Title
Design, simulation, and control of a vertically balancing treaded rover

Permalink
https://escholarship.org/uc/item/6pf6j70w

Author
Grinberg, Matt R.

Publication Date
2009

Peer reviewed|Thesis/dissertation
UNIVERSITY OF CALIFORNIA, SAN DIEGO

Design, Simulation, and Control of a Vertically Balancing Treaded Rover

A thesis submitted in partial satisfaction of the requirements for the degree
Master of Science

in

Engineering Sciences (Mechanical Engineering)

by

Matt R. Grinberg

Committee in charge:
Professor Thomas R. Bewley, Chair
Professor Mauricio de Oliveira
Professor Frank E. Talke

2009
The thesis of Matt R. Grinberg is approved, and it is acceptable in quality and form for publication on microfilm:

__________________________________________

__________________________________________

__________________________________________

Chair

University of California, San Diego

2009
# TABLE OF CONTENTS

Signature Page ........................................ iii
Table of Contents .................................. iv
List of Figures ...................................... vi
Acknowledgements ................................... viii
Abstract ........................................... ix

Chapter 1  Introduction .............................. 1
  1.1 Shortfalls of Existing Designs ................. 2
  1.2 Initial Concept ................................ 3
     1.2.1 Viper Mode .............................. 4
     1.2.2 Cobra Mode ............................. 11

Chapter 2  Mechanical Design ...................... 12
  2.1 Design Constraints ............................ 13
  2.2 CAD Model .................................. 13
  2.3 Fabrication .................................. 18

Chapter 3  Dynamic Simulation ..................... 21
  3.1 Lagrangian Mechanics ........................ 21
     3.1.1 Defining Variables ....................... 22
     3.1.2 Equations of Motion ..................... 23
  3.2 Linearization ................................ 25
     3.2.1 Linear Equations of Motion .......... 26
  3.3 Linear Quadratic Regulators ................. 28
     3.3.1 Choosing Control Costs ................ 30
     3.3.2 Duality of Kalman Filter .............. 32
  3.4 Runge Kutta .................................. 34
     3.4.1 Identifying Parameters ................ 35
     3.4.2 Control Authority ....................... 38
     3.4.3 Disturbance Rejection ................. 40
     3.4.4 Upright Maneuver ....................... 41

Chapter 4  Control Implementation ................. 45
  4.1 Electronics .................................. 45
     4.1.1 Microcontroller ......................... 46
     4.1.2 Sensors ................................ 48
     4.1.3 Motor Drivers .......................... 51
  4.2 Design Modifications ........................ 52
4.3 LabView .............................................. 54
Chapter 5 Conclusion .................................. 55
5.1 Future Work ........................................ 55
Appendix A Motion Equations and Control Law Derivations ...... 57
Appendix B Open Loop Simulation .......................... 61
Appendix C Closed Loop Simulation ........................ 67
Appendix D Disturbance Rejection Simulation .................... 73
Appendix E Upright Maneuver Simulation ....................... 79
Bibliography ............................................. 85
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Existing Rover Designs</td>
<td>2</td>
</tr>
<tr>
<td>1.2</td>
<td>Initial Concept Design</td>
<td>3</td>
</tr>
<tr>
<td>1.3</td>
<td>Initial Concept Maneuvers</td>
<td>4</td>
</tr>
<tr>
<td>1.4</td>
<td>Viper Concept Design</td>
<td>5</td>
</tr>
<tr>
<td>1.5</td>
<td>V1 Schematic Design</td>
<td>6</td>
</tr>
<tr>
<td>1.6</td>
<td>V1 CAD Model</td>
<td>7</td>
</tr>
<tr>
<td>1.7</td>
<td>V1 Variable Diagram</td>
<td>8</td>
</tr>
<tr>
<td>1.8</td>
<td>V1 Logic Layout</td>
<td>10</td>
</tr>
<tr>
<td>1.9</td>
<td>V1 Wiring Diagram</td>
<td>10</td>
</tr>
<tr>
<td>1.10</td>
<td>Cobra Concept Design</td>
<td>11</td>
</tr>
<tr>
<td>2.1</td>
<td>V2 Concept Design</td>
<td>12</td>
</tr>
<tr>
<td>2.2</td>
<td>V2 CAD Model</td>
<td>13</td>
</tr>
<tr>
<td>2.3</td>
<td>VEX Tread Kit</td>
<td>14</td>
</tr>
<tr>
<td>2.4</td>
<td>High-torque Beetle Gearmotor</td>
<td>14</td>
</tr>
<tr>
<td>2.5</td>
<td>Beetle Gearmotor Placement</td>
<td>15</td>
</tr>
<tr>
<td>2.6</td>
<td>Main Geartrain Components</td>
<td>16</td>
</tr>
<tr>
<td>2.7</td>
<td>Slip Ring Diagram</td>
<td>16</td>
</tr>
<tr>
<td>2.8</td>
<td>Main Boom Components</td>
<td>17</td>
</tr>
<tr>
<td>2.9</td>
<td>Rechargeable Tenergy Batteries</td>
<td>17</td>
</tr>
<tr>
<td>2.10</td>
<td>Switchblade V2 Prototype</td>
<td>18</td>
</tr>
<tr>
<td>2.11</td>
<td>Prototype Fabrication Tools</td>
<td>19</td>
</tr>
<tr>
<td>3.1</td>
<td>Variables and Constants</td>
<td>23</td>
</tr>
<tr>
<td>3.2</td>
<td>Closed Loop Simulation</td>
<td>38</td>
</tr>
<tr>
<td>3.3</td>
<td>Open Loop Simulation</td>
<td>40</td>
</tr>
<tr>
<td>3.4</td>
<td>Open Loop Estimation</td>
<td>41</td>
</tr>
<tr>
<td>3.5</td>
<td>Closed Loop Estimation</td>
<td>41</td>
</tr>
<tr>
<td>3.6</td>
<td>Disturbance Rejection Estimation</td>
<td>42</td>
</tr>
<tr>
<td>3.7</td>
<td>Disturbance Rejection Simulation</td>
<td>43</td>
</tr>
<tr>
<td>3.8</td>
<td>Upright Maneuver Estimation</td>
<td>43</td>
</tr>
<tr>
<td>3.9</td>
<td>Upright Maneuver Simulation</td>
<td>44</td>
</tr>
<tr>
<td>4.1</td>
<td>System Block Diagram</td>
<td>46</td>
</tr>
<tr>
<td>4.2</td>
<td>NI Single-board RIO</td>
<td>46</td>
</tr>
<tr>
<td>4.3</td>
<td>Analog Input Modules</td>
<td>47</td>
</tr>
<tr>
<td>4.4</td>
<td>ALPS RDC80 Potentiometer</td>
<td>49</td>
</tr>
<tr>
<td>4.5</td>
<td>Analog Devices Gyroscope</td>
<td>50</td>
</tr>
<tr>
<td>4.6</td>
<td>Analog Devices Accelerometer</td>
<td>51</td>
</tr>
<tr>
<td>4.7</td>
<td>STMicroelectronics Motor Driver</td>
<td>52</td>
</tr>
<tr>
<td>4.8</td>
<td>Switchblade V3 Design</td>
<td>53</td>
</tr>
</tbody>
</table>
ACKNOWLEDGEMENTS

Thanks to Professor Bewley for his indispensable advice and counsel.
I am especially thankful to Chris Schmidt-Wetekam for his guidance and Kassi Marshall for her assistance.

This would not be possible without the 2008 team (Anson Brune, Scott Fall, and William Hayes) in getting the project started and the 2009 team (Omar Abolhosn, Colter Cederlof, Evan Coons, Devin Uehling, and William Wojciak) in bringing it to life.
ABSTRACT OF THE THESIS

Design, Simulation, and Control of a Vertically Balancing Treaded Rover

by

Matt R. Grinberg

Master of Science in Engineering Sciences (Mechanical Engineering)

University of California San Diego, 2009

Professor Thomas R. Bewley, Chair

An original design based on mechanical proficiency and algorithmic control of a continuous track rover enables it to perform self-balancing operations and large obstacle negotiations. While the tank-like treads are capable of overcoming basic terrain roughness, it is the rover’s ability to manipulate its center of mass that allows for unorthodox vertical balance. An extended rotating boom, housing heavy on-board batteries, can be pivoted in order to effectively change the location of the center of mass. A sequence of control laws takes advantage of this movement using sensor information from potentiometers, accelerometers, and gyroscopes to properly upright and stabilizes the vehicle. In order to accurately estimate the rover’s orientation, Kalman filtering techniques are applied to sensor measurements eliminating vibration and noise errors. A feedback gain is derived using a linear-quadratic regulator from a state-space linear model of the physical system and utilized to direct power to the motors. This enables both deliberate maneuvers and immediate response to impacts or other outside disturbances, making it possible for the rover to be agile and dexterous over various topographies and over-sized impediments. For processing, the NI single-board RIO allows for estimation and control computations to be implemented onboard in real time. Once programmed, the robot can function as a standalone unit capable of adjusting to its environment.
Chapter 1

Introduction

In combat situations, terrestrial exploration, and even some household chores often the best approach is to use a small unmanned vehicle for the task. Military bots can keep soldiers out of harms way on the battlefield and assist with extremely dangerous bomb and landmine diffusions. The diminutive size of exploratory rovers can allow them to investigate tight excavation and natural cave openings. Other rovers can be used in planetary studies to where human travel has not yet been developed. Personal assistant robots for the disabled can provide support and mobility while everyday jobs such as vacuuming, dusting, and lawn mowing can be automated. There are many other possible applications for maneuverable autonomous or semi-autonomous robotic systems, including scouting within burning buildings, monitoring of nuclear waste storage facilities, and fun inexpensive toys.

With each of these miniature vehicles, it is important that the basic platform be capable of negotiating its environment. Because of their small stature simply climbing stairs or trekking over debris can be troubling. Thus it becomes imperative that these platforms be proficient in overcoming obstacles almost as large as themselves while still carrying necessary cargo. The best method for solving this conundrum is by leveraging the existing payload for agility with an intelligent feedback control system. The goal of the research presented here is to find a realistic design for such a system platform using off the shelf components and easily manufactured parts.
1.1 Shortfalls of Existing Designs

Currently very few effective system solutions have been fielded to tackle obstacle negotiation. Existing designs either do not attempt this problem, like the household Roomba seen in Figure 1.1(d), or use brut force and size to get over its obstacles, like the PackBot seen in Figure 1.1(a) employed by the United States military. Bigger motors and more batteries are required in order to forcefully overpower obstructions, thus creating an even bulkier and more oversized construction. In this work, we will take an alternate avenue and design a vehicle that can leverage
its payload for agility rather than fall victim to it. Although some static stability will be sacrificed, the overall result greatly improves maneuverability, efficiency, and significantly reduced weight load.

1.2 Initial Concept

![Initial Concept Design](image)

Figure 1.2: Initial Concept Design

The original design was that of a treaded robot seen in Figure 1.2. It would be capable of going back and forth using the treads to maneuver like a tank. It would also have an extended boom with the payload, such as batteries or other nonmoving necessities, housed at the tip. The possible mobility of the vehicle can be seen in Figure 1.3(a). By pivoting the boom we could easily change the position of the center of mass. If the weight shift is done correctly, the tank portion of the robot could pop a wheelie and be able to drive over small obstacles as sketched out in Figure 1.3(b). However the best results would be gained by balancing in a vertical stance seen in Figure 1.3(c). By putting its center of mass over the edge of the obstruction, as seen in Figure 1.3(d), the robot would be able to climb onto obstacles nearly as tall as its length.

To improve the maneuverability of the original concept we place the boom
(a) Possible Motor Actuations  
(b) Overcoming Small Obstacles  
(c) Vertically Balancing Stance  
(d) Overcoming Large Obstacles

Figure 1.3: Initial Concept Maneuvers

pivot at one end of the tank. This allows for a longer boom arm and thus a larger extension of the payload. With a farther reaching offshoot we have greater command of the center of mass and overall movement.

The adjusted schematic of the robot, seen in Figure, resembles a switchblade automatic knife. The tank chassis represents the knife handle while the boom is the blade. The automated nature of raising the boom off the tank to gain vertical stability becomes analogous to the opening of a switchblade, thus giving the project its namesake: Switchblade.

1.2.1 Viper Mode

In order to set up the robot for large obstacle climbing, we must define the manner with which it can balance vertically. In one mode of operation, shown in Figure 1.4, the robot maintains stability on the back wheels of its treads. This was dubbed Viper mode because of the V shape formed by the tank and boom.
Changing this angle along with moving treads can manipulate the position of the tank with respect to the ground, thus allowing the robot to come up to obstacles at a desired incline. This approach is tested with mathematical modeling and a functioning prototype. Because this is the first version of the robot, it receives the name Switchblade V1.

The original concept, design, and prototype layout the steps we take with the second and third iterations of Switchblade. Although this first iteration is focused primarily on the Viper maneuver, the main ideas of Switchblade become solidified through this exercise. It gives us a greater understanding of the theoretical and practical complications involved with building a new unusual robot, thus we gain overall approaches when looking at future designs.

**V1 Mechanical Design**

In order to validate the robot’s ability to self-balance on its back axle, a physical system is constructed. The Switchblade V1 prototype is built from scratch to act as a specimen so that control algorithms could be tested on its platform. The mechanical design simplifies the original concept by only using wheels, instead of treads, to balance itself on one axle. A mechanical schematic for Switchblade V1 is shown in Figure 1.5. It includes a central boom and two arms to simulate the
weight of the tank chassis. The majority of the robot’s mass is placed on the boom as far away from the pivot point as possible. This allows for the robot to change its center of gravity more easily by simply moving the boom. Both arms of the tank chassis are designed to be rigidly connected through a shaft. The final CAD model can be seen in Figure 1.6.

