The Design and Implementation of a Scalable Two Degree of Freedom Position Control Mechanism

Gabriel E. Baraban, David J. Beck, Anna D. Cardinal and
Joshua A. Zimmer, '15

Submitted to the
Department of Mechanical and Aerospace Engineering
Princeton University
in partial fulfillment of the requirements of
Undergraduate Independent Work.

Final Report

April 30, 2015

Prof. Michael Littman
Prof. Egemen Kolemen
MAE 444D
106 pages
File Copy
Color Images

This senior project represents our own work in accordance with University regulations.

Abstract

This report details the invention of a scalable two degree of freedom mechanism that accurately controls the position of an end effector. This invention consists of two motors, each of which controls the linear position of one arm via a gear rack and pinion. The arms are joined at a hinge, which together with the two motors forms a triangle. The position of the hinge is determined by its distance from each motor. The motors are mounted to turntables, giving them the ability to rotate passively with the system's motion. Comparing the rack and pinion mechanism to a five-bar linkage showed that the Arduino-based rack and pinion mechanism more accurately reproduced shapes, but the Quanser-based five-bar linkage performed better at recording motion. The mechanical and computational limits of the two systems provided context for a comparison of their performance.

Acknowledgements

We would like to thank Professor Littman for advising us, Professor Rowley and Professor Houck for providing resources and consultation, and Glenn Northey and Professor Martinelli for their assistance in the shop and with choosing materials. Many thanks to Jon Prevost for electronics mentoring, providing us with the Quanser setup and required computer, and his dry sense of wit. Thanks to Mike Vocaturo for being our technical point of contact. Thanks to Jo Ann Kropilak-Love for her administrative assistance and help with ordering materials. Thanks to the MAE Department and SEAS for project funding and resources.

Gabe derived and developed the dynamics and control algorithms for both systems and tested the Quanser system. David performed design using CAD software, manufactured the mechanical systems, derived the geometry, and tested and optimized the rack and pinion system. Annie managed the project logistically as project lead, designed and manufactured the mechanical systems, and made the poster. Josh developed the electrical and software components for and tested and optimized the control of the Arduino-based rack and pinion system.

Gabe: I could have spent all month thanking all of the people who have made this possible, but I want to especially thank: Samuels, for putting up with who I am, Annie, for helping me towards who I strive to be, and Matt Walsh and Tim Matchen for teaching me Lagrangian dynamics in one night. Finally, I want to thank my loving parents for their love and support over the last twenty two years. I guess this is it. Wow.

David: First, I would like to thank all of the friends (Holder 10, Clockwork Ultimate, and Terrace) who have been there for me not just throughout the past year, but also during my entire time at Princeton. A special thanks goes to Suzie Shoffner for her advice, her sense of humor, and for putting up with me day after day. Finally, I would like to thank my parents Lynn and John Beck. I would not be here today without their support, trust, love, and understanding.

Annie: I would like to thank Glenn for his support and advice throughout the entire process. My parents, Dave and Lorrie, have been a wonderful sounding board for ideas and encouragement. Thank you to Silken and Gabe for bringing me up when I'm down. I'd also like to thank Kristin, Riley, Mike, Andra, Matthew, Matt, Leann, Graham, and everyone at Charter for their support when times were tough. And of course, many thanks to my awesome project team for a great year!

Josh: I cannot believe that my collegiate journey is rapidly approaching its end, and so, I would like to take a moment to thank everyone that has assisted me throughout. Mom, thank you for all of your help, support, and guidance. Your assistance has played an essential role in the journey that has brought me this point. Every day, you inspire me to try harder, better myself, and improve the lives of those around me. To My Professors: thank you all for providing me with the best education that I can imagine. Your influence, instruction, and passions have expanded my world-view and impacted my life immeasurably. To Mr. Glickman and Mr. Arechabaleta: thank you both for guiding my optimism, my interest in engineering, and my intellectual curiosity overall. Without your inspiration, I would not be the man that I am today. To Gabe, David, and Annie: We did it! Thank you for your dedication, teamwork, and passion. The three of you have been the best teammates I can imagine, and I hope that our friendships will continue into the foreseeable future.

Till the Break of Dawn

Contents

List of Tables

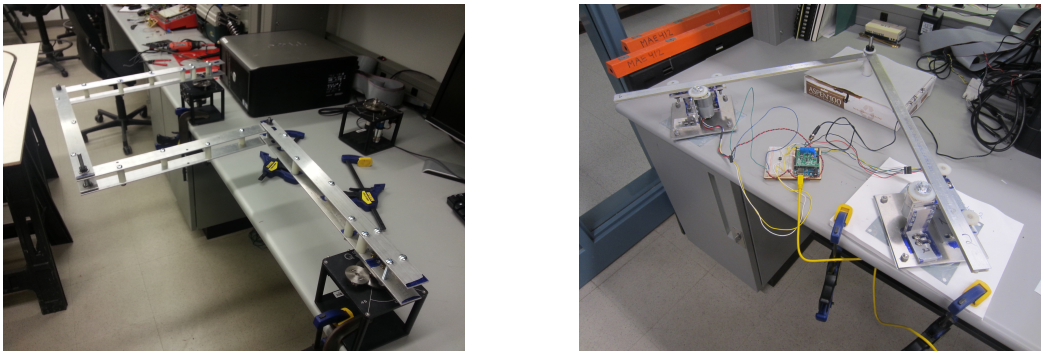
List of Figures

Chapter 1

Introduction

1.1 Project Overview and Purpose

This report details the invention of a scalable two degree of freedom mechanism that accurately controls the position of an end effector. After examining existing systems, we noticed the opportunity to invent a new type of mechanism for recording and playing back motion. For a controlled comparison with our invention–the rack and pinion mechanism–we prototyped a traditional five-bar linkage.



(a) Quanser Five-Bar Linkage (b) Arduino Rack and Pinion Mechanism

Figure 1.1: A Visual Comparison of the Two Mechanisms

The five-bar linkage manufactured in the first phase of this project interfaces with a Quanser Easy-PCE data acquisition board. The Quanser board connects to MATLAB and Simulink models to read encoders, interpret data, and output motor voltages with millisecond precision (see Appendix B). The rack and pinion mechanism manufactured in the second phase of the project interfaces with an Arduino Uno (see Appendix D). This mechanism is more accurate and incorporates what we learned from the

development and testing of the five-bar linkage. While the comparison of these two mechanisms is not entirely fair due to differences in the controllers and motors, testing nonetheless revealed the advantages and disadvantages of each mechanical system. The two systems are shown side by side in Figure 1.1.

We compared the ability of each system to record and play back different types of motion. We predicted that the rack and pinion mechanism would allow for more accurate control of the end effector's position and would reduce amplification of physical error but that the five-bar linkage would be easier to move when recording motion.

Although our motivation for this project was to learn about designing, manufacturing, and controlling mechanisms, we wanted to link it to possible applications. This project relates to a wide range of applications as many situations call for the accurate capture and playback of physical motion. Though our project is a prototype, similar apparatuses could be used to accurately reproduce, scale, and control physical motion. Possible applications include robotic surgery, training motor skills, and automated manufacturing.

1.2 Background Research and Influence on Design

Our initial project was inspired by a Japanese trash can robot that tracks crumpled paper ball projectiles and moves autonomously to catch them [14]. The robot uses computer vision algorithms to find a user-tossed ball, so they toss it and leave its coordinates unknown to the robot [11]. Our team wanted to re-derive the concept, obtain more consistent results, and apply the knowledge to a more interesting application, such as hitting the projectile back to the user. The five-bar linkage was initially designed and manufactured with this intent in mind. However, the computer vision portion of the project did not coincide with the interests of team members, leading to our decision to pivot our project concept and to find a way of repurposing the five-bar linkage.

After researching many different uses of two degree of freedom mechanisms, we learned about the Polygraph used in the 1800s by Thomas Jefferson to copy physical writing. John Isaac Hawkins patented the Polygraph in 1803, and Jefferson started using it soon after to copy his letters and papers. The original Polygraph even replicated the angle of the pen, producing a very accurate copy. Through the years, other important figures used electrical versions of the Polygraph, now called the Autopen, to sign documents remotely. Most famously, President Obama used an Autopen to remotely sign the Fiscal Cliff compromise bill of 2013 into effect while on

vacation in Hawaii [7]. Photographs are shown in Figure 1.2.

Because the Autopen uses a five-bar linkage, we repurposed the Quanser mechanism to test its ability to record and play back motion. After testing, we decided to make a second mechanism better suited to the task and developed the concept of the rack and pinion mechanism. Our design requires less space to achieve the same range of accurate motion and is also easy to implement using store-bought parts.

(a) President Jefferson's Polygraph [7] (b) President Obama's Autopen [7]

Figure 1.2: A Visual Comparison of the Polygraph and Autopen

The decision to initially proceed with a five-bar linkage was based on Professor Littman's recommendation and on our access to the Quanser servo motors [12]. Instead of purchasing new motors and encoders, the Quanser servo motors were a pre-built solution that saved time and money in the first phase of the project. However, we desired a larger range of motion than that provided by Quanser's 2DOF Robot kit, which is 10 inches across, so we manufactured and constructed our own three-foot wide mechanism.

Using materials provided by Professor Clarence Rowley of Princeton University, we derived control laws for the five-bar linkage [4]. Derivations of the effective moment of inertia, J, for each system are included in sections 2.3 and 3.3. The derivation of J for the five-bar linkage is based on that found in "Experimental Study of a Two-DOF Five Bar Close-Loop Mechanism" [13].

1.3 Design Constraints and Specifications

The budget for the entire project was $2100. $1600 came from the MAE Department Fund, and $500 from the SEAS Fund. This was sufficient funding for both phases, but the team opted to spend as little as possible on the first phase in order to build an entirely new mechanism in the second phase.

As a result, the motors and computers for the first phase were loaned from the Princeton University Controls Laboratory, as they are the most expensive components. The five-bar linkage was designed specifically to connect to the Quanser motors. In addition, the Quanser system gave the team a solid framework to interface motors and sensors with Simulink and MATLAB. The first phase cost approximately $350, leaving $1850 for the second phase, of which we spent $998. We chose to make each arm of the linkage 18 inches long, giving the linkage a span of three feet across to provide a comfortable range of motion for writing.

In the second phase, we chose an Arduino primarily to facilitate transportation of our final product because the Quanser system and computers were too heavy to move and set up for demonstrations. Using Arduino also allowed us to program our own system and use third party motors. The second mechanism has a smaller range of overall motion than the five-bar linkage but a similar range of accurate motion.

Chapter 2

Design of the Quanser Five-Bar Linkage System

2.1 System Overview

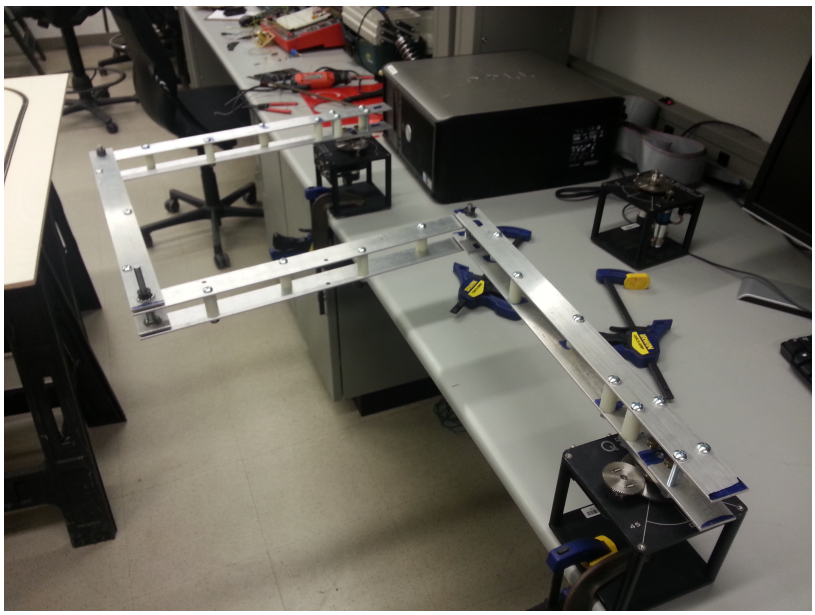


Figure 2.1: Quanser Five-Bar Linkage System

The five-bar linkage provides two degrees of freedom and a large range of motion. Five-bar linkages are used in many applications today, making it an ideal choice for controlled testing. The use of readily available motors and software was optimal for

prototyping, understanding control algorithms, and determining the accuracy of a typical five-bar linkage. It is controlled by a Quanser Easy-PCE board and connects to a computer running Simulink within MATLAB. Smooth bearing joints allow for precise and rapid movement.

2.2 Mechanical System

2.2.1 Design Concept

We chose the five-bar linkage because it provides the necessary two degrees of freedom with the minimum number of links and allowed us to use stationary motors. We designed the five-bar linkage to be simple, robust, and easily disassembled.

2.2.2 Physical Design

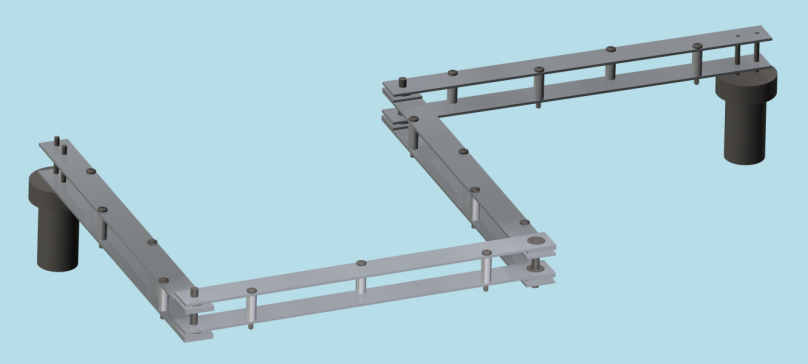


Figure 2.2: Creo Model of the Five-Bar Linkage

The five-bar linkage consists of four physical arms and a fifth arm implicitly connecting the two fixed motors. This allows for two degrees of freedom that are controlled by the motor angles. The Quanser 2DOF Robot kit design has a very small range of motion with the two motors only ten inches apart. Given our initial goal of catching projectiles, we required a larger range of motion, shown in Figure 2.7. Our modified five-bar linkage is three feet across with each linkage measuring 18 inches in length. The total coverage area is approximately six square feet. Each linkage consists of two eighth-inch thick aluminum bars 18 inches long connected by staggered plastic spacers that hold them 1.5 inches apart. This provides the necessary stiffness to prevent bending due to excessive loading.

The four physical linkage arms are connected by joints that consist of two self-aligning bearings, an 8mm diameter steel shaft, and a spring to maintain constant separation. Shaft collars constrain the joint on either side. The joint, illustrated in Figure 2.3, has very little play and allows for easy maintenance and minimal manufacturing due to the style of assembly.

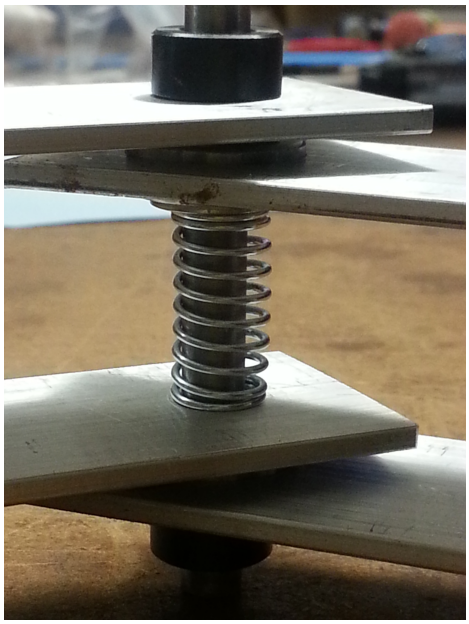


Figure 2.3: Self-Aligning Bearing Joint

The connection to the motors consists of a reinforcing plate that is machined to match the holes in the drive gear on the Quanser servo. The final setup is shown in Figure 2.1.

The four linkage bars add up to about four pounds in weight. This could be reduced by cutting material out of the aluminum bars. Eliminating weight would prevent the bars from bending and also require less torque from the motors for control.

2.2.3 Geometry and Range of Motion

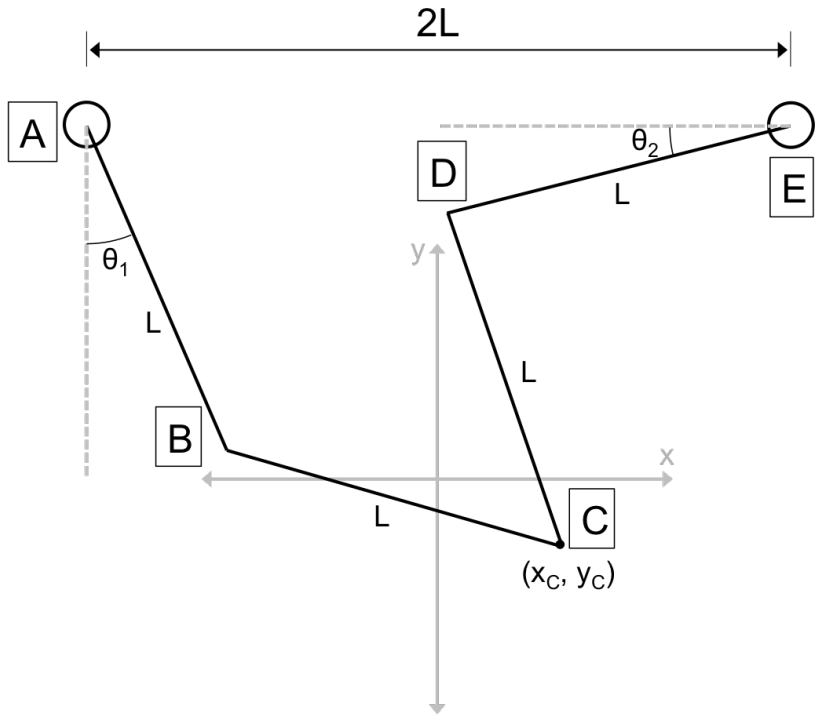


Figure 2.4: Geometry of the Five-Bar Linkage

Figure 2.4 shows the layout of the five-bar linkage. For the calculations in this section, the left motor is labeled "A", the three hinges are labeled "B", "C", and "D" (from left to right along the linkage), and the right motor is labeled "E". θ_1 is the angle of motor 1 from the vertical, and θ_2 is the angle of motor 2 from the horizontal. The position of interest is the position of hinge C, which has coordinates (x_C , y_C). The origin is the position of this hinge when $\theta_1 = \theta_2 = 0$.

Using this mechanism for position control requires two different transformations: one to convert (x_C, y_C) to (θ_1, θ_2), and one to convert (θ_1, θ_2) to (x_C, y_C).

Converting Between Cartesian Position and Motor Angles

The following equations are used to solve for the motor angles (θ_1, θ_2) that will result in a given Cartesian coordinate for the middle hinge. This calculation is used to recreate a sequence of positions. The angles that correspond to each position are calculated, and then the motors are forced to those angles.

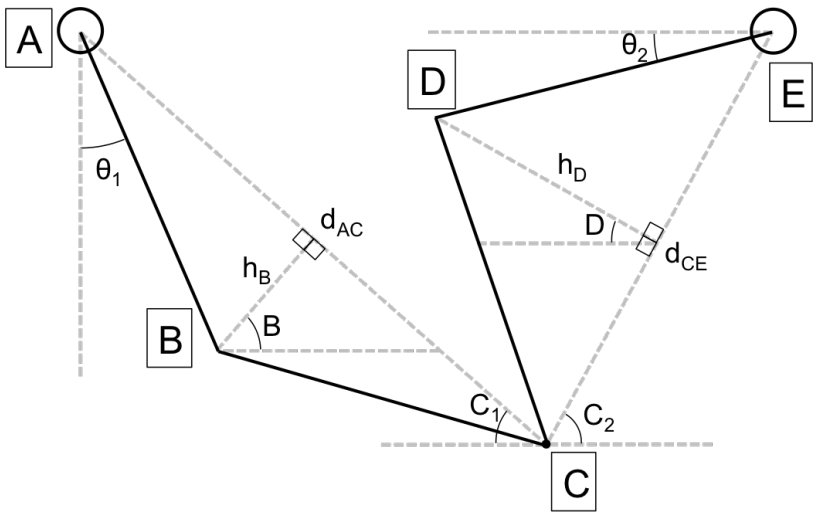


Figure 2.5: Geometry for Converting Position to Motor Angles

Figure 2.5 and the equations that follow describe the conversion from position (x_C, y_C) to motor angles (θ_1, θ_2).

$$d_{AC} = \sqrt{(L + x_C)^2 + (L - y_C)^2} \qquad d_{CE} = \sqrt{(L - x_C)^2 + (L - y_C)^2}$$

$$C_1 = \cos^{-1}\left(\frac{L + x_C}{d_{AC}}\right) \qquad C_2 = \cos^{-1}\left(\frac{L - x_B}{d_{CE}}\right)$$

$$h_B = \sqrt{L^2 - \left(\frac{d_{AC}}{2}\right)^2} \qquad h_D = \sqrt{L^2 - \left(\frac{d_{CE}}{2}\right)^2}$$

$$B = \frac{\pi}{2} - C_1 \qquad D = \frac{\pi}{2} - C_2$$

$$x_B = x_C - \frac{d_{AC}}{2}\cos(C_1) - h_B \cos(B) \qquad x_D = x_C + \frac{d_{CE}}{2}\cos(C_2) - h_D \cos(D)$$

$$y_B = y_C - \frac{d_{AC}}{2}\sin(C_1) - h_B \sin(B) \qquad y_D = y_C + \frac{d_{CE}}{2}\sin(C_2) + h_D \sin(D)$$

$$\theta_1 = \sin^{-1}\left(\frac{L + x_B}{L}\right) \qquad \theta_2 = \sin^{-1}\left(\frac{L - y_D}{L}\right)$$

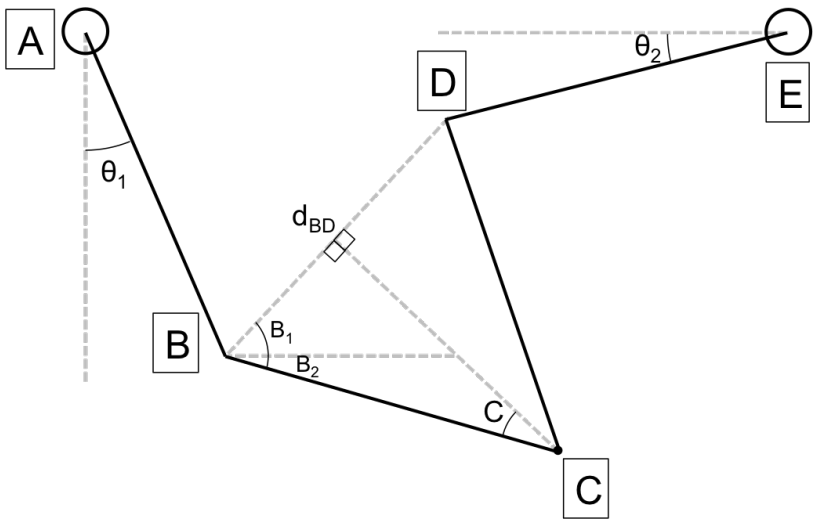


Figure 2.6: Geometry for Converting Motor Angles to Position

Figure 2.6 and the equations that follow describe the conversion from motor angles (θ_1, θ_2) to position (x_C, y_C).

$$x_B = L(\sin(\theta_1) - 1) \qquad y_B = L(1 - \cos(\theta_1))$$

$$x_D = L(1 - \cos(\theta_2)) \qquad y_D = L(1 - \sin(\theta_2))$$

$$d_{BD} = \sqrt{(x_D - x_B)^2 + (y_D - y_B)^2}$$

$$C = \sin^{-1}\left(\frac{d_{BD}}{2L}\right)$$

$$B_1 = \tan^{-1}\left(\frac{y_D - y_B}{x_D - x_B}\right) \qquad B_2 = \frac{\pi}{2} - C - B_1$$

$$x_C = x_B + L\cos(B_2) \qquad y_C = y_B - L\sin(B_2)$$

Range of Motion

One complication of the five-bar linkage mechanism is that for every pair of motor angles there are two possible positions of the middle hinge, and for every hinge position there are four possible combinations of motor angles (two angles per motor). In order to account for this, we limit the range of the mechanism so that ∠ABC, ∠BCD, and ∠CDE never cross 180°. Under these conditions, the position of hinge C is uniquely determined by the motor angles, and similarly, the motor angles are uniquely determined by the hinge position. The restricted range of the mechanism is shown in Figure 2.7.

