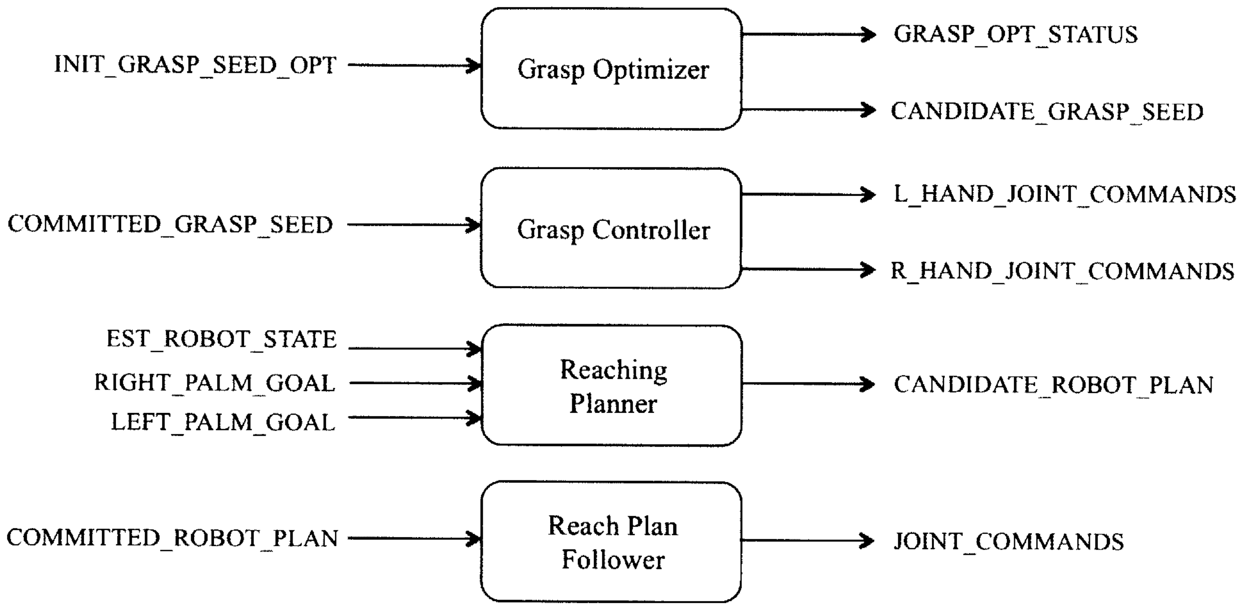


Figure 3-3: LCM channel inputs and outputs of the Drake grasping systems.

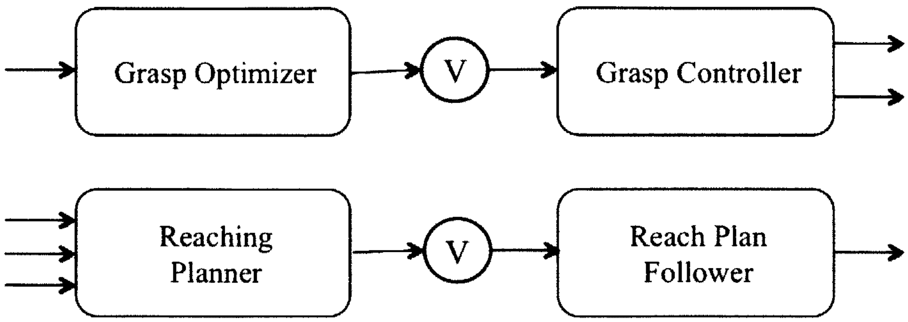
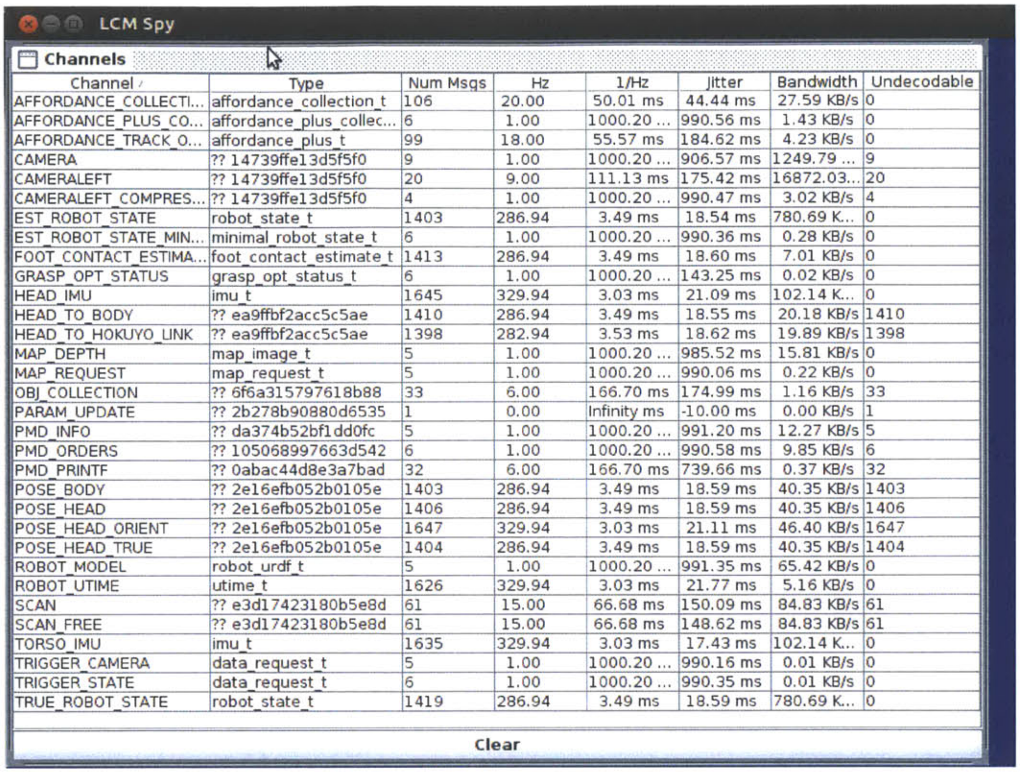