The basic design consists of the large red weighted boom and the blue tank chassis. They are connected via a rigid axel. The large geared motor shown on the end of the boom is responsible for the motion of the boom around the rigid axel via a belt drive. The boom can twist around the axel via journal bearings press fit into the boom. Bearing boxes are on the outsides of the chassis for the wheels with the motors are attached to each of the wheels individually.

Many of the parts involved in the design are obtained in ready form, needing little alteration to be used effectively in the robot. These off the shelf components include the motors, belt drive pulleys, shafts, and bearings. Other pieces are manufactured from the CAD drawings using basic machine tools. We are able to assemble these parts together to build the framework of the robot and house all necessary components. This results in a complete working prototype.
V1 Dynamic Simulation

Control over the two-mass system can be implemented by first solving for the equations of motion involved. We identify our variables such that $\theta_1$ is the angle between vertical and the chassis, $\theta_2$ is the angle between vertical and the boom, $\theta_3$ is the rotation angle of the wheel. We also identify the necessary constants: $r$ is the radius of the wheel; $J_1$, $J_2$, and $J_3$ are the respective moments of inertia; $u_1$ is the torque applied from the tank to the treads and $u_2$ is the torque applied from the tank to the boom. This set of variables is shown in Figure 1.7, with the blue circles defining the centers of mass for the boom and the tank.
Figure 1.7: V1 Variable Diagram

\[ a_1 = J_1 + m_1 L_1^2 \]
\[ a_2 = J_2 + m_2 L_2^2 \]
\[ b_1 = m_1 r L_1 \]
\[ b_2 = m_2 r L_2 \]
\[ c_1 = m_1 g L_1 \]
\[ c_2 = m_2 g L_2 \]
\[ d = J_3 + (m_1 + m_2 + m_3) r^2 \]

(1.1)

We simplify the constants with Equations 1.1 so that we may derive the equations of motion of each mass and the treads. Using a method we will discuss later on in the text, we derive Equation 1.2 for the angular dynamics of the tank, Equation 1.3 for the angular dynamics of the boom, and Equation 1.4 for the horizontal dynamics.
\[ a_1 \ddot{\theta}_1 + b_1 \ddot{\theta}_3 \cos \theta_1 = c_1 \sin \theta_1 - u_1 - u_2 \quad (1.2) \]

\[ a_2 \ddot{\theta}_2 + b_2 \ddot{\theta}_3 \cos \theta_2 = c_2 \sin \theta_1 + u_2 \quad (1.3) \]

\[ d \ddot{\theta}_3 + b_1 \ddot{\theta}_1 \cos \theta_1 + b_2 \ddot{\theta}_2 \cos \theta_2 = b_1 \dot{\theta}_1 \sin \theta_1 + b_2 \dot{\theta}_2 \sin \theta_2 + u_1 \quad (1.4) \]

These results can be linearized and put into state-space form as shown in Equation 1.5, so that the system can be easily used with built-in functions within Matlab. These equations are used to build a suitable controller for Viper mode actuation.

\[
\begin{bmatrix}
a_1 & 0 & b_1 & 0 & 0 & 0 \\
0 & a_2 & b_2 & 0 & 0 & 0 \\
b_1 & b_2 & d & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
\dot{\theta}_1 \\
\dot{\theta}_2 \\
\dot{\theta}_3
\end{bmatrix}
= \begin{bmatrix}
0 & 0 & 0 & c_1 & 0 & 0 \\
0 & 0 & 0 & 0 & c_2 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0
\end{bmatrix}
\begin{bmatrix}
\dot{\theta}_1 \\
\dot{\theta}_2 \\
\dot{\theta}_3
\end{bmatrix}
+ \begin{bmatrix}
-1 & -1 \\
0 & 1 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0
\end{bmatrix}
\begin{bmatrix}
u_1 \\
u_2
\end{bmatrix} \quad (1.5)
\]

To create the control system, it cannot be assumed that all of the state variables are available for feedback. The sensors do not measure the actual state directly, so therefore an observer is needed to estimate the unavailable variables. The estimated variables are used in the feedback. Two gain matrices must be used for this system: the first being the feedback gain matrix \(K_c\) which yields the desired characteristic equation and the second being the observer gain matrix, \(K_e\). Finally, the controller can be implemented with the actual robot through Simulink.

**V1 Control Implementation**

Each of the wheels is driven by a dedicated gear train and motor. A potentiometer capable of constant rotation is associated with each motor. The gears train has a one to one ratio so the potentiometer could be attached to the drive axel rather than the motor itself. The motor responsible for \(V\) motion is mounted
at the end of the boom to make the center of gravity easily adjustable. By spinning the motor, the belt moves the boom relative to the tank chassis. This motion is monitored by another potentiometer placed around the motor shaft. A gyro is placed on the boom to measure angular velocity. Lastly in order to measure translational accelerations, a two axis accelerometer is also placed on the boom.

The placement of all the logic onto the physical prototype of Switchblade V1 is shown in Figure 1.8. One umbilical carries power for all three motors, 6 volts for the drive motors and 9 volts for the boom motor. A second umbilical carries the logic wiring from the robot to the National Instruments I/O Connector Block responsible for interfacing with Simulink on a PC. Onboard the robot, all of the various sensor wiring is organized into an umbilical. The wire diagram for these components is shown in Figure 1.9.
The result shows us that even though the robot can balance in Viper mode, it is not the best option for overcoming obstacles. The maneuver to lift the tank chassis off the ground is cumbersome and does not lift the center of mass high enough to get over barriers.

1.2.2 Cobra Mode

![Figure 1.10: Cobra Concept Design](image)

The other alternative for vertical balance is to stabilize on the front wheel of the treads. As shown in Figure 1.10, the resulting stance forms the letter C, thus christening it Cobra mode. With this method the robot would be able to fully extend the boom, lifting the center of mass much higher than with the Viper mode. This allows for a much easier way to overcome large obstacles. We are interested in studying this maneuver in particular, through the development of Switchblade V2.
Chapter 2

Mechanical Design

Figure 2.1: V2 Concept Design

In order to facilitate a working prototype for Switchblade V2, detailed concept schematics and CAD models must be produced. Since Switchblade V2 will now focus on the Cobra maneuver several key design changes become immediately apparent. As shown in Figure 2.1, two motors drive the treads allowing for forward, reverse, and turning capabilities. A third motor is geared down to provide necessary torque to a long shaft which rotates two boom arms. These arms hold the heavy batteries that supply power to both the motors and the onboard electronics.
2.1 Design Constraints

For Switchblade V2 to function properly and be able to perform its up righting maneuver, the center of mass must be placed such that while horizontal it is already near the contact point of the rear tread wheel and the ground. However this weight distribution is not simple. We are restricted by the physical weight of available motors and their respective torques.

In order to pull the center of mass towards the desired location more mass must be added to the end of the boom. This puts an additional torque load on the V motion motor. A more powerful, heavier, motor must be chosen to be able to lift the boom effectively. The additional weight of the motor shifts the center of mass closer to the rear tread wheels, detracting from the original intent. This dilemma has to be resolved with a careful motor selection and gear-train design, as the right amount of torque must be balanced with the appropriate weight distribution.

2.2 CAD Model

![V2 CAD Model](image)

Figure 2.2: V2 CAD Model
Figure 2.2 shows the finished CAD model for Switchblade V2. It considers weight distribution and manufacturability in order to satisfy the requirements for fabrication and performance. The design incorporates off the shelf part necessary to for the robot to function, including treads, motors, and batteries. Some parts needing to be modified in order to be better integrated with other components.

The pronounced tank treads seen in the CAD model are part of a VEX Tank Tread Kit shown in Figure 2.3. Every section of these treads is a master link, enabling us to create treads of any length. With each link able to grip the surface over which it is traveling, these treads enable the robot to explore much more demanding terrain than ordinary wheels. The width of the treads increases their surface area giving the robot more traction for hauling heavy loads.

Figure 2.4: High-torque Beetle Garmotor
We utilize the sleek miniature design of Beetle Gearmotors, seen in Figure 2.4, to balance the necessary torque output with acceptable physical weight. Two different kinds of Beetle Gearmotors are used for actuation in the robot. A pair of Beetle B104 Gearmotors, emphasized in Figure 2.5(a), powers the tread movement. Both motors come with an inline 104:1 gear ratio planetary transmission and a 4mm output shaft. The motors are mounted to the side plates of the chassis by four screws on their front faces. Using two independently powered motors, one on each tread, allows the robot to turn right and left in place.

![Figure 2.5: Beetle Gearmotor Placement](image)

A second type of Beetle Gearmotor, a B231 emphasized in Figure 2.5(b), is used for pivoting the boom about the tank chassis. Although the attached gearbox for the motor has a large 231:1 gear ratio, an additional 3:1 gear-train is used to supply enough torque to lift the heavy boom. The spur gear set up is highlighted in Figure 2.6(a). Along with providing a gear reduction, it also transfers the power of the motor to the shaft, pointed out in Figure 2.6(b), which is responsible for rotating the boom. The shaft runs through the bearings located at the rear tread wheels and comes out on both sides of the tank chassis to link to the boom arms.

The shaft also acts to provide power to the robot through a slip ring, shown in
Figure 2.6: Main Geartrain Components

Figure 2.7. The wire enters the shaft, the bearing keeps the shaft aligned, and the wire exists through a second set of holes. It is stripped and wrapped around the non-conductive Delrin shell. A copper ring keeps the stripped wire in place, while conducting electricity to the brush and spring system that leads into the rest of the robot. In this way we can provide power to both the motors and the electronics.

Figure 2.7: Slip Ring Diagram

The main function of the shaft, however, is to transmit torque to the boom
arms located on the outside of the treads. These arms, highlighted in Figure 2.8(a), support the battery case and essentially define the length of boom. They are designed to be aluminum L beams in order to resist twisting without adding unnecessary weight to the robot.

![Boom Arm Pair and SubC NiMH Batteries](image)

Figure 2.8: Main Boom Components

The arms hold the battery case at the end of the boom, where the batteries sit as highlighted in Figure 2.8(b).

![Rechargeable Tenergy Batteries](image)

Figure 2.9: Rechargeable Tenergy Batteries

The Tenergy rechargeable batteries, shown in Figure 2.9, are used to provide power to the robot. Each battery is able to supply 1.2 volts so that a total of 14
batteries organized in series are needed to supply a minimum of 12 volts to the motors.

Several of the premade components need to be modified in order to fit the design properly. Specifically the VEX drive wheels need tailoring. The front wheels have attachments to motor hubs so they can transmit power to the treads. The back wheels, on the other hand, have to accommodate an axle bearing to support the main boom shaft. Most other pieces of the robot are designed for flat sheets of Delrin for manufacturing ease. For instance, by incorporating potentiometers into the side mountings of the motors, we eliminate the need for addition housing mechanisms, thus maintaining a compact design.

2.3 Fabrication

![Figure 2.10: Switchblade V2 Prototype](image)

Along with supplying premade components, we must manufacture all other
parts of the robot, including the chassis, housings, shafts, arms, and gears. The tools that are used in this practice allow us to produce parts quickly and efficiently, so that we can adjust to any design changes that may occur. Aside from basic hand tools, the lathe, mill, and laser cutter facilitate the means of fabrication. The first prototype of Switchblade V2 is shown in Figure 2.10.

The LaserCAMM machine, shown in Figure 2.11(a), is the most instrumental tool in aiding quick, clean, and customized part production for the robot. The system uses a laser beam to cut and scribe a variety of sheet materials into intricate patterns. Any shape that is drawn on in CAD can be precisely cut relatively fast compared to other methods, like the band saw featured in Figure 2.11(b).

![Prototype Fabrication Tools](image)

(a) Design Studio LaserCAMM   (b) Walker-Turner Band Saw
(c) Bridgeport Vertical Mill   (d) Hardinge Precision Lathe

Figure 2.11: Prototype Fabrication Tools

Flat sheets of Delrin plastic are the most common material on the robot because of its strength and superior machining properties. The use of the LaserCAMM makes it possible to incorporating potentiometers into the side mountings of the
motors. This requires square cut outs, which would otherwise be difficult and time consuming to be performed through other means. Even complicated shapes, like the gears shown in Figure 2.6(a), are made of Delrin. The custom design allows specialized bores to be made in each gear. It also allows for easy reproductions in case individual pieces fail, or a redesign is made.

After pieces are cut out on the LaserCAMM, further treatment is done on the Bridgeport vertical mill, shown in Figure 2.11(c). With the spindle axis oriented vertically, the milling cutters are held in the spindle and rotate on its axis. The spindle can be extended allowing plunge cuts and drilling. In the turret Bridgeport mill the spindle remains stationary during operation while the table is moved to accomplish cutting. The side holes in the Delrin sheet pieces are made with this mill. Other modification and parts are also made on this multifunctional tool.

For other circular cuts we use a Hardinge precision lathe, seen in Figure 2.11(d). By spinning a held piece of material while applying a cutting tool to it, we can create objects which have symmetry about an axis of rotation. The lathe is used to make the main shaft used for pivoting the boom. Aside from forming the correct outside diameter, an inside bore was also made as part of the slip ring design, seen in Figure 2.7. All VEX drive wheel modifications are also made on the lathe. The wheels original inner components are bored out so the proper housing can be placed inside.
Chapter 3

Dynamic Simulation

In order to ensure that the design is realizable we must perform a full mathematical simulation. We must identify the physical and dynamic characteristics governing the system as well as the appropriate control laws which will enable Switchblade V2 to perform the desired Cobra maneuver. To do this, we can simplify our problem by looking at a 2D depiction, assuming symmetry between the two treads.