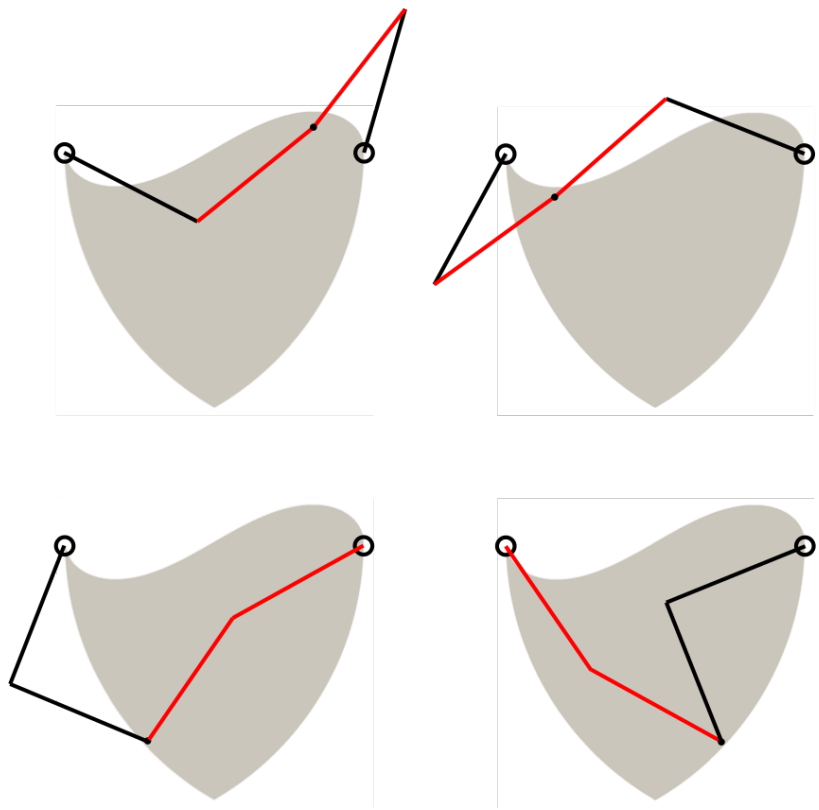


Figure 2.7: Range of motion of the five-bar linkage mechanism. The arms limiting the range of the mechanism in each region are highlighted in red.

To determine whether or not a point is within the range of the mechanism, a vector is first defined for each arm:

$$\overrightarrow{AB} = \langle x_B - x_A,\, y_B - y_A \rangle \qquad \overrightarrow{BC} = \langle x_C - x_B,\, y_C - y_B \rangle$$
$$\overrightarrow{CD} = \langle x_D - x_C,\, y_D - y_C \rangle \qquad \overrightarrow{DE} = \langle x_E - x_D,\, y_E - y_D \rangle$$

By taking the cross product between an arm's vector and the vector for the adjacent arm, we can characterize the angle formed at the hinge between the two arms. For example, $(\overrightarrow{AB} \times \overrightarrow{BC}) \cdot \hat{k} > 0$ and $(\overrightarrow{AB} \times \overrightarrow{BC}) \cdot \hat{k} < 0$ would indicate a counter-clockwise and clockwise turn, respectively, between arms AB and BC. Given the initial geometry of our linkage, the following three constraints define the range of our mechanism:

$$(\overrightarrow{AB} \times \overrightarrow{BC}) \cdot \hat{k} > 0 \qquad (\overrightarrow{BC} \times \overrightarrow{CD}) \cdot \hat{k} > 0 \qquad (\overrightarrow{CD} \times \overrightarrow{DE}) \cdot \hat{k} < 0$$

where $\hat{k}$ is the unit vector in the positive z direction.

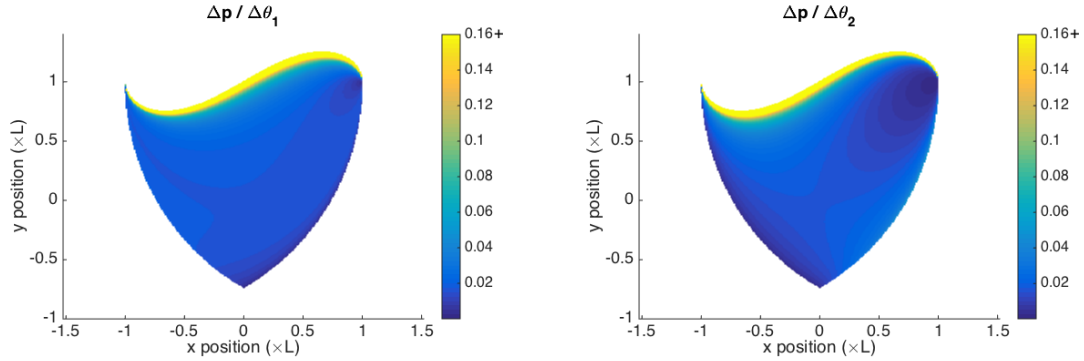
Error as a Function of Position

For certain regions in the range of the five-bar linkage mechanism, small changes in the motor angle can result in large changes in the position of the middle hinge. Knowing how error propagates through the system helps avoid regions in which error is most amplified.

For this mechanism, two different values were plotted over the entire range to represent the expected error at each point. $\frac{\Delta p}{\Delta \theta_1}$ and $\frac{\Delta p}{\Delta \theta_2}$ represent the magnitude of the change in position divided by the magnitude of the change in motor angle for motors 1 and 2, respectively.

The values for these plots were found using MATLAB. For each point within the range of the mechanism, we calculated values of θ_1 and θ_2 corresponding to that point. The angle of one motor was changed by a small amount (in this case 10^{-5} degrees), and a new position was calculated. The distance from the original position to the new position was divided by the change in motor angle and plotted over the range of the mechanism.

These values represent a numerical estimate of the partial derivative of position with respect to motor angle. We chose this method because it is faster than explicitly calculating the exact partial derivatives of x and y with respect to θ_1 and θ_2 and still provides accurate results.



(a) Error with Respect to Motor 1 (b) Error with Respect to Motor 2

Figure 2.8: Amplification of Error for the Five-Bar Linkage Mechanism

The plots in Figure 2.8 show that the region of highest error is at the top boundary of the range of the mechanism. Near these points, small errors in the angle of the motors result in large errors in position. The error near the origin is relatively small, as is the error near the boundary opposite the motor of interest.

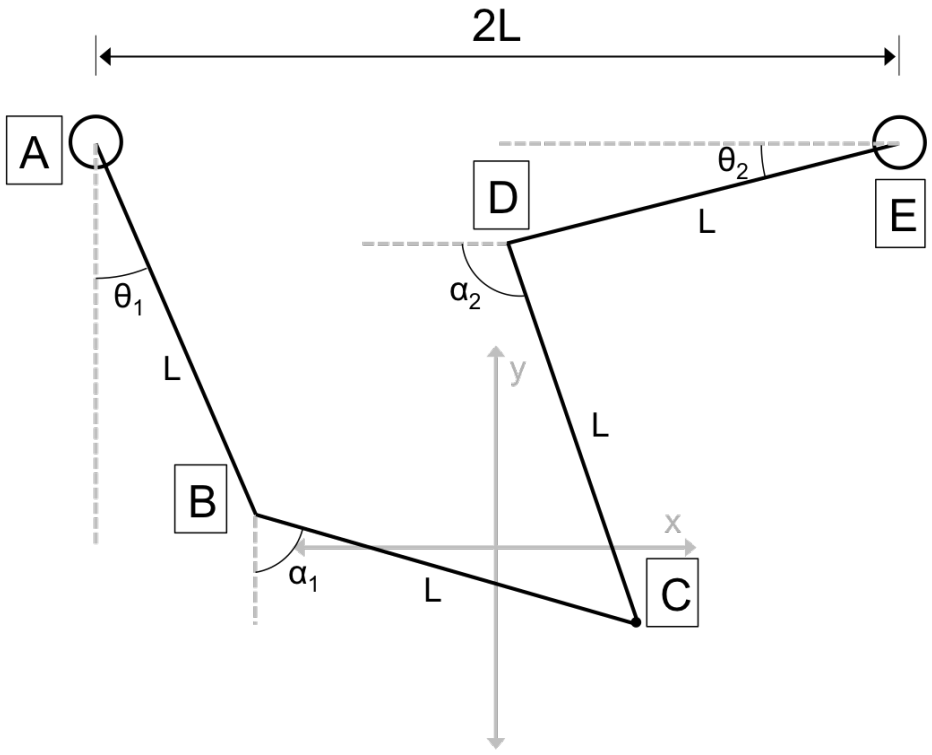


Figure 2.9: The geometry used in the moment of inertia derivation

2.3 Dynamics

Conservation of energy and momentum were not enough to determine the equations of motion for the system due to the unknown reaction forces on the pinned motors. Instead, after much algebraic manipulation, the study of the mechanism discarded Newtonian mechanics and transitioned to use of the Lagrangian method.

For this derivation we use two sets of angles: the exterior angles θ_1 and θ_2, and the interior ones, defined in Figure 2.9, termed α_1 and α_2. By continuity, the two sets are related by:

$$\sin(\theta_1) + \sin(\alpha_1) + \cos(\theta_2) + \cos(\alpha_2) = 2$$

$$\cos(\theta_1) + \cos(\alpha_2) = \sin(\theta_2) + \sin(\alpha_2)$$

Solving for α_1 and α_2 in terms of θ_1 and θ_2 explicitly is difficult, particularly because the system is not uniquely defined. For any set of θ values, there are two valid α coordinate pairs that correspond to the intersections of two circles.

To relate the derivatives of α_1 and α_2 to θ_1, θ_2, $\dot{\theta}_1$, $\dot{\theta}_2$, there are two equivalent routes–differentiating the position continuity equation or deriving a velocity continuity equation:

$$\dot{\theta}_1 \cos(\theta_1) + \dot{\alpha}_1 \cos(\alpha_1) - \dot{\theta}_2 \sin(\theta_1) - \dot{\alpha}_2 \sin(\alpha_2) = 0$$

$$-\dot{\theta}_1 \sin(\theta_1) - \dot{\alpha}_2 \sin(\alpha_2) = \dot{\theta}_2 \cos(\theta_2) + \dot{\alpha}_2 \cos(\alpha_2)$$

Solving for $\dot{\alpha}_1$ and $\dot{\alpha}_2$ yields:

$$\dot{\alpha}_1 = \frac{-\dot{\theta}_1 \cos(\alpha_2 - \theta_1) + \dot{\theta}_2 \sin(\theta_2 - \alpha_2)}{\cos(\alpha_1 - \alpha_2)}$$

$$\dot{\alpha}_2 = \frac{\dot{\theta}_1 \sin(\alpha_1 - \theta_1) - \dot{\theta}_2 \cos(\theta_2 - \alpha_1)}{\cos(\alpha_1 - \alpha_2)}$$

The Lagrangian, $\mathcal{L}$, of the system is the total kinetic energy, as there are no potential energy effects.

$$\mathcal{L} = \frac{I}{2}(\dot{\theta}_1^2+\dot{\theta}_2^2+\dot{\alpha}_1^2+\dot{\alpha}_2^2)+\frac{mL^2}{8}(5\dot{\theta}_1^2+5\dot{\theta}_2^2+4\dot{\theta}_1\dot{\alpha}_1\cos(\alpha_1-\theta_1)+4\dot{\theta}_2\dot{\alpha}_2\cos(\alpha_2-\theta_2)+\dot{\alpha}_1^2+\dot{\alpha}_2^2)$$

The Lagrangian is related to the input torque by the equations:

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial \mathcal{L}}{\partial \dot{\theta}_1}\right) - \frac{\partial \mathcal{L}}{\partial \theta_1} = \tau_1 \qquad \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial \mathcal{L}}{\partial \dot{\theta}_2}\right) - \frac{\partial \mathcal{L}}{\partial \theta_2} = \tau_2$$

Differentiating by the angular velocity and then time yields:

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial \mathcal{L}}{\partial \dot{\theta}_1}\right) = \ddot{\theta}_1 J_1 \qquad \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial \mathcal{L}}{\partial \dot{\theta}_2}\right) = \ddot{\theta}_2 J_2$$

Where J_1 and J_2 are defined as:

$$J_1 = I\left(1 + \frac{\sin^2(\alpha_1 - \theta_1) + \cos^2(\alpha_2 - \theta_1)}{\cos^2(\alpha_1 - \alpha_2)}\right) + \frac{mL^2}{8}\left(10 + 2\left(\frac{\sin^2(\alpha_1 - \theta_1) + \cos^2(\alpha_2 - \theta_1)}{\cos^2(\alpha_1 - \alpha_2)}\right) - 8\left(\frac{\cos(\alpha_1 - \theta_2)\sin(\alpha_2 - \theta_1)}{\cos(\alpha_1 - \alpha_2)}\right)\right)$$

$$J_2 = I\left(1 + \frac{\sin^2(\alpha_2 - \theta_2) + \cos^2(\alpha_1 - \theta_2)}{\cos^2(\alpha_1 - \alpha_2)}\right) + \frac{mL^2}{8}\left(10 + 2\left(\frac{\sin^2(\alpha_2 - \theta_2) + \cos^2(\alpha_1 - \theta_2)}{\cos^2(\alpha_1 - \alpha_2)}\right) - 8\left(\frac{\cos(\alpha_2 - \theta_1)\sin(\alpha_1 - \theta_2)}{\cos(\alpha_1 - \alpha_2)}\right)\right)$$

The next step in a Lagrangian analysis would be differentiating $\frac{\partial \mathcal{L}}{\partial \theta}$, producing the centrifugal terms in the equations of motion. However, by dimensional analysis, it becomes obvious that these terms will all contain terms of $\dot{\theta}_1^2$, $\dot{\theta}_2^2$, and $\dot{\theta}_1\dot{\theta}_2$. For a linearized analysis where the velocities are assumed to be small, these higher order terms can be neglected. Under this simplifying assumption, the full dynamics of the system become two straightforward equations:

$$J_1\ddot{\theta}_1 = \tau_1 \qquad J_2\ddot{\theta}_2 = \tau_2$$

The completion of this modeling allowed us to proceed to designing the control laws.

2.4 Controls

The first step in modeling the system was implementing a proportional control loop in Simulink. Although we knew the control would be non-optimal, it allowed us to test the reliability of the motors and encoders. We discovered that simple proportional control was inadequate, as it led to oscillations in the motor response. We improved this behavior by using a low-pass derivative filter to implement PD control. These filters, while more noisy than an estimator, had the benefit of being blind to the model. An asymptotically stable controller was found after tinkering with the gains for PD control. Finally, in an effort to remove any steady state error, an integrator was added into the Simulink model, allowing for full PID control. The code can be seen in Appendix B.

For a more refined control scheme, we took advantage of the symmetry of the system to derive a control law for a single motor. With the exception of certain degenerate cases, θ_1 and θ_2 are independent. Using the model for a DC motor provided

in Professor Rowley's lab notes for MAE 433B [4], the state space model is:

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix}\theta\\ \dot{\theta}\end{bmatrix} = \begin{bmatrix}0 & 1\\ 0 & -K_1\end{bmatrix}\begin{bmatrix}\theta\\ \dot{\theta}\end{bmatrix} + \begin{bmatrix}0\\ K_2\end{bmatrix} u$$

$K_2 = \frac{K_t}{JR}$ and $K_1 = K_e K_2$. K_t, K_e, and R are the torque constant, the EMF constant, and the resistance of the motor, respectively. J is the effective moment of inertia, which is a highly non-linear function of θ_1 and θ_2. For the purposes of design, the state space was linearized about the origin point, $\theta_1 = 0$ and $\theta_2 = 0$, where $J_1 = J_2 = 2I + \frac{3ml^2}{2} = 0.177\text{kgm}^2$. The actual variation in J can be seen in Figure 2.10. At degenerate pairs of angles as demonstrated in section 2.2.3 the denominator in the J formula, $\cos(\alpha_1 - \alpha_2)$, goes to 0. This creates a singularity in J. However, in the normal space of operation around the origin, the value of J ranges from $\approx 0.07\text{kgm}^2$ to $\approx .248\text{kgm}^2$. Thus, when designing a controller, we consider a range of systems as discussed in Doyle, Francis, and Tannenbaum [8].

The transfer functions using these J values are as follows:

$$P_{avg}(s) = \frac{0.01667}{s^2 + (1.278 \times 10^{-4})s}$$

$$P_{min}(s) = \frac{0.04164}{s^2 + (3.19 \times 10^{-4})s}$$

$$P_{max}(s) = \frac{0.04164}{s^2 + (5.11 \times 10^{-5})s}$$

The Bode plot of these plants is found in Figure 2.11.

As evident in Figure 2.11, the bandwidth of all three systems, which differ only in the placement of one pole, is approximately 10^{-1} radians per second and the phase margin is very small. Our two priorities for the system were stability and tracking, so the task of designing a controller became one of optimizing these quantities. For this task, the obvious choice was a PD controller of the form $C(s) = k_d s + k_p$. The effects of various controllers can be seen in Figure 2.12. By placing a zero at $\frac{-k_p}{k_d}$, the slope of the magnitude plot became more shallow and the phase increased. During initial testing, the reduced steady state error provided by integral feedback was deemed unnecessary for the purposes of the mechanism. In hindsight, the additional pole and zero from a PID controller would allow for more detailed shaping of the loop function but was not applied in this project, leaving it for future experimentation. Through physical testing, the k_p–k_d pair of 200 and 10 was found to provide the best performance, despite having the same crossover frequency as the other tested controllers and a

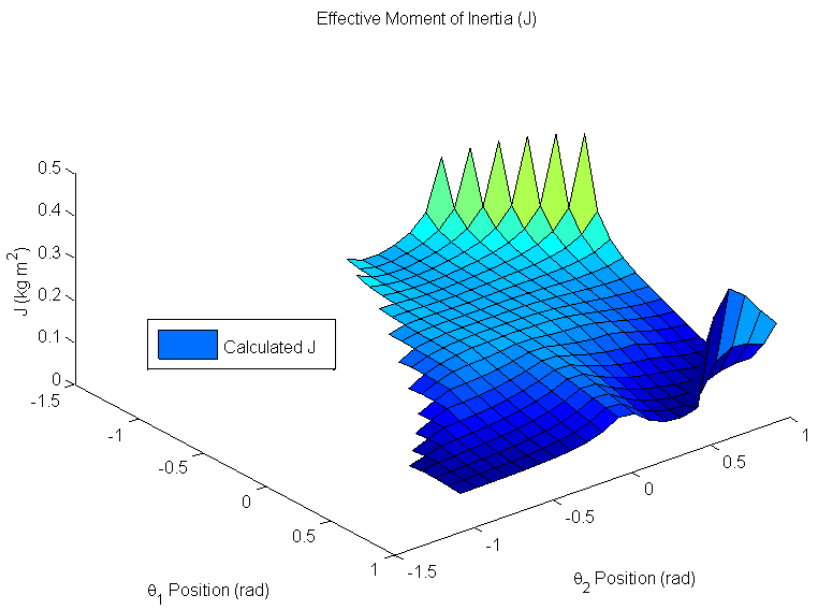


Figure 2.10: The variation in the effective moment of inertia

smaller phase margin. The near-instability this causes would be troublesome for an idealized system. However, the friction of the bearings, the slight slop in the arms, and other non-linearities counteracted this tendency towards oscillation, leaving just the benefit of increased rise time. LQR controllers were tested with this model but performed considerably worse, as the framework of LQR places a higher priority on removing overshoot. While important in this context, overshoot held a lower priority than other factors. The performance of LQR controllers with various weights can be seen in Figure 2.13. In the end, this analysis directed the testing approach described later, but its particular insights were sometimes proven overly idealized for such a complicated system.