Figure 3-4: The Grasp Optimizer and the Reaching Planner output candidate plans and are communicated to the user via the Viewer, which is represented by a V in this diagram. The user then commits the plans, which are intersected by the Grasp Controller and the Reach Plan Follower and sent as position commands to be executed in the Gazebo simulation environment.

Because the majority of inputs and outputs to all the systems required for the manipulation qualification task reside in the LCM Cloud, a tool "useful for logging, replaying, and inspecting traffic," known as the LCM-spy was used to determine whether systems were publishing as expected [14]. If a particular system weren't publishing to its LCM channel, the developer would interpret it as an error in that particular system. This tool, along with the modular design of the end-to-end system allowed bugs to be located and solved in a more efficient manner. Figure 3-5 shows an image of the LCM-spy being used to see which channels were being published during end-to-end operation of the manipulation task.



LCM Spy

Channels

Channel	Type	Num Msgs	Hz	1/Hz	Jitter	Bandwidth	Undecodable
AFFORDANCE_COLLECTI...	affordance_collection_t	106	20.00	50.01 ms	44.44 ms	27.59 KB/s	0
AFFORDANCE_PLUS_CO...	affordance_plus_collec...	6	1.00	1000.20 ...	990.56 ms	1.43 KB/s	0
AFFORDANCE_TRACK_O...	affordance_plus_t	99	18.00	55.57 ms	184.62 ms	4.23 KB/s	0
CAMERA	?? 14739ffe13d5f5f0	9	1.00	1000.20 ...	906.57 ms	1249.79 ...	9
CAMERALEFT	?? 14739ffe13d5f5f0	20	9.00	111.13 ms	175.42 ms	16872.03...	20
CAMERALEFT_COMPRES...	?? 14739ffe13d5f5f0	4	1.00	1000.20 ...	990.47 ms	3.02 KB/s	4
EST_ROBOT_STATE	robot_state_t	1403	286.94	3.49 ms	18.54 ms	780.69 K...	0
EST_ROBOT_STATE_MIN...	minimal_robot_state_t	6	1.00	1000.20 ...	990.36 ms	0.28 KB/s	0
FOOT_CONTACT_ESTIMA...	foot_contact_estimate_t	1413	286.94	3.49 ms	18.60 ms	7.01 KB/s	0
GRASP_OPT_STATUS	grasp_opt_status_t	6	1.00	1000.20 ...	143.25 ms	0.02 KB/s	0
HEAD_IMU	imu_t	1645	329.94	3.03 ms	21.09 ms	102.14 K...	0
HEAD_TO_BODY	?? ea9ffbf2acc5c5ae	1410	286.94	3.49 ms	18.55 ms	20.18 KB/s	1410
HEAD_TO_HOKUYO_LINK	?? ea9ffbf2acc5c5ae	1398	282.94	3.53 ms	18.62 ms	19.89 KB/s	1398
MAP_DEPTH	map_image_t	5	1.00	1000.20 ...	985.52 ms	15.81 KB/s	0
MAP_REQUEST	map_request_t	5	1.00	1000.20 ...	990.06 ms	0.22 KB/s	0
OBJ_COLLECTION	?? 6f6a315797618b88	33	6.00	166.70 ms	174.99 ms	1.16 KB/s	33
PARAM_UPDATE	?? 2b278b90880d6535	1	0.00	Infinity ms	-10.00 ms	0.00 KB/s	1
PMD_INFO	?? da374b52bf1dd0fc	5	1.00	1000.20 ...	991.20 ms	12.27 KB/s	5
PMD_ORDERS	?? 105068997663d542	6	1.00	1000.20 ...	990.58 ms	9.85 KB/s	6
PMD_PRINTF	?? 0abac44d8e3a7bad	32	6.00	166.70 ms	739.66 ms	0.37 KB/s	32