3.1 Lagrangian Mechanics

The equations of motion of a moving system can often be derived in a simple manner in terms of generalized coordinates by the use of Lagrange’s equations. For an \( n \) degree of freedom system Lagrange’s equations can be stated as Equation 3.1, where \( T \) is the total kinetic energy, \( U \) is the total potential energy, \( \dot{q}_j = \frac{\partial q_j}{\partial t} \) is the generalized velocity, and \( G^{(n)}_j \) is the neoconservative generalized force corresponding to the generalized coordinate \( q_j \).

\[
\frac{d}{dt} \left( \frac{\partial T}{\partial \dot{q}_j} \right) - \frac{\partial T}{\partial q_j} + \frac{\partial U}{\partial q_j} = G^{(n)}_j, \quad j = 1, 2, ..., n \tag{3.1}
\]

The forces represented by \( P^{(n)}_j \) may be damping forces or other external forces that are not derivable from a potential function. If \( F_{xk} \), \( F_{yk} \), and \( F_{zk} \) represent the external forces acting on the \( k \)th mass of the system in the \( x \), \( y \), and \( z \) directions,
respectively, then the generalized force \( G^{(n)}_j \) can be computed by using Equation 3.2.

\[
G^{(n)}_j = \sum_k \left( F_{xk} \frac{\partial x_k}{\partial q_j} + F_{yk} \frac{\partial y_k}{\partial q_j} + F_{zk} \frac{\partial z_k}{\partial q_j} \right)
\] (3.2)

For a torsion system, the force \( F_{ik} \) is replaced by \( M_{ik} \), the moment acting about the \( i \) axis. The displacement \( i_k \) is replaced by \( \theta_{ik} \), the angular displacement about the \( i \) axis. The computation takes the form of Equation 3.3.

\[
G^{(n)}_j = \sum_k \left( M_{xk} \frac{\partial \theta_{xk}}{\partial q_j} + M_{yk} \frac{\partial \theta_{yk}}{\partial q_j} + M_{zk} \frac{\partial \theta_{zk}}{\partial q_j} \right)
\] (3.3)

Equation 3.1 represents a system of \( n \) differential equations, one corresponding to each of the \( n \) generalized coordinated. Thus the equations of motion of the system can be derived, provided the energy expressions are available.

### 3.1.1 Defining Variables

In order to derive our equations of motion, we must first define the parameters of our system. The physical characteristic such as the lengths, masses, and inertias are unchanging constants. \( m_1 \) and \( J_1 \) are the mass and inertia of the boom while \( m_2 \) and \( J_2 \) are the mass and inertia of the tank. The lengths of the boom and the tank are \( L_b \) and \( L_t \). The radius of the tread wheels is \( r \). Additionally, \( L_1 \) is the measured distance between the boom pivot and the center of mass of the boom, \( L_2 \) is the measured distance between the axis of the grounded tread wheel and the center of mass of the tank, and \( r \) is the radius of the tread wheels. Next we define the generalized coordinates of the system. \( \alpha \) is the angle between the ground and the tank, \( \phi \) is the position of the tread wheels with respect to the tank, and \( \theta \) is the angle between the tank and the boom. Lastly, the system input are defined as the two rotating elements that can be actuated via the onboard motors. \( u_1 \) acts on the angle between the boom and the tank, while \( u_2 \) actuates the tread wheels. Figure 3.1 shows a clear diagram of the listed characteristic.
3.1.2 Equations of Motion

Now that our variables and constants are defined, we characterize the position of the two centers of mass in Cartesian coordinates. Equations 3.4 and 3.5 are associated with the boom, while Equations 3.6 and 3.7 are associated with the tank.

\[
x_1 = -r(\alpha - \phi) + L_t \cos \alpha - L_1 \cos (\theta - \alpha) \\
y_1 = r + L_t \sin \alpha + L_t \sin (\theta - \alpha)
\]  
\[
x_2 = -r(\alpha - \phi) + L_2 \cos \alpha \\
y_2 = r + L_2 \sin \alpha
\]  

Next we compute the velocities for both the boom and the tank. In order to put into a convenient form, we square the velocities as shown Equations 3.8 and 3.9.
\[ v_1^2 = \left( \frac{dx_1}{dt} \right)^2 + \left( \frac{dy_1}{dt} \right)^2 \]  
\[ v_2^2 = \left( \frac{dx_2}{dt} \right)^2 + \left( \frac{dy_2}{dt} \right)^2 \]  

(3.8)  

The translational kinetic energy associated with rectilinear motion of a body with constant mass \( m_i \), whose center of mass is moving at speed \( v \), is \( T = \frac{1}{2} m_i v^2 \). Similarly, when the same rigid body is rotating about the center of mass, it has a rotational kinetic energy given by \( T = \frac{1}{2} J_i \omega^2 \), where \( \omega \) is the body’s angular velocity and \( J_i \) is the body’s moment of inertia. Using these concepts from classical mechanics, we compute the total kinetic energy of the system in Equation 3.10.

\[ T = \frac{1}{2} m_1 v_1^2 + \frac{1}{2} m_2 v_2^2 + \frac{1}{2} J_1 \theta^2 + \frac{1}{2} J_2 \alpha^2 \]  

(3.10)

The potential energy of the system, when accounting for mass, gravity, and altitude, is expressed in Equation 3.11, where \( g \) is standard gravity.

\[ U = m_1 g y_1 + m_2 g y_2 \]  

(3.11)

By applying Equation 3.1, we can now use our three variable angles to define the Lagrange’s equations. Each of these angles describes a different movement of the system. Thus we can express the dynamics of the boom with Equation 3.12, the dynamics of the tank with Equation 3.13, and the horizontal dynamics with Equation 3.14.

\[ \frac{d}{dt} \left( \frac{\partial T}{\partial \alpha} \right) - \frac{\partial T}{\partial \alpha} + \frac{\partial U}{\partial \alpha} = G_{\alpha} \]  

(3.12)

\[ \frac{d}{dt} \left( \frac{\partial T}{\partial \phi} \right) - \frac{\partial T}{\partial \phi} + \frac{\partial U}{\partial \phi} = G_{\phi} \]  

(3.13)

\[ \frac{d}{dt} \left( \frac{\partial T}{\partial \theta} \right) - \frac{\partial T}{\partial \theta} + \frac{\partial U}{\partial \theta} = G_{\theta} \]  

(3.14)

The next step is to define the generalized forces \( G_{\alpha}, G_{\phi}, \) and \( G_{\theta} \). To do this we must accurately model the robot’s motor torque output as a function of voltage inputs \( u_1 \) and \( u_2 \). We identify the stall torques as \( s_v \) for the \( V \) motion motor and
s_w for both of the wheel motors. We also identify the total damping constants as 
\( b_v \) for the V motion motor and \( b_w \) for the wheel motors. With the maximum power 
as \( \mathcal{E} \) and motor resistance as \( \Omega \), we can write the motor models as Equations 3.15 
and 3.16.

\[
\tau_v = \frac{s_v}{\mathcal{E}} u_1 + \frac{b_v^2}{\Omega} \dot{\theta} \quad (3.15)
\]

\[
\tau_w = 2 \left( \frac{s_w}{\mathcal{E}} u_2 + \frac{b_w^2}{\Omega} \dot{\phi} \right) \quad (3.16)
\]

We can write out the full equations of motion by combining these models with 
the kinetic friction at the wheels, \( f \), which inherently depends on wheel speed 
\( \dot{\alpha} - \dot{\phi} \). Equation 3.17 describes the dynamics of the boom, Equation 3.18 describes 
the dynamic of the tank, and Equation 3.19 describes the horizontal dynamic.

\[
\frac{d}{dt} \left( \frac{\partial T}{\partial \dot{\alpha}} \right) - \frac{\partial T}{\partial \alpha} + \frac{\partial U}{\partial \alpha} = -f \quad (3.17)
\]

\[
\frac{d}{dt} \left( \frac{\partial T}{\partial \dot{\phi}} \right) - \frac{\partial T}{\partial \phi} + \frac{\partial U}{\partial \phi} = f - \tau_w \quad (3.18)
\]

\[
\frac{d}{dt} \left( \frac{\partial T}{\partial \dot{\theta}} \right) - \frac{\partial T}{\partial \theta} + \frac{\partial U}{\partial \theta} = -\tau_v \quad (3.19)
\]

A Matlab code which following the presented procedure can be found in the 
Appendix. Both symbolic and individual cases can be run by this Matlab code to 
numerically compute the equation of motion.

### 3.2 Linearization

We consider the linearization of the nonlinear ordinary differential equation 
\( \dot{x}_t = f(x_t, u_t) \). Using Taylor’s theorem to linearize the solution about a current 
(\( \bar{x}_t, \bar{u}_t \)) value, we can write \( x_t = \bar{x}_t + \tilde{x}_t, \ u_t = \bar{u}_t + \tilde{u}_t \), and expand \( f \) as Equation 
3.20.

\[
\dot{x}_t = \dot{\bar{x}}_t + \tilde{x}_t = f(\bar{x}_t, \bar{u}_t) + \left. \frac{\partial f}{\partial x} \right|_{\bar{x}, \bar{u}} \bar{x}_t + \left. \frac{\partial f}{\partial u} \right|_{\bar{x}, \bar{u}} \bar{u}_t \quad (3.20)
\]
This ordinary differential equation describes how the solution $x_t$ evolves with control $u_t$ in comparison with the nominal solution $\bar{x}_t$ from control $\bar{u}_t$. Equation 3.21 shows us the linearized system.

$$\dot{x}_t = \frac{\partial f}{\partial x}\bigg|_{\bar{x},\bar{u}} \bar{x}_t + \frac{\partial f}{\partial u}\bigg|_{\bar{x},\bar{u}} \bar{u}_t + f(\bar{x}_t, \bar{u}_t) - \dot{\bar{x}}_t$$ (3.21)

The nominal solution enters as a driving function of time to the linearization. For equilibrium or steady-state solutions, we can write $\dot{\bar{x}} = 0$ and $f(\bar{x}_t, \bar{u}_t) = 0$, so that there are no additive terms. Taking the current control value as $\bar{u}_t$ and current state estimate as $\bar{x}_t$, there will be an additive driving term $f(\bar{x}_t, \bar{u}_t) = 0$ to linearize the ordinary differential equation. This term will be a constant over the timeframe of evolution of the linearized system. The control solution $\bar{u}_{t+\tau}$ is an additive to the constant current input, so that $u_{t+\tau} = \bar{u}_t + \bar{u}_{t+\tau}$.

### 3.2.1 Linear Equations of Motion

In order to linearize our equations of motion, we must rewrite them into the matrix form seen in Equation 3.22.

$$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{d}{dt} \left( \frac{\partial T}{\partial \alpha} \right) - \frac{\partial T}{\partial \alpha} + \frac{\partial U}{\partial \alpha} + f \\ \frac{d}{dt} \left( \frac{\partial T}{\partial \phi} \right) - \frac{\partial T}{\partial \phi} + \frac{\partial U}{\partial \phi} - f + \tau_w \\ \frac{d}{dt} \left( \frac{\partial T}{\partial \theta} \right) - \frac{\partial T}{\partial \theta} + \frac{\partial U}{\partial \theta} + \tau_v \end{bmatrix} = O_{(3\times1)}$$ (3.22)

We must also rewrite the generalized coordinates into vector form, shown in Equation 3.23.

$$q_{(3\times1)} = \begin{bmatrix} \alpha \\ \phi \\ \theta \end{bmatrix}$$ (3.23)

Along with the generalized coordinates, we can also define the state vector and control vector of our system as shown in Equations 3.24 and 3.25.
\[ x_{(6\times1)} = \begin{bmatrix} \dot{q}_{(3\times1)} \\ q_{(3\times1)} \end{bmatrix} \quad (3.24) \]

\[ u_{(2\times1)} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \quad (3.25) \]

Next we identify the acceleration terms in Equation 3.26, in order to separate them out.

\[ E_{(3\times3)} = \frac{\partial O_{(3\times1)}}{\partial \ddot{q}_{(3\times1)}} \quad (3.26) \]

We can now write our equations of motion in the form of Equation 3.27, where \( N_{(3\times1)} = E_{(3\times3)} \ddot{q}_{(3\times1)} - O_{(3\times1)} \). In this form the right hand side of the equation is not dependent on \( \ddot{q} \), but rather on \( x, u, \) and \( t \).

\[ E\ddot{q} = N \quad (3.27) \]

With the equations of motion now presented as a differential equation, we can linearize about a point of equilibrium. In our system that point is when the robot is perfectly vertical and not moving, that is \( x_0 = [0, 0, 0, \pi/2, \pi/2, \pi]^T \). In Equations 3.28, 3.29, and 3.30 we evaluate coefficients of the linearized dynamics about \( x_0 \).

\[ \bar{E}_{(6\times6)} = \frac{\partial [E \ddot{q}, \dot{q}]^T}{\partial \ddot{x}} \bigg|_{x_0} \quad (3.28) \]

\[ \bar{A}_{(6\times6)} = \frac{\partial [N, \dot{q}]^T}{\partial x} \bigg|_{x_0} \quad (3.29) \]

\[ \bar{B}_{(6\times2)} = \frac{\partial [N, \dot{q}]^T}{\partial u} \bigg|_{x_0} \quad (3.30) \]

When these coefficients are put together we can write our linearized equations of motion in the form shown in Equation 3.31.

\[ \bar{E}\ddot{x} = \bar{A}x + \bar{B}u \quad (3.31) \]
3.3 Linear Quadratic Regulators