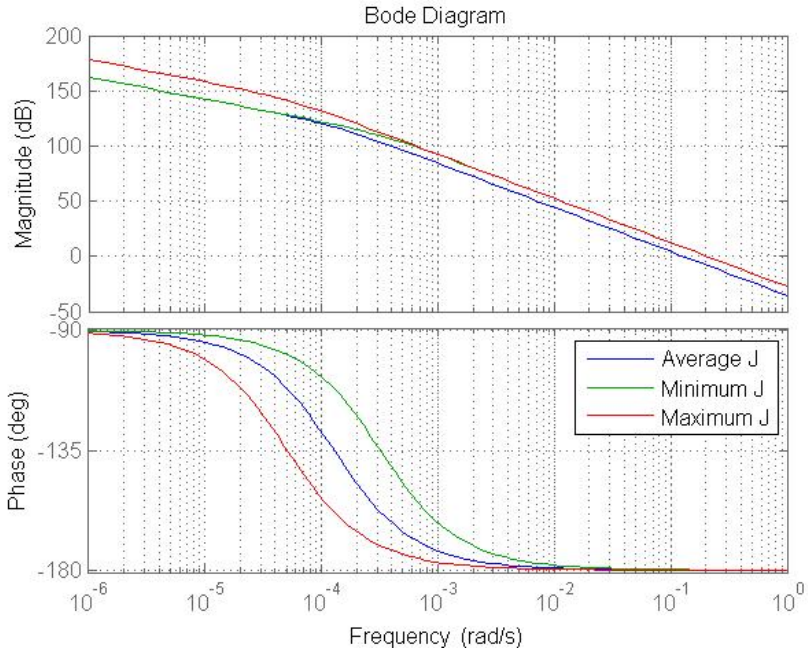


Figure 2.11: The Bode plots of the plant with different values of J

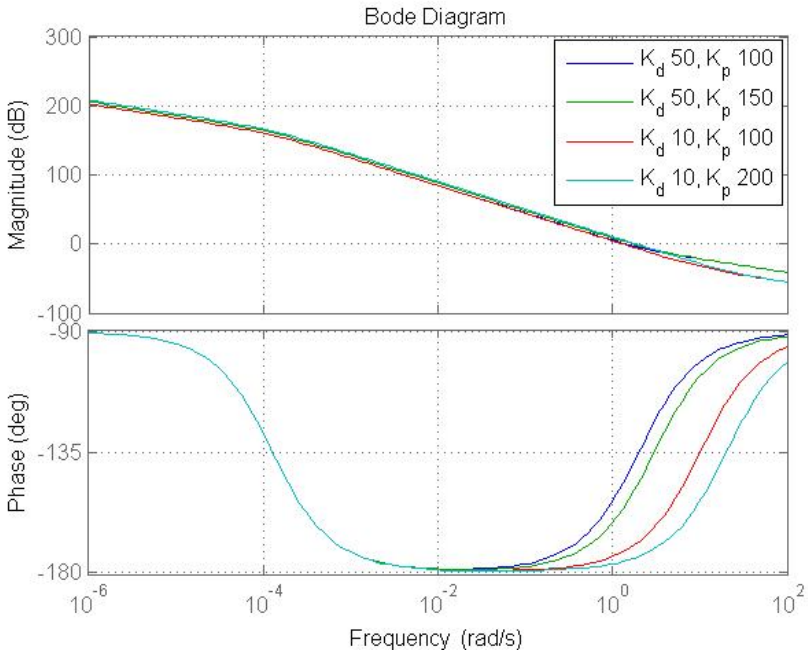
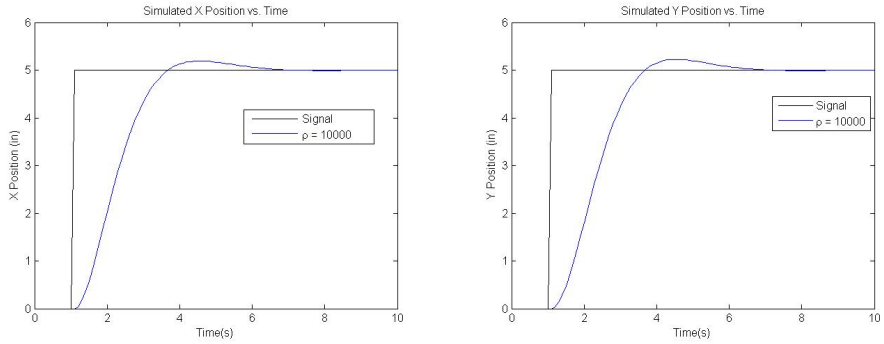
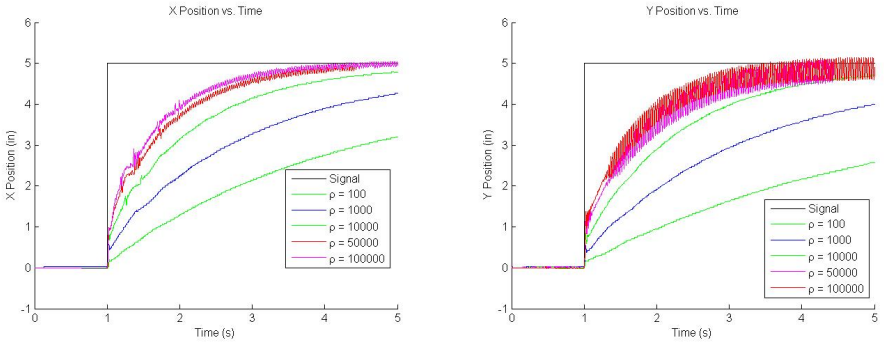


Figure 2.12: Bode plots of the average plant with different controllers



(a) Simulated step response of the LQR controllers



(b) Experimental data from LQR controllers

Figure 2.13: The results of the attempt at LQR control. High values of ρ yield better rise times, but require such high voltages that the motors saturate and shake. ρ is the relative weight, Q/R.

Chapter 3

Design of the Arduino Rack and Pinion System

3.1 System Overview

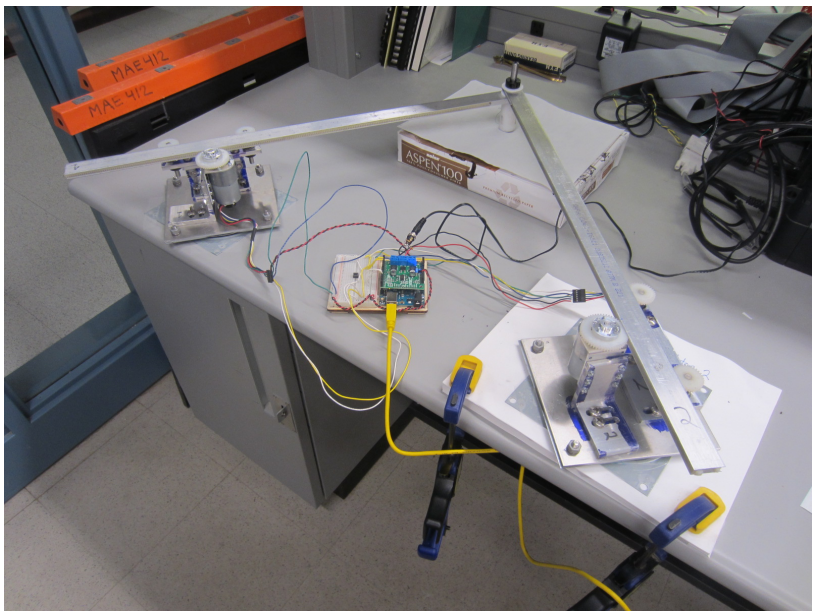


Figure 3.1: Arduino Rack and Pinion System

The rack and pinion mechanism provides two degrees of freedom and is designed for accurate position control. This invention consists of two motors, each of which controls the linear position of one arm via a gear rack and pinion. The arms are

joined at a hinge, which together with the two motors forms a triangle. The position of the hinge is determined by its distance from each motor. The motors are mounted to turntables, giving them the ability to rotate passively with the system's motion. Instead of using rotational motion as the Quanser five-bar linkage does, the rack and pinion mechanism converts the rotational motion to linear translation. In addition, the rack and pinion mechanism is powered by an Arduino, making it portable and increasing flexibility in coding methods.

3.2 Mechanical System

3.2.1 Design Concept

We developed the rack and pinion mechanism with the goals of the five-bar linkage in mind, but with the added goal of improving accuracy for recording and playing back motion. The pen mount supports the hinge, which negates the effects of bending due to gravity. We developed the idea from scratch and chose the gear rack due to its ability to be accurately controlled. In addition, the error in the mechanism is only dependent on the position accuracy of the motors and not the size of the mechanism itself. The use of Arduino allows us to design our own control system and use third party motors and components.

3.2.2 Physical Design

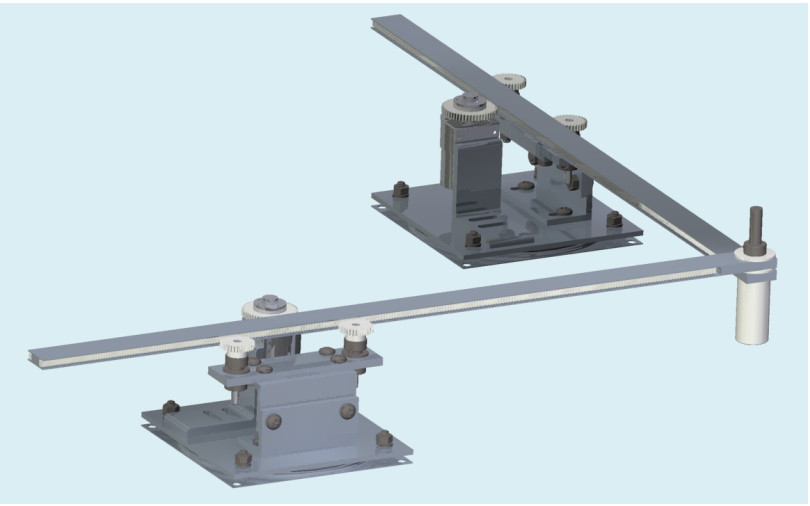
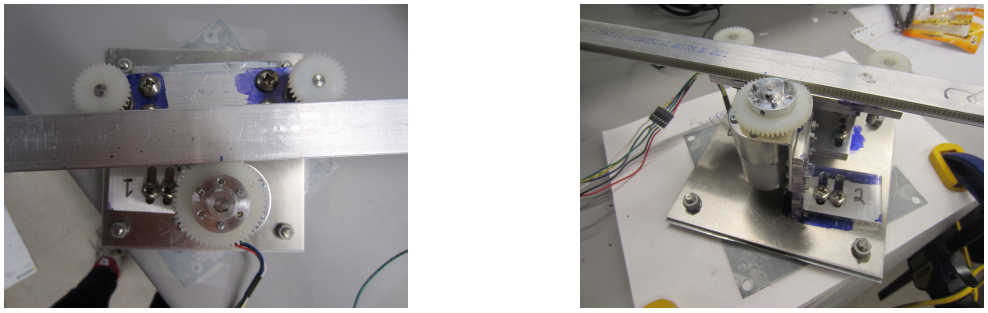


Figure 3.2: Creo Model of the Rack and Pinion Mechanism

The mechanism's arms are made out of milled aluminum with inset nylon gear racks secured with epoxy. Each rack is sandwiched between three gears, two of which rotate freely, and one of which is driven by a gear motor. The mounts, shown in Figure 3.3, are centered on turntables, allowing them to rotate with the motion of the pen.



(a) Gear rack motor mount viewed from top (b) Gear rack motor mount viewed from side

Figure 3.3: Views of the Motor Mounts

This system allows for two degrees of freedom with a range of accurate motion that easily covers a standard sheet of paper, which is ideal for its use as an Autopen. Each arm is two feet long, one inch wide, and 5/16 inches thick with a 3/16 inch channel milled into each side. The choice to machine the racks out of one piece of aluminum stemmed from the difficulty in machining three pieces and sandwiching them together. This solution provided more accuracy in machining and reduced the number of components.

The two gear rack arms are connected by a joint, shown in Figure 3.4, with a steel shaft that is press fit into one arm and loosely fit into the other. The shaft has a hole milled through the center that leaves room for a spring-loaded ballpoint pen nib. The bottom of the joint is a Delrin cylinder with a hole milled through the center large enough to let the pen tip through but small enough so that the pen is held securely inside the steel shaft. Two PTFE washers provide lubrication, and a shaft collar holds the arms together. Many joint options were discussed, but this one was by far the simplest, as it required no bearings and minimal machining.

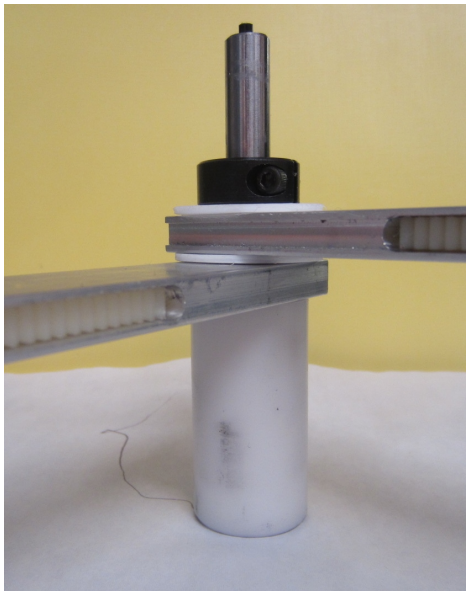


Figure 3.4: Joint of the Rack and Pinion Mechanism

The arms are supported by the Delrin cylinder, which negates the loading due to gravity. This prevents the arms from relying entirely on the motor mounts for support. The gear rack is designed to derail after a pen breaches the bounds of the range of motion, ensuring that the motor is not damaged. The final setup of the rack and pinion mechanism is shown in Figure 3.1.

3.2.3 Geometry and Range of Motion

Figure 3.5 shows the geometry of the rack and pinion mechanism. L_1 and L_2 represent the distance from each motor to the pen. α is the angle between the left arm and the horizontal, β is the angle between the right arm and the horizontal, and γ is the angle formed by the two arms. The origin of the coordinate system represents the location of the pen when $L_1 = L_2 = L_0$, where L_0 is the distance that causes the arms to form a right angle. d is the spacing between the motors, which is equal to $L_0\sqrt{2}$.

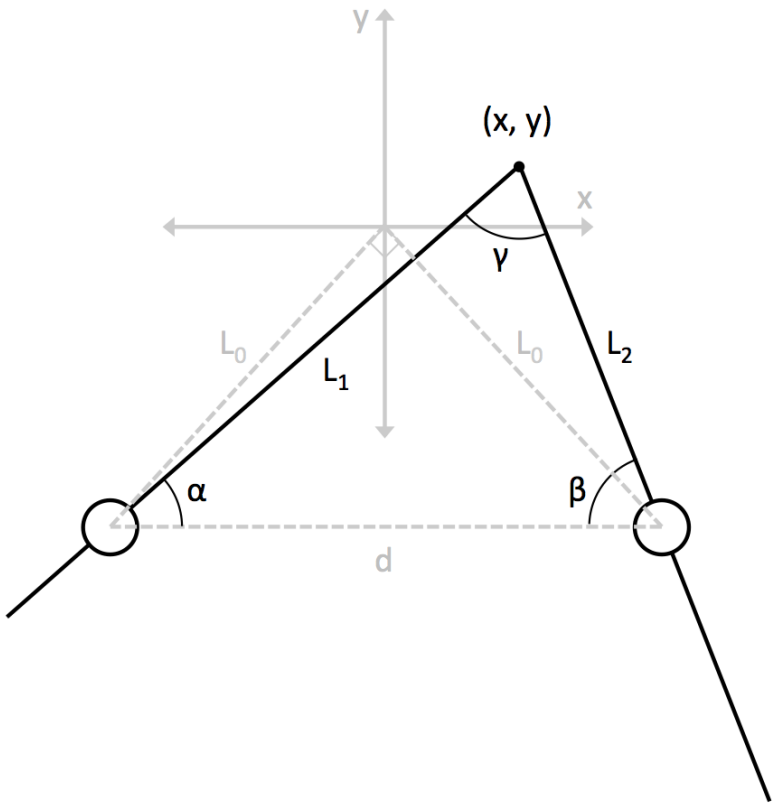


Figure 3.5: Geometry of the Rack and Pinion Mechanism

Converting Between Arm Lengths and Pen Position

The position of the pen is controlled by changing lengths L_1 and L_2. The relationship between the lengths (L_1, L_2) and the position of the pen (x, y) can be found using basic trigonometry.

The conversion from arm lengths to pen position is as follows:

$$\alpha = \cos^{-1}\left(\frac{L_1^2 + d^2 - L_2^2}{2dL_1}\right)$$

$$x = L_1\cos(\alpha) - \frac{d}{2} \qquad y = L_1\sin(\alpha) - \frac{d}{2}$$

The conversion from pen position to arm lengths is as follows:

$$\alpha = \tan^{-1}\left(\frac{d+2y}{d+2x}\right) \qquad \beta = \tan^{-1}\left(\frac{d+2y}{d-2x}\right)$$

$$L_1 = \frac{d+2x}{2\cos(\alpha)} \qquad L_2 = \frac{d-2x}{2\cos(\beta)}$$

The length of each arm is controlled by a gear of diameter D that rides along the arm's gear rack. In order to control the position of the pen, the length of each arm is changed by rotating the gear to a certain angle, θ_i. The relationship between the gear angle and the arm length is given below for both arms.

$$L_1 = L_0 + \theta_1 \frac{D}{2} \qquad L_2 = L_0 + \theta_2 \frac{D}{2}$$

In order to control the angle of the gear using the motor, the angle is converted to encoder counts, c_i. There are 477 counts in one revolution of the gear motor shaft, so the relationship between θ_i and c_i is as follows.

$$c_1 = \theta_1 \frac{477}{2\pi} \qquad c_2 = \theta_2 \frac{477}{2\pi}$$

Range of Motion

To ensure that the pen position is uniquely determined by lengths L_1 and L_2, the y-position of the pen is constrained to the area above the two motors. For the geometry specified in Figure 3.5, this means that the y-position of the pen must always be greater than $\frac{-L_0\sqrt{2}}{2}$. The x-position of the pen is limited by the maximum lengths of arms 1 and 2, L_{max}, so the left and right boundaries are formed by two circular arcs centered on the motors. The point at which the arcs meet represents the farthest point in the mechanism's range. This range is shown below in Figure 3.6.

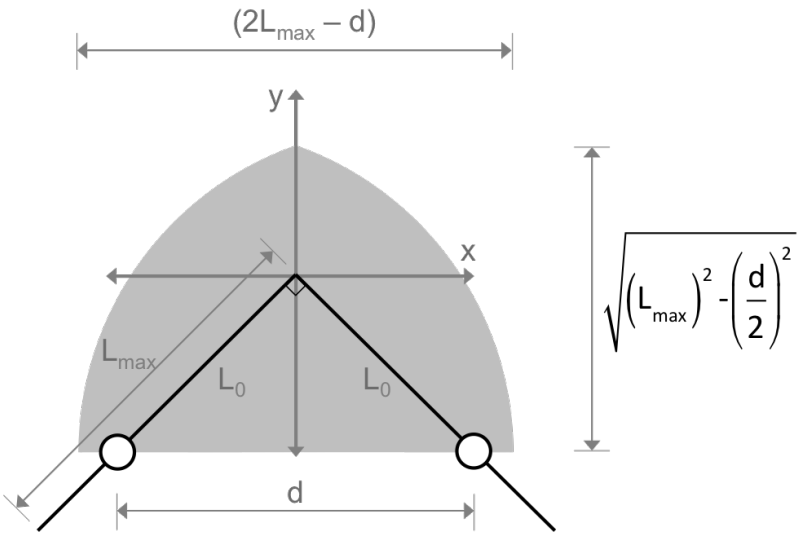


Figure 3.6: Range of Motion of the Rack and Pinion Mechanism (Shaded in Gray)

Error as a Function of Position

To quantify the error of this mechanism, a similar approach to that of the five-bar linkage was used, as shown in section 2.2.3. The partial derivative of the pen position with respect to the lengths of each arm was found numerically using MATLAB, and the values were plotted over the range of the mechanism. One important difference is that for this mechanism, the partial derivative values are non-dimensional whereas for the previous mechanism they were expressed in units of length.

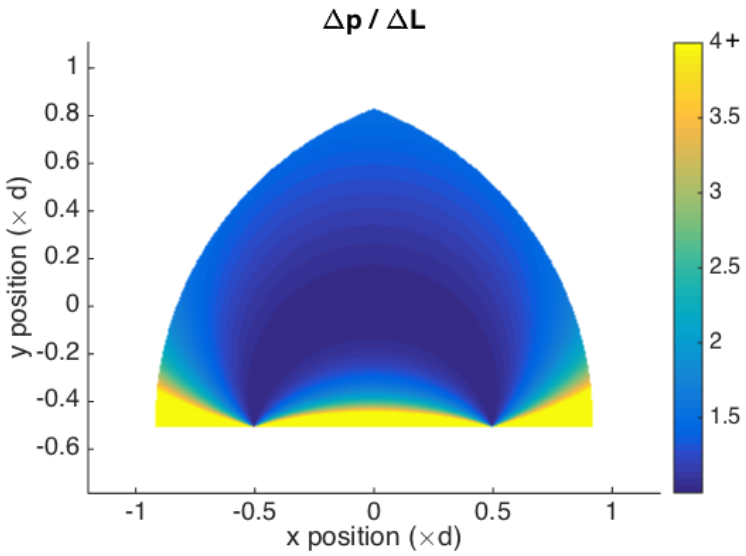


Figure 3.7: Amplification of Error for the Rack and Pinion Mechanism

As shown in Figure 3.7, the error in the position for this mechanism is smallest when a right angle is formed by the arms. At these points, which form an arc through the origin, the non-dimensional error is equal to one. The highest error values occur near the bottom of the range. This plot is useful because it shows that in order to have the most accurate position control, the angle formed by the arms should be kept as close to 90° as possible.