POSE_BODY	?? 2e16efb052b0105e	1403	286.94	3.49 ms	18.59 ms	40.35 KB/s	1403
POSE_HEAD	?? 2e16efb052b0105e	1406	286.94	3.49 ms	18.59 ms	40.35 KB/s	1406
POSE_HEAD_ORIENT	?? 2e16efb052b0105e	1647	329.94	3.03 ms	21.11 ms	46.40 KB/s	1647
POSE_HEAD_TRUE	?? 2e16efb052b0105e	1404	286.94	3.49 ms	18.59 ms	40.35 KB/s	1404
ROBOT_MODEL	robot_urdf_t	5	1.00	1000.20 ...	991.35 ms	65.42 KB/s	0
ROBOT_UTIME	utime_t	1626	329.94	3.03 ms	21.77 ms	5.16 KB/s	0
SCAN	?? e3d17423180b5e8d	61	15.00	66.68 ms	150.09 ms	84.83 KB/s	61
SCAN_FREE	?? e3d17423180b5e8d	61	15.00	66.68 ms	148.62 ms	84.83 KB/s	61
TORSO_IMU	imu_t	1635	329.94	3.03 ms	17.43 ms	102.14 K...	0
TRIGGER_CAMERA	data_request_t	5	1.00	1000.20 ...	990.16 ms	0.01 KB/s	0
TRIGGER_STATE	data_request_t	6	1.00	1000.20 ...	990.35 ms	0.01 KB/s	0
TRUE_ROBOT_STATE	robot_state_t	1419	286.94	3.49 ms	18.59 ms	780.69 K...	0

Clear

Figure 3-5: LCM-spy showing the traffic. LCM-spy was designed to show the channels, the LCM type associated with its channel, the number of messaged published from the initiation of the spy, the frequency at which messages are being published, and some other information.

3.3 Gain Tuning

Even though PD and PID controllers are linear controllers, for this particular case, they are being used to control a nonlinear plant as the presence of revolute joints produces nonlinearities. Although standard mathematical tools for control analysis for linear controller-plant combinations may be used in general, the behavior of this system, which has a linear controller and nonlinear plant, was tuned empirically for better performance due to the complexity of the nonlinear plant. A gain-tuning script was utilized for this purpose.[7]

This gain-tuning script sends joint position commands via LCM to the robot in a Gazebo simulation environment. The script then reads the state of the robot, which is published by Gazebo. The designer along with knowledge about joint referencing determined by a plug-in provided by the DRC developers provides information, which is used to develop a simple trajectory for a specific joint to follow. This trajectory may be a series of step functions or may more closely assimilate a sinusoid wave depending on parameters established by the designer. The script then takes its position command trajectory and plots it against the joint position as published by Gazebo. The designer could then compare the two signals and determine how to

---

[7] This Drake MATLAB script that was written by Scott Kuindersama.

change the controller gains so as to match the signals as closely as possible. Figure 3-6 shows an example plot created by the script when attempting to tune the gains for one of the elbow joints of the robot.