A most effective and widely used technique of linear control system design is the optimal linear quadratic regulator. An advantage of the quadratic optimal control method over the pole-placement method is that the former provides a systematic way of computing the state feedback control gain matrix. Using this method we consider a problem that determines the matrix $K_c$ of the optimal control vector defined in Equation 3.33 given the system dynamics of Equation 3.32. An optimal solution can be found by minimizing the performance cost shown in Equation 3.34, which accounts for the expenditure of the energy of the control signals. The matrices $Q_c$ and $R_c$ determine the relative importance of the error and the expenditure of this energy.

\[
\dot{x} = Ax + Bu 
\]

\[
u = -K_c x
\]

\[
J_c = \int_0^\infty (x^T Q_c x + u^T R_c u) dt
\]

Substituting Equation 3.33 into Equation 3.32, we obtain Equation 3.35.

\[
\dot{x} = Ax - BK_c x = (A - BK_c) x
\]

Assuming that the matrix $A - BK_c$ is stable, with its eigenvalues having negative real parts, we first substitute Equation 3.33 into Equation 3.34 yields Equation 3.36.

\[
J_c = \int_0^\infty (x^T Q_c x + x^T K^T_c R_c K_c x) dt = \int_0^\infty x^T (Q_c + K^T_c R_c K_c) x dt
\]

Next we define $X$ such that it stratifies Equation 3.37.

\[
x^T (Q_c + K^T_c R_c K_c) x = \frac{d}{dt} (x^T X x)
\]

We are then able to obtain Equation 3.38.
\[
\frac{d}{dt}(x^TX) = -x^TXx - x^T \dot{x} = -x^T[(A - BK_c)^T X + X(A - BK_c)]x \quad (3.38)
\]

Combining Equation 3.37 and Equation 3.38, we can write Equation 3.39. It can be proved that since \(A - BK_c\) is a stable matrix, there must exist a positive-definite matrix \(X\) that satisfies Equation 3.39.

\[
(A - BK_c)^T X + X(A - BK_c) = -(Q_c + K_c^T R_c K_c) \quad (3.39)
\]

Assuming \(R_c = R_c^{T/2} R_c^{1/2}\) where \(R_c^{1/2}\) is a nonsingular matrix, we can rewrite Equation 3.39 first as Equation 3.40 and then as Equation 3.41.

\[
(A^T - K_c^T B^T) X + X(A - BK_c) + Q_c + K_c^T R_c^{T/2} R_c^{1/2} K_c = 0 \quad (3.40)
\]

\[
A^T X + X A + [R_c^{1/2} K_c - R_c^{-T/2} B^T X]^T[R_c^{1/2} K_c - R_c^{-T/2} B^T X] - X B R_c^{-1} B^T X + Q_c = 0 \quad (3.41)
\]

The minimization of \(J_c\) with respect to \(K_c\) requires the minimization of \(x^T [R_c^{1/2} K_c - R_c^{-T/2} B^T X]^T[R_c^{1/2} K_c - R_c^{-T/2} B^T X] x\). Since this expression is nonnegative, the minimum occurs when it is zero or when Equation 3.42 holds true.

\[
R_c^{1/2} K_c = R_c^{-T/2} B^T X \quad (3.42)
\]

Thus Equation 3.43 gives the optimal matrix \(K_c\).

\[
K_c = R_c^{-1/2} R_c^{-T/2} B^T X = R_c^{-1} B^T X \quad (3.43)
\]

The optimal control law is given by Equation 3.44.

\[
u = -K_c x = -R_c^{-1} B^T X x \quad (3.44)
\]

The matrix \(X\) must satisfy Equation 3.39 or the reduced-matrix Riccati Equation 3.45.

\[
A^T X + X A - X B R_c^{-1} B^T X + Q_c = 0 \quad (3.45)
\]
3.3.1 Choosing Control Costs

In order to get our linearized equation of motion into the form of Equation 3.32 we define $A = \bar{E}^{-1}\bar{A}$ and $B = \bar{E}^{-1}\bar{B}$. We are then able to write Equation 3.46 as the linearized dynamic model of Switchblade V2.

\[
\dot{x} = \bar{E}^{-1}\bar{A}x + \bar{E}^{-1}\bar{B}u = Ax + Bu \tag{3.46}
\]

Next we must choose $Q_c$ and $R_c$ to fulfill Equation 3.34 so that we may solve the linear quadratic regulator problem of Equations 3.44 and 3.45 for $K_c$. One reasonable method to obtain acceptable values of $x$ and $u$ through design interactions is suggested by Bryon’s rule. An appropriate choice is to initially choose diagonal matrices $Q_c$ and $R_c$ such that they reflect the maximum acceptable values of $x$ and $u$ by satisfying Equations 3.47 and 3.48.

\[
Q_{(ii)} = 1/|x_i|_{max}^2 \tag{3.47}
\]
\[
R_{(ii)} = 1/|u_i|_{max}^2 \tag{3.48}
\]

In our system the states representing velocities are limited by motor speeds. Defining the no load speed of the V motor as $v_v[\text{rpm}]$ and the tread motors as $v_w[\text{rpm}]$, we can write the maximum values of the velocities states in Equations 3.49, 3.50, and 3.51. We can also define the maximum value of the position states in Equations 3.52, 3.53, and 3.54.

\[
[\dot{\alpha}]_{max} = v_v[\text{rpm}] \times (2\pi[\text{rad/rev}]/(60[\text{sec/min}])) \tag{3.49}
\]
\[
[\dot{\phi}]_{max} = v_w[\text{rpm}] \times (2\pi[\text{rad/rev}]/(60[\text{sec/min}])) \tag{3.50}
\]
\[
[\dot{\theta}]_{max} = v_v[\text{rpm}] \times (2\pi[\text{rad/rev}]/(60[\text{sec/min}])) \tag{3.51}
\]
\[
[\alpha]_{max} = (\pi/2)[\text{rad}] \tag{3.52}
\]
\[
[\phi]_{max} = \pi[\text{rad}] \tag{3.53}
\]
\[
[\theta]_{max} = (\pi/2)[\text{rad}] \tag{3.54}
\]

Similarly we identify the maximum control signal inputs as the maximum motor power with Equation 3.55.
\[ [u_1]_{max} = [u_2]_{max} = \mathcal{E} \]  

We can then compose the matrices \( Q_c \) and \( R_c \) as shown in Equations 3.56 and 3.57.

\[
Q_c = \begin{bmatrix}
\left( \frac{30}{v_c \pi} \right)^2 & 0 & 0 & 0 & 0 & 0 \\
0 & \left( \frac{30}{v_w \pi} \right)^2 & 0 & 0 & 0 & 0 \\
0 & 0 & \left( \frac{30}{v_c \pi} \right)^2 & 0 & 0 & 0 \\
0 & 0 & 0 & \left( \frac{2}{\pi} \right)^2 & 0 & 0 \\
0 & 0 & 0 & 0 & \left( \frac{1}{\pi} \right)^2 & 0 \\
0 & 0 & 0 & 0 & 0 & \left( \frac{2}{\pi} \right)^2 \\
\end{bmatrix} \tag{3.56}
\]

\[
R_c = \begin{bmatrix}
\mathcal{E}^{-2} & 0 \\
0 & \mathcal{E}^{-2} \\
\end{bmatrix} \tag{3.57}
\]

Both matrices now need to be modified to achieve an acceptable trade-off between system performance and control effort. Since the goal of the control law is to stabilize the system about the point of equilibrium we must consider which states are crucial to meeting that end. All of the velocities need to be driven to zero for the robot to be at rest. Also we must drive \( \alpha \) to be 90° so that the tank is perpendicular with the ground. This will naturally drive \( \theta \) to be 180° and the boom to be held vertically over the tank chassis, but it is not essential to command that angle. Similarly it is irrelevant where the robot is able to perform its maneuver, so the angle \( \phi \) has little overall influence on the system. With this information we can add a weight \( \rho \) to the appropriate states in Equation 3.56 so that we can write it as Equation 3.58.
3.3.2 Duality of Kalman Filter

To find an observer that will approximate our states from limited measurements, we utilize the inherent duality between optimal estimation and optimal control. As we have seen the optimal control problem begins with the system Equation 3.32 where \( u \) is the control variable. The optimal control problem tries to find the control \( u \) that minimizes the cost function seen in Equation 3.34. The matrices \( Q_c \) and \( R_c \) provide the specified weighting in performance. The optimal controller is given as Equations 3.45, 3.43, and 3.44.

The optimal estimation problem begins with the system Equation 3.59 and measurement Equation 3.60.

\[
\dot{x} = Ax + w_x, \quad \text{where} \quad w_x \sim N(0, Q_e) \tag{3.59}
\]
\[
y = Cx + w_y, \quad \text{where} \quad w_y \sim N(0, R_e) \tag{3.60}
\]

It tries to find the state estimate \( \hat{x} \) that minimizes the cost function Equation 3.61.

\[
J_e = \int_0^\infty E[(x - \hat{x})^T(x - \hat{x})]dt \tag{3.61}
\]

The Kalman filter is then given by Equations 3.62, 3.63, and 3.64.
\[0 = AY + Y A^T - YC^T R_e^{-1} CY + Q_e\]  
(3.62)

\[K_e = YC^T R_e^{-1}\]  
(3.63)

\[\dot{x} = A\hat{x} + K_e(y - C\hat{x})\]  
(3.64)

We solve the differential Riccati equation for \(Y\) to get the optimal estimator \(K_e\). There is a clear relationship to the optimal control solution. The differential equations have the same form, except \(A\) and \(B\) are replaced by \(A^T\) and \(C^T\). The estimator gain \(K_e\) and the controller gain \(K_c\) also have very similar forms. The \(Q_e\) and \(R_e\) covariance matrices in the estimation problem have duels in the cost function weighting matrices of the optimal control problem. When the optimal control and optimal estimation problems are combined the control \(u\) is now based off of the estimated value \(\hat{x}\), such as to adhere to Equation 3.65.

\[u = -K_e\hat{x}\]  
(3.65)

**Choosing Estimator Costs**

In order to choose the appropriate \(Q_e\) and \(R_e\) we define the measurement signal \(y\) through Equation 3.60. The coefficient matrix \(C\) identifies the knowledge we are able to gather from onboard sensors. Because we should have an accelerometer and gyro on the tank, we are able to read \(\alpha\) and its velocity \(\dot{\alpha}\). Potentiometers are also present on each motor, giving us information about the positions of the associated angles, \(\phi\) and \(\theta\). When combined, the \(C\) matrix can be expressed through Equation 3.66.

\[C_{(4\times6)} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}\]  
(3.66)

From Equation 3.59 we know that \(Q_e\) and \(R_e\) is the covariance of the system’s plant noise and measurement noise. Since we have no knowledge of this noise, we take the identity matrix as our covariance as seen in Equation 3.67.
\[ Q_e = I_{(6 \times 6)} \quad (3.67) \]

On the other hand, the covariance of the measurement can be quantified by the deviations of accuracy taken by our sensors. We can define the covariance in Equation 3.68 where \( w_p, w_g, \) and \( w_a \) is the variance of the potentiometer, gyroscope, and accelerometer measurements.

\[
R_e = \begin{bmatrix}
   w_g^2 & 0 & 0 & 0 \\
   0 & w_a^2 & 0 & 0 \\
   0 & 0 & w_p^2 & 0 \\
   0 & 0 & 0 & w_p^2
\end{bmatrix} \quad (3.68)
\]

### 3.4 Runge Kutta

In order to simulate a realistic response of our system we will use a predictor-corrector method. The idea is to advance the solution over a certain time interval \( h \) by executing a sequence of steps. Certain of these steps are provisional or exploratory, whereas others are educated, in the sense that they sensible correct rough estimates. The correction is based on knowledge that has been accumulated during the provisional or exploratory steps. In the Runge-Kutta method specifically we abandon all previous knowledge and collect information on the imminent motion by performing exploratory steps.

When the cost of a function evaluation is reasonable and extremely high accuracy is not required, the fourth order Runge-Kutta method is considered the best compromise. This is a genuine predictor-corrector method involving three provisional steps and one final educated step. As shown in Equations 3.69, 3.70, 3.71, and 3.72, we first compute four auxiliary quantities \( k_1, k_2, k_3, \) and \( k_4 \). We then add them together in the new value \( x_{n+1} \) as shown in Equation 3.73. The method is well suited because it needs no special starting procedure, makes light demand on storage, and repeatedly used the same straightforward computational procedure. It is also numerically stable when using a sufficiently small time step, with each complete step introducing a numerical error on the order of \( h^5 \).
\[ k_1 = h f(x_n, t_n) \]  
(3.69)

\[ k_2 = h f(x_n + \frac{1}{2}k_1, t_n + \frac{1}{2}h) \]  
(3.70)

\[ k_3 = h f(x_n + \frac{1}{2}k_2, t_n + \frac{1}{2}h) \]  
(3.71)

\[ k_4 = h f(x_n + k_3, t_n + h) \]  
(3.72)

\[ x_{n+1} = x_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \]  
(3.73)

From our equations of motion we define \( \bar{N} = E^{-1}N \) symbolically, so that with each computation of \( k \) we can numerically evaluate the current value of \( \bar{N} \). This way we can just calculate the value of one matrix, rather than inverting a matrix and then performing matrix multiplication. We are able to model the dynamics of the nonlinear system while incorporating the linear control \( u \), which is based upon the linearized system. Additionally to simulate actual physical constraints of the motors, the controls \( u_1 \) and \( u_2 \) are limited to \( \mathcal{E} \) regardless of the outcome of \( u = -K_c \dot{x} \).

