Tabletop Manipulation

Practices and Applications

Dylan Holmes

December 2015

Contents

List of Figures

List of Tables

Acknowledgements

1 Introduction

Tabletop manipulation is a class of problems in robotics which concerns interacting with objects located on top of a table. Such problems have a large variety of practical application areas such as manufacturing, agriculture and the food industry.

Tabletop manipulation problems can involve structured or unstructured environments. In a structured environment, everything about the environment is known in advance. In terms of tabletop manipulation, this would mean that the initial state of the environment is known and nothing will unexpectedly leave or enter the workspace, in other words the environment is essentially a closed system. An unstructured environment is the opposite of a structured environment. In terms of table top manipulation, anything could be happening in environment, for example another person could also be manipulating items in the same environment.

Some challenges applicable to tabletop manipulation in structured environments are obstacle avoidance and object interaction through contacts. Obstacle avoidance encompasses all efforts to not make contact between a manipulator and an object or between two objects during a task. It is generally a part of path planning, in which one wishes to find a path through an object riddled environment without knocking into any of the objects. Interacting with objects through contacts is one of the most difficult problems in tabletop manipulation [1]. There are a variety of methods for limiting interaction with objects to simplify the problem. Interaction through contacts is a hot topic at the forefront

of state of the art object manipulation research [1]. Unstructured environments present the additional challenge of an unpredictable environment.

There are a number of known methods for approaching tabletop manipulation problems. Some methods are be more suited towards structured or unstructured environments.

One of the simplest control methods for tabletop manipulation is kinematic planning. Kinematic planning methods work by planning trajectories of an end effector and using inverse kinematics to recover the joint angles. Trajectories generated by kinematic planners can be executed using PID control. PID control essentially uses feedback gains to override the system dynamics in order to achieve a desired joint configuration. This is the same as assuming locally linear dynamics, which is an appropriate assumption for small displacements. As velocities increase, so do displacements for a fixed timestep, thus PID control for tracking the higher velocity trajectories.

The artificial potential field technique is collision free path planning method invented by Khatib [2]. The idea is to attribute a higher cost to paths that pass through objects, which allows gradient descent to be used to find the closest collision-free path given an initial path. Because this approach relies on gradient descent, which is a local optimization method, it is subject to local minima.

CHOMP, short for Covariant Hamiltonian Optimization for Motion Planning, is a motion planner that is capable of optimizing higher-order dynamics of trajectories in real-time [3]. CHOMP accomplishes this by refining an initial sampled trajectory using a covariant gradient descent and relies on the assump-

tion of a sparse distribution of obstacles for better performance. STOMP, short for Stochastic Trajectory Optimization for Motion Planning was later developed as an extension of CHOMP, which relies on generating noisy trajectories for space exploration [4].

Optimal control theory is a computational framework that formalizes control problems mathematically as optimization problems, allowing one to find a control policy that is optimal with respect to a given cost function [5]. Using a cost function allows tasks to be specified declaratively, by designing the terms of a cost function using intuitive mathematical analogies derived from the goal of the task itself.

Like many problems in engineering, selecting a control strategy for a particular tabletop manipulation application depends on multiple factors. For structured environments, such as in a factory, a simple kinematic planner and PID control may be simple enough. For other research and development applications, such as working in an unstructured environment and in the presence of humans or animals, kinematic planning may not be enough and more advance method may be required.

The work in this thesis summarizes my undergraduate research experience in applying model-based control on a robotic manipulator. My experience focused on the control aspects of tabletop manipulation, but also heavily relied on all aspects of practical robotics including modeling, calibration, system identification, communication, and robotic framework development. My project revolved around applying known control methods to the simple, yet high level tabletop

manipulation task of picking up a coffee cup from a table and placing it back down in a structured environment.

This paper is organized as follows. The following section gives an overview of the system, including the hardware, framework development, modeling, and system identification. Section 3 outlines the basics of control methods applied during the project. Section 4 covers the actual application of control methods towards a tabletop manipulation task. Section 5 concludes the paper.

2 System

2.1 Hardware

The robotic arm is composed of a ceiling mounted 4-DOF Barrett WAM arm, custom machined 3-DOF wrist, and 3D printed 8-DOF Yale Open Hand Model T [6, 7]. As a whole, the arm has 15 degrees of freedom. The arm and wrist are fully-actuated and account for 7 of the 15-DOFs. The hand is underactucted, with only one actuator controlling all of it's 8 DOFs (2 DOFs per finger, 4 fingers). The assembled system can be seen in figure 1. The Barrett WAM arm uses high velocity brush-less motors with a cable driven winch system to achieve very high compliance. The arm is capable of millimeter repeatability in theory, which turned out to be around 2cm in practice. The arm has a minimum joint resolution 0.005 degrees. The wrist and hand use dynamical MX-64 servo actuators, which support current-based torque control. The dynamixel joint encoders are capable of 0.087 degree resolution. The wrist's actuator supplying

pronation/supination movements is housed in a custom machined steel casing, designed specifically as an attachment for the WAM arm.