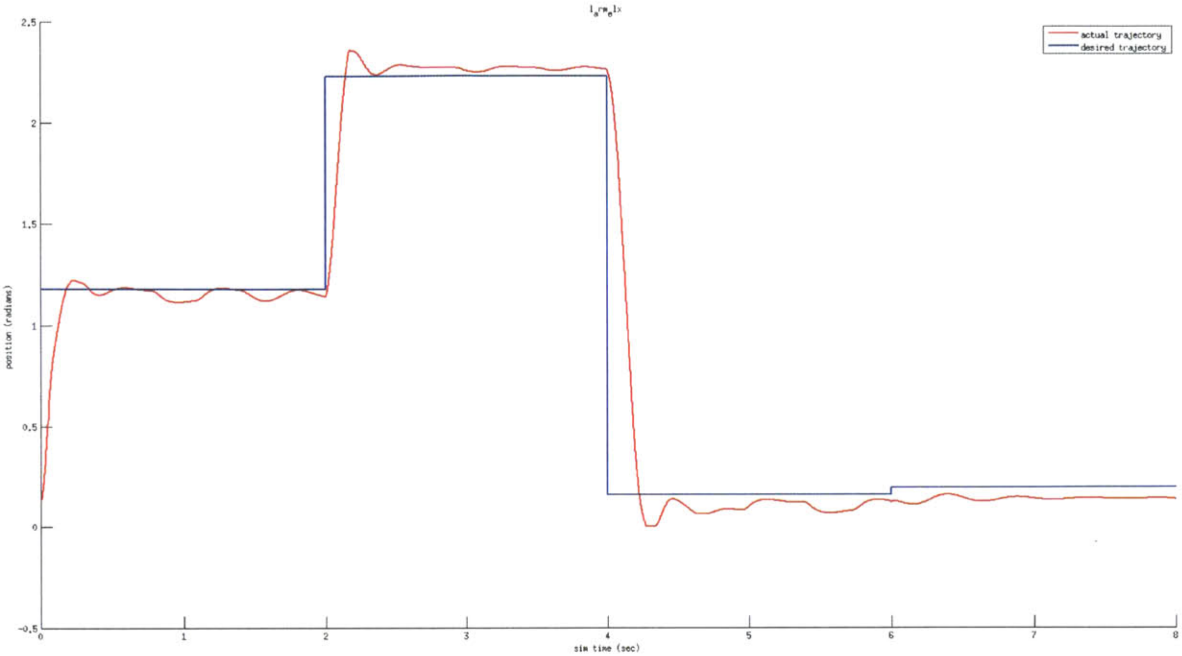


Figure 3-6: MATLAB plot created by the gain-tuning script in which the blue signal represents the joint position trajectory sent to Gazebo and the red signal represents the position of the joint as output by Gazebo. The gains should be tuned such that the two signals match as closely as possible.

After interpreting plots like the one in Figure 3-6 and making changes to the gains, the process was repeated until the designer was satisfied with the plot. Then the designer would re-execute the entire process for each relevant joint (for manipulation, gain tuning for the leg joints of the robot was unnecessary as the robot was pinned from the hip). Table 3-1 contains the values of the tuned proportional and derivative gains for the robot and the fingers of the Sandia hand.

Table 3-1: Tuned gains for the manipulation qualification task using the gain-tuning script described above.

Joint Name	K_p Gain	K_d Gain
Upper Shoulder Y	400	70
Shoulder X	2000	70
Elbow Y	200	5
Elbow X	400	15
Upper Wrist Y	15	6
Middle Wrist X	300	15
Upper Hip Z	100	30
Middle Hip X	250	50
Lower Hip Y	500	25
Knee Y	120	5
Upper Ankle Y	15	2
Lower Ankle X	15	2
Neck Y	100	5
Lower Back Z	5000	45
Middle Back Y	3000	45
Upper Back X	6000	45
Finger Joint 0	150	0.75
Finger Joint 1	100	0.45
Finger Joint 2	100	0.30

3.4 Grasping for the Qualification Manipulation Task

Since the GO system obtained specific grasps to execute, position control was used to command the Sandia hand to attain the desired position and orientation. As such, the user, after planning and executing a full body trajectory for the robot to reach a pre-grasp or palm touch configuration, would then command the Sandia hand to close into the desired grasp as calculated by the GO system. However, since environmental objects were simplified into more basic geometries for grasp optimization, the optimized grasp from the GO wasn't completely accurate with respect to constraint points. Figure 3-7 represents an object and the simple shape that was fitted to it by the perception systems. Because of this simplification, the optimized grasps, though appropriate, contained intrinsic errors that originated from the slightly inaccurate determination of contact points. Figure 3-8 represents the actual vs. the desired position of one joint, of the twelve, in the robotic Sandia hand. It shows how the joint never reaches the desired position because of an early, unexpected collision. Such plots for other joints depict similar information but shall not be shown without loss of generality.