### 3.4.1 Identifying Parameters

From the CAD model of Switchblade V2 we can define the constants in our equations of motion. This not only includes lengths and masses, but also inertias calculated around the center of mass of the desired object. The constants in Equations 3.74 correspond to the dimensions in Figure 3.1.

\[ r = 0.031[m] \]

\[ L_1 = 0.201[m] \]

\[ L_2 = 0.091[m] \]

\[ m_1 = 1.096[kg] \]  
(3.74)

\[ m_2 = 0.827[kg] \]

\[ J_1 = 0.006[kg \text{ m}^2] \]

\[ J_2 = 0.004[kg \text{ m}^2] \]
The overall length of the boom and tank are also necessary for commuting the equations of motion. Their values are given in Equation 3.75 and 3.76.

\[
L_b = 0.232\text{[m]} \quad (3.75) \\
L_t = 0.167\text{[m]} \quad (3.76)
\]

Standard gravity is defined nominal acceleration due to gravity at the Earth’s surface at sea level given in SI units by Equation 3.77.

\[
g = 0.232\text{[m/s}^2]\] \quad (3.77)

To model the motor torques we use the values provided by Equations 3.78. These equations consider the motors’ associated gearboxes and additional gearing for total torque output.

\[
s_v = 3 \times 2.613 = 7.838\text{[Nm]} \\
s_w = 1.744\text{[Nm]} \\
b_v = 3 \times 231 \times 0.014 = 9.702\text{[Nm/A]} \quad (3.78) \\
b_w = 104 \times 0.014 = 1.456\text{[Nm/A]} \\
\mathcal{E} = 12\text{[V]} \\
\Omega = 11\text{[ohms]}
\]

Furthermore, the no load speeds of our two motors are given by Equations 3.79 and 3.80.

\[
v_v = 70/3 = 23.333\text{[rpm]} \quad (3.79) \\
v_w = 155\text{[rpm]} \quad (3.80)
\]

For the sensors we must identify the degree of variance that is expected from each type of measurement. The values can be seen in Equations 3.81.
Choosing \( \rho = 100 \), we can now compute our equations of motion. The resulting nonlinear system is complicated and does not easily reduce to a compact form. However, from these equations we can obtain our linearized system, described by Equations 3.82 and 3.83.

\[
A = \begin{bmatrix}
19.3 & -179.2 & -698.1 & 193.4 & 0 & 6 \\
165.9 & -1437.3 & -2407.4 & 1337.8 & 0 & -289 \\
11.5 & -119.9 & -1059.2 & 114.9 & 0 & 91.3 \\
1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 \\
\end{bmatrix}
\]

(3.82)

\[
B = \begin{bmatrix}
-53.3 & -120.6 \\
-183.8 & -958.8 \\
-80.9 & -81.8 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
\end{bmatrix}
\]

(3.83)

Additionally we can also find our controller and estimator. They are given numerically in Equations 3.84 and 3.85.
The controllability matrix of the linearized dynamic system is given by:

\[ K_c = \begin{bmatrix}
  0.133 & -0.011 & -0.068 & 0.401 & -0.006 & -5.464 \\
  -22.249 & 1.906 & 10.333 & -115.187 & 0.637 & 52.985 \\
 \end{bmatrix} \]  

(3.84)

\[ K_e = \begin{bmatrix}
  9.961 & 2.011 & 0.551 & -0.256 \\
  3.15 & 0.662 & 0.17 & -0.113 \\
  -0.052 & 0.028 & -0.001 & 0.0943 \\
  2.011 & 0.564 & 0.114 & 0.228 \\
  0.551 & 0.114 & 0.355 & 0.001 \\
  -0.256 & 0.228 & 0.001 & 0.69 \\
\end{bmatrix} \]  

(3.85)

### 3.4.2 Control Authority

(a) At 0.0 Seconds  
(b) At 1.9 Seconds  
(c) At 3.9 Seconds  
(d) At 6.0 Seconds

**Figure 3.2: Closed Loop Simulation**

To find the controllably matrix of our linearized dynamic system, we use Equation 3.86.
\[ C = \begin{bmatrix} B & AB & \cdots & A^{n-1}B \end{bmatrix} = \begin{bmatrix} B & AB & A^2B & A^3B & A^4B & A^5B \end{bmatrix} \] (3.86)

If the controllability matrix has full rank than the closed looped system is controllable, allowing the unconstrained control vector to transfer the system from any initial state to any other state in a finite interval of time. When we calculate our controllability matrix, however, we find that it only has a rank of 2, making it rank deficient and not satisfying the requirements for full controllability. To understand which states are unstable and therefore require controllability we examine the eigenvalues of \( A \).

Through the use of the Schur decomposition we find Equation 3.87. This shows us that the velocity states are already passively stable. Thus we only need to have control authority over two states, associated with \( \alpha \) and \( \theta \), for the system to be stabilizable. We see that this is possible with the controller \( K_c \), when we take the eigenvalues of the closed loop system \( A - BK_c \). Again using the Schur decomposition, we see Equation 3.88 and that our closed loop system is stable.

\[
\lambda_A = (-1796.4, -680.7, -5.1, 4.9, 0, 0.1) \quad (3.87)
\]

\[
\lambda_{A-BK} = (-1796.4, -680.7, -5.3, -4.8, -0.4, -0.3) \quad (3.88)
\]

Although this analysis guarantees stabilizability for the linearized system, our actual dynamics are nonlinear. Using the Runge-Kutta method we can more realistically simulate the performance of our system. First we take a loop at the open loop system in Figure 3.3.

In Figure 3.3, the blue dot represents the center of gravity of the entire system. We see that without controls the robot topples over in less than a second. This is a reflection of what would happen with a physical prototype and is an easy verification of our dynamics model. If the same system with the same initial conditions was set up with our designed control laws the simulation would look like Figure 3.2.

Performing a closed loop test on the same initial conditions, as shown in Figure 3.2, shows the robot capable of gaining balance. The center of mass aligns itself
with the pivot point on the ground. Over 6 seconds the robot slowly straightens out into a vertical stance. Figure 3.5 shows that the estimated states are closely mimic the actual states of the system. The Matlab code for both the open loop and closed loop simulations can be found in the Appendix.

### 3.4.3 Disturbance Rejection

The next step of analyzing the success of the closed loop system is accessing its ability to reject disturbances. A disturbance is a signal that tends to adversely affect the value of the output of our system. An external disturbance is generated outside the system and is considered an input into the system. This represents actual turbulences, model uncertainty, and the non-rigid elements of the robot.

To simulate the disturbance the system is set up at equilibrium and random
noise is applied to $\alpha$. As we can see in Figure 3.6, the system needs to be able to correctly estimate the states in order to then make necessary adjustment via the controller. This is the key to stability as seen by the simulation in Figure 3.7. The Matlab code for this simulation can be found in the Appendix.

### 3.4.4 Upright Maneuver

The main drawback of a linearized model is the inability to describe changes in the nonlinear movement occurring away from the point of equilibrium. The implementation of a controller based on this model usually requires a gain scheduling algorithm, where pre-computed time-varying gains are selected based on the cur-
rent estimated trajectory. This requires a library of pre-computed gain sequences and a method for interpolating between data points.

However it also possible to avoid this complexity through a clever system design which allows us to keep the center of mass close to the robot’s ground pivot in any Cobra mode configuration. Thus we can perform a full upright maneuver by starting in a horizontal position and using the same control setup as the previous simulations. As seen in Figure 3.9, the robot is able to roll forward and start raising the boom. This offsets the balance, allowing for the front end of the tank to lift off of the ground. Once the center of mass is over the point where the treads and ground touch, the maneuver naturally switches to slowly opening up $\theta$ while holding stability. Over the time the robot is able to straighten into a vertical stance.

Only one set of control laws through $K_c$ is needed for the entire maneuver. Also only one estimator is need as seen in Figure 3.8. This is possible because of the mechanical design which places the center of mass of the entire robot near the balancing toe of the tread. Even when taking into consideration the stall torques of the motors, as this model does, the closed loop system is able to move the center of mass to the correct spot and then maintain it there. The Matlab code for this simulation can be found in the Appendix.
Figure 3.7: Disturbance Rejection Simulation

(a) At 0 Seconds
(b) At 1 Seconds
(c) At 2 Seconds
(d) At 3 Seconds

Figure 3.8: Upright Maneuver Estimation

(a) Angular Position Estimation
(b) Angular Velocity Estimation
Figure 3.9: Upright Maneuver Simulation
Chapter 4

Control Implementation

With a viable design and corresponding dynamic simulations of performance, we must now apply the control laws we found to the physical prototype. In order to do this, we must integrate measurement sensors and motor drivers into our system, as well as a way to process the information. To this effect, we are using the NI single-board RIO microcontroller for all onboard computing. This allows us to program our algorithms in LabView and upload them directly to the robot. Once operational the robot becomes a standalone system with all estimation and control occurring onboard the RIO console.

4.1 Electronics

The system includes one accelerometer, two gyroscopes, and four potentiometers, as well as four output motors. All of the sensors are analog and will feed into an A/D converter before feeding into the microcontroller. The NI microcontroller then outputs a digital signal to the motor drivers which supplies the required 12 volts to the motors. A pulse width modulator is also incorporated in the microcontroller programming to run the motors at appropriate speeds. Using a state-space model of our system and the linear quadratic regulator method we have already computed the necessary estimator and controller gains. This logic, as well as the calibration of the sensors, is included in the programming of the microcontroller.

The control algorithm used to balance the robot utilizes six states, four of
which are measured. Figure 3.1 shows the angles being measured. The angle between the tank and the ground, $\alpha$, are measured with the accelerometer, and the angular velocity with the gyroscopes. The position of the wheels with respect to the tank, $\phi$, are measured with two of the potentiometers while the velocity must be estimated. The position of the boom will be measured with respect to the tank, $\theta$, using the other two potentiometers. All states are run through an estimator before outputting a signal to the motors. A system block diagram, shown in Figure 4.1, lays out the flowchart of data through the circuitry of the robot.

4.1.1 Microcontroller
Our microcontroller, model type sbRIO-9602 shown in Figure 4.2, is made by National Instruments. The specific model is selected because it is the smallest and lightest board available that can handle the number of inputs and outputs needed for this project.

The A/D converter, shown in Figure 4.3 and also made by National Instruments, is used with this microcontroller. It is a model type 9205 and plugs in directly to the board. This module has an approximate sampling rate of 250 kilohertz, which is the sampling rate that can be approximated for the accelerometer, the potentiometers and the gyros. The resolution is 16 bits, which exceeds what is needed by the sensors and the control algorithm. The power consumption of this module is accounted for in the total power of the microcontroller.

![Figure 4.3: Analog Input Modules](image)

The power consumption by the microcontroller is computed by adding the internal power consumption of the device, $P_i$, the consumption by the digital outputs, $P_d$, the consumption by the 5 volt output, $P_v$, and the power consumed by the A/D converter, $P_s$. The internal power consumption is 6 watts. The power consumption by the digital outputs and by the 5 volt outputs is given by the Equations 4.1 and 4.2, where $I_d$ is total digital output current and $I_v$ is the total 5 volt output current. The power consumed by the A/D Converter is 1.1 watts. The total power consumed by the board, after considering all the sensors involved, is 7.21 watts.
\[ P_d[W] = \frac{I_d[A] \times 3.3[V]}{0.85} \]  
\[ P_v[W] = \frac{I_v[A] \times 5[V]}{0.9} \] (4.1)  
(4.2)

Three of the sensors each require one analog input channel and the four potentiometers each require two channels. The module has 32 channels but only eleven channels are needed in total. The microcontroller has 110 digital channels, although only six are used, two for each motor driver and one for each direction.

The microcontroller employs intricate programming in order to output a logical signal to the motors. The calibration of all of the signals as well as the conversion of the signals into useable units is done within the microcontroller. The program uses these signals in conjecture with an estimator to estimate the states and a gain controller to implement the balancing maneuvers.

The sampling rate of all the sensors is 250 kilohertz, as determined by the A/D converter. Since the resolution is 16 bits, the data rate for each of the sensors is 500[Kbyte/sec]. This application does not require any data storage because the system is closed-loop. All measurements that are taken by the sensors are nearly instantaneously converted to output for the motors. After output to the motors the data is no longer useful. There are a few temporary variables that are continuously updated. These values are stored within the microcontroller.

### 4.1.2 Sensors

The potentiometers are used to measure the position of one rotating body with respect to another. In this application, they are used to measure the position of the boom with respect to the tank and the position of the wheel with respect to the tank. Altogether, four ALPS RDC80 potentiometers will be responsible for two of the six states of the control model. This type of potentiometer, seen in Figure 4.4, includes two signals so that multiple rotations may be measured. The potentiometer is simple to mount, although care must be taken when press fitting a motor shaft because it is sensitive and easy to break. If the plastic is deformed in the center, unwanted friction is added between the center and the sensor. The
rotating center acts as a variable resistor. At different resistances, the sensor passes a range of voltages. Based on this signal, which is proportional to the angle at which the center is turned, the position of one component can be measured with respect to another. Our potentiometers have two signals, which are $180^\circ$ out of phase from each other, so that they can measure multiple rotations.

![ALPS RDC80 Potentiometer](image)

Figure 4.4: ALPS RDC80 Potentiometer

The total resistance in the sensor is 10 kilohms, and when 5 volts is applied to the sensor, the power it consumes is 2.5 milliwatts. The maximum voltage that may be applied to the sensor is 16 volts, but for this application, only 5 volts will be used. The output of the potentiometer is analog and the sensor requires an A/D converter. It outputs a signal with a range of 0 to 5 volts, with a variance of $\pm 0.019$ radians.

A gyroscope is used to measure the angular velocity of the tank chassis. This velocity is one of the states in the control model. The Analog Devices ADXRS150 gyroscope, shown in Figure 4.5, is highly sensitive and has built-in signal conditioning. The gyroscope is mounted by attaching the sensor to a board and soldering wires to the required pins. The gyroscope includes a vibrating mass that resists changes to its axis of rotation. The mass moves linearly while the sensor rotates
about some axis. The centrifugal forces, caused by the Coriolis Effect, exerted on the mass as the sensor rotates about that axis are proportional to the angular velocity of the sensor.