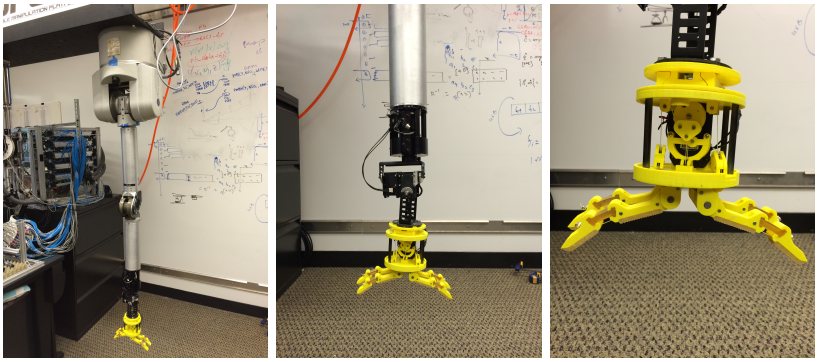

Figure 1: ***Left:*** *Fully assembled robotic arm.* ***Center:*** *Custom machine wrist with Openhand end effector.* ***Right:*** *The Yale Openhand Model T end effector.*

Gimbal lock, the loss of one degree of freedom, occurs between joints 0, 2, and 4 when their central axis line up. This is illustrated in figure 2. The jacobian, which relates joint velocities to end effector velocities, becomes singular when gimbal-lock occurs. For kinematic planners that use the Jacobian to translate a direction in terms of the end effector to a direction in joint space this is less than favorable because planning through them can result in erratic, unpredictable behaviors. Such singularities inside the configuration space also require special attention with optimal control methods as well. When designing a cost function, singularities can prevent the optimizer from being able to find a gradient.

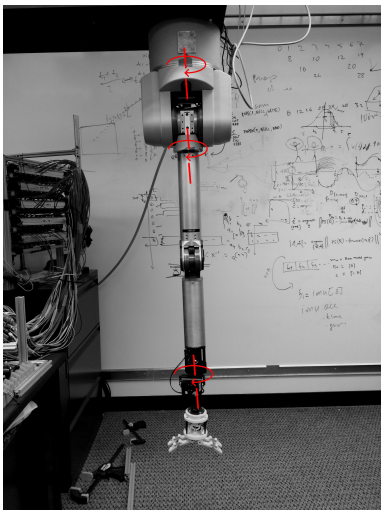

Figure 2: An illustrative diagram of joints 0, 2, and 4 (numbered vertically downward), which have central axis that align perfectly in the shown configuration. This is an instance of gimbal lock between the all three joints (and therefore the loss of 2 degrees of freedom).

The Yale Openhand model T is a light weight open source end effector, designed as a 3D printed version of the original SMD Hand [7]. A single actuator provides a gripping motion to four underactuated finger. The resulting grasp naturally conforms to objects of varying shape without intelligent control.

During system identification and calibration, I used the Phasespace motion capture system. The Phasespace motion capture system is capable of tracking at 480Hz through the use of an active infrared LED marker system. The system has 8 cameras, each of which has a resolution of 12 Megapixels [8].

The integrated system was distributed over three separate computers. Control and synchronization is achieved by integrating device data on one com-

puter, which communicates with the WAM's low level controller through a CAN bus, Dynamixel through USB, and the Phasespace through Ethernet. A block-diagram depiction of the integrated system can be viewed in figure 3.

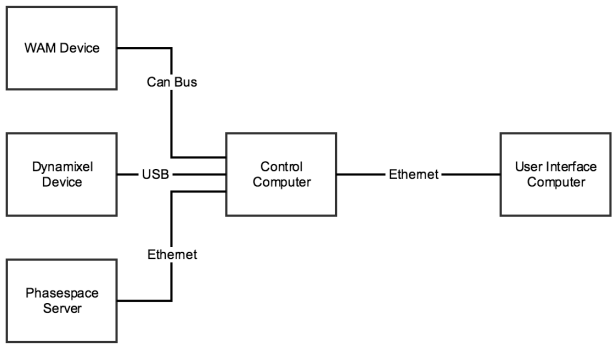

Figure 3: Block diagram of the integrated system, depicting communication between hardware.