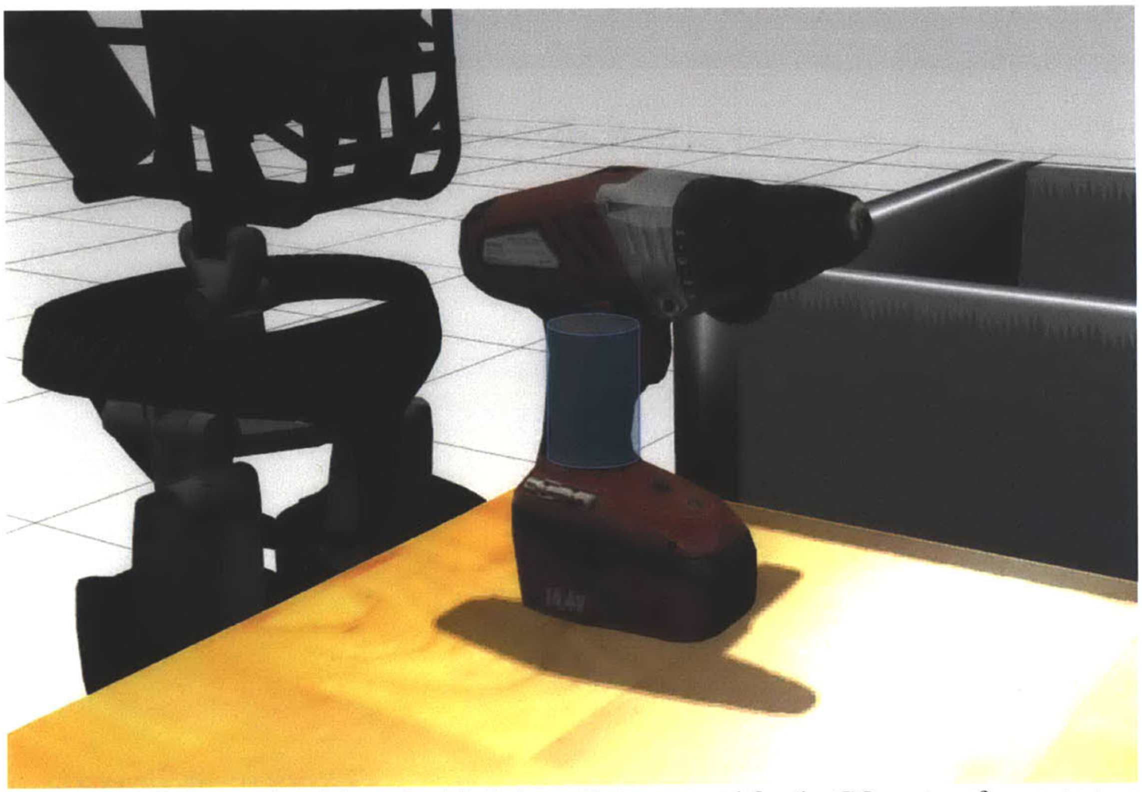


Figure 3-7: Example of the cylindrical shape that was used for the GO system for contact determination. As shown, the simplified shape is, though appropriate, not completely representative of the actual environmental object. However, this doesn't cause any problems for grasp execution and is even preferred for simplifying the grasp optimization process, which would otherwise be very computationally expensive.