3.3 Dynamics

As with the five-bar linkage, the dynamics of this mechanism can be derived by finding the effective moment of inertia. We started by rotating one motor while keeping the other stationary. The inertia is divided into contributions from the near moving apparatus and the far stationary apparatus. The components of the near inertia are

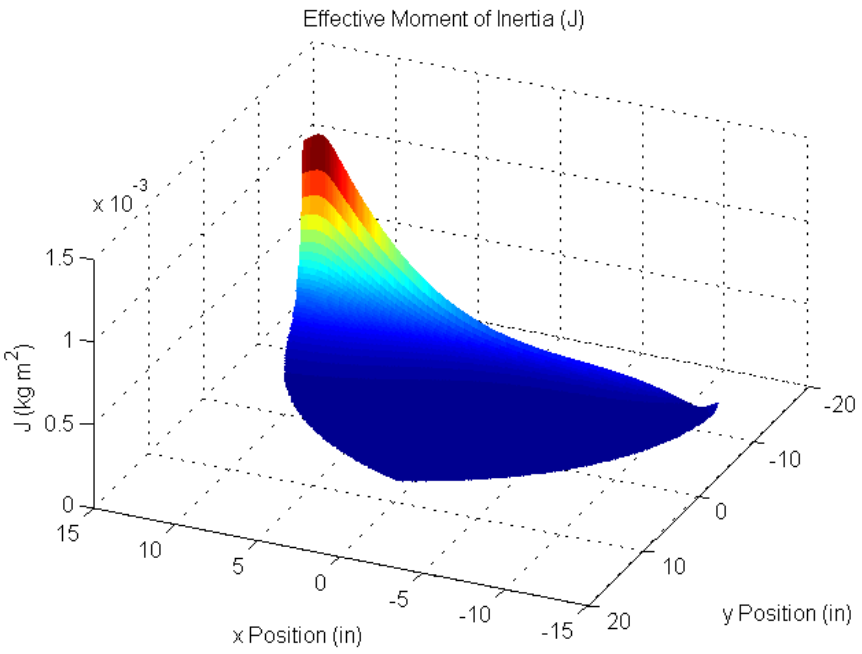


Figure 3.8: The effective moment of inertia for the left motor. The J for the right motor would be flipped about the centerline, due to the symmetry of the system.

the gear (I_G) and the translation of the near arm ($m_A(\frac{D}{2})^2$). In the far apparatus, there is only rotational motion, so the only components are the moments of inertia of the turntable apparatus (I_S) and the arm, accounting for its rotation around a point other than its center ($I_A + m_A(L - \frac{L_{max}}{2})^2$). The rotation rate in the far turntable is not the same as that of the spinning motor, but they are related by a factor of $\frac{D\sin(\gamma)}{2L}$. In this analysis, L is the distance from the pen holder to the far turntable center, L_{max} is the total length of the arm, and γ is the angle between the two arms at the pen holder (equivalent to $\frac{\pi}{2} - (\alpha + \beta)$). Combining these pieces yields:

$$J_{near} = I_G + m_a(\frac{D}{2})^2 + \frac{D}{2L\sin(\gamma)}(I_S + I_A + m_A(L - \frac{L_{max}}{2})^2)$$

For this specific mechanism, the J is calculated at the origin to have a value of $1.035 \times 10^{-4}\,\mathrm{kgm}^2$. A plot of J over the range of the motion is found in Figure 3.8. The plot is relatively flat except for its singularity at the far motor. Because of this feature, the linearized model is considered to hold for the typical range of motion.

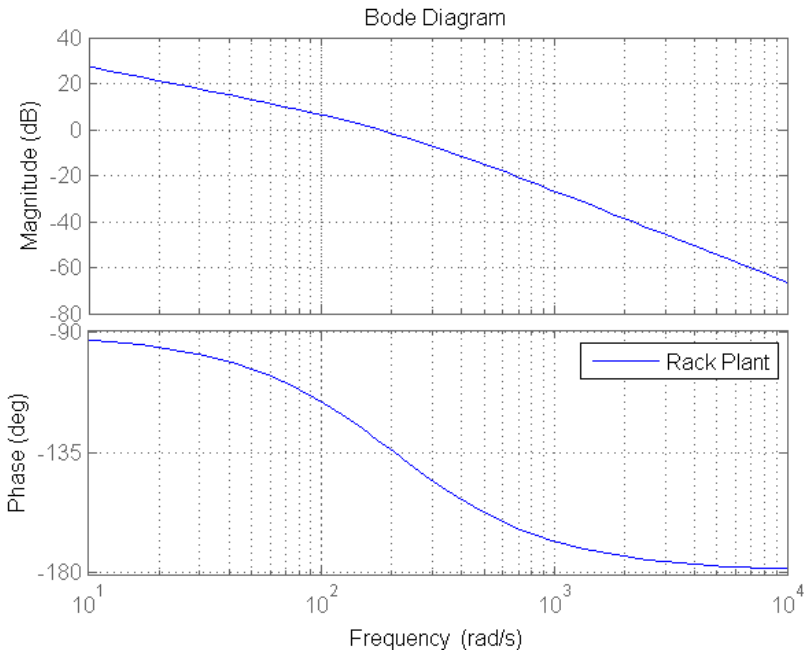


Figure 3.9: Plant Transfer Function for the Rack and Pinion Mechanism

3.4 Controls

With the dynamics quantified, the modeling of a control system for the rack mechanism followed the same pattern as the linkage. Using the motor constants and the new J value, the system still maintained the form

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix}\theta\\ \dot{\theta}\end{bmatrix} = \begin{bmatrix}0 & 1\\ 0 & -K_1\end{bmatrix}\begin{bmatrix}\theta\\ \dot{\theta}\end{bmatrix} + \begin{bmatrix}0\\ K_2\end{bmatrix}u$$

with $K_1 = 204.7$ and $K_2 = 8.236$. In this model, the transformation between θ and encoder counts is incorporated into the C matrix, so that the state space is still kept in radians. The transfer function of the plant, plotted in Figure 3.9, is

$$P(s) = \frac{4.747 \times 10^4}{s^2 + 204.7s}$$

As with the five-bar linkage, PD control was used to place a zero before the crossover frequency in order to increase the phase margin and bandwidth. The resulting plots of the control system are found in Figure 3.10.

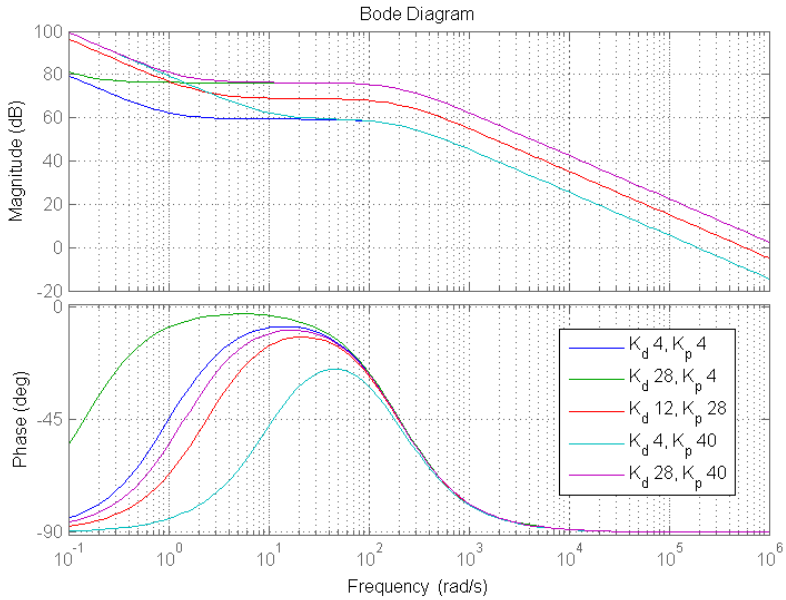


Figure 3.10: The different controllers applied to the rack mechanism plant. Notice the large increase in bandwidth and phase margin.

3.5 Arduino Interface

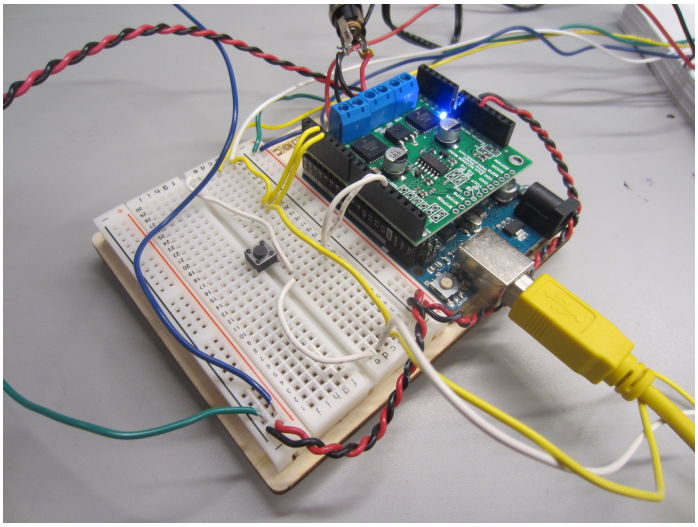


Figure 3.11: Arduino Circuit

We selected an Arduino for three reasons. First, a microcontroller provides portability and minimal power drain. Second, Arduino motor shields are more readily available than motor shields for other microcontrollers and microprocessors. Third, the Arduino made it easy to use third party motors.

3.5.1 Arduino-Motor Interaction

Using the Dual MC33926 Motor Shield and complementing Arduino library facilitated interactions between the Arduino and the two motors. This enabled direct port reading of the motor encoders and simplified writing of PWM signals to the motors. Attaching the motors' Hall effect sensor encoders to the two falling-edge interrupt pins while maintaining direct read on the secondary sensor pins allowed us to accurately control the motor speed while rapidly and precisely updating information about the orientations of the motors. This design was necessary because interfacing an Arduino with motors requires manual position encoding from the raw signal output of the motors' sensors [5].

3.5.2 Coding the Arduino

Using a Deterministic Finite Automaton (DFA) of states, the Arduino system distinguishes between two primary states: gathering data from the serial port and updating and outputting the motor's position.

While in the data collection state, the Arduino continuously gathers the position data that is sent via USB to the Arduino's serial port. With the serial port's maximum buffer of 64 characters and the Arduino's limited $2\,kb$ of memory, which can only maintain 350 lines of input or 700 integers corresponding to input motor counts, the maximum rate of data intake is severely limited. Because of these space limitations, our system requires a peripheral C program that pauses every 350 input points and waits for the Arduino to draw the portions that have already been sent.

When the Arduino receives the termination string or reaches the maximum capacity of data points, the information transfer pauses and the state transitions. Before the data transfer resumes, the motors track the loaded trajectory, clearing space for more points. The code uses two independent PD loops to control the counts of each motor. In the update loop, our control algorithm implements discrete time PD control. The code can be seen in Appendix D.

3.5.3 Implementing Control

To ensure that PD updates take place at approximately consistent intervals despite inconsistent delays that might occur due to the serial writing of data or encoder interrupts, we defined a set of constants: the frequency at which PD updates ($LT \geq 10ms$), the frequency at which current position data writes to the serial port ($TSO \geq$

$10ms$), and the frequency at which the reference position updates ($TSI \geq 30ms$). LT was selected to provide as close a simulation of continuous PD control as possible. TSO was selected to minimize the delay of LT while maximizing the frequency of the output. TSI was selected to coincide with the settling time of each point.

Beginning initially with PD control of a single motor's speed based on the motor shield documentation [5], we expanded the system to handle two motors simultaneously. Next, we modified the program to work with PD position control rather than speed control and implemented a position-reading interface on the Arduino that received information over the Serial Monitor. After writing a C program that would automatically output time-delayed CSV count data over the serial interface [3], we optimized LT, TSO, and TSI. We found $K_p = 28$, $K_d = 12$ to be the most promising constants for PD control on our given LT timescale. Further discussion on experimentation is discussed in Chapter 4.

Chapter 4

Testing Methods and Results

4.1 Preliminary Testing of the Five-Bar Linkage

After the completion of the five-bar linkage mechanism, some K_p–K_d pairs were tested with simple step function inputs. The results in Figure 4.1 demonstrate the following characteristic trends, which informed our choices of K_p and K_d. Increasing K_p provided faster rise time and lower steady state error. At K_p values that were too high, the motor would shake as shown in the LQR tests in Figure 2.13. Increasing K_d reduced overshoot and oscillations. Decreasing K_d provided the fastest rise times but caused a more oscillatory response.

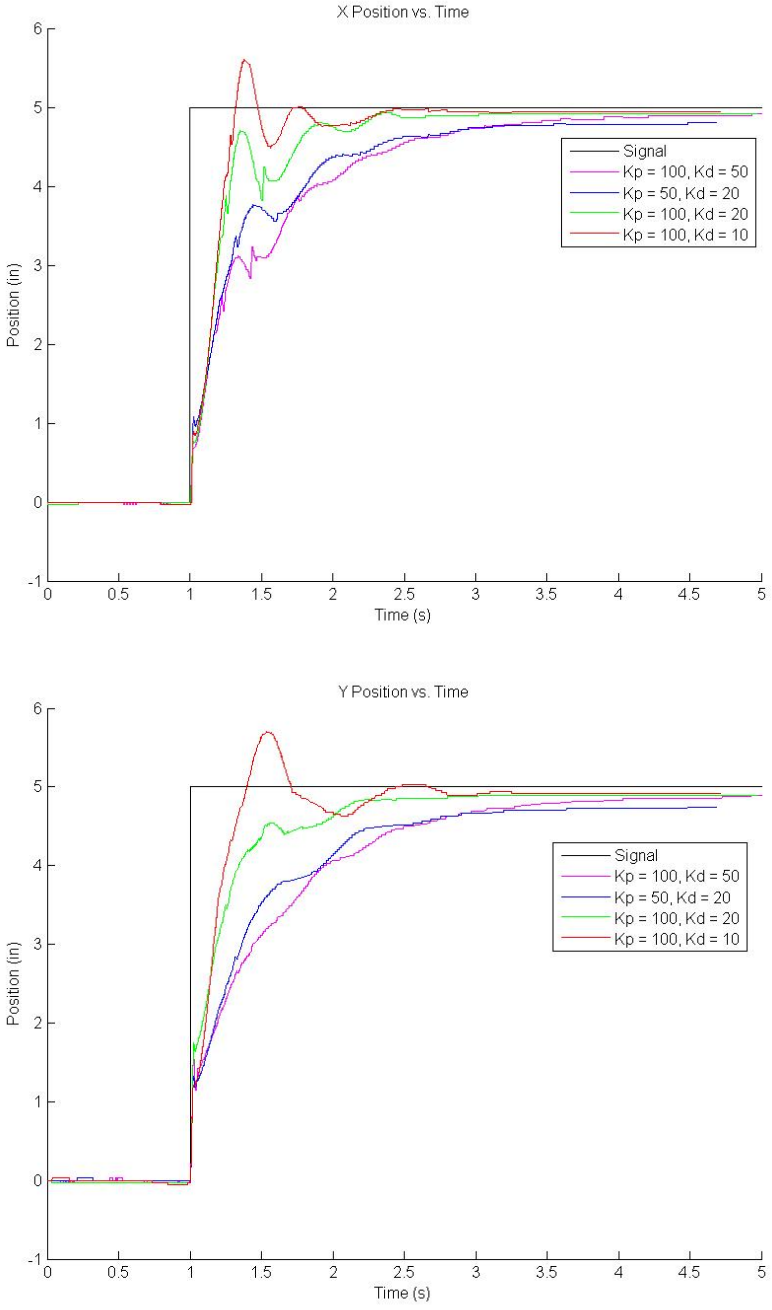


Figure 4.1: The step response of the Quanser system to a reference of $x = 5$, $y = 5$. These runs demonstrate the trends associated with changing K_p and K_d.

4.2 Preliminary Testing of the Rack and Pinion System

After the early trials of the rack and pinion system, two key observations informed our K_p and K_d selection. First, the output signals were constrained to the 400 to -400 range (corresponding to maximum and minimum PWM signals). Second, at lower inputs, the motors were subject to stick friction. These two concepts are demonstrated in Figure 4.2:

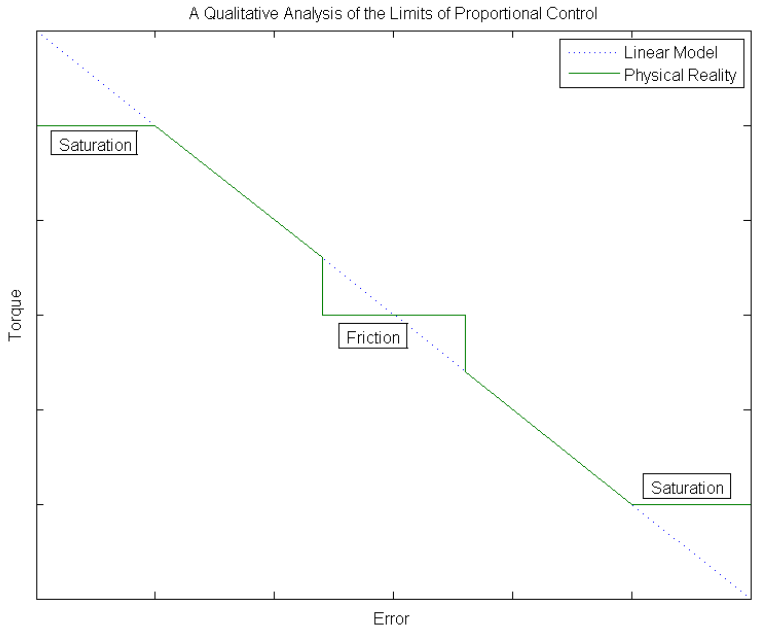


Figure 4.2: Unitless Diagram of Motor Saturation/Stick Friction

A control law that is accurately reflected by motor outputs requires a K_p value that never causes saturation and yet results in minimal sticking. Thus, we conceived of the concept of a "saturation frontier." Found experimentally from the circle trajectory (see section 4.3), this frontier separates the pairs of K_p and K_d that cause the system's output signal to saturate from those that do not. This frontier later guided our choice of K_p–K_d. The saturation frontier is shown in Figure 4.3.

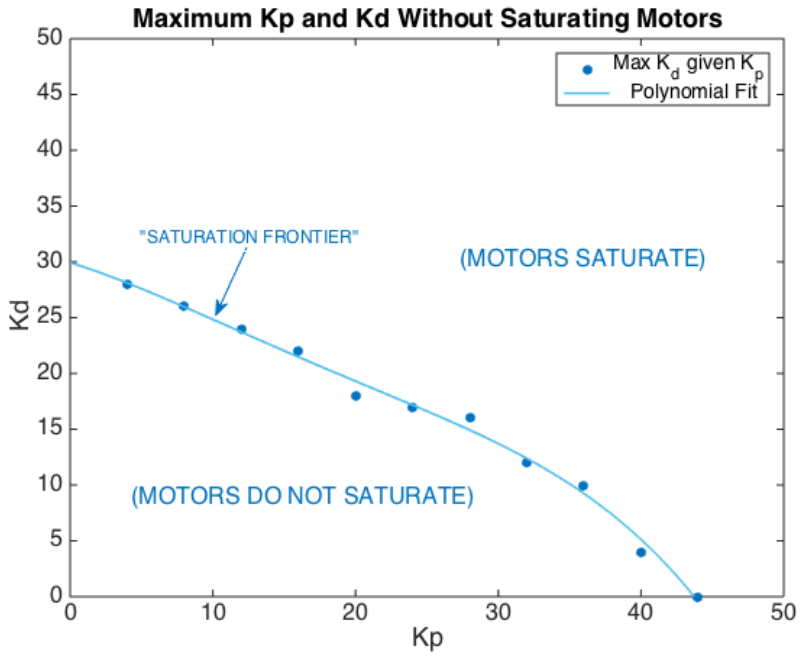


Figure 4.3: Saturation Frontier for K_p and K_d Values

For a series of K_p–K_d pairs on the saturation frontier, we calculated the Root Mean Square error between the output trajectory and the idealized circle. With the RMS values shown in Table 4.1, we had a basis of comparison to conclude that $K_p = 28$, $K_d = 18$ was the most accurate PD tracking combination on the frontier that would not lead to an output speed saturation. We selected $K_p = 28$, $K_d = 12$ as our ideal gains in order to provide a slight buffer against saturation.

Table 4.1: Root Mean Square Error of a Circular Trajectory on the Saturation Frontier

K_p	K_d	RMS Error (inches)
4	28	0.0563
12	24	0.0200
20	18	0.0119
28	16	**0.0089**
36	10	0.0092
44	0	0.0104

The serial interface provided communication between our Arduino and a computer. Initially, the Serial Monitor that comes bundled with the Arduino IDE was

sufficient to examine individual points and ensure that the code of our Arduino-motor system worked as desired. Using the code from TodBot [3] as a baseline and modifying the end user code to suit our particular Arduino needs, we developed a means of sending entire CSV files to the Arduino from a UNIX based command prompt. To automate testing of the system's shape-drawing capabilities in their entirety, we would need to design a computer program that could interface with the Arduino serial port.