2.2 Framework Development

Systems integration is an integral part of any robotics endeavour. From a physical perspective, robots are most often a composition of many different devices, with different manufacturers, interfaces, latencies, update rates, precisions, and repeatabilities. Post-manufacturing, there is really little one can do to improve the quantitative features, which means that one may be limited to the lowest common denominator in the integrated system. For example, a system with one component that has a very large latency and you need to plan a trajectory for the entire system, you may not be able to take advantage of the lower latency provided by the other components. Some devices may not even include a device

driver for you platfom. Luckily, all of the devices used during the course of this project were developed enough to come with manufacturer implemented drivers and APIs. From a software perspective robots rely on a wide variety of different software components, such as the device drivers mentioned, communication software for distributed applications, user interfaces for testing and development, and control software. Most of these software component need to be running simultaneously, thus safely handling of concurrency is important to ensure the low latency that is desirable for running high fidelity trajectories.

The developed framework focused on several features: system abstraction, latency, concurrency, and communication.

System abstraction was needed primarily for system integration purposes. It is more useful to be able to treat the manipulator as a single piece of hardware rather than a slew of components, each of which must be controlled separately. For this purpose, the first step was to create a single interface that abstracts out the device details. However the abstracting low level details must be done with care to achieve low latency and safe concurrency practices.

To abstract a fragmented robotic system while not adding latency to any of the low level controllers, my solution was to add a layer of indirection between the main control thread and the device drivers. This layer of indirection is simply a thread for each device, which acts as a proxy between the main loop and the device. This prevents a device from blocking communication with other devices, by avoiding attempting to read or write to any devices directly from the main control thread.

Only using seperate threads does not completely eliminate the possibility for another thread to block the main thread however. Data still needs to be synchronized safely between concurrent threads through the use of mutexes, which present another possibility of blocking the main thread. To prevent this, I added double buffers to proctor communication between the main thread and the device threads ensures that main thread can not be blocked by a busy device. Double buffers achieve this by limiting the amount of data being manipulated while locking the mutex from the main thread to a single pointer.

Communication was needed to ease development by allowing remote control of the main control process from outside applications such as user interfaces. This part is important not only for interactive development, but is also important for monitoring activity by plotting data and visualizing simulations.

2.3 Modeling

Model-based control relies on a complete mathematical description of a dynamic system, which is used to develop control strategies accordingly. This mathematical description must represent all of the kinematic and dynamic properties of the system. The most common approach involves modeling a robotic system as rigid bodies connect by joints to form an articulated rigid body. This is the approach taken by the physics engine of choice, MuJoCo, which is used throughout this project.

The MuJoCo physics engine greatly simplifies the modeling process [9]. Instead of approaching modeling the kinematic and dynamic properties of the

system as an ad-hoc procedure, MuJoCo has developed it's modeling format to encompass the minimum required parameters for state of the art algorithms used in articulated rigid body simulation with joints and contacts. Modeling with MuJoCo consists of working in a readable XML format, that allows the specification of kinematics properties such as link lengths and geometry, and dynamic properties such as inertial properties in a single file. MoJoCo's API also comes equipped with OpenGL visualization routines. A view of the finished model of the robotic arm can be seen in figure 4.

Initially, model parameters are gathered from manufacturer data sheets if available and measured by hand if applicable. Some model parameters such as joint armature that are either not directly measurable or too difficult to measure because of workspace constraints, must be estimated relative to the other aspects of the model and simulation performance. System identification is the next step in the modeling process, which uses data driven techniques to find the values of unknown model parameters, which is the subject of an section 2.5.

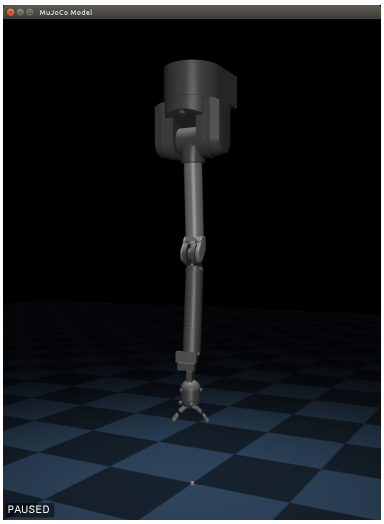

Figure 4: OpenGL mesh rendering of the model, created using the MuJoCo physics engine package.