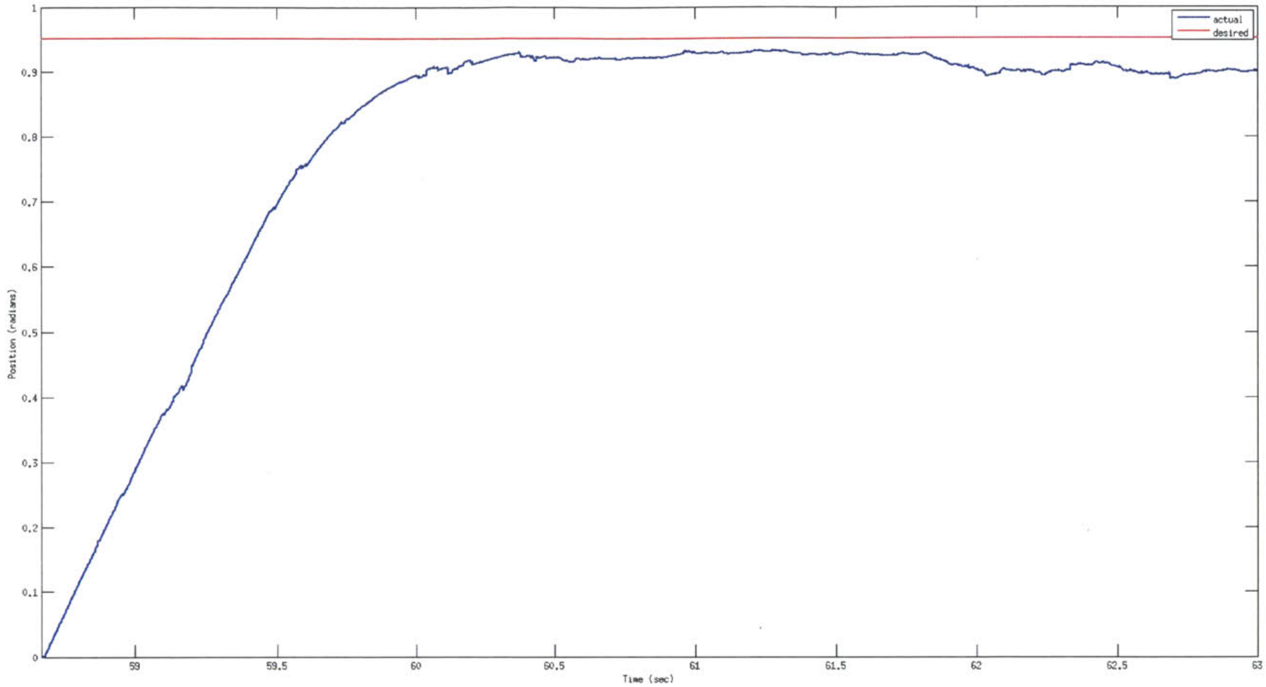


Figure 3-7: Actual vs. desired trajectory (as computed by the GO system) for a specific joint in the robotic Sandia hand during grasp execution.

4. Summary and Future Work

For end-to-end operation, first a grasp around an object, which could be modeled as a shape primitive such as a cylinder, rectangular prism, or torus, would be computed. Then, the state trajectory of the robot between the current and desired states was determined using spline interpolation and inverse kinematics. The robot was then given joint position commands to execute the plan in sequences. Because the human user is such an important component of the control loop, modular computation and execution proved convenient as it facilitated human interference in case there were any errors with the optimizers and planners. Even when errors were detected, the human user didn't necessarily have to adapt to switching between higher-level to lower-level control; the user would simply re-establish the optimization parameters to be re-sent to the optimizer/planner systems.

The gains were optimized with no restrictions to the degrees of freedom of the robotic fingers, however these gains may not be appropriate for situations in which the fingers must collide with objects, presenting a slight drawback for the use of linear position control. A suggestion for future work would be to introduce dynamic gains calculated using feedback from simulated force sensors in the robotic fingers. Currently, the GO maximizes the number of contact points between the Sandia hand and the interactive object. For future work, it might be possible to allow the GO to also consider force feedback and perhaps include optimization for the grasp wrench space describe in the background section of this paper. Other proposed suggestions include allowing the user to modify the orientation of the optimized grasp and require the GO to re-optimize the grasp given the orientation constraints defined by the human user.

5. Conclusions

Even though liner controllers can be appropriate for nonlinear robots such as Boston Dynamic's ATLAS and the Sandia hand developed by the Sandia Laboratories, the standard mathematical tools available for control system development for linear plants might not applicable or efficient. Instead, controller optimization relies on designer judgment and empirical gain tuning. For robots to be effective in real world situations, complex, and sometimes nonlinear, optimization techniques must be employed for trajectory planning. Nevertheless, human intuition and input are useful for detecting errors in optimization systems.

6. Appendices

6.1 Appendix A: Drake Toolbox

The Drake MATLAB toolbox developed by the Robot Locomotion Group of the MIT Computer Science and Artificial Intelligence Laboratory has applications in modeling and simulation, planning, stability analysis, controller design, system identification, and state estimation [20]. Since this toolbox has very extensive applications, this appendix will focus on detailing Drake's modeling and simulation capabilities, which will present enough information to the reader to understand Drake usability and procedures.