4.3 Experimental Procedure

After both mechanisms were completed, the following tests were performed in order to determine their relative performance. For a variety of gains, the mechanisms were fed a trajectory of points in the following shapes:

A Square

The first trajectory was a square generated in MATLAB by `squaregen.m` (found in Appendix C). For the five-bar linkage mechanism, the points defining the square were intentionally sparse to allow for some settling between points. This allowed us to see the step response of each control law in a repeatable fashion. For the rack and pinion mechanism, the points defining the square had to be more densely packed because the gains chosen for the Arduino controller responded to the sparse points with too much overshoot.

A Circle

The second trajectory allows for a direct comparison between the two mechanisms. Both mechanisms were fed a six inch diameter circle consisting of 200 points. This served as a test of each mechanism's ability to perform smooth, constant motions. Because the mechanism is constantly moving, this test places more significance on rise time than on steady-state behavior.

An Organic Shape

This organic trajectory was recorded on the linkage mechanism in Quanser, taking one data point every 0.001 seconds. The points followed no general pattern and formed several intersecting loops. This test is the closest to the intended use case of these

mechanisms, which is copying human-generated motions. By testing this path, we examined the fidelity of each mechanism to both small and large scale details. For the rack and pinion mechanism, the shape was scaled down by a factor of two to fit on a piece of printer paper.

A Fast Shape

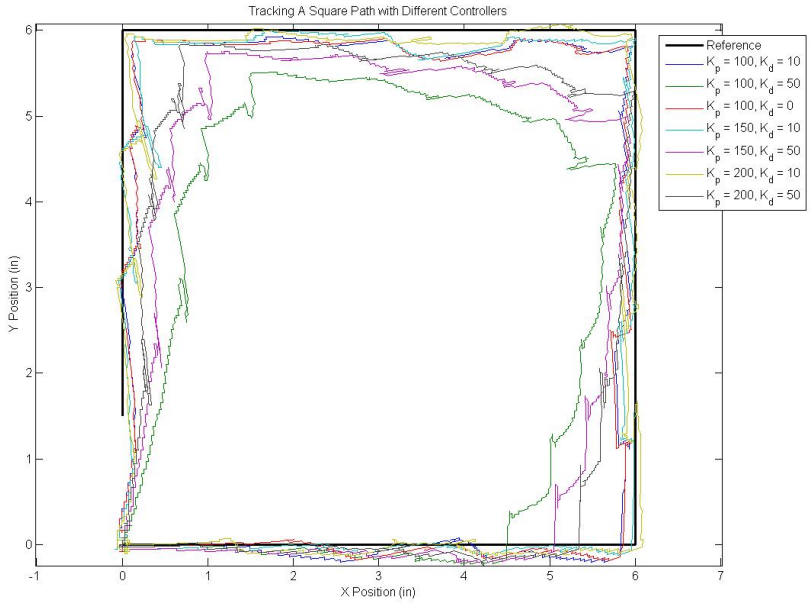
The final test is another hand-recorded shape from the Quanser system, but the arm was moved intentionally quickly, providing a time series that would be more difficult for the motors to copy. This allowed us to see the behavior of the two systems at the extremes of their capabilities. As in the case of the organic shape, the fast shape was scaled down by a factor of two for the rack and pinion mechanism.

4.4 Results of Testing

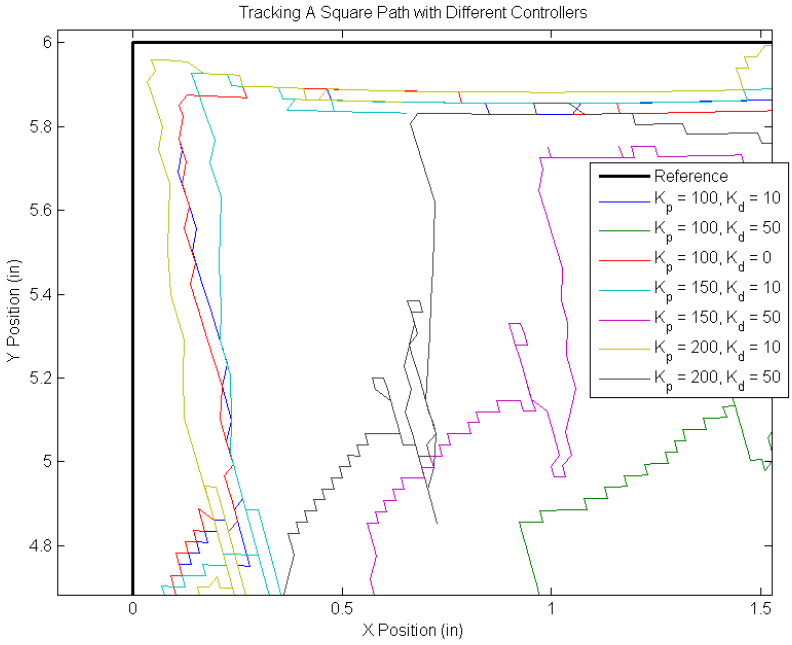
4.4.1 Quanser System Tests

The testing of the arm from a mechanical standpoint was very straightforward. After assembly, the bearings worked smoothly. The bars themselves sagged under the weight of the linkage, and future work may be done to lighten and strengthen the design. This sagging pulls the linkage away from the motors when the system is unpowered, but not with enough force to affect the output. The testing for the PD control, following the above procedure, can be seen in Figures 4.4 through 4.7.

Square



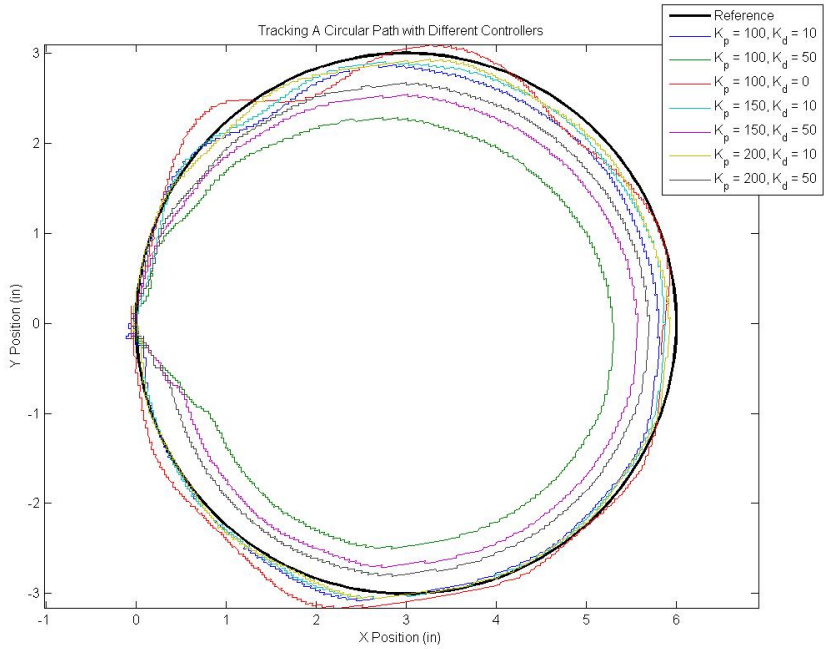
(a) Path Overview



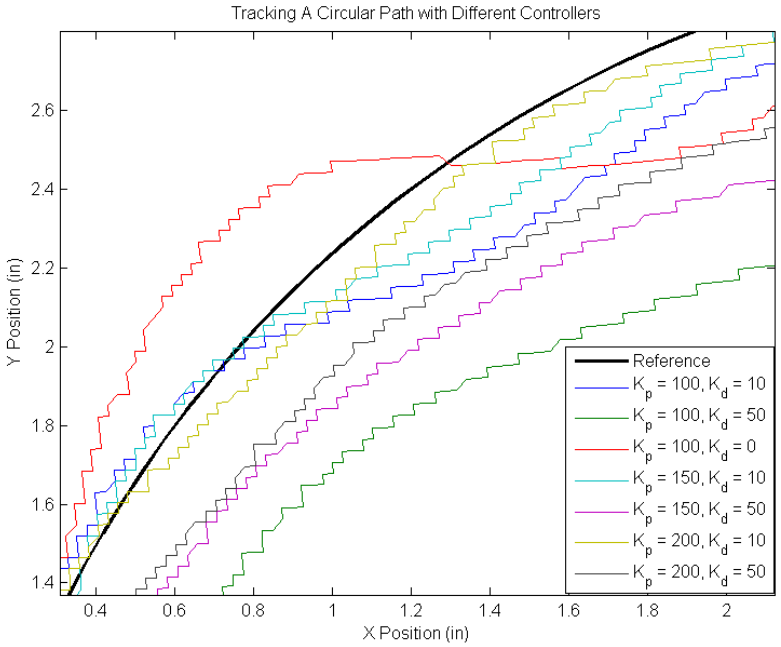
(b) Closeup View

Figure 4.4: Different K_p–K_d pairs following the same square trajectory

Circle



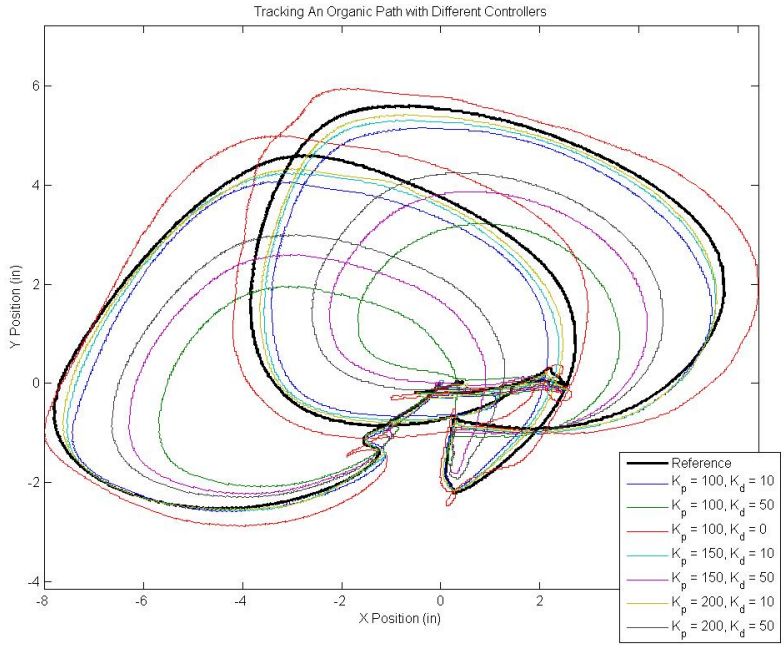
(a) Path Overview



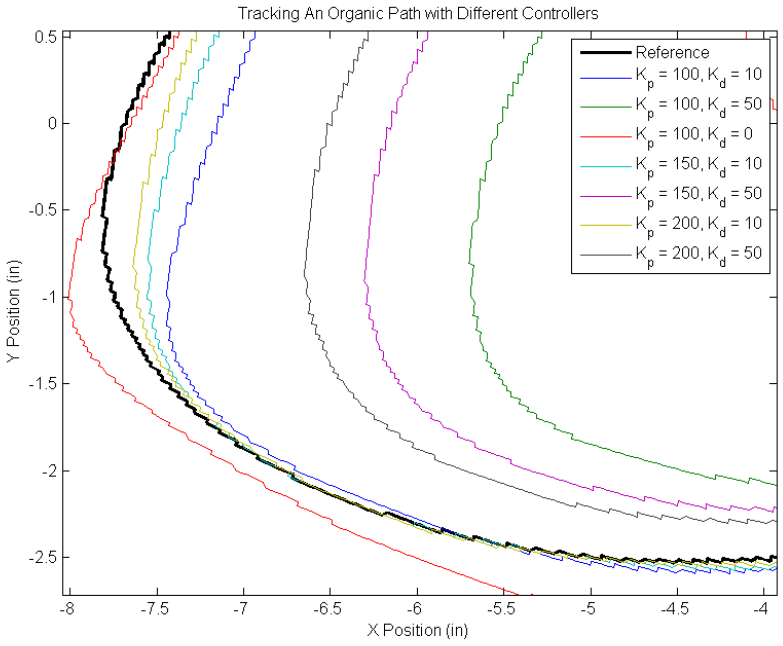
(b) Closeup View

Figure 4.5: Different K_p–K_d pairs following the same circular trajectory

Organic Shape



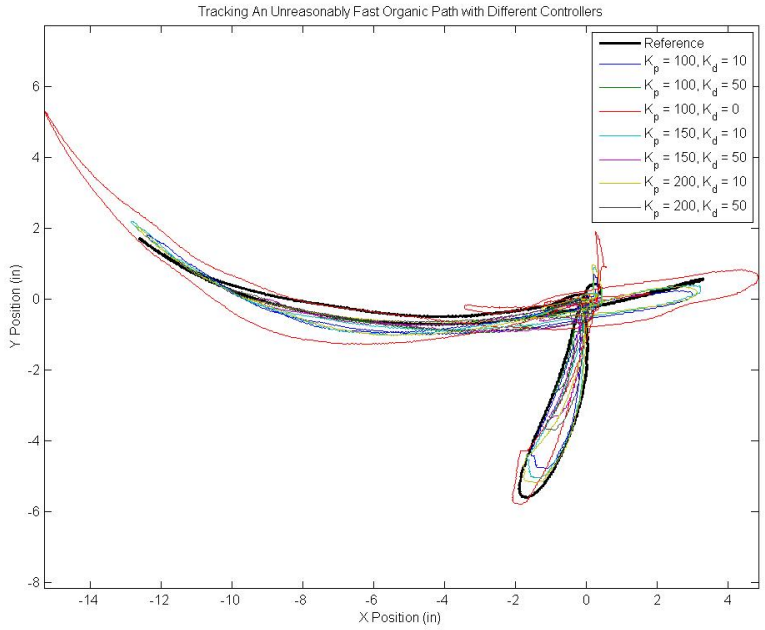
(a) Path Overview



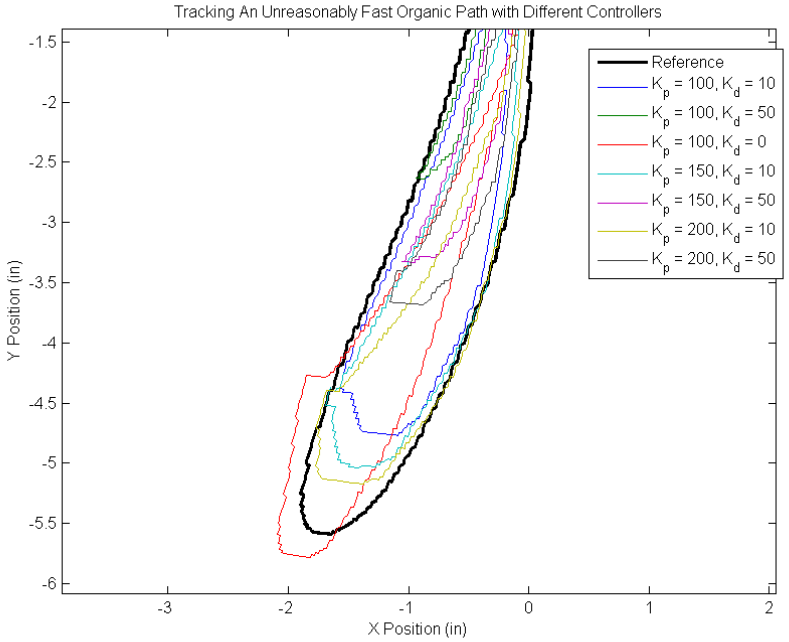
(b) Closeup View

Figure 4.6: Different K_p–K_d pairs following the same hand recorded trajectory

Fast Shape



(a) Path Overview



(b) Closeup View

Figure 4.7: Different K_p–K_d pairs following the same unreasonably fast trajectory

Trends

The first three plots show that the worst tracking pair ($K_p = 100$, $K_d = 50$), drawn in dark green) lagged consistently on the interior of the reference shape. Increasing K_p to 150, as shown by the purple line, improved tracking slightly, but there was still significant lag. Reducing K_d increased the rise time which improved the tracking more than increasing K_p. A value of $K_d = 10$ provided good tracking, as evident in the K_p lines of 100, 150, and 200. For contrast, a $K_d = 0$ line was also plotted, and while it fared decently on the square, it oscillated in and out of the circle and remained on the outside of the organic shape.

Unlike the other test trajectories, the fast shape provided a less clear trend, but ultimately followed the same overall behavior. Because the motors saturated so quickly, the different K_p values made little difference in the performance of the mechanism. The pairs at $K_d = 50$ provided the worst tracking. With less damping, the $K_d = 10$ gains tracked better. The significant outlier was the $K_d = 0$ gain, which, because it had no derivative dependence, overshot the reference trajectory.

After weighing all of this data, the optimal pair of gains was found to be $K_p = 200$ and $K_d = 10$. The Bode plots of this controller based on the three variants of the plant model are shown in Figure 4.8.

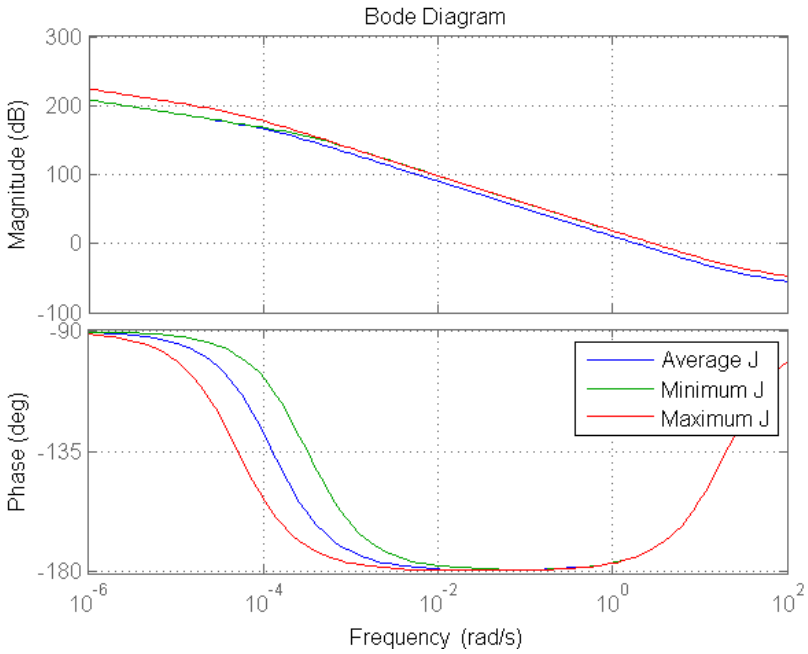


Figure 4.8: Bode plots of the best controller, $10s + 200$

Accuracy and Precision

Figures 4.4 through 4.7 represent a single trial for each gain pair, but they are all representative of multiple trials with little variance. As an example, Figure 4.9 shows two step responses with the same K_p and K_d values. The plots are practically identical, demonstrating the precision of the linkage system.

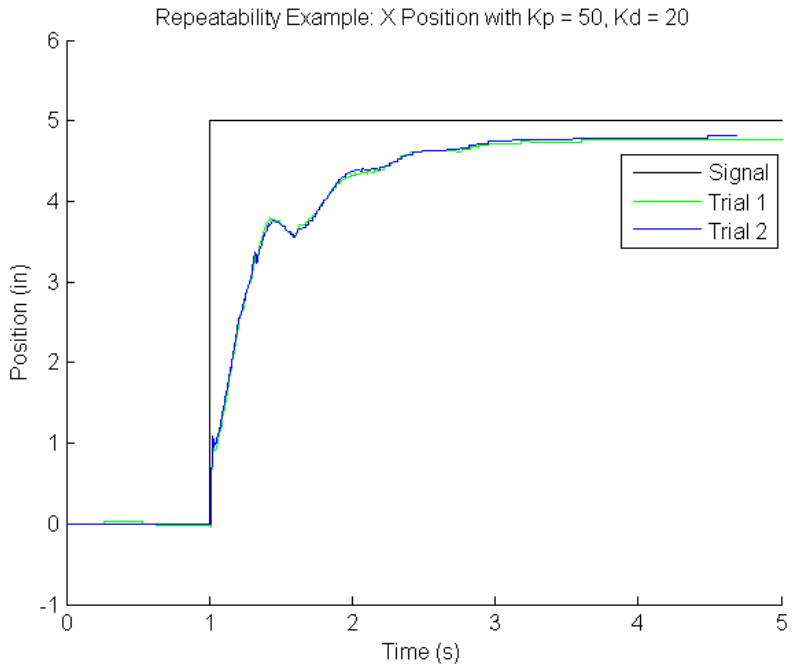


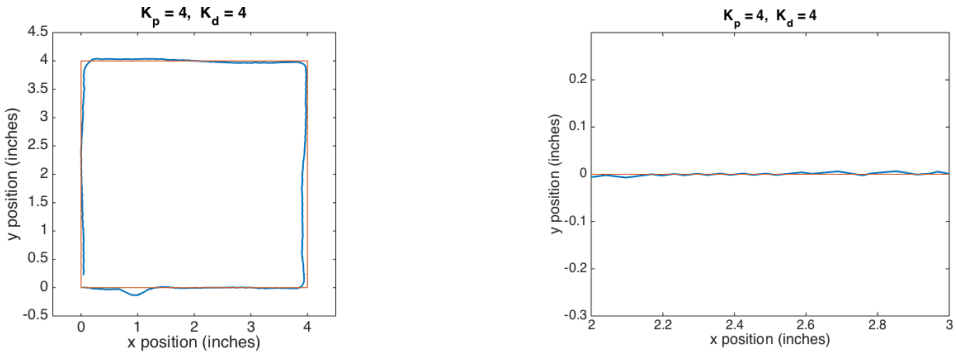
Figure 4.9: Repeatability Testing of Quanser System

On the other hand, the accuracy of the linkage was not as strong. When running more than five to ten inches away from the origin, small errors in the zeroing of the encoders became more pronounced. While the coordinates were internally consistent, plotting a square could have easily become a parallelogram if the mechanism were not set up to the proper specifications.