2.4 Calibration

A sufficient model for the particular application must be accurate enough to allow manipulation of objects on the scale of 5 to 10cm. Given that the end effector is capable of gripping a sphere of at most 12cm in diameter, this leaves approximately +/-1cm room for error. Therefore the preliminary goal was to improve the accuracy of the end effector to a maximum error of +/-1cm. The error in the end effector is a result of two types of errors. Improperly calibrated sensors are one source. Modeling error is another source, although the type of modeling error depends on the control strategy used. Kinematic planning control methods that rely in PID control only rely on kinematic model parameters. Control methods that rely on accurate dynamics parameters will have

error from them as well.

The calibration process concerns finding the parameters, $\hat{c}$, which minimize the error between a calibration function $calib_s(q, c)$ for sensor s given those parameters and set of sensor values q and true values $\hat{q}$. The true values are never known precisely, but are instead taken from another measurement device that is assumed to be closer to the true value than the original device, in other words another calibrated device.

$$\hat{c} = \underset{c}{\operatorname{argmin}} \left|\left|\hat{q} - calib_s(q, c)\right|\right| \tag{1}$$

All of the devices used in this project came with calibration procedures specified by their manufacturers. Calibration of the system was performed intermittently throughout the course of the project as needed.

Coordinate system registration is the process of finding an affine transformation between two coordinate systems [10]. The Phasespace motion capture system and MuJoCo physics engine each have their own coordinate system. Thus an affine transformation is needed in order to get the location of a mocap marker relative to the model. Coordinate system registration can be considered part of the calibration process. This was accomplished by constructing a frame in the model coordinate space, getting the corresponding frame in the motion capture space, then finding the transformation between the two. An alternative approach is to use forward kinematics to get a coordinate transformation while simultaneously performing system identification.

2.5 System Identification

In it's most general sense, system identification is the process of building a description of a dynamics model from data. Luckily, the system at hand is a special case, it has a model described by physics, and only the true values of the model parameters need to be identifiedd. This can be approached as an a nonlinear optimization problem, in which we wish minimize the error between a recorded trajectory and predicted trajectory given a set of model parameters. For kinematic parameters, the error between trajectories is taken to be the sum of square differences between a recorded trajectory of motion capture marker positions, r and marker positions as predicted by the model given the corresponding recorded joint angles, $forward(m, q)$.

$$\hat{m} = \underset{m}{\operatorname{argmin}} \left|\left| r - forward(m, q) \right|\right| \tag{2}$$

.

Identifying parameters was done on a need to know basis, due to the large amount of model parameters (on the scale of hundreds), many of which have little effect on certain tasks.

One additional parameter not included explicitly in the model is the latency of motion capture data collection, which can be established by optimizing over a shift in the data collected from the motion capture system and the state trajectory.

3 Control Methods

3.1 Kinematic Planning

A common method for generating trajectories, is keyframe interpolation. A keyframe is a single frame of a trajectory at a certain time. Keyframe interpolation forms a trajectory by interpolating multiple keyframe together through time.

Here I describe a kinematic planner implementation that was used during this project. This kinematic planner uses an augmented version of keyframe interpolation to generate a trajectory of joint configurations. The generated trajectory can then be executed on the manipulator using PID control. The keyframes can be specified in terms of the end effector in the workspace or in terms of specific joint positions. This process involves three steps, generating an initial trajectory from the keyframes, optionally converting the initial trajectory into a trajectory of joint positions if the initial trajectory was specified in terms of the end effector in the workspace, and ultimately running the trajectory on the hardware using PID control.

In generating trajectories, we want them to be smooth, satisfy kinematic and dynamic constraints, avoid objects (if desired), as well as be dynamically stable. This interpolation method assumes that the kinematic constraints are being considered before input. The dynamic constraints and stability will be satisfied as long as resulting trajectory is not too fast, since it is intended to be run using PID control. Smoothness, however, must be handled here. Defining this

as an optimization problem, the smoothness of a trajectory can be interpreted as minimizing the square jerk over the duration of a trajectory [11]. Jerk, $\dddot{x}$, is defined as the third derivative of position with respect to time:

$$\dddot{x} = \frac{d^3x}{dt^3} \tag{3}$$

The smoothness over a trajectory from initial state $(x_{t_0}, \dot{x}_{t_0}, \ddot{x}_{t_0})$ to final state $(x_{t_f}, \dot{x}_{t_f}, \ddot{x}_{t_f})$ is then defined as:

$$\int_{t_0}^{t_f} \dddot{x}^2 dt \tag{4}$$

Since this term includes an integral, finding the resulting minimum jerk trajectory can be solved by minimizing the functional:

$$H(x) = \frac{1}{2} \int_{t_0}^{t_f} \dddot{x}^2 dt \tag{5}$$

Techniques for minimizing functionals come from the calculus of variations. First we define a variation, $\eta(t)$, such that $(\eta_{t_0}, \dot{\eta}_{t_0}, \ddot{\eta}_{t_0}) = (\eta_{t_f}, \dot{\eta}_{t_f}, \ddot{\eta}_{t_f}) = (0, 0, 0)$. This will allow the variation to leave the boundary conditions of the desired trajectory as we defined them. Replacing x by the variation $x \mapsto x + e\eta$:

$$\begin{aligned}
H(x + e\eta) &= \frac{1}{2} \int_{t_0}^{t_f} (\dddot{x} + e\dddot{\eta})^2 dt \\
\frac{dH(x + e\eta)}{e} &= \int_{t_0}^{t_f} (\dddot{x} + e\dddot{\eta})\dddot{\eta}\, dt \\
\frac{dH(x + e\eta)}{e}\Big|_{e=0} &= \int_{t_0}^{t_f} \dddot{x}\dddot{\eta}\, dt
\end{aligned}$$

Then applying integration by parts three times to simplify the right hand side:

$$\begin{aligned}
&\int_{t_0}^{t_f} \dddot{x}\dddot{\eta}\,dt \\
\dddot{x}\ddot{\eta}\Big|_{t_0}^{t_f} &- \int_{t_0}^{t_f} x^{(4)}\ddot{\eta}dt \\
&- \int_{t_0}^{t_f} x^{(4)}\ddot{\eta}dt \\
-x^{(4)}\dot{\eta}\Big|_{t_0}^{t_f} &+ \int_{t_0}^{t_f} x^{(5)}\dot{\eta}dt \\
&\int_{t_0}^{t_f} x^{(5)}\dot{\eta}dt \\
x^{(6)}\eta\Big|_{t_0}^{t_f} &- \int_{t_0}^{t_f} x^{(6)}\eta dt \\
&\int_{t_0}^{t_f} x^{(6)}\eta dt
\end{aligned}$$

Shows that following property is true for any functional, $\eta(t)$:

$$H(x) = \int_{t_0}^{t_f} x^{(6)}\eta dt = 0 \tag{6}$$

Thus any trajectory with initial state $(x_{t_0}, \dot{x}_{t_0}, \ddot{x}_{t_0})$ to final state $(x_{t_f}, \dot{x}_{t_f}, \ddot{x}_{t_f})$ will have minimum jerk when the sixth derivative of position with respect to time is zero. This is gives us the differential equation:

$$x^{(6)} = 0 \tag{7}$$

The general solution of which is:

$$x = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5 \tag{8}$$

Using the derivatives of x, we end up with a system of linear equations that can give us a minimum jerk trajectory for any initial state $(x_{t_0}, \dot{x}_{t_0}, \ddot{x}_{t_0})$ and final

state $(x_{t_f}, \dot{x}_{t_f}, \ddot{x}_{t_f})$:

$$\begin{bmatrix} 1 & t_0 & t_0^2 & t_0^3 & t_0^4 & t_0^5 \\ 0 & 1 & 2t_0 & 3t_0^2 & 4t_0^3 & 5t_0^4 \\ 0 & 0 & 2 & 6t_0 & 12t_0^2 & 20t_0^3 \\ 1 & t_f & t_f^2 & t_f^3 & t_f^4 & t_f^5 \\ 0 & 1 & 2t_f & 3t_f^2 & 4t_f^3 & 5t_f^4 \\ 0 & 0 & 2 & 6t_f & 12t_f^2 & 20t_f^3 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} = \begin{bmatrix} x_{t_0} \\ \dot{x}_{t_0} \\ \ddot{x}_{t_0} \\ x_{t_f} \\ \dot{x}_{t_f} \\ \ddot{x}_{t_f} \end{bmatrix} \tag{9}$$

Thus, $x(t)$ for any $t_0 \leq t \leq t_f$:

$$x(t) = \begin{bmatrix} 0 \\ t \\ t^2 \\ t^3 \\ t^4 \\ t^5 \end{bmatrix}^T \begin{bmatrix} 1 & t_0 & t_0^2 & t_0^3 & t_0^4 & t_0^5 \\ 0 & 1 & 2t_0 & 3t_0^2 & 4t_0^3 & 5t_0^4 \\ 0 & 0 & 2 & 6t_0 & 12t_0^2 & 20t_0^3 \\ 1 & t_f & t_f^2 & t_f^3 & t_f^4 & t_f^5 \\ 0 & 1 & 2t_f & 3t_f^2 & 4t_f^3 & 5t_f^4 \\ 0 & 0 & 2 & 6t_f & 12t_f^2 & 20t_f^3 \end{bmatrix}^{-1} \begin{bmatrix} x_{t_0} \\ \dot{x}_{t_0} \\ \ddot{x}_{t_0} \\ x_{t_f} \\ \dot{x}_{t_f} \\ \ddot{x}_{t_f} \end{bmatrix} \tag{10}$$