![Analog Devices Gyroscope](image)

**Figure 4.5: Analog Devices Gyroscope**

The maximum voltage that the gyroscope can run on is 5.25 volts. The gyroscope for this robot uses 5 volts. The current that the gyroscope uses is 8 milliamps so the maximum power that can be supplied to it is about 40 milliwatts. The output of the gyroscope is analog and the sensor requires an A/D converter. It outputs a signal with a range of 0 to 5 volts, with a variance of ±0.262 radians per second.

The accelerometer is used to sense the angular position of the tank with respect to the ground, which is one of the six states of the control model. The sensor constantly senses the acceleration of gravity and when tilted at different angles, it passes a corresponding voltage values, signaling a change in position. The Analog Devices dual-axis iMEMS accelerometer, shown in Figure 4.6, is very sensitive and low power, making it suitable for this application. As we can see, the accelerometer is already mounted to a board with the required capacitors and has through-holes for easy mounting. The structure includes a small cantilever beams
with attached seismic masses. When acceleration is felt, the masses deflect from their original positions. The capacitance between the masses is measured and output as an analog signal.

![Analog Devices Accelerometer](image)

Figure 4.6: Analog Devices Accelerometer

The accelerometer runs on 5 volts. The maximum current supply for this sensor is 1.1 milliamps. The absolute maximum amount of power required for the sensor, using 5 volts, will be 5.5 milliwatts. The output of the accelerometer is analog and the sensor requires an A/D converter. It outputs a signal with a range of 0 to 5 volts, with a variance of ±0.063 radians.

### 4.1.3 Motor Drivers

Motors are the output device used to stabilize our robot. They are used to drive the left and right treads, and move the boom about the tank. Three motor drivers are used for these three motor directions. The VNH3SP30 motor drivers, shown in Figure 4.7, are made by STMicroelectronics. The model is lightweight and can handle the amount of current coming into the terminals for the motors.
It has three through holes for mounting. The motor driver gets either a high or low signal from the microcontroller digital output. When it receives a high signal, it passes the voltage supplied to the power terminal to the motors.

Figure 4.7: STMicroelectronics Motor Driver

The total voltage applied to the motors is 14.4 volts and the average amount of current that the motors use is about 400 milliamps. The total power for each motor is about 5.76 watts. The motor drivers receive 5 volts for the signal and use 0.01 milliamps. The total power consumption for each motor driver is 50 milliwatts. The microcontroller outputs a digital signal to the motor drivers. The microcontroller is programmed with pulse-width modulations to control the speed of the motor. This is implemented by generating a square wave with a high frequency within the microcontroller, with the duty cycle of the square wave proportionally determining the motor speed.

4.2 Design Modifications

Because Switchblade V2 is not large enough to hold the NI microcontroller, it was redesigned into Switchblade V3. This third version of the robot, as shown in Figure, is wider to be compatible with the provided circuit board and all other
components. There are a number of design improvements and alterations from the Switchblade V2 design. The A/D converter module is tucked underneath the microcontroller and mounted with various slots in the chassis. Motor driver housings are designed and mounted in convenient locations. Additional batteries are needed to power the microcontroller at 18 volts or higher, which cannot be provided by the subC batteries alone. Two low current 9 volt batteries independently tailor to the electronic power needs, while the remaining batteries power the motors. The weighted rotating boom is also modified in this version of the robot. The adjusted boom now contains both the two 9 volt batteries that power the circuit board and the fourteen subC batteries that drive the motors, substantially increases the weight of the boom. To combat this, two geared motors are now responsible for pivoting the boom, providing double the torque.

Other aspects of the design are also updated as a fail-safe to help eliminate possible weak spots in the rigid body. Additional holes are added to mount treads to provide increased control over the tautness of the tread. Gears are resized so that they mesh with the pinion driven by the motor. A press press-fit connection with a redundant set screw now attaches the main shaft to the boom arms. All of the electronics are now integration onto this Switchblade V3 design. The microcontroller housing maximizes the use of the small inner chassis space. All
together the robot houses three motor drivers, four motors, four accelerometers, four potentiometers, and two gyroscopes. These components are properly soldered and wired to the electronics board and are attached to the new rover.

### 4.3 LabView

![LabView Block Diagram](image)

A LabView interface, seen in Figure 4.9, is used to program the microcontroller. In it all of the signals are presented as graphical nodes, with a specialized syntax strictly enforced during the editing process to ensure compliably. The graphical code is then translated into a machine language program. This executable works with the help of the LabView run-time engine, which contains precompiled code to perform common tasks. The run-time engine reduces compile time and also provides a consistent interface with the hardware components.

Many libraries with a large number of functions for data acquisition, signal generation, and statistics are provided in the LabView package. In addition, LabView includes a text-based programming component called MathScript with additional functionality for signal processing, analysis, and mathematics. MathScript is integrated with our graphical programming using a syntax that is similar to Matlab code.
Chapter 5

Conclusion

We have seen the development of a vertical balancing treded rover. Building upon each other, three different prototypes are manufactured. With each version, improvements in structure design and component integration are clearly visible. Utilizing dynamics, control theory, and numerical computations we are able to fully analyze the theoretical potential of our system, or a similar system with different parameters. Once we have gained an understanding of the inputs, outputs, and control laws, the third version of the robot becomes the platform for testing our maneuvering capabilities. The implementation of the NI single-board RIO allows us to create a completely standalone rover. Our custom algorithms have enabled us some basic balancing accomplishment. The robot is able to hold its equilibrium and reject disturbances in a fixed Cobra mode position with its treads at an angle with the ground. This original concept of vertical tread travel has begun to show signs of success. The future generations of Switchblade can use this success as a stepping stone to greater achievement.

5.1 Future Work

Aside from working out the details of the Cobra upright maneuver, other methods and movements can be explored. For instance, the incorporation of a Viper mode in the rover can furnish a choice of operations for any given situation. The ability balance on edges as well as function upside-down can also make Switch-
blade more agile. With more dexterity we can plan trajectories for the robot over specific obstacles, such as stair climbing. The addition of cameras and network communication, can allow for multiple nimble Switchblades to solve complex real work coordination problems.
Appendix A

Motion Equations and Control Law Derivations

%%%Nonlinear Equations of Motion

clear all

L1 = 0.20136242; % meters
L2 = 0.09091696; % meters
Lt = 0.16660000; % meters
r = 0.03050835; % meters
m1 = 1.09601699; % kilograms
m2 = 0.82729460; % kilograms
J1 = 0.00629730; % kilograms * square meters
J2 = 0.00371840; % kilograms * square meters
g = 9.80665; % meters / (second ^ 2)
v_s = 3 * 2.61277417; % newton meters
v_k = 3 * 231 * .014; % newton meters / amps
w_s = 1.7442033; % newton meters
w_k = 104 * .014; % newton meters / amps

% syms L1 L2 Lt r m1 m2 J1 J2 g v_s v_k w_s w_k real
syms aalpha phi theta aalpha_d phi_d theta_d aalpha_dd phi_dd...
    theta_dd u1 u2 real

x1 = - r * (aalpha - phi) + Lt * cos(aalpha) - L1 * cos(theta -...
aalpha);
y1 = r + Lt * sin(aalpha) + L1 * sin(theta - aalpha);
\[ x_2 = -r \ast (\alpha - \phi) + L_2 \ast \cos(\alpha); \]
\[ y_2 = r + L_2 \ast \sin(\alpha); \]

%Define Generalized Coordinates, q, Velocities, qt, %and Accelerations, qtt
q = [\alpha \phi \theta];
qt = [\alpha_d \phi_d \theta_d];
qtt = [\alpha_dd \phi_dd \theta_dd];

%Compute Velocity of Boom COM
xt1 = jacobian(x1, q) \ast qt \ast [1 1 1]';
yt1 = jacobian(y1, q) \ast qt \ast [1 1 1]';

%Compute Velocity of Tank COM
xt2 = jacobian(x2, q) \ast qt \ast [1 1 1]';
yt2 = jacobian(y2, q) \ast qt \ast [1 1 1]';

%Square Velocities and Simplify
v1_sq = simple(xt1 \ast 2 + yt1 \ast 2);
v2_sq = simple(xt2 \ast 2 + yt2 \ast 2);

%Compute Total Kinetic Energy
L = .5 \ast m_1 \ast v_1_sq + .5 \ast m_2 \ast v_2_sq + .5 \ast J_1 \ast (\theta_d) \ast 2 + .5 \ast J_2 \ast (\alpha_d) \ast 2;

%Compute Total Potential Energy
U = m_1 \ast g \ast y_1 + m_2 \ast g \ast y_2;

%Use Lagrange’s Method to Compute Equations of Motion:
\[ \frac{d}{dt}(\frac{dL}{dq}) - \frac{dL}{dq} - \frac{dU}{dq} = 0 \]
one = jacobian(jacobian(L, \alpha_d), [q qt]) \ast [qt qtt] \ast [1 1 1 1 1]' - jacobian(L, \alpha) + jacobian(U, \alpha);
two = jacobian(jacobian(L, \phi_d), [q qt]) \ast [qt qtt] \ast [1 1 1 1 1]' - jacobian(L, \phi) + jacobian(U, \phi);
three = jacobian(jacobian(L, \theta_d), [q qt]) \ast [qt qtt] \ast [1 1 1 1 1]' - jacobian(L, \theta) + jacobian(U, \theta);

v\_motor\_torque = v_s \ast u1 / 12 + \theta_d \ast v_k ^ 2 / 11;
w\_motor\_torque = 2 \ast (w_s \ast u2 / 12 + \phi_d \ast w_k ^ 2 / 11);
\[ f = 0.1 \times r \times (m1 + m2) \times g \times (\alpha_d - \phi_d); \]

% Arrange Three Equations of Motion into Matrix Form

\[
\text{motion\_eq} = \begin{bmatrix}
\text{one}; \\
\text{two}; \\
\text{three}
\end{bmatrix} + \\
[f; w\_motor\_torque - f; v\_motor\_torque];
\]

% Identify Acceleration Terms

\[ E = \text{jacobian(motion\_eq, qtt)}; \]

\% E * qtt' = N(x, t, u)
\[ \text{lhs} = E \times \text{qtt'}; \]
\[ \text{rhs} = \text{motion\_eq + lhs}; \]
\[ \text{E\_N} = \text{simple([E zeros(3); zeros(3) eye(3)]) \ldots} \]
\[ \text{[rhs; aalpha\_d; phi\_d; theta\_d]);} \]
\[ \text{disp(E\_N)} \]

% Linearize about x0
\[ x0 = [0, 0, 0, \pi / 2, \pi / 2, \pi]; \]
\[ \text{E\_bar} = \text{subs(jacobian([lhs; aalpha\_d; phi\_d; theta\_d],...} \]
\[ \text{[qtt qt]), [qt q], x0}); \]
\[ \text{A\_bar} = \text{subs(jacobian([rhs; aalpha\_d; phi\_d; theta\_d],...} \]
\[ \text{[qt q]), [qt q], x0}); \]
\[ \text{B\_bar} = \text{subs(jacobian([rhs; aalpha\_d; phi\_d; theta\_d],...} \]
\[ \text{[u1 u2]), [qt q], x0}); \]

\[ \text{A = E\_bar \backslash A\_bar}; \]
\[ \text{B = E\_bar \backslash B\_bar}; \]

\[ \text{disp('Rank of Controllability Matrix')} \]
\[ \text{rank(ctrb(A, B))} \]

% Measuring
\[ \text{C = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0; \\
0 & 0 & 0 & 1 & 0 & 0; \\
0 & 0 & 0 & 0 & 1 & 0; \\
0 & 0 & 0 & 0 & 0 & 1;
\end{bmatrix};} \]

\[ \text{disp('Rank of Observability Matrix')} \]
\[ \text{rank(obsv(A, C))} \]

% Optimal State Feedback Control
\[ \text{K = lqr(A, B, diag([100 * ((40 * 120 * pi / 3) ^ - 2), 100 *...} \]
\[ ((90 * 120 * pi) ^ - 2), 100 * ((40 * 120 * pi / 3) ^ -...} \]
\[ 2), 100 * ((pi / 2) ^ - 2), (pi / 2) ^ - 2]);} \]
\[ \text{diag([1, 2 ^ - 2]));} \]
% Optimal State Estimation
L = lqr(A', C', diag([(40 * 120 * pi / 3)^-2], (90 * ...
120 * pi)^-2), ((40 * 120 * pi / 3)^-2), ((pi / 2)^...
- 2), pi^-2, (pi / 2)^-2), eye(4));
L = L';
Appendix B

Open Loop Simulation

%Nonlinear Equations of Motion
%Open(A, B, C, K, L)

function [] = Open(a, b, c, k, l)

close all;

global A B C K L L1 L2 Lb Lt r m1 m2 x0

A = a;
B = b;
C = c;
K = k;
L = l;

L1 = 0.20136242; % meters
L2 = 0.09091696; % meters
Lb = 0.23177500; % meters
Lt = 0.16660000; % meters
r = 0.03050835; % meters
m1 = 1.09601699; % kilograms
m2 = 0.82729460; % kilograms

t_final = .5;
x0 = [0, 0, 0, pi / 2, pi / 2, pi];
x = [0, 0, 0, pi / 4, pi / 4, pi / 3]';
x_hat = [0, 0, 0, pi / 4, pi / 4, pi / 3]';
x_star = [x; x_hat];
%Simulation Time Step Size
    h = .001;
    disp('Simulating nonlinear model...')