4.4.2 Arduino System Tests

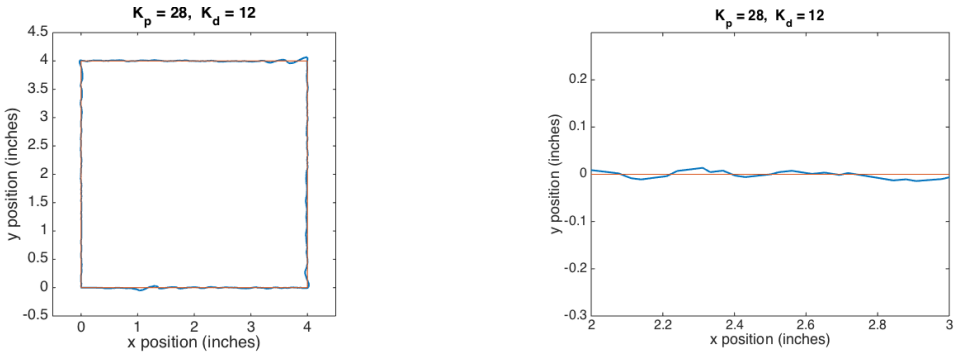
In the plots in this section, the red line represents the reference trajectory and the blue line represents the actual trajectory followed by the rack and pinion mechanism. Section 4.4.3 includes an in-depth analysis of the Arduino tests, and section 4.3 includes a description of each test.

Square



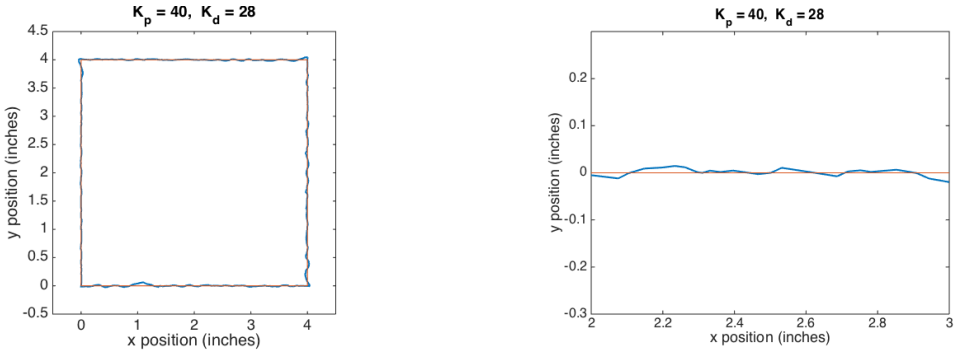
(a) Path Overview (b) Closeup View

Figure 4.10: Reference and Actual Position for $K_p = 4$, $K_d = 4$



(a) Path Overview (b) Closeup View

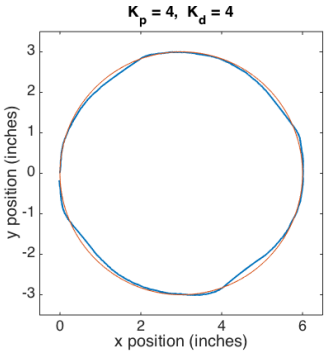
Figure 4.11: Reference and Actual Position for $K_p = 28$, $K_d = 12$



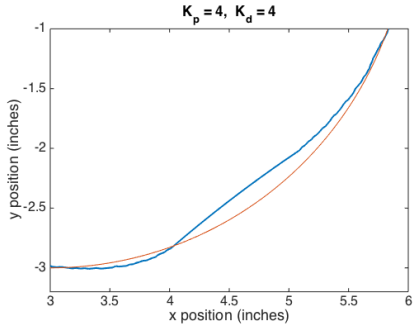
(a) Path Overview (b) Closeup View

Figure 4.12: Reference and Actual Position for $K_p = 40$, $K_d = 28$

Circle

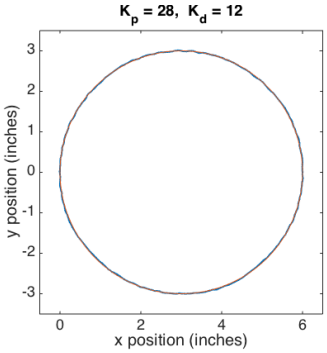


(a) Path Overview

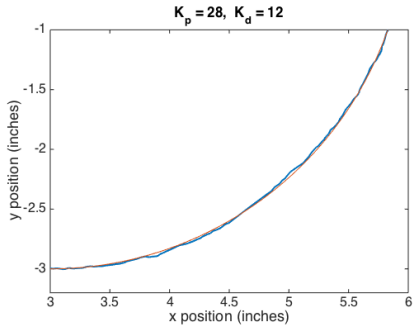


(b) Closeup View

Figure 4.13: Reference and Actual Position for $K_p = 4$, $K_d = 4$

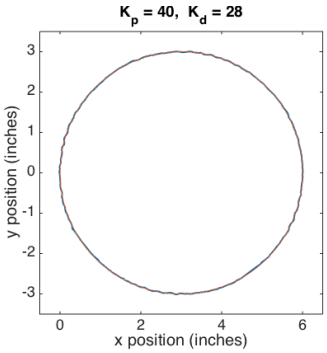


(a) Path Overview

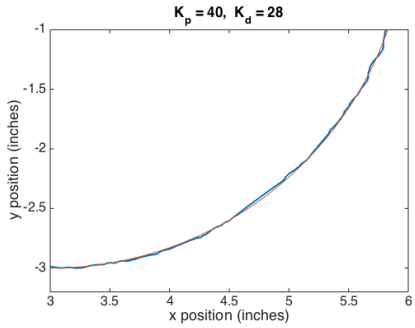


(b) Closeup View

Figure 4.14: Reference and Actual Position for $K_p = 28$, $K_d = 12$



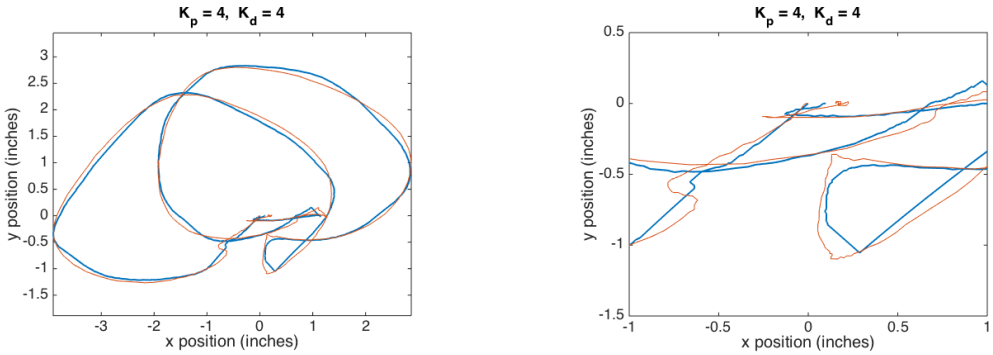
(a) Path Overview



(b) Closeup View

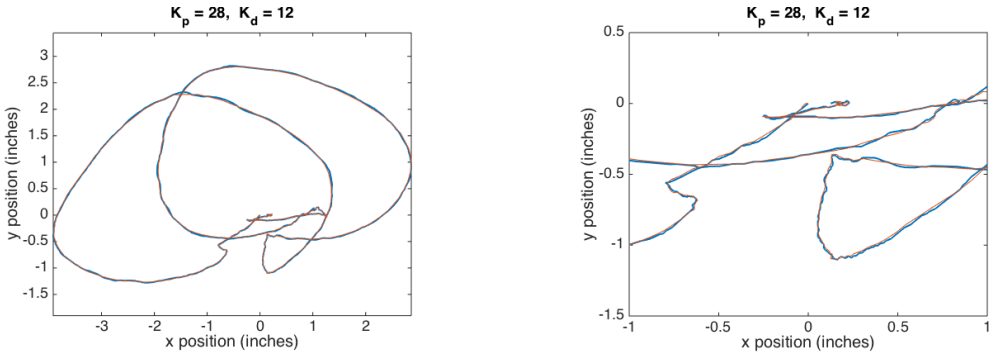
Figure 4.15: Reference and Actual Position for $K_p = 40$, $K_d = 28$

Organic Shape



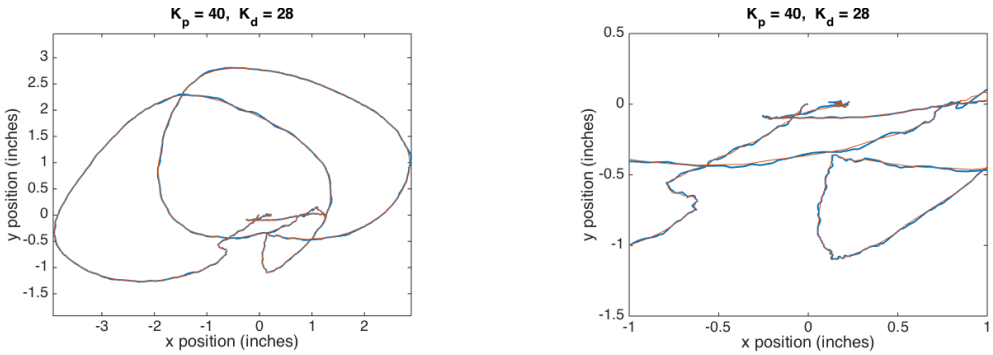
(a) Path Overview (b) Closeup View

Figure 4.16: Reference and Actual Position for $K_p = 4$, $K_d = 4$



(a) Path Overview (b) Closeup View

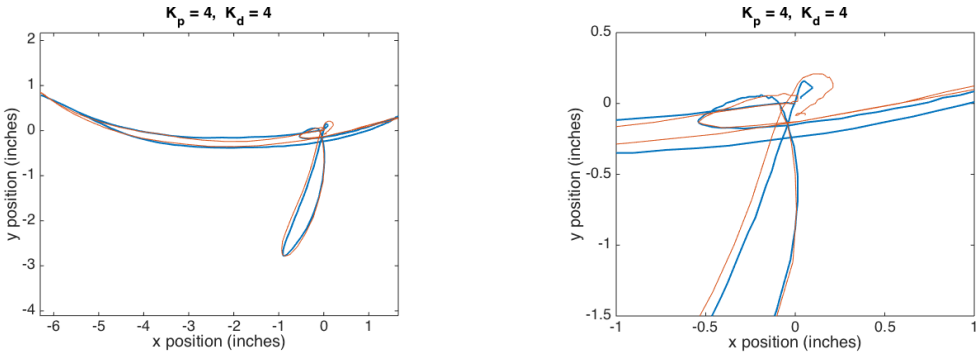
Figure 4.17: Reference and Actual Position for $K_p = 28$, $K_d = 12$



(a) Path Overview (b) Closeup View

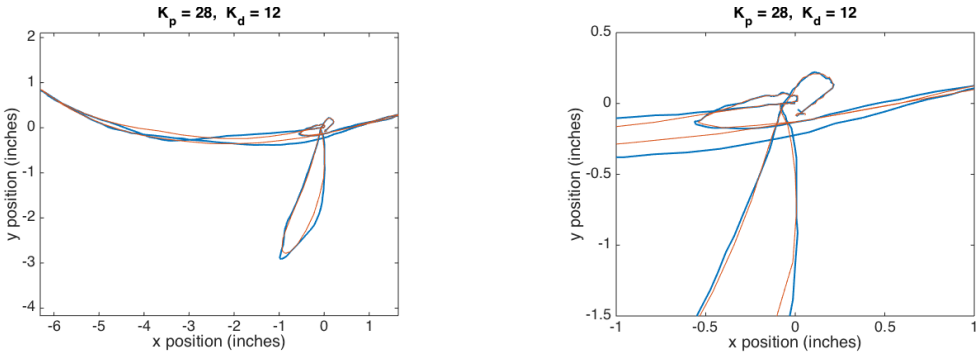
Figure 4.18: Reference and Actual Position for $K_p = 40$, $K_d = 28$

Fast Shape



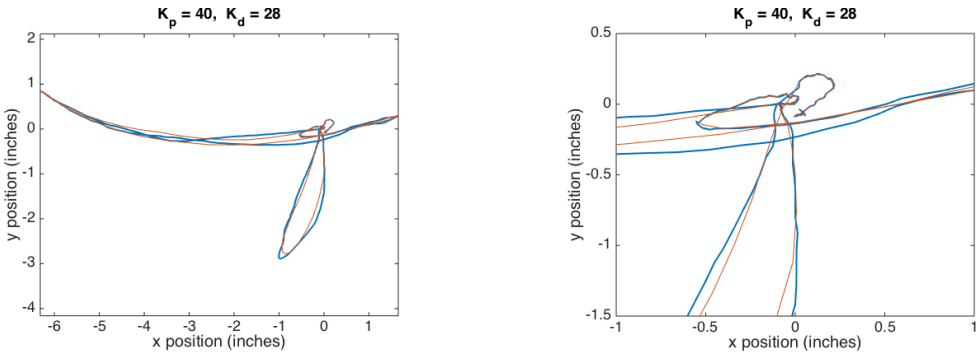
(a) Path Overview (b) Closeup View

Figure 4.19: Reference and Actual Position for $K_p = 4$, $K_d = 4$



(a) Path Overview (b) Closeup View

Figure 4.20: Reference and Actual Position for $K_p = 28$, $K_d = 12$



(a) Path Overview (b) Closeup View

Figure 4.21: Reference and Actual Position for $K_p = 40$, $K_d = 28$

4.4.3 Trends

Figures 4.10 through 4.21 show that while lower values of K_p resulted in a smoother line, this line was not always accurate because the position sometimes remained far from the reference trajectory over a long distance. Higher values of K_p generally caused quick oscillation of the mechanism, resulting in a bumpy line that never strayed far from the reference trajectory.

These tests made it clear that there is a tradeoff between smoothness and accuracy when selecting values for K_p and K_d. A relatively high value for K_p paired with a value for K_d that avoids saturation of the motors provides the closest tracking. The ideal pair of values that we found for this case was $K_p = 28$, $K_d = 12$. However, if the goal is to form smooth lines that qualitatively mimic human motions, then it is best to choose a lower value for K_p to avoid vibration of the motors.

Chapter 5

Analysis and Comparison of Mechanisms

5.1 Physical Mechanism

One main difference between the five-bar linkage and the rack and pinion mechanism is the effective moment of inertia of the two systems. There are two factors that contribute to the higher moment of inertia of the five-bar linkage: the first is that the system has more mass, and the second is that the center of mass of the linkage arms is located far from the axes of rotation of the motors.

When one motor of the 5-bar linkage rotates, it affects three of the four bars. The motors of the rack and pinion mechanism, on the other hand, only translates the near bar along its axis and rotates the far bar by applying a force at its end. The rack and pinion mechanism provides greater mechanical advantage. This can be seen in the accuracy of the position results in Figure 5.1.

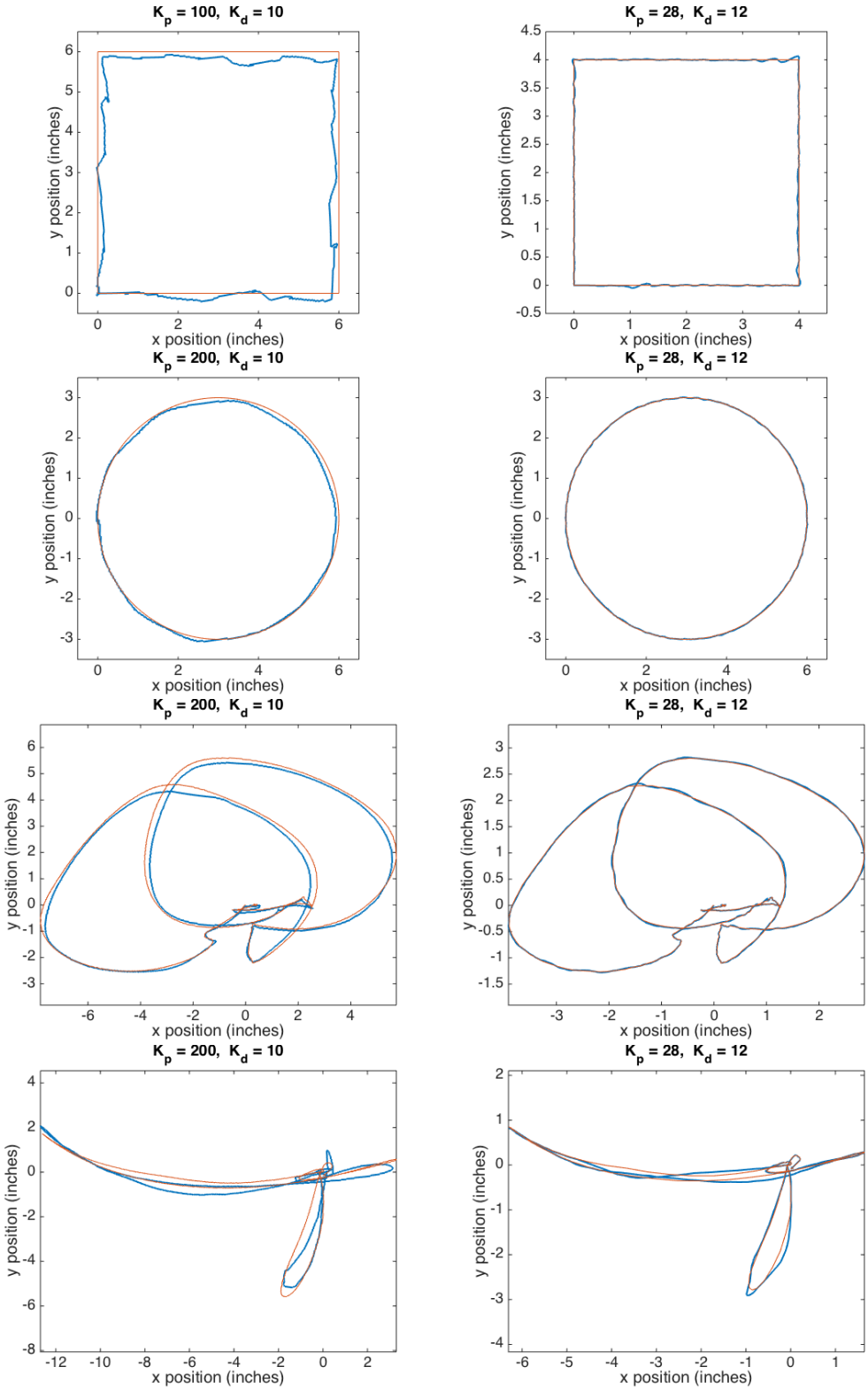


Figure 5.1: Comparisons of the two systems on the four trajectories. The left column is the Quanser system, and the right column is the Arduino system.

Other characteristics are also affected by the difference in mechanical advantage between the two systems. One positive feature of the five-bar linkage setup is that it is much easier for a user to move the end effector compared to the rack and pinion mechanism. This comparison demonstrates the tradeoff between the torque required from the motors to move the system and the force required from a user to move the system. For this reason, the five-bar linkage mechanism appears to be better for recording user inputs, while the rack and pinion mechanism is better for reproducing trajectories.

There is also a difference in scalability between the two mechanisms. If a larger version of the five-bar linkage mechanism were constructed, the inherent error would be amplified by a constant factor proportional to the change in size. This is because the angle of the arms is being controlled, so any error in the angle is amplified along the length of the arms. The inherent error in the rack and pinion mechanism, on the other hand, does not scale with the size of the mechanism. The error in the rack and pinion only depends on how precisely the lengths of the arms can be controlled, not on the overall size of the mechanism. In fact, a larger mechanism could be even more accurate because if the arms are longer, there is a larger region in which the arms are at nearly a right angle to one another (i.e. $\gamma \approx 90°$).

5.2 Controls

From an analysis of the linear models of the two plants in sections 2.4 and 3.4, a significant difference is immediately clear. The five-bar linkage, with its greater mass, has a pole at 1.3×10^{-4} rad/s. The rack and pinion mechanism, on the other hand, has the same pole at -204 rad/s. The dramatic increase in pole speed provides more flexibility in placing a controller zero, allowing for a much greater bandwidth. This results in higher rise times and better tracking. By selecting different gains for the rack and pinion mechanism, a wider variety of behaviors can be achieved, ranging from smooth, gentle control to aggressive, oscillatory, highly accurate control.

5.3 Electronics and Software

Microprocessor-based systems and microcontroller-based systems have inherent trade-offs. In general, microprocessors have more processing power, more data storage space, and a higher data transfer rate [6]. Meanwhile, the Arduino microcontroller

consumes much less power and due to its lack of an operating system has negligible overhead [2]. Due to the above criteria with regard to overall accuracy, one would expect microprocessors to outperform microcontrollers. However, our aim for the second design was to implement a portable system based on readily available third party motors, leading to our decision to choose an Arduino.

Despite these limitations and the less accurate encoders, the Arduino system was able to maintain a high level of accuracy and in general performed better than the Quanser system. We attribute the success of the system to the mechanical features of the rack and pinion mechanism. However, in order to be certain, further studies would be required.

Chapter 6

Conclusion

During the course of this project, we developed and tested an entirely new mechanism capable of motion with two degrees of freedom. We tested this mechanism against a five-bar linkage. By optimizing the proportional and derivative gains for each system, we were able to accurately record and play back a variety of simple and complex trajectories.

The five-bar linkage and rack and pinion mechanism have comparable ranges of motion, but the latter is significantly lighter. Its use of translational instead of rotational motion increases its accuracy, as can be seen in the test results. Unlike the five-bar linkage, error in the rack and pinion does not scale with the size of the mechanism.