If the input keyframes were in terms of end effector configurations, then the trajectory must be converted into a trajectory of joint positions. This is done using the extended Jacobian control method [12]. Inverse kinematics may possibly have zero, one, or many solutions, thus a nominal pose is required as a regularizer, to select a preferred solution from the Jacobian's pseudo-inverse nullspace.

This suffices being able to generate trajectories offline for execution via PID control. It is possible, however, to take this approach further and construct feedback control system that will always be executing a minimum jerk trajectory

given a target final state, for details see [13].

3.2 Trajectory Optimization

Trajectory optimization is the process of minimizing a given cost function over a trajectory of states and controls of a dynamic system. The cost function can be designed to produce trajectory for solving arbitrary tasks that are only limited by the inherent properties of the system and the corresponding model. Methods for solving trajectory optimization problems can lumped into two categories: direct methods and indirect methods. Direct methods are numerical methods for solving approximate optimal control problems using nonlinear programming. Indirect methods consist of both analytical and numerical methods for solving trajectory optimization problems and are derived from calculus of variations or Pontryagin's maximum principle.

To see the formulation of an example optimal control problem, consider a discrete time dynamic system with state $x \in \mathbb{R}^n$, control $u \in \mathbb{R}^m$ and system dynamics $\dot{x} = f(t, u, x)$. We can define an optimal control problem for this system from time t_0 to time t_f as follows:

$$J = L_f(x(t_f)) + \sum_{t=t_0}^{t_f-1} L(t, x(t), u(t)) \tag{11}$$

subject to:

$$\dot{x} = f(t, u, x), x(t_0) = x_0 \tag{12}$$

This defines a cost functional as a composition of a running cost $L : (x, u, t) \mapsto \mathbb{R}$ and special terminal cost $L_f : (x, u, t) \mapsto \mathbb{R}$ for handling the terminal state

separately.

4 Task Implementation

4.1 Kinematic Planning

Kinematic planning methods are commonly used in conjunction with state machines to perform high level tasks. This is done by systematically breaking up the task into possible states, then by defining the transition between those states. For grasping transitions, the kinematic planner can be used to generate trajectories in joint space. For reaching transitions, the kinematic planner can be used to generate trajectories of the end effector in the workspace.

To test the kinematic planner outlined earlier, I designed a state machine to pick up a coffee cup from a pre-specifed location, then place it at another pre-specified location, which worked as designed. A sequence of video frames depicting the trajectory execution is shown in figure 5.

Figure 5: Demonstration of tabletop manipulation on hardware using kinematic planning. The task is to move the coffee cup from one location on the table to another.

4.2 Trajectory Optimization

Using Matlab software packages developed with [5], I was able to approach solving the pick and place task from an optimal control perspective. The package consists of an implementation of a trajectory optimization method called control-limited differential dynamic programming, which is an indirect method for solving trajectory optimization problems. The software only requires you to specify the system dynamics, cost, and associated derivatives. Used in conjunction with MuJoCo; it is even easier. MuJoCo, being a physics engine, can compute the system dynamics and it's derivative for the optimizer. In addition, MuJoCo has an extension to it's XML modeling format which allows the cost function to be specified in XML as well. This provides a readable, structured

environment that allows the user to focus on developing cost function instead of associated plumbing.