Drake systems, which are dynamical systems, are represented in code using the standard state-space model. These dynamical systems have an internal state, represented as a MATLAB array, and can have inputs and outputs. Drake has the capability to allow MIMO (multiple-input multiple-output) systems to be created, and to make them LCM capable. Drake is built as a class hierarchy in MATLAB and supports both discrete, continuous and hybrid systems. Figure A-1 shows a Drake class that implements the nonlinear continuous system described by [20]

$$\dot{x} = -x + x^3 \tag{8}$$

$$y = x. \tag{9}$$

```matlab
classdef SimpleCTExample < DrakeSystem
    methods
        function obj = SimpleCTExample()
        % call the parent class constructor:
        obj = obj@DrakeSystem(...
        1, ... % number of continuous states
        0, ... % number of discrete states
        0, ... % number of inputs
        1, ... % number of outputs
        false, ... % because the output does not depend on u
        true);%because the dynamics and output do not depend on t
        end

        function xdot = dynamics(obj,t,x,u)
            xdot = -x+x^3;
        end

        function y=output(obj,t,x,u)
            y=x;
        end
    end
end
```

Figure A-1: Drake (MATLAB) implementation of the nonlinear continuous system described by equations (2) and (3). As shown in the code, `DrakeSystem` instances require the specification of the number of states, both discrete and continuous, inputs, and outputs. Furthermore the user must specify whether the system is time dependant and whether the output directly depends on the input [20].

6.2 Appendix B: Universal Robot Description Format (URDF)

All of the Gazebo simulations conducted in this project utilized URDF files to represent, the ATLAS humanoid robot, the robotic Sandia hand, and all other unactuated objects in the simulated environment. URDF requires specifications for the number of bodies, or links, of a robot, which are contained in link fields. Link fields may also contain other optional information such as visual and collision properties and inertial characteristics such as mass and moment of inertia. Once all the links are specified, joints can be created to connect links with each other. Each joint can only connect two links, the parent link and the child link, and only provides one degree of freedom. Although for simulation this can be cumbersome because of the need to create “dummy links” to simulate multi-DOF joints, it is still appropriate for real life applications as most individual robotic joints only have a single DOF. Figure B-1 shows the XML formatted code that describes a simple robot with four bodies, or links, and three joints and Figure B-2 provides a figure that abstractly represents the description [21].

```xml
<robot name="test_robot">
  <link name="link1" />
  <link name="link2" />
  <link name="link3" />
  <link name="link4" />

  <joint name="joint1" type="continuous">
    <parent link="link1"/>
    <child link="link2"/>
    <origin xyz="5 3 0" rpy="0 0 0" />
    <axis xyz="-0.9 0.15 0" />
  </joint>

  <joint name="joint2" type="continuous">
    <parent link="link1"/>
    <child link="link3"/>
    <origin xyz="-2 5 0" rpy="0 0 1.57" />
    <axis xyz="-0.707 0.707 0" />
  </joint>

  <joint name="joint3" type="continuous">
    <parent link="link3"/>
    <child link="link4"/>
    <origin xyz="5 0 0" rpy="0 0 -1.57" />
    <axis xyz="0.707 -0.707 0" />
  </joint>
</robot>
```

Figure B-1: XML formatted code of a simple robot with four links and three joints [21].

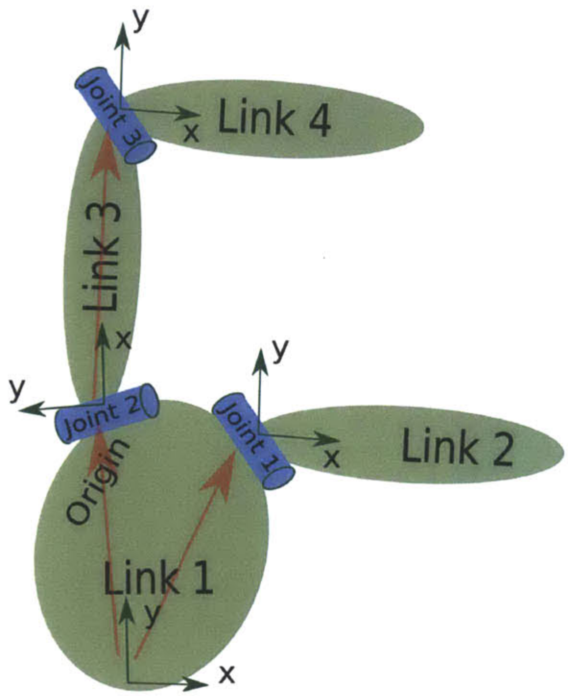


Figure B-2: Abstract representation of the robot described by the code in Figure B-1 [21].