The differences between these two mechanisms illustrated a key tradeoff. The five-bar linkage affords the user greater leverage in turning the motors, allowing for easier recording of user motion at the cost of playback accuracy. In contrast, the rack and pinion mechanism provides less leverage, making user input more difficult but providing more accurate control. One simple modification that could be made to the rack and pinion mechanism to address this tradeoff would be to have two modes for the mechanism: one for recording and one for playback. The gear motor could be disengaged during recording so that only the encoder needed to be turned by the user.

Table 6.1: Microcontroller and Microprocessor Usage

	Microprocessor	**Microcontroller**
Five-Bar Linkage	Examined Here	Future Work
Rack and Pinion	Future Work	Examined Here

Future experimenters might consider making a five-bar linkage that is lighter weight and optimized for its use as an Autopen. Since this project started as a method of catching projectiles, the five-bar linkage was not fine-tuned for this purpose. In addition, using the same controller for both systems would produce more useful results. Arduinos are inexpensive, and controlling the five-bar linkage using Arduino would have been no more complex than it was with Quanser. It is unclear which differences in output are due to the physical setup and which are due to the chosen controller. Table 6.1 shows the scope of our testing and opportunities for future experimentation. One might also compare the rack and pinion mechanism to other two degree of freedom systems, such as the track-based mechanisms found in laser cutters.

Overall, the rack and pinion mechanism is simple, portable, and accurate. Our prototype lays the groundwork for future testing and application of this two degree of freedom mechanism.

Bibliography

[1] "30:1 Metal Gearmotor 37Dx52L mm with 64 CPR Encoder." *Pololu.* https://www.pololu.com/product/1443.

[2] "Arduino Uno." *Arudino.* http://www.arduino.cc/en/Main/ArduinoBoardUno.

[3] "Arduino-serial: C code to talk to Arduino." *todbot blog.* http://todbot.com/blog/2006/12/06/arduino-serial-c-code-to-talk-to-arduino/. December 6, 2006.

[4] Clarence Rowley and Michael Littman. "Lab Handout." *MAE 433: Automatic Control Systems.* Princeton University, 2014.

[5] "DC motor control with PID." *Arduino Project Forum.* http://forum.arduino.cc/index.php?topic=8652.0. August 12, 2010.

[6] "FAQs — Raspberry Pi" *Raspberry Pi Foundation.* https://www.raspberrypi.org/help/faqs.

[7] Jimmy Stamp. "President Obama's Autopen: When is an Autograph Not an Autograph?" *Smithsonian.com* 8 January, 2013. http://www.smithsonianmag.com/arts-culture/president-obamas-autopen-when-is-an-autograph-not-an-autograph-574822/?no-ist. Accessed April 2015.

[8] John Doyle, Bruce Francis, and Allen Tannenbaum. *Feedback Control Theory.* Dover Publications, 2009.

[9] "Pololu Dual MC33926 Motor Driver Shield for Arduino." *Pololu.* https://www.pololu.com/product/2503/specs.

[10] "Pololu Dual MC33926 Motor Driver Shield User's Guide." *Pololu Corporation.* https://www.pololu.com/docs/pdf/0J55/dual_mc33926_motor_driver_shield.pdf.

[11] Sir Mix-A-Lot. "Baby Got Back." *Mack Daddy*, Def American Recordings, 1992.

[12] Quanser 2DOF Robot. Accessed October 2014. http://www.quanser.com/products/2dof_robot.

[13] R. Moazed. *Experimental Study of a Two-DOF Five Bar Close-Loop Mechanism*, Unpublished Master's Thesis, University of Saskatchewan, Canada. 2006.

[14] Smart Trash Can Robot. Last modified February 20, 2013. https://www.youtube.com/watch?v=j9kNCnOWuO8.

[15] "SRV02-Series Rotary Servo Plant User Manual." *Quanser.* Revision 01.

Appendix A

Project Budget and List of Materials

This table illustrates the funding for the project and the amount remaining. We spent a minimal amount on Phase One leaving as much a possible for Phase Two.

Source of Funding	Amount of Funding
MAE Fund	$1,600.00
SEAS Fund	$500.00
Total	$2,100.00
Amount Spent:	$998.14
Remaining Funds:	$1,101.86
% Spent:	47.53%
% Remaining:	52.47%

Figure A.1: Project Budget Summary

Part Number	Description	Quantity	Vendor	Date Ordered	Price (per item)	Total Price
8975K34	Aluminum Bar Stock 6' by 1/8" by 1.5"	3	McMaster	11/13/2014	$9.33	$27.99
90272A550	1/4-20 Machine Screws (pack)	1	McMaster	11/13/2014	$9.92	$9.92
90480A029	1/4-20 Hex Nut (pack)	1	McMaster	11/13/2014	$3.15	$3.15
94639A818	Nylon Spacers (pack)	2	McMaster	11/13/2014	$7.95	$15.90
6112K45	Steel Shaft 8mm D, 400mm L	1	McMaster	11/13/2014	$12.12	$12.12
57485K67	Shaft Collars for 8mm D	8	McMaster	11/13/2014	$1.87	$14.96
93475A270	Steel Flat Washer 8.4mm ID 16.0mm OD (pack)	1	McMaster	11/13/2014	$7.90	$7.90
9657K372	Steel Compression Spring (pack)	1	McMaster	11/13/2014	$5.99	$5.99
92146A029	Steel Split Lock Washer 1/4-20 (pack)	1	McMaster	11/13/2014	$4.03	$4.03
86985K437	Aluminum Rod 1' L .5" D	1	McMaster	11/13/2014	$6.88	$6.88
108	Self-Aligning Ball Bearings	7	eBay	11/15/2014	$9.90	$69.30
N/A	Pixy Camera	2	Amazon	11/12/2014	$69.00	$138.00
5544T273	Steel Rod 8mm diameter, 3ft long, machinable	1	McMaster	12/1/2014	$6.80	$6.80
9657K371	Steel Compression Spring (pack), less strength	1	McMaster	12/1/2014	$5.63	$5.63
470132-378	Advanced Ping-Pong Ball Launcher	1	VWR International	12/1/2014	$12.59	$12.59
470132-450	Ping-Pong Balls (6-pack)	1	VWR International	12/1/2014	$4.29	$4.29
10UM-02103BK	CableWholesale 5 Pin 3-Feet USB Type A Male/Mini-B Male Cable, Black	2	Amazon	12/9/2014	$1.90	$3.80
AMB610	Prism Backdrops 6X10' Black Muslin Photo Video Backdrop Background	1	Amazon	2/9/2015	$22.96	$22.96
	HDMI Cable	1	Amazon	2/13/2015	$5.49	$5.49
	HDMI to DVI Adapter	1	Amazon	2/13/2015	$9.03	$9.03
	HC-SR04 Ultrasonic Sensors (for sonic position finding)	2	eBay	2/15/2015	$8.99	$17.98
	Pololu Motor Components			3/6/2015		$141.97
	Components for the Gear Rack Mechanism		McMaster	3/8/2015		$307.48
	Components for the Gear Rack Mechanism		McMaster	3/28/2015		$49.86
	Materials for Rack Joint		McMaster	4/3/2015		$36.06
	Epoxy	2	Amazon	4/1/2015		$20.26
	Arduino Kit		SparkFun	4/17/2015		$37.80
					Total Spent:	$998.14

Figure A.2: List of Materials

Appendix B

Quanser Simulink Models

B.1 `Record.mdl`

The recording model has three nested models, starting at the XY level, then the $\theta_1\theta_2$, then finally the physical encoder reading level.

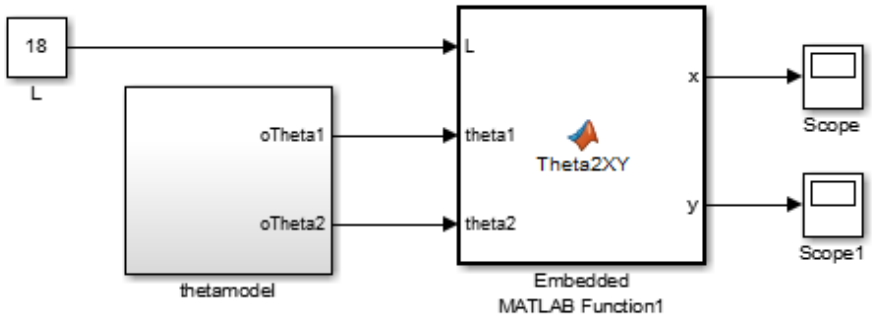


Figure B.1: Quanser Recording Simulink Model: Cartesian Coordinates

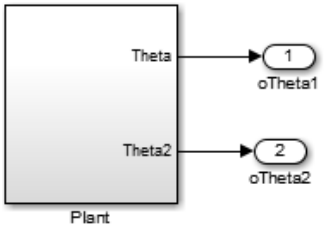


Figure B.2: Quanser Recording Simulink Submodel: Angular Coordinates

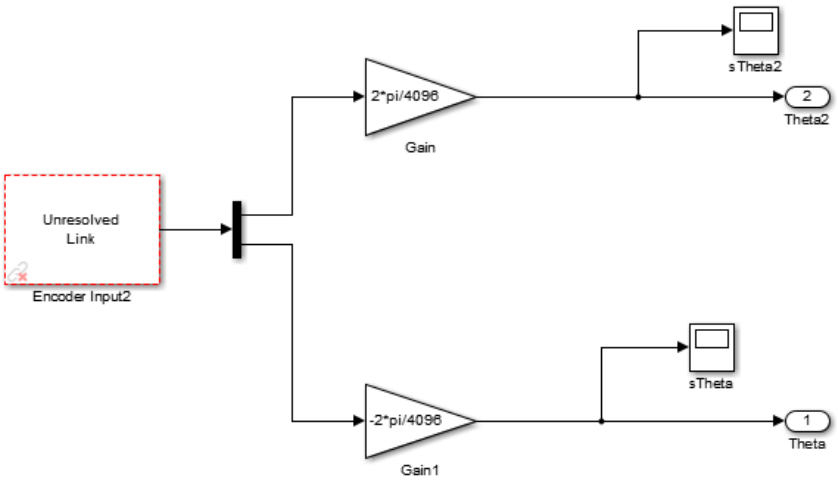


Figure B.3: Quanser Recording Simulink Submodel: Encoder Capture

B.2 `Playback.mdl`

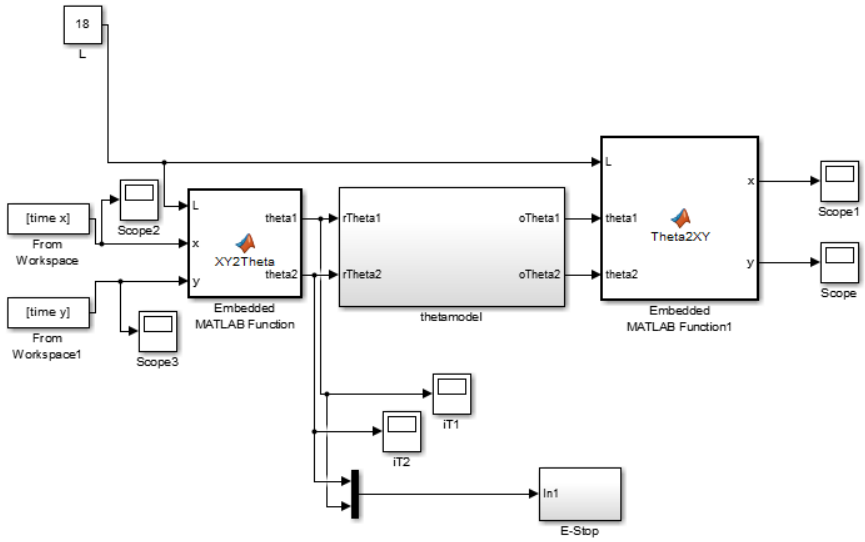


Figure B.4: Quanser Playback Simulink Model: Cartesian Coordinates

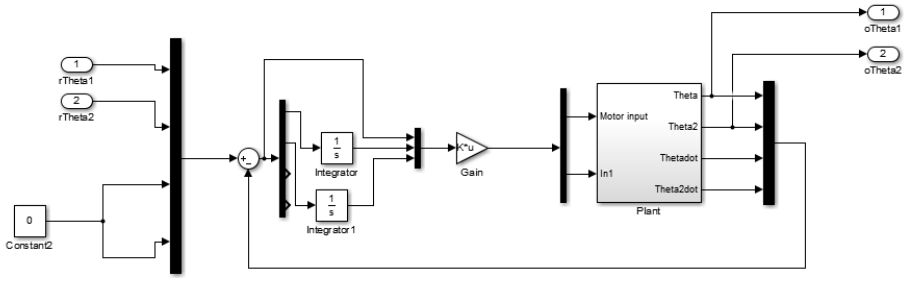


Figure B.5: Quanser Playback Simulink Model: Angular Coordinates

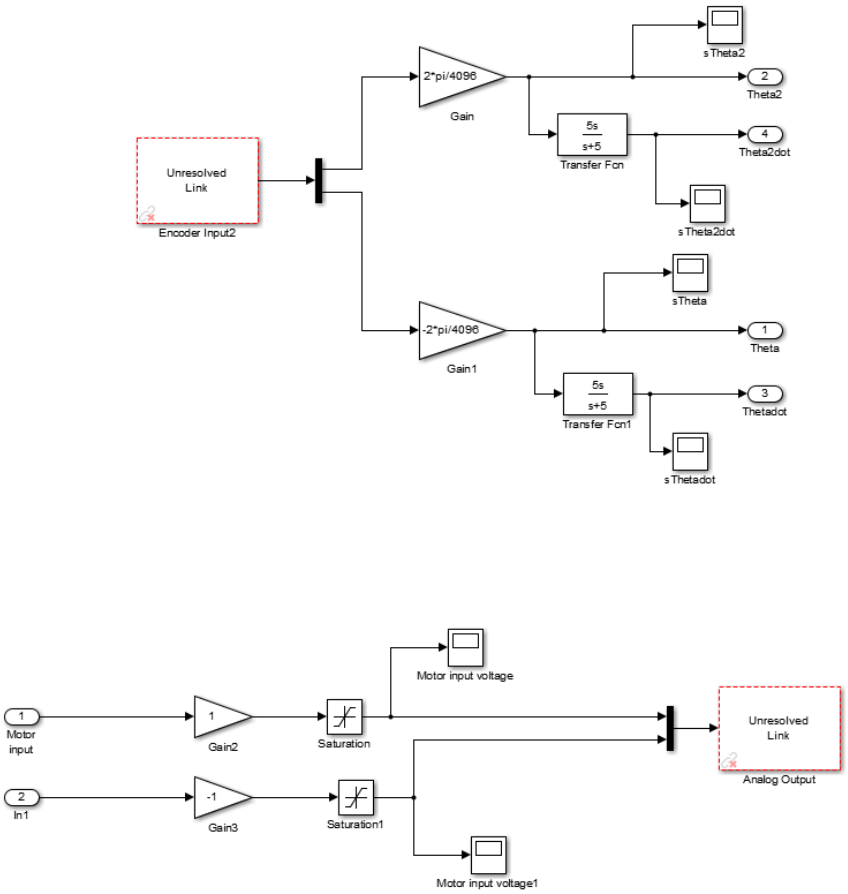


Figure B.6: Quanser Playback Simulink Model: Voltage Output and Encoder Capture

Appendix C

Matlab Library

This project used a lot of MATLAB infrastructure, short programs that generated trajectories, converted between coordinate systems, or plotted various quantities.

C.1 Trajectories

C.1.1 circlegen.m

Creates a csv file with x-y pairs in a circle.

```
numpoints = 200;
r = 2;
dtheta = 2*pi/numpoints;
theta = 0:dtheta:2*pi;
x = 1 - cos(theta);
y = sin(theta);
x = r.*(x');
y = r.*(y');
csvwrite('circle.csv',[x y]);
```

C.1.2 squaregen.m

Creates a csv file with x-y pairs in a square.

```
numPoints = 4;
len = 1;
dx = len/numPoints;
side1x = 0:len/numPoints:len-dx;
side1y = zeros(size(side1x));
side2x = len*ones(size(side1x));
side2y = side1x;
side3x = len - side1x;
side3y = len - side1y;
side4x = len - side2x;
side4y = len - side2y;
x = [side1x side2x side3x side4x]';
y = [side1y side2y side3y side4y]';
csvwrite('square.csv',[x y]);
```

C.2 Conversions

C.2.1 Theta2XY.m

Converts a θ_1, θ_2 pair to X,Y for the Quanser system.

```
function [x,y] = Theta2XY(theta1,theta2)


L = 18;


a = zeros(1,2);
b = zeros(1,2);
c = zeros(1,2);
rCslashA = zeros(1,2);
rCsAnorm = 0;
```

```
m1 = [-L L];
m2 = [L,L];

a = m1 + L.*[sin(theta1) -cos(theta1)];
c = m2 + L.*[-cos(theta2) -sin(theta2)];
rCslashA = c - a;
rCsAnorm = sqrt(rCslashA(1).*rCslashA(1) + rCslashA(2).*rCslashA(2))./2;

rPerp = [rCslashA(2) -rCslashA(1)];
rPerp = rPerp./sqrt((rCslashA(2)*rCslashA(2) + rCslashA(1)*rCslashA(1)));

b = a + rCslashA./2 + sqrt(L*L - rCsAnorm^2).*rPerp;

x = b(1);
y = b(2);

end
```

C.2.2 XY2Theta.m

Converts an X,Y pair to θ_1, θ_2 for the Quanser system.

```
function [theta1,theta2] = XY2Theta(x,y)

L = 18;

X = [x y];

m1 = zeros(1,2);
m2 = zeros(1,2);
r1 = 0;
r2 = 0;
```

```
phi1 = 0;
phi2 = 0;
beta1 = 0;
beta2 = 0;

m1 = [-L L];
m2 = [L,L];

r1v = X-m1;
r2v = X-m2;
r1 = sqrt(r1v(1).*r1v(1) + r1v(2).*r1v(2));
r2 = sqrt(r2v(1).*r2v(1) + r2v(2).*r2v(2));

phi1 = asin(r1/(2*L)); phi1 = phi1*2;
phi2 = 2*asin(r2./(2*L));

beta1 = asin((x - m1(1))/r1);
beta2 = asin((y - m2(2))/r2);

theta1 = (beta1 - pi/2 + phi1/2);
theta2 = (-beta2 - pi/2 + phi2/2);

end
```

C.2.3 AlphaFromTheta

Calculates the values of the interior angles based on the outer angles.

```
function [alpha1 alpha2] = AlphaFromTheta(theta1, theta2)
    [x y] = Theta2XY(theta1,theta2);

    L = 18;
```

```
    m1 = [-L L];
    m2 = [L,L];

    alpha1 = asin(x/L - m1(1)/L - sin(theta1));
    alpha2 = asin(-y/L + m2(2)/L - sin(theta2));
end
```

C.2.4 XY2Counts.m

Converts an X,Y pair to encoder counts in the Arduino system.

```
function [count1, count2] = XY2Counts(x, y)

L = 24;
r0 = 16;
m1x = -sqrt(2)*r0/2;
m1y = -sqrt(2)*r0/2;
m2x = sqrt(2)*r0/2;
m2y = -sqrt(2)*r0/2;
D = 1.5;
TicksPerRev = 477; %Found experimentally
H = sqrt((m1x - m2x)^2 + (m1y - m2y)^2);

dx1 = m1x - x;
dx2 = m2x - x;
dy1 = m1y - y;
dy2 = m2y - y;

r1 = sqrt(dx1*dx1 + dy1*dy1);
r2 = sqrt(dx2*dx2 + dy2*dy2);

theta1 = (2/D)*(r1 - r0);
```

```
theta2 = (2/D)*(r2 - r0);

count1 = -theta2*TicksPerRev/(2*pi);
count2 = theta1*TicksPerRev/(2*pi);

end
```

C.2.5 Counts2XY.m

Converts Arduino encoder counts to XY coordinates.

```
function [x y] = counts2XY(counts1, counts2)

L = 24;
r0 = 16;
m1x = -sqrt(2)*r0/2;
m1y = -sqrt(2)*r0/2;
m2x = sqrt(2)*r0/2;
m2y = -sqrt(2)*r0/2;
D = 1.5;
TicksPerRev = 477; %Found experimentally
H = sqrt((m1x - m2x)^2 + (m1y - m2y)^2);

theta1 = counts2*2*pi/TicksPerRev;
theta2 = -counts1*2*pi/TicksPerRev;

r1 = r0 + D*theta1/2;
r2 = r0 + D*theta2/2;

h = asin((m2y - m1y)/H);

alpha = acos((r2^2 - r1^2 - H^2)/(-2*r1*H));
```

```
y = r1*sin(alpha - h) + m1y;
x = r1*cos(alpha - h) + m1x;

end
```

C.2.6 fileParser.m

Takes in a CSV file of encoder data from the arduino and imports it as XY pairs for plotting.

```
function [x y] = fileParser(filename)

data = csvread(filename);

len = size(data,1);

x = zeros(len,1);
y = zeros(len,1);

for i = 1:len
    [x(i) y(i)] = counts2XY(data(i,1),data(i,2));
end

end
```

C.2.7 fileconvert.m

Converts a CSV file of XY data to encoder counts.

```
function fileconvert(filename,newname)

xy = csvread(filename);
```

```
counts = zeros(size(xy));
len = size(xy,1);
for i = 1:len
    [temp1, temp2] = XY2Counts(xy(i,1),xy(i,2));
    counts(i,:) = [round(temp1) round(temp2)];
end

csvwrite(newname,counts);

end
```