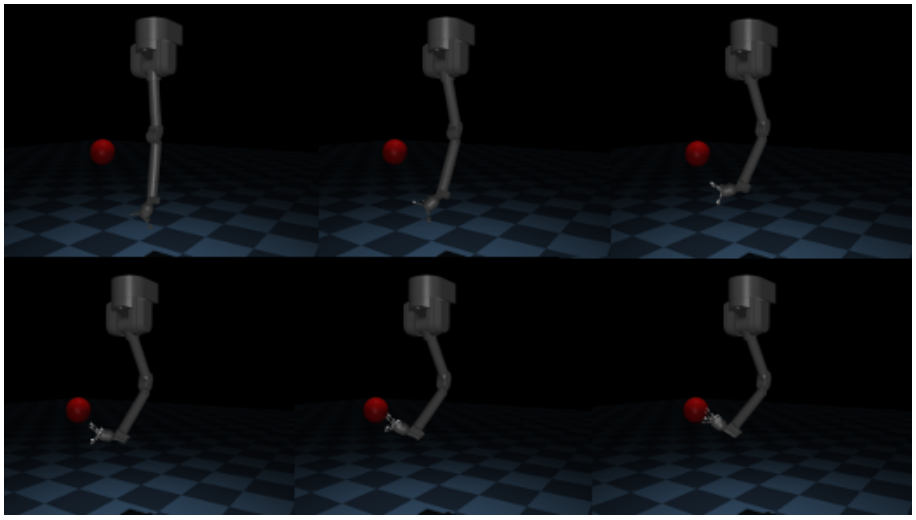

Figure 6: Image sequence of a reaching task trajectory generated with trajectory optimization.

Figure 6 shows an image sequence taken of the robotic arm performing a reaching task, in which the target is the red sphere. The trajectory was produced using trajectory optimization. Terms from the cost function for this particular trajectory are given in table 1. Figures 7 and 8 each depict a sequence of plots during the optimization process. Each plot in figure 7 shows the states and running cost over the length of the trajectory at different iterations during the optimization, starting from the beginning of optimization from top to bottom, left to right. Similarly, each plot in figure 8 shows the controls and linear feedback gains. The total time taken during the optimization process was roughly 5 seconds, with plotting consuming approximately 90% of the run-time.

quantity	***norm***	***coefficient***
control	quadratic	1×10^{-4}
momentum	quadratic	1×10^{-4}
distance from nominal pose	quadratic	1×10^{-4}
distance from target to end effector	quadratic	1

Table 1: Terms of a cost function used to accomplished a reaching task.

Designing cost function is more of an art than a science. Half of the process is intuition based on domain knowledge and experimentation, and the other half is tweaking parameters when you know you're on the right track. The first step is to translate the goal of the task into a physical metaphor in the form of a term in the cost function. For example, in designing a cost function to perform the reaching transition from the state machine in the section 4.1, the first term would be exactly the quantity you care about, the distance between the end effector position and target position. This is essentially achieving the same end result as the kinematic planner from section 3.1, apart from the path. There are differences however. This naive approach will most likely produced an undesirable trajectory, but logically so. The optimizer only cares about the cost, exactly as specified. It will do what ever it can to reduce it. This means for example, not penalizing control can result in an aggressive, forceful trajectories. This is because it would be suboptimal for the optimizer to not exploit it's freedom and get to the target as fast as possible to keep the total cost down.

Intuitions such as the example above come quite naturally through experimentation. An additional bonus to this method is that once a cost function is found that accomplishes at task, it can then be augmented to perform it in alternative styles. For example in the reaching task, you may decide that movement efficiency is more important and you can slowly crank up a cost on energy.

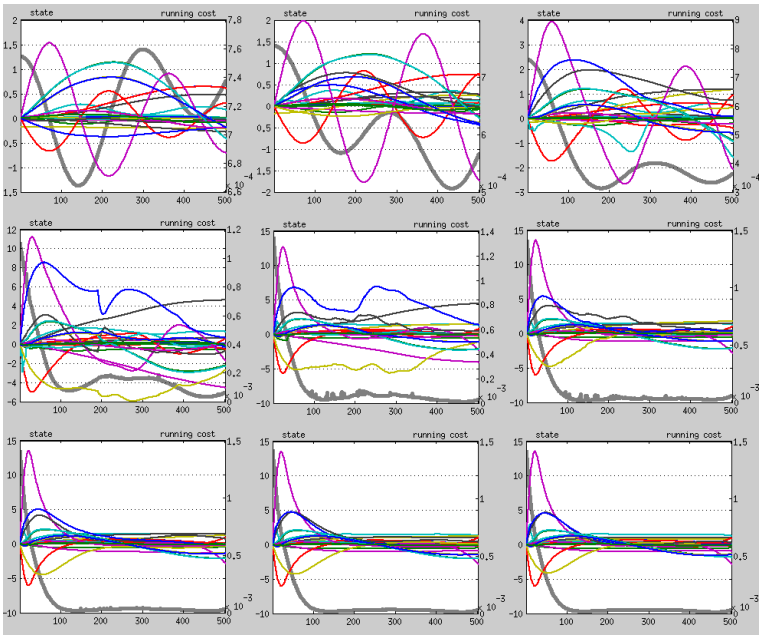

Figure 7: Sequence of plots depicting the states and running cost over the length of the trajectory during different iterations during the optimization. Sequence is ordered sequentially top to bottom, left to right.