Depending of the capabilities of the parser being used to read URDFs, it is possible to include sensors, mesh descriptions for visual and collision properties, and even plug-ins in these descriptions as well. For the DRC, enhanced parsing capabilities are employed to make the robot description as representative of the hardware system as possible. URDF is part of open source software, thus is it possible to find tutorials and more information about it online [21].

7. References

[1] *Artist's Concept for Robots Competing in the DARPA Robotic Challenge* (2012).

[2] *Autonomous Robotic Manipulation Software (ARM-S) Description.* (2013). Retrieved May 16, 2013, from https://www.rec.ri.cmu.edu/projects/arms/description/

[3] Borst, C., Fischer, M., & Hirzinger, G. (2004). Grasp planning: How to Choose a Suitable Task Wrench Space. *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference.* pp. 319-325 Vol.1.

[4] *Boston Dynamics: PETMAN.* (2013). Retrieved April 30, 2013, from http://www.bostondynamics.com/robot_petman.html

[5] *DARPA Autonomous Robotic Manipulation.* (2013). Retrieved April 30, 2013, from http://thearmrobot.com/index.html

[6] *DARPA Robotics Challenge.* (2013). Retrieved April 29, 2013, from http://www.darpa.mil/Our_Work/TTO/Programs/DARPA_Robotics_Challenge.aspx

[7] *DARPA Urban Challenge.* Retrieved April 29, 2013, from http://archive.darpa.mil/grandchallenge/

[8] *Find Minimum of Constrained Nonlinear Multivariable Function - MATLAB fmincon.* (2013). Retrieved May 16, 2013, from http://www.mathworks.com/help/optim/ug/fmincon.html

[9] *Game Physics Simulation.* (2013). Retrieved May 16, 2013, from http://bulletphysics.org/wordpress/

[10] Grasping. (2008). In B. Siciliano, & O. Khatib (Eds.), *Springer Handbook of Robotics*. Berlin: Springer.

[11] *Gazebo Wiki.* (2013). Retrieved April 30, 2013, from http://gazebosim.org/wiki/Main_Page

[12] Greenemeier, L. (2012). *4-Fingered Robot can Replace Flashlight Batteries [video].* Retrieved April 30, 2013, from http://blogs.scientificamerican.com/observations/2012/08/19/four-fingered-robot-can-replace-flashlight-batteries-video/

[13] Huang, A. S., Antone, M., Olson, E., Moore, D., Fletcher, L., Teller, S., et al. *DARPA Grand Challenge Viewer*

[14] Huang, A. S., Olson, E., & Moore, D. C. *LCM: Lightweight Communications and Marshalling*

[15] *LCM - Lightweight Communications and Processing.* (2012). Retrieved April 29, 2013, from https://code.google.com/p/lcm/

[16] Miller, A. T., & Christensen, H. I. (2003). Implementation of Multi-Rigid-Body Dynamics Within a Robotic Grasping Simulator. *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference.* pp. 2262-2268 vol.2.

[17] Nise, N. S. (2008). *Control Systems Engineering* (5th ed.). United States of America: Wiley.

[18] *PETMAN Tests Camo - YOUTUBE.* . (2013). [Video/DVD] Retrieved from http://www.youtube.com/watch?feature=player_embedded&v=tFrjrgBV8K0

[19] *Sandia Labs News Releases: Lifelike, Cost-effective Robotic Sandia Hand can Disable IEDs.* (2013). Retrieved April 30, 2013, from https://share.sandia.gov/news/resources/news_releases/robotic_hand/

[20] Tedrake, R. (2013). *Drake: A Planning, Control, and Analysis Toolbox for Nonlinear Dynamical Systems.* Retrieved May 16, 2013, from http://drake.mit.edu/

[21] *URDF - ROS Wiki.* (2013). Retrieved April 30, 2013, from http://www.ros.org/wiki/urdf

[22] Vahrenkamp, N., Asfour, T., & Dillmann, R. (2012). Simultaneous Grasp and Motion Planning: Humanoid Robot ARMAR-III. *Robotics & Automation Magazine, IEEE, 19*(2), 43-57.