C.3 J Functions

C.3.1 JFunction.m

For a given $\theta_1 - \theta_2$ pair, computes the effective moment of inertia about one of the motors.

```
function [J1, J2] = JFunction(theta1,theta2)

    [alpha1, alpha2] = AlphaFromTheta(theta1,theta2);

    lbm = 0.4535;
    in = 0.0254;
    I = 44.5250*lbm*in*in;
    m = 1.0623*lbm;
    L = 18*in;

    J1 = I.*(1 + (sin(alpha1 - theta1).^2 + cos(alpha2 - theta1).^2)...
    /(cos(alpha1 - alpha2)^2)) + (m*L^2/8).*(10 + 2.*(sin(alpha1 - theta1).^2...
    + cos(alpha2 - theta1).^2)./(cos(alpha1-alpha2).^2)...
    - 8.*(cos(alpha1 - theta1).*cos(alpha2 - theta1))./cos(alpha1 - alpha2));
```

```
    J2 = I.*(1 + (sin(alpha2 - theta2).^2 + cos(alpha1 - theta2).^2)...
    /(cos(alpha1 - alpha2)^2)) + (m*L^2/8).*(10 + 2.*(sin(alpha1 - theta1).^2...
    + cos(alpha2 - theta1).^2)./(cos(alpha1-alpha2).^2)...
    - 8.*(cos(alpha1 - theta1).*cos(alpha2 - theta1))./cos(alpha1 - alpha2));

end
```

C.3.2 rackJfunc.m

For a given r_1, r_2 pair and motor distance h, determines the effective moment of inertia on the motor.

```
function J = rackJfunc(r1,r2,h)
    lbm = 0.4535;
    in = 0.0254;
    I_turntable = 3.087e-3;
    I_arm = 28.15*lbm*in*in;
    m_arm = 0.240;
    %I'm neglecting the penholder and stuff because it's tiny
    l = (r2-12)*in;
    r_gear = 0.75*in;
    alpha = acos((h^2 - r1^2 - r2^2)/(-2*r1*r2));
    J = m_arm*r_gear*r_gear + (r_gear/(sin(alpha)*r2))*(I_turntable...
    + I_arm + m_arm*l*l);
end
```

Appendix D

Arduino and Serial Code

D.1 Arduino Code

```
/*
 * Encoder_Control_Two_Motors
 * --------------
 * Author: Joshua A. Zimmer
 * Version: 1.0
 * Date: 26 April 2015
 *
 * Significant Adaptations From:
 *    * http://forum.arduino.cc/index.php?topic=8652.0
 *    * Lab Handouts -- MAE 433B Instructor Versions --
 *      Professor Clarence Rowley, Princeton University
 *    * Pololu Dual MC33926 Motor Driver Shield User's Guide
 *
 * This program draws etch-a-sketch style figures from a set of
 * CSV input data points corresponding to the rotations of
 * two motors, making use of PD control to ensure output accuracy.
 * After the final data point is sent, the motors return their
 * rotations to 0 to ensure that future tests start from the
```

```
 * appropriate orientation.
 *
 * On startup, this program zeros the rotations of the two
 * connected motor encoders, and prepares to accept data from the
 * serial port. The program recieves a two column CSV file encoded
 * as ASCII data over the serial port, which is preceded by the
 * initialization string "0000" and succeeded by the finalizing
 * string "FFFF". The initialization string indicates the start of
 * the CSV data and the finalizing string indicates the end of the
 * CSV data. After receiving "FFFF", the system will reset the two
 * motors' rotations to 0.
 *
 * If the CSV file sends more data than the Arduino's memory
 * can store, the Arduino will cease to interpret data, draw the
 * currently stored data points, and then resume intaking data.
 *
 * (Please note that the following code only pertains to discussed
 * analysis and does not allow for direct position recording.)
 *
 *
 * IMPORTANT NOTE: Due to the wiring of the system, Motors 1 & 2 in
 * the following code represent the opposite motors as are reflected
 * in the text.
 *
 */

#include "DualMC33926MotorShield.h"

DualMC33926MotorShield md;
```

```
static String inputString = ""; // a string to store incoming data

#define TIMEENTRIES 350 // maximum entries to be stored at once
#define TIMESCALEIN 30 // millisecionds available for settling per entry
#define TIMESCALEOUT 10 // milliseconds between data recording per entry
#define LOOPTIME 10 // milliseconds between PD update loops

#define TOL 15 // count tolerance for re-zeroing PD update loop

//CONTROL VALUES
#define Kp 28.0 // proportional control constant (found experimentally)
#define Kd 12.0 // derivative control constant (found experimentally)

unsigned long lastMilli = 0; // maintains time of last PD update
unsigned long lastMilliPrint = 0; // maintains time of last data output
unsigned long lastMilliIn = 0; // maintains time of last CSV point rotation

/*
 * Shield Pin Reference:
 * ----------------------
 * I(Motor 1) = A0 (M1FB)
 * I(Motor 2) = A1 (M2FB)
 * BrakeOff = D4 (nD2)
 * DIR(Motor 1) = D7 (M1DIR)
 * DIR(Motor 2) = D8 (M2DIR)
 * SPD(Motor 1) = D9 (M2PWM)
 * SPD(Motor 2) = D10 (M2PWM)
 */
int encodPinA1 = 2;
int encodPinB1 = 11;
```

```
int encodPinA2 = 3;
int encodPinB2 = 12;

// default reference rotation for Motor 1 to attain (misleading name)
int defSpd1 = 0;
// speed acutually sent to Motor 1 to attain defSpd1
int sntSpd1 = 0;
// the current rotation position of Motor 1
volatile long count1 = 0;

// default reference rotation for Motor 2 to attain (misleading name)
int defSpd2 = 0;
// speed acutually sent to Motor 2 to attain defSpd1
int sntSpd2 = 0;
// the current rotation position of Motor 2
volatile long count2 = 0;




/* ---------------------------- FUNCTIONS --------------------------- */

/* Initialize input/output pins, attatch falling-edge interrupts to
 * the encoders in order maintain motor rotation counts, initialize
 * motor variables and speeds to 0, and reserve memory for the input
 * string.*/
void setup()
{
  analogReference(DEFAULT); // Should be 5 Volts -- USE EXTERNAL OTHERWISE
```

```
  Serial.begin(115200); // Initialize serial communication to shield default

  // Setup Encoders for Motor 1 and turn on pullup resistors
  pinMode(encodPinA1, INPUT);
  pinMode(encodPinB1, INPUT);
  digitalWrite(encodPinA1, HIGH);
  digitalWrite(encodPinB1, HIGH);

  // Setup Encoders for Motor 2 and turn on pullup resistors
  pinMode(encodPinA2, INPUT);
  pinMode(encodPinB2, INPUT);
  digitalWrite(encodPinA2, HIGH);
  digitalWrite(encodPinB2, HIGH);

  // Attach Interrupts for HE sensor encodor A for Motors 1 and 2
  attachInterrupt(0, rencoder1, FALLING); // Interrupt on DP 2 -- Motor 1
  attachInterrupt(1, rencoder2, FALLING); // Interrupt on DP 3 -- Motor 2

  // Initialize Motors
  md.init();
  md.setM1Speed(0); // Set initial speed on M1 to zero
  md.setM2Speed(0); // Set initial speed on M2 to zero

  inputString.reserve(200); // reserve 200 bytes for the inputString:
  delay(1000); // Allow for debugging prep/Serial Monitor
}



/* Maintains a DFA state variable that causes thes system to correspndingly:
```

```
 *    1. Take in and record CSV data OR
 *    2. Update PD Control, Output Recorded Data, and Set Motor Speeds
 */
void loop()
{
  static boolean stringComplete = false;  // maintains if line is complete

  static int state = 0; // DFA State

  static int i = 0; // iteration var -- current number of CSV input lines
  static int j = 0; // iteration var -- motor # of next input count
  static int l = 0; // iteration var -- line # of current input tracked

  static int xy[TIMEENTRIES][2]; // array of counts for input motors (x-1, y-2)
  static boolean done = true; // maintains whether or not "FFFF" recieved

  // Determines whether or not we should be waiting for the "0000" string
  if(state == 0)
  {
    if(done)
      state = waitForStart(state);
    else
      state ++;
    done = false;
  }

  // Intakes and parses the CSV file from the serial port
  else if(state == 1)
  {
    while (Serial.available()) {
```

```
// get the new byte:
char inChar = (char)Serial.read();

// add it to the inputString:
inputString += inChar;

// if the input string is "FFFF", we have reached
// file end:
if (inputString.substring(inputString.length()-5,
inputString.length()) == "FFFF\n")
{
  stringComplete = true;
  done = true;

  state ++;
  j = 0;
}

// if the incoming character is a newline, set a flag
// so the main loop can do something about it, inc i,
// set j = 0:
else if (inChar == '\n')
{

  xy[i][j] = inputString.toInt();

  stringComplete = true;
  i++;
  j = 0;
```

```
}

// if the incoming character is a ',', inc j:
else if (inChar == ',')
{
  xy[i][j] = inputString.toInt();

  inputString = "";
  j++;
}

// deal with the newline flag:
if(stringComplete == true)
{
  //Serial.print(inputString);
  inputString = "";
  stringComplete = false;
}

// if the input data exceeds Arduino memory,
// write out all of the data that has been
// brought in, thus far:
if(i + 1 >= TIMEENTRIES)
{
  state ++;
  j = 0;

  inputString = "";
  break;
}
```

```
  }
}

// Dummy state for debugging
else if(state == 1)
{
  state++;
  l = 0;
}

// PD update loop
else if(state == 2)
{
  // if time-out has elapsed, output position data
  if(millis() - lastMilliPrint >= TIMESCALEOUT && l > 1 && l <= i+1)
  {
    lastMilliPrint = millis();

    //send data to file
    pXY(count1,count2);
  }

  // if time-in has elapsed, progress to next CSV position
  if(millis() - lastMilliIn >= TIMESCALEIN)
  {
    lastMilliIn = millis();

    if(l >= i)
    {
      if(done)
```

```
      {
        defSpd1 = 0;
        defSpd2 = 0;
      }
      if(l > i+1)
      {
        state ++;
      }
    }
    else
    {
        defSpd1 = xy[l][0]; //x
        defSpd2 = xy[l][1]; //y
    }

    l++;
    delay(1);
  }

  // if loop-time has elapsed, update PD outputs for M1/M2
  if(millis()-lastMilli >= LOOPTIME)
  {
    lastMilli = millis();
    sntSpd1 = updatePid1(sntSpd1, defSpd1, count1);
    md.setM1Speed(sntSpd1);
    sntSpd2 = updatePid2(sntSpd2, defSpd2, count2);
    md.setM2Speed(sntSpd2);
  }
}
```

```
  // additional PD settling time to allow for continued input
  else if(state == 3)
  {
    if(millis()-lastMilli >= LOOPTIME)
    {
      lastMilli = millis();
      sntSpd1 = updatePid1(sntSpd1, defSpd1, count1);
      md.setM1Speed(sntSpd1);
      sntSpd2 = updatePid2(sntSpd2, defSpd2, count2);
      md.setM2Speed(sntSpd2);
    }
    if(abs(count1 - defSpd1) < TOL && abs(count2 - defSpd2) < TOL)
    {
      md.setM1Speed(0);
      md.setM2Speed(0);

      state ++;
    }
  }

  // Re-zero and return to initial state
  else
  {
    i = 0;
    j = 0;
    l = 0;
    state = 0;
    //set back to 0,0 -- go to state 0
  }
}
```

```
// compute desired PWM value for Motor 1 output
int updatePid1(int command, int targetValue, int currentValue)
{
  float pidTerm = 0; // PD correction
  int error = 0;
  error = targetValue - currentValue;
  static int last_error = error;
  pidTerm = (Kp * error) + (Kd * (error - last_error));
  last_error = error;
  return constrain(int(pidTerm), -400, 400);
}

// compute desired PWM value for Motor 2 output
int updatePid2(int command, int targetValue, int currentValue)
{

  float pidTerm = 0; // PD correction
  int error = 0;
  error = targetValue - currentValue;
  static int last_error = error;
  pidTerm = (Kp * error) + (Kd * (error - last_error));
  last_error = error;
  return constrain(int(pidTerm), -400, 400);
}

// interrupt routine to maintian Motor 1 rotation value
void rencoder1()
{
  if(PINB & 0b00001000) count1++; // Pin 11 falling edge
```

```
  else                  count1--;
}

// interrupt routine to maintian Motor 1 rotation value
void rencoder2()
{
  if(PINB & 0b00010000) count2++; // Pin 12 falling edge
  else                  count2--;
}

// parameter current state, waits for "0000" to increment input and return
int waitForStart(int cState)
{
  while (true) {
    if(Serial.available())
    {
      // get the new byte:
      char inChar = (char)Serial.read();
      // add it to the inputString:
      inputString += inChar;
      // if the input string is "0000", commence CSV reading
      if (inputString.substring(inputString.length()-5,
      inputString.length()) == "0000\n") {
        break;
      }
    }
  }
  inputString = "";

  return cState + 1;
```

```
}

// prints "X, Y" out over the serial port
void pXY(int x, int y)
{
  Serial.print(String(x)); Serial.print(", "); Serial.println(String(y));
  Serial.flush();
}
```

D.2 C-Interfacing Code

```
/*
 * spc
 * --------------
 * Author: Joshua A. Zimmer
 * Version: 1.0
 * Date: 26 April 2015
 *
 * Significant Adaptations From:
 *    * The arduino-serial library
 *
 * A simple command-line program to send CSV file count data to
 * an Arduino board. Works on any POSIX system (Mac/Unix/PC)
 *
 * Compiles with something like:
 *   gcc -o spc arduino-serial-lib.c arduino-serial.c
 * or use the included Makefile
 *
 * Mac: make sure you have Xcode installed
 * Windows: try MinGW to get GCC
```

```
 *
 */

#include <stdio.h>    // Standard input/output definitions
#include <stdlib.h>
#include <string.h>
#include <string.h>   // String function definitions
#include <unistd.h>   // for usleep()
#include <getopt.h>

#include "arduino-serial-lib.h"


void usage(void)
{
    printf("Usage: spc -b <bps> -p <serialport> [OPTIONS]\n"
    "\n"
    "Options:\n"
    "  -h, --help                 Print this help message\n"
    "  -b, --baud=baudrate        Baudrate (bps) of Arduino (default 9600)\n"
    "  -p, --port=serialport      Serial port Arduino is connected to\n"
    "  -q  --quiet                Don't print out as much info\n"
    "\n");
    exit(EXIT_SUCCESS);
}

// Prints out an error message corresponding to string parameter
void error(char * msg)
{
    fprintf(stderr, "%s\n",msg);
```

```
    exit(EXIT_FAILURE);
}

// Opens the serial port at the specified baudrate
int openPort(const char* serialport, int baudrate, int quiet)
{
    int fd = -1;
    fd = serialport_init(serialport, baudrate);
    if( fd==-1 ) error("couldn't open port");
    if(!quiet) printf("opened port %s\n",serialport);
    serialport_flush(fd);

    return fd;
}

// Recieves output fron the serial port
int getOutput(int fd, char* buf, const int buf_max, int timeout, int quiet)
{
    if( fd == -1 ) error("serial port not opened");
    memset(buf,0,buf_max);
    serialport_read_until(fd, buf, '\n', buf_max, timeout);
    if( !quiet ) printf("read string:");
    printf("%s", buf);
    return 0;
}


// Beginning of file = "0000", End of file = "FFFF", CSV included between
int rwStdIn(int fd, char* buf, const int buf_max, int timeout, int quiet)
{
```

```
int inputSize = 0;
int charLen = 0;
int rc = -1;
if(fd == -1) error("serial port not opened");

// Writes out "0000\n" until successfully recieved by Arduino
while((rc = serialport_write(fd,"0000\n")) < 0)
{
    printf("%i\n", rc);
    usleep(2); // wait until connection established
}

usleep(20); // wait 20 microseconds

// Continue in the file write loop until no information left to write
while(fgets(buf, buf_max, stdin)) {

    charLen += strlen(buf);

    // Wait if the data is too much for Arduino to handle
    if(inputSize >= 350)
    {
        usleep(10000000);
        charLen = 0;
        inputSize = 0;
    }
    // Wait for a little while if too many characters on serial buffer
    else if(charLen >= 64)
    {
        usleep(300000);
```

```
            charLen = 0;
        }
        // Brief wait between characters
        else
        {
            usleep(30000);
        }

        if( !quiet ) printf("send string:%s\n", buf);
        if((rc = serialport_write(fd, buf)) == -1)
        {
            error("error writing");
            usleep(2);
        }
        inputSize += 1;
    }

    // Writes out "FFFF\n" until successfully recieved by Arduino
    while((rc = serialport_write(fd,"FFFF\n")) < 0)
    {
        printf("%i\n", rc);
        usleep(2); // wait until confirmed sending
    }

    return inputSize;
}


int main(int argc, char *argv[])
{
```

```
const int buf_max = 256;

int fd = -1;
char serialport[buf_max];
int baudrate = 115200;          // default
char quiet = 0;
int timeout = 2000;
char buf[buf_max];

strcpy(serialport, "/dev/tty.usbmodemfa131"); // default

/* Parse options pt. 1*/
int option_index = 0, opt;
static struct option loptions[] = {
    {"help",       no_argument,       0, 'h'},
    {"port",       required_argument, 0, 'p'},
    {"baud",       required_argument, 0, 'b'},
    {"quiet",      no_argument,       0, 'q'},
    {NULL,         0,                 0, 0}
};

// Parse options pt. 2
while(1)
{
    opt = getopt_long (argc, argv, "hp:b:s:S:i:rFn:d:qe:t:",
                       loptions, &option_index);
    if (opt==-1) break;

    switch (opt)
    {
```

```
        case '0':
            break;
        case 'q':
            quiet = 1;
            break;
        case 'h':
            usage();
            break;
        case 'b':
            baudrate = strtol(optarg,NULL,10);
            break;
        case 'p':
            if( fd!=-1 ) {
                serialport_close(fd);
                if(!quiet) printf("closed port %s\n",serialport);
            }
            strcpy(serialport,optarg);
            break;
    }
}

int inputSize = 0;

// Open serial port at baud rate
fd = openPort(serialport, baudrate, quiet);

// Send initialization byte to serial port --
// wait for response, open input file, read/write to serial
inputSize = rwStdIn(fd,buf,buf_max,timeout,quiet);
```

```
    // Close input file
    // Send closing byte to serial port -- wait for response


    // Open output file
    // Output serial from Arduino to output file
    do
    {
        getOutput(fd,buf,buf_max,timeout,quiet); //REMOVED FOR DEBUG
    }while(*buf != 0);


    // Close output file
    // Close serial port
    serialport_close(fd);

    exit(EXIT_SUCCESS);
} // end main
```

Appendix E

Technical Specifications

E.1 Quanser System Specifications

Symbol	Name	Value	Units	Variation
K_t	Motor Torque Constant	**0.00767**	N·m	±12%
K_m	Back EMF Constant	**0.00767**	V/(rd/s)	±12%
R_m	Armature Resistance	**2.6**	Ω	±12%
K_{gi}	Gearbox Ratio (Internal)	**14:1**	N/A	
$K_{ge\text{-}low}$	External Low Gear Ratio	**1:1**	N/A	
$K_{ge\text{-}high}$	External High Gear Ratio	**5:1**	N/A	
J_m	Motor Inertia	**3.87 e-7**	kg·m^2	±10%
J_{tach}	Tachometer Inertia	**0.7 e-7**	kg·m^2	±10%
Jeq_{low}	Equivalent Low Gear Inertia	**9.3 e-5**	kg·m^2	±10%
Jeq_{high}	Equivalent High Gear Inertia	**2.0 e-3**	kg·m^2	±10%
Beq_{low}	Viscous Damping Coefficient (Low)	**1.5 e-3**	N·m/(rd/s)	±20%
Beq_{high}	Viscous Damping Coefficient (High)	**4.0 e-3**	N·m/(rd/s)	±20%
Eff_g	Gearbox Efficiency	**0.85**	N/A	±10%
Eff_m	Motor Efficiency	**0.69**	N/A	±5%
K_{pot}	Potentiometer Sensitivity	**35.2**	Deg/V	±2%
K_{Enc}	Encoder Resolution (E option)	**4096**	Counts/rev.	
K_{EncH}	Encoder Resolution (EHR option)	**8192**	Counts/rev.	
K_{tach}	Tachometer Sensitivity	**1.5**	V/1000RPM	±2%
M_{max}	Maximum Load on Output Shaft	**5**	kg	
f_{max}	Maximum Input Frequency	**50**	Hz	
V_{rated}	Rated Motor Voltage	**6**	V	

Figure E.1: Quanser SRV02 Rotary Servo Plant Specifications [15]

E.2 Arduino Rack and Pinion Mechanism Specifications

Table E.1: Pololu Dual MC33926 Motor Driver Shield Specifications [9]

Motor driver:	MC33926
Motor channels:	2
Minimum operating voltage:	5 V
Maximum operating voltage:	28 V
Current sense:	0.525 V/A
Maximum PWM frequency:	20 kHz
Minimum logic voltage:	2.5 V
Maximum logic voltage:	5.5 V

Table E.2: Pololu 12V, 30:1 Gear Motor w/ Encoder Specifications [1]

Gear ratio:	30:1
Free-run speed @ 12V:	350 rpm
Free-run current @ 12V:	300 mA
Stall current @ 12V:	5000 mA
Stall torque @ 12V:	110 oz-in
Free-run speed @ 6V:	175 rpm
Free-run current @ 6V:	250 mA
Stall current @ 6V:	2500 mA
Stall torque @ 6V:	55 oz-in
Lead length:	11 in
Counts per Revolution:	1920

Table E.3: Arduino Uno - R3 Specifications [2]

Microcontroller:	ATmega328
Operating Voltage	5V
Input Voltage (recommended):	7-12V
Input Voltage (limits):	6-20V
Digital I/O Pins:	14 (of which 6 provide PWM output)
Analog Input Pins:	6
DC Current per I/O Pin:	40 mA
DC Current for 3.3V Pin:	50 mA
Flash Memory:	32 KB (ATmega328) of which 0.5 KB used by bootloader
SRAM:	2 KB (ATmega328)
EEPROM:	1 KB (ATmega328)
Clock Speed:	16 MHz
Length:	68.6 mm
Width:	53.4 mm
Weight:	25 g