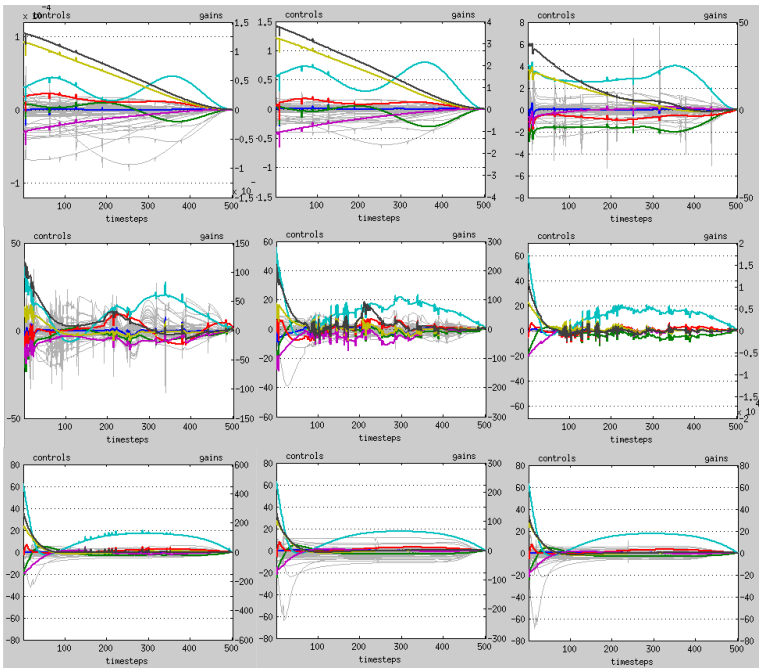

Figure 8: Sequence of plots depicting the controls and linear feedback gains over the length of the trajectory during different iterations during the optimization. Sequence is ordered sequentially top to bottom, left to right.

5 Conclusion

This main goal of this work was to give an overview of the aspects of applying model-based control to a tabletop manipulation problem. Throughout the process I touched on the practical areas of robotics such as hardware, framework development, the modelling process, as well as control strategies to present an overall introductory picture to tabletop manipulation. Applications of two different control methods towards the same task were attempted to highlight

the different practical aspects of choosing a particular control method for an application.

References

[1] Kumar, Vipin, et al., "Real-time behaviour synthesis for dynamic Hand-Manipulation" in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, IEEE, 2014.

[2] Khatib, Oussama. "Real-time obstacle avoidance for manipulators and mobile robots." *The international journal of robotics research* 5.1 (1986): 90-98.

[3] Ratliff, Nathan, et al. "CHOMP: Gradient optimization techniques for efficient motion planning." *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on.* IEEE, 2009.

[4] Kalakrishnan, Mrinal, et al. "STOMP: Stochastic trajectory optimization for motion planning." *Robotics and Automation (ICRA), 2011 IEEE International Conference on.* IEEE, 2011.

[5] Tassa, Yuval, Nicolas Mansard, and Emo Todorov. "Control-limited differential dynamic programming." *Robotics and Automation (ICRA), 2014 IEEE International Conference on.* IEEE, 2014.

[6] Barrett Technology Inc. "WAM Arm datasheet" Feb. 2011.

[7] Ma, Raymond R., Lael U. Odhner, and Aaron M. Dollar. "A modular, open-source 3d printed underactuated hand." *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013.

[8] PhaseSpace Inc., http://phasespace.com/

[9] Todorov, Emanuel, Tom Erez, and Yuval Tassa. "MuJoCo: A physics engine for model-based control." *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012.

[10] Bachrach, Jonathan, and Christopher Taylor. "Localization in sensor networks." *Handbook of sensor networks: Algorithms and Architectures* 1 (2005).

[11] Shadmehr, Reza, and Steven P. Wise. *The computational neurobiology of reaching and pointing: a foundation for motor learning*. MIT press, 2005.

[12] Buss, Samuel R. "Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods." *IEEE Journal of Robotics and Automation* 17.1-19 (2004): 16.

[13] Hoff, Bruce, and Michael A. Arbib. "Models of trajectory formation and temporal interaction of reach and grasp." *Journal of motor behavior* 25.3 (1993): 175-192. Bruce Hoff, Michael Arbib, "Models of Trajectory Formation and Temporal Interaction of Reach and Grasp,"

[14] GFantoni, Gualtiero, et al. "Grasping devices and methods in automated production processes." CIRP Annals-Manufacturing Technology 63.2 (2014): 679-701.