%Initialize Elapsed Time Vector
    t = zeros(t_final / h + 1, 1);
    n = 0;
    p_aalpha_d = zeros(t_final / h, 1);
    p_phi_d = zeros(t_final / h, 1);
    p_theta_d = zeros(t_final / h, 1);
    p_aalpha = zeros(t_final / h, 1);
    p_phi = zeros(t_final / h, 1);
    p_theta = zeros(t_final / h, 1);
    p_aalpha_d_hat = zeros(t_final / h, 1);
    p_phi_d_hat = zeros(t_final / h, 1);
    p_theta_d_hat = zeros(t_final / h, 1);
    p_aalpha_hat = zeros(t_final / h, 1);
    p_phi_hat = zeros(t_final / h, 1);
    p_theta_hat = zeros(t_final / h, 1);

%Run Nonlinear Simulation
    while t(n + 1) < t_final

        %Update RK4 Iteration Counter
        n = n + 1;

        %RK4: evaluate slope at four points from z_n to z_n + 1, to get
        %k1, k2, k3, k4, then solve for z_n + 1
        k1 = f_of_x(x_star);
        k2 = f_of_x(x_star + (h / 2) * k1);
        k3 = f_of_x(x_star + (h / 2) * k2);
        k4 = f_of_x(x_star + h * k3);

        %Calculate z at new interval using linear combination of slopes
        %k1 - k4 to draw a line from zn to zn + 1
        x_star = x_star + (h / 6) * k1 + (h / 3) * (k2 + k3) + (h / 6)...
        * k4;

        x = x_star(1 : 6);
        x_hat = x_star(7 : 12);

        %Update Elapsed Time Vector
t(n + 1) = t(n) + h;
t(n)

% Parse state vector, x, into named state variables
p_aalpha_d(n) = x(1);
p_phi_d(n) = x(2);
p_theta_d(n) = x(3);
p_aalpha(n) = x(4);
p_phi(n) = x(5);
p_theta(n) = x(6);
p_aalpha_d_hat(n) = x_hat(1);
p_phi_d_hat(n) = x_hat(2);
p_theta_d_hat(n) = x_hat(3);
p_aalpha_hat(n) = x_hat(4);
p_phi_hat(n) = x_hat(5);
p_theta_hat(n) = x_hat(6);

end

% Remove last extra element in time vector, t
   temp = t;
clear t;
t = temp(1 : end - 1);
clear temp;

plot_switchblade(t, p_aalpha, p_phi, p_theta);
figure;
plot(t, [p_aalpha p_phi p_theta], '-', t,...
   [p_aalpha_hat p_phi_hat p_theta_hat], '--', 'linewidth', 2);
set(gca, 'FontSize', 12, 'FontName', 'Californian FB');
xlabel('Time [sec]', 'FontWeight', 'b');
ylabel('Angular Position [rad]', 'FontWeight', 'b');
set(legend='$\alpha$', '$\phi$', '$\theta$', '$\dot{\alpha}$',
       '$\dot{\phi}$', '$\dot{\theta}$'), 'Interpreter', 'LaTeX');
figure;
plot(t, [p_aalpha_d p_phi_d p_theta_d], '-', t,...
   [p_aalpha_d_hat p_phi_d_hat p_theta_d_hat], '--', 'linewidth',... 2);
set(gca, 'FontSize', 12, 'FontName', 'Californian FB');
xlabel('Time [sec]', 'FontWeight', 'b');
ylabel('Angular Velocity [rad/sec]', 'FontWeight', 'b');
set(legend='$\dot{\alpha}$', '$\dot{\phi}$', '$\dot{\theta}$',
       '$\dot{\hat{\alpha}}$', '$\dot{\hat{\phi}}$',...  '$\dot{\hat{\theta}}$'), 'Interpreter', 'LaTeX');
\[
\dot{\hat{\theta}}
\]

\textit{'Interpreter', 'LaTeX');

---

%F_OF_X
function [dx_dt] = f_of_x(x_star)

global A B C K L x0

x = x_star(1 : 6);
x_hat = x_star(7 : 12);

\[x = [\alpha_d; \phi_d; \theta_d; \alpha_d; \phi; \theta]\]
aalpha_d = x(1);
phi_d = x(2);
theta_d = x(3);
aalpha = x(4);
% phi = x(5);
theta = x(6);

u = -K * (x_hat - x0');

for j = 1 : length(u)
    if abs(u(j)) > 12
        u(j) = 12 * sign(u(j));
        disp(['STALL ' num2str(j)]n)
    end
end

u1 = u(1);
u2 = u(2);

u1 = 0;
u2 = 0;

E_N = [...;
...;
...;
aalpha_d;
phi_d;
theta_d];

% E_N = E_N + [100 * (rand - 0.5); zeros(5, 1)];

dx_hat_dt = (A - L * C) * (x_hat - x0') + B * u + L * C * (x - x0');
dx_dt = [E_N; dx_hat_dt];

%Plot Simulation Results
function [] = plot_switchblade(t, aalpha, phi, theta)

global L1 L2 Lb Lt r m1 m2

x3 = - r * (aalpha - phi);
y3 = x3 - x3 + r;

x2 = x3 + Lt * cos(aalpha);
y2 = y3 + Lt * sin(aalpha);

x1 = x2 - Lb * cos(theta - aalpha);
y1 = y2 + Lb * sin(theta - aalpha);

xc = x3 + (m2 * L2 * cos(aalpha) + m1 * (Lt * cos(aalpha) - L1 * cos(theta - aalpha))) / (m1 + m2);
yc = y3 + (m2 * L2 * sin(aalpha) + m1 * (Lt * sin(aalpha) + L1 * sin(theta - aalpha))) / (m1 + m2);

%Animate Simulation Results
for j = 1 : round((1 + length(t)) / 4) : length(t)
  figure;
  boom = [x1(j) y1(j); x2(j) y2(j)];
  tank1 = [x2(j) + r * cos(aalpha(j) + pi / 2), y2(j) + r * sin(aalpha(j) + pi / 2);
            x3(j) + r * cos(aalpha(j) + pi / 2), y3(j) + r * sin(aalpha(j) + pi / 2)];
  tank2 = [x2(j) - r * cos(aalpha(j) + pi / 2), y2(j) - r * sin(aalpha(j) + pi / 2);
            x3(j) - r * cos(aalpha(j) + pi / 2), y3(j) - r * sin(aalpha(j) + pi / 2)];
  wheel1 = [x1(j) + r * cos(0 : 1 : 2 * pi)', y1(j) + r * sin(0 : 1 : 2 * pi)'];
  wheel1(size(wheel1, 1) + 1, :) = wheel1(1, :);
  wheel2 = [x2(j) + r * cos((0 : 1 : 2 * pi) - pi / 2 + aalpha(j) - phi(j)),
            y2(j) + r * sin((0 : 1 : 2 * pi) - pi / 2 + aalpha(j) - phi(j))];

wheel2(size(wheel2, 1) + 1, :) = wheel2(1, :);
wheel3 = [x3(j) + r * cos((0 : .1 : 2 * pi) - pi / 2 +
    aalpha(j) - phi(j))', y3(j) + r * sin((0 : .1 : 2 * pi) -
    pi / 2 + aalpha(j) - phi(j))'];
wheel3(size(wheel3, 1) + 1, :) = wheel3(1, :);
hold off;
plot(boom(:, 1), boom(:, 2), 'linewidth', 2, 'color', 'k');
hold on;
plot(tank1(:, 1), tank1(:, 2), 'linewidth', 2, 'color', 'g');
plot(wheel2(:, 1), wheel2(:, 2), 'linewidth', 2, 'color', 'g');
plot(wheel1(:, 1), wheel1(:, 2), 'linewidth', 2, 'color', 'k');
plot(wheel2(1, 1), wheel2(1, 2), 'y.');
plot(wheel3(:, 1), wheel3(:, 2), 'linewidth', 2, 'color', 'g');
plot(wheel3(1, 1), wheel3(1, 2), 'y.');</p>
plot(xc(1:j), yc(1:j), 'linewidth', 2, 'color', 'b');
plot(xc(j), yc(j), 'b.');
plot(xc(j), yc(j), 'bo');
axis([-0.3, 0.3, 0, 0.6]);
set(gca, 'FontSize', 12, 'FontName', 'Californian FB');
text(0, .55, ['t = ', num2str(floor(10 * t(j)) / 10),
    ' sec.'], 'Color', 'r', 'FontSize', 12, 'FontWeight', 'b');
xlabel('x [m]', 'FontWeight', 'b')
ylabel('y [m]', 'FontWeight', 'b')
axis equal;
grid;
drawnow;
end
Appendix C

Closed Loop Simulation

%%%Nonlinear Equations of Motion
%%%Close(A, B, C, K, L)

function [] = Close(a, b, c, k, l)

close all;

global A B C K L L1 L2 Lb Lt r m1 m2 x0

A = a;
B = b;
C = c;
K = k;
L = l;

L1 = 0.20136242; % meters
L2 = 0.09091696; % meters
Lb = 0.23177500; % meters
Lt = 0.16660000; % meters
r = 0.03050835; % meters
m1 = 1.09601699; % kilograms
m2 = 0.82729460; % kilograms

t_final = 8;
x0 = [0, 0, 0, pi / 2, pi / 2, pi];
x = [0, 0, 0, pi / 4, pi / 4, pi / 3]';
x_hat = [0, 0, 0, pi / 4, pi / 4, pi / 3]';
x_star = [x; x_hat];
%Simulation Time Step Size
    h = .001;
    disp('Simulating nonlinear model...')

%Initialize Elapsed Time Vector
    t = zeros(t_final / h + 1, 1);
    n = 0;
    p_aalpha_d = zeros(t_final / h, 1);
    p_phi_d = zeros(t_final / h, 1);
    p_theta_d = zeros(t_final / h, 1);
    p_aalpha = zeros(t_final / h, 1);
    p_phi = zeros(t_final / h, 1);
    p_theta = zeros(t_final / h, 1);
    p_aalpha_d_hat = zeros(t_final / h, 1);
    p_phi_d_hat = zeros(t_final / h, 1);
    p_theta_d_hat = zeros(t_final / h, 1);
    p_aalpha_hat = zeros(t_final / h, 1);
    p_phi_hat = zeros(t_final / h, 1);
    p_theta_hat = zeros(t_final / h, 1);

%Run Nonlinear Simulation
    while t(n + 1) < t_final
        %Update RK4 Iteration Counter
        n = n + 1;

        %RK4: evaluate slope at four points from z_n to z_n + 1, to get
        %k1, k2, k3, k4, then solve for z_n + 1
        k1 = f_of_x(x_star);
        k2 = f_of_x(x_star + (h / 2) * k1);
        k3 = f_of_x(x_star + (h / 2) * k2);
        k4 = f_of_x(x_star + h * k3);

        %Calculate z at new interval using linear combination of slopes
        %k1 - k4 to draw a line from zn to zn + 1
        x_star = x_star + (h / 6) * k1 + (h / 3) * (k2 + k3) + (h / 6)... * k4;
        x = x_star(1 : 6);
        x_hat = x_star(7 : 12);

        %Update Elapsed Time Vector
t(n + 1) = t(n) + h;

% Parse state vector, x, into named state variables
p_aalpha_d(n) = x(1);
p_phi_d(n) = x(2);
p_theta_d(n) = x(3);
p_aalpha(n) = x(4);
p_phi(n) = x(5);
p_theta(n) = x(6);

p_aalpha_d_hat(n) = x_hat(1);
p_phi_d_hat(n) = x_hat(2);
p_theta_d_hat(n) = x_hat(3);
p_aalpha_hat(n) = x_hat(4);
p_phi_hat(n) = x_hat(5);
p_theta_hat(n) = x_hat(6);

end

% Remove last extra element in time vector, t
   temp = t;
clear t;
t = temp(1:end - 1);
clear temp;

plot_switchblade(t, p_aalpha, p_phi, p_theta);
figure;
plot(t, [p_aalpha p_phi p_theta], '-', t,...
    [p_aalpha_hat p_phi_hat p_theta_hat], '--', 'linewidth', 2);
set(gca, 'FontSize', 12, 'FontName', 'Californian FB');
xlabel('Time [sec]', 'FontWeight', 'b');
ylabel('Angular Position [rad]', 'FontWeight', 'b');
set(legend('$\alpha$', '$\phi$', '$\theta$', '$\hat{\alpha}$','$
    \hat{\phi}$', '$\hat{\theta}$'), 'Interpreter', 'LaTeX');
figure;
plot(t, [p_aalpha_d p_phi_d p_theta_d], '-', t,...
    [p_aalpha_d_hat p_phi_d_hat p_theta_d_hat], '--', 'linewidth',... 
    2);
set(gca, 'FontSize', 12, 'FontName', 'Californian FB');
xlabel('Time [sec]', 'FontWeight', 'b');
ylabel('Angular Velocity [rad/sec]', 'FontWeight', 'b');
set(legend('$\dot{\alpha}$', '$\dot{\phi}$', '$\dot{\theta}$','$
    \dot{\hat{\alpha}}$', '$\dot{\hat{\phi}}$', '$\dot{\hat{\theta}}$'),... 
    'Interpreter', 'LaTeX');
function [dx_dt] = f_of_x(x_star)

global A B C K L x0

x = x_star(1:6);
x_hat = x_star(7:12);

% \textit{x} = [\alpha_d; \phi_d; \theta_d; \alpha; \phi; \theta]
alpha_d = x(1);
phi_d = x(2);
theta_d = x(3);
alpha = x(4);
% \phi = x(5);
theta = x(6);

u = -K * (x_hat - x0');

for j = 1 : length(u)
    if abs(u(j)) > 12
        u(j) = 12 * sign(u(j));
        disp(['STALL ' num2str(j)])
    end
end

u1 = u(1);
u2 = u(2);

% u1 = 0;
% u2 = 0;

E_N = [...;
    ...;
    ...;
    alpha_d;
    phi_d;
    theta_d];

% E_N = E_N + [100 * (rand - 0.5); zeros(5, 1)];

dx_hat_dt = (A - L * C) * (x_hat - x0') + B * u + L * C * (x - x0');
dx_dt = [E_N; dx_hat_dt];

%%%%Plot Simulation Results
function [] = plot_switchblade(t, aalpha, phi, theta)

global L1 L2 Lb Lt r m1 m2

x3 = - r * (aalpha - phi);
y3 = x3 - x3 + r;

x2 = x3 + Lt * cos(aalpha);
y2 = y3 + Lt * sin(aalpha);

x1 = x2 - Lb * cos(theta - aalpha);
y1 = y2 + Lb * sin(theta - aalpha);

xc = x3 + (m2 * L2 * cos(aalpha) + m1 * (Lt * cos(aalpha) - L1 ... cos(theta - aalpha))) / (m1 + m2);
yc = y3 + (m2 * L2 * sin(aalpha) + m1 * (Lt * sin(aalpha) + L1 ... sin(theta - aalpha))) / (m1 + m2);

%%%%Animate Simulation Results
for j = 1 : round((1 + length(t)) / 4) : length(t)
    figure;
    boom = [x1(j) y1(j); x2(j) y2(j)];
    tank1 = [x2(j) + r * cos(aalpha(j) + pi / 2), y2(j) + r *... sin(aalpha(j) + pi / 2);
             x3(j) + r * cos(aalpha(j) + pi / 2), y3(j) + r *... sin(aalpha(j) + pi / 2)];
    tank2 = [x2(j) - r * cos(aalpha(j) + pi / 2), y2(j) - r *... sin(aalpha(j) + pi / 2);
             x3(j) - r * cos(aalpha(j) + pi / 2), y3(j) - r *... sin(aalpha(j) + pi / 2)];
    wheel1 = [x1(j) + r * cos(0 : .1 : 2 * pi)', y1(j) + r *... sin(0 : .1 : 2 * pi)'];
    wheel1(size(wheel1, 1) + 1, :) = wheel1(1, :);
    wheel2 = [x2(j) + r * cos((0 : .1 : 2 * pi) - pi / 2 +... aalpha(j) - phi(j)'), y2(j) + r * sin((0 : .1 : 2 * pi)... - pi / 2 + aalpha(j) - phi(j)')]:
wheel2(size(wheel2, 1) + 1, :) = wheel2(1, :);
wheel3 = [x3(j) + r * cos((0 : .1 : 2 * pi) - pi / 2 +
aalpha(j) - phi(j))', y3(j) + r * sin((0 : .1 : 2 * pi) -... pi / 2 + aalpha(j) - phi(j))'];
wheel3(size(wheel3, 1) + 1, :) = wheel3(1, :);
hold off;
plot(boom(:, 1), boom(:, 2), 'linewidth', 2, 'color', 'k');
hold on;
plot(tank1(:, 1), tank1(:, 2), 'linewidth', 2, 'color', 'g');
plot(tank2(:, 1), tank2(:, 2), 'linewidth', 2, 'color', 'g');
plot(wheel1(:, 1), wheel1(:, 2), 'linewidth', 2, 'color', 'k');
plot(wheel2(:, 1), wheel2(:, 2), 'linewidth', 2, 'color', 'g');
plot(wheel2(1, 1), wheel2(1, 2), 'y.');
plot(wheel3(:, 1), wheel3(:, 2), 'linewidth', 2, 'color', 'g');
plot(wheel3(1, 1), wheel3(1, 2), 'y.');
plot(xc(1:j), yc(1:j), 'linewidth', 2, 'color', 'b');
plot(xc(j), yc(j), 'b.');
plot(xc(j), yc(j), 'bo');
axis([-0.3, 0.3, 0, 0.6]);
set(gca, 'FontSize', 12, 'FontName', 'Californian FB');
text(0, .55, ['t = ', num2str(floor(10 * t(j)) / 10),...
'sec.'], 'Color', 'r', 'FontSize', 12, 'FontWeight', 'b');
xlabel('x [m]', 'FontWeight', 'b')
ylabel('y [m]', 'FontWeight', 'b')
axis equal;
grid;
drawnow;
end
Appendix D

Disturbance Rejection Simulation

% Nonlinear Equations of Motion
% Reject(A, B, C, K, L)

function [] = Reject(a, b, c, k, l)

close all;

global A B C K L L1 L2 Lb Lt r m1 m2 x0

A = a;
B = b;
C = c;
K = k;
L = l;

L1 = 0.20136242; \% meters
L2 = 0.09091696; \% meters
Lb = 0.23177500; \% meters
Lt = 0.16660000; \% meters
r = 0.03050835; \% meters
m1 = 1.09601699; \% kilograms
m2 = 0.82729460; \% kilograms

t_final = 4;
x0 = [0, 0, 0, pi / 2, pi / 2, pi];
x = [0, 0, 0, pi / 2, pi / 2, pi]';
x_hat = [0, 0, 0, pi / 2, pi / 2, pi]';
x_star = [x; x_hat];
%Simulation Time Step Size
h = .001;
disp('Simulating nonlinear model...')

%Initialize Elapsed Time Vector
n = 0;
t = zeros(t_final / h + 1, 1);
p_aalpha_d = zeros(t_final / h, 1);
p_phi_d = zeros(t_final / h, 1);
p_theta_d = zeros(t_final / h, 1);
p_aalpha = zeros(t_final / h, 1);
p_phi = zeros(t_final / h, 1);
p_theta = zeros(t_final / h, 1);
p_aalpha_d_hat = zeros(t_final / h, 1);
p_phi_d_hat = zeros(t_final / h, 1);
p_theta_d_hat = zeros(t_final / h, 1);
p_aalpha_hat = zeros(t_final / h, 1);
p_phi_hat = zeros(t_final / h, 1);
p_theta_hat = zeros(t_final / h, 1);

%Run Nonlinear Simulation
while t(n + 1) < t_final

  %Update RK4 Iteration Counter
  n = n + 1;

  %RK4: evaluate slope at four points from z_n to z_n + 1, to get %k1, k2, k3, k4, then solve for z_n + 1
  k1 = f_of_x(x_star);
  k2 = f_of_x(x_star + (h / 2) * k1);
  k3 = f_of_x(x_star + (h / 2) * k2);
  k4 = f_of_x(x_star + h * k3);

  %Calculate z at new interval using linear combination of slopes %k1 - k4 to draw a line from zn to zn + 1
  x_star = x_star + (h / 6) * k1 + (h / 3) * (k2 + k3) + (h / 6)... * k4;

  x = x_star(1 : 6);
  x_hat = x_star(7 : 12);

  %Update Elapsed Time Vector
t(n + 1) = t(n) + h;
t(n)

% Parse state vector, x, into named state variables
p_aalpha_d(n) = x(1);
p_phi_d(n) = x(2);
p_theta_d(n) = x(3);
p_aalpha(n) = x(4);
p_phi(n) = x(5);
p_theta(n) = x(6);
p_aalpha_d_hat(n) = x_hat(1);
p_phi_d_hat(n) = x_hat(2);
p_theta_d_hat(n) = x_hat(3);
p_aalpha_hat(n) = x_hat(4);
p_phi_hat(n) = x_hat(5);
p_theta_hat(n) = x_hat(6);
end

% Remove last extra element in time vector, t
  temp = t;
clear t;
t = temp(1 : end - 1);
clear temp;

plot_switchblade(t, p_aalpha, p_phi, p_theta);
figure;
plot(t, [p_aalpha p_phi p_theta], '-', t,...
    [p_aalpha_hat p_phi_hat p_theta_hat], '--', 'linewidth', 2);
set(gca, 'FontSize', 12, 'FontName', 'Californian FB');
xlabel('Time [sec]', 'FontWeight', 'b');
ylabel('Angular Position [rad]', 'FontWeight', 'b');
set(legend('$\alpha$', '$\phi$', '$\theta$', '$\hat{\alpha}$',...
            '$\hat{\phi}$', '$\hat{\theta}$'), 'Interpreter', 'LaTeX');
figure;
plot(t, [p_aalpha_d p_phi_d p_theta_d], '-', t,...
    [p_aalpha_d_hat p_phi_d_hat p_theta_d_hat], '--', 'linewidth',... 2);
set(gca, 'FontSize', 12, 'FontName', 'Californian FB');
xlabel('Time [sec]', 'FontWeight', 'b');
ylabel('Angular Velocity [rad/sec]', 'FontWeight', 'b');
set(legend('$\dot{\alpha}$', '$\dot{\phi}$', '$\dot{\theta}$',...
            '$\dot{\hat{\alpha}}$', '$\dot{\hat{\phi}}$',...
            '$\dot{\hat{\theta}}$'), 'Interpreter', 'LaTeX');
'\dot{\hat{\theta}}', 'Interpreter', 'LaTeX');

\%F_OF_X
function [dx_dt] = f_of_x(x_star)

global A B C K L x0

x = x_star(1 : 6);
x_hat = x_star(7 : 12);

% x = [alpha_d; phi_d; theta_d; alpha; phi; theta]
aalpha_d = x(1);
phi_d = x(2);
theta_d = x(3);
aalpha = x(4);
% phi = x(5);
theta = x(6);

u = - K * (x_hat - x0');

for j = 1 : length(u)
    if abs(u(j)) > 12
        u(j) = 12 * sign(u(j));
        disp(['STALL ' num2str(j)])
    end
end

u1 = u(1);
u2 = u(2);

% u1 = 0;
% u2 = 0;

E_N = [...;
    ...;
    ...;
aalpha_d;
phi_d;
theta_d];

E_N = E_N + [100 * (rand - 0.5); zeros(5, 1)];

dx_hat_dt = (A - L * C) * (x_hat - x0') + B * u + L * C * (x - x0');
dx_dt = [E_N; dx_hat_dt];

%Plot Simulation Results
function [] = plot_switchblade(t, aalpha, phi, theta)

global L1 L2 Lb Lt r m1 m2

x3 = -r * (aalpha - phi);
y3 = x3 - x3 + r;

x2 = x3 + Lt * cos(aalpha);
y2 = y3 + Lt * sin(aalpha);

x1 = x2 - Lb * cos(theta - aalpha);
y1 = y2 + Lb * sin(theta - aalpha);

xc = x3 + (m2 * L2 * cos(aalpha) + m1 * (Lt * cos(aalpha) - L1 *... 
cos(theta - aalpha))) / (m1 + m2);
yc = y3 + (m2 * L2 * sin(aalpha) + m1 * (Lt * sin(aalpha) + L1 *... 
sin(theta - aalpha))) / (m1 + m2);

%Animate Simulation Results
for j = 1 : round((1 + length(t)) / 4) : length(t)
    figure;
    boom = [x1(j) y1(j); x2(j) y2(j)];
    tank1 = [x2(j) + r * cos(aalpha(j) + pi / 2), y2(j) + r *... 
sin(aalpha(j) + pi / 2);
            x3(j) + r * cos(aalpha(j) + pi / 2), y3(j) + r *... 
sin(aalpha(j) + pi / 2)];
    tank2 = [x2(j) - r * cos(aalpha(j) + pi / 2), y2(j) - r *... 
sin(aalpha(j) + pi / 2);
            x3(j) - r * cos(aalpha(j) + pi / 2), y3(j) - r *... 
sin(aalpha(j) + pi / 2)];
    wheel1 = [x1(j) + r * cos(0 : .1 : 2 * pi)'', y1(j) + r *... 
sin(0 : .1 : 2 * pi)''];
    wheel1(size(wheel1, 1) + 1, :) = wheel1(1, :);
    wheel2 = [x2(j) + r * cos(0 : .1 : 2 * pi) - pi / 2 +... 
aalpha(j) - phi(j))', y2(j) + r * sin((0 : .1 : 2 * pi)... 
- pi / 2 + aalpha(j) - phi(j))'];
wheel2(size(wheel2, 1) + 1, :) = wheel2(1, :);
wheel3 = [x3(j) + r * cos((0 : .1 : 2 * pi) - pi / 2 +...   
   aalpha(j) - phi(j))’, y3(j) + r * sin((0 : .1 : 2 * pi) -...   
   pi / 2 + aalpha(j) - phi(j))]';
wheel3(size(wheel3, 1) + 1, :) = wheel3(1, :);
hold off;
plot(boom(:, 1), boom(:, 2), 'linewidth', 2, 'color', 'k');
hold on;
plot(tank1(:, 1), tank1(:, 2), 'linewidth', 2, 'color', 'g');
plot(tank2(:, 1), tank2(:, 2), 'linewidth', 2, 'color', 'g');
plot(wheel1(:, 1), wheel1(:, 2), 'linewidth', 2, 'color', 'k');
plot(wheel2(:, 1), wheel2(:, 2), 'linewidth', 2, 'color', 'g');
plot(wheel2(1, 1), wheel2(1, 2), 'y.');
plot(wheel3(:, 1), wheel3(:, 2), 'linewidth', 2, 'color', 'g');
plot(wheel3(1, 1), wheel3(1, 2), 'y.');
plot(xc(1:j), yc(1:j), 'linewidth', 2, 'color', 'b');
plot(xc(j), yc(j), 'b.');
plot(xc(j), yc(j), 'bo');
axis([-0.3, 0.3, 0, 0.6]);
set(gca, 'FontSize', 12, 'FontName', 'Californian FB');
text(0, .55, [t = ', num2str(floor(10 * t(j)) / 10),...   
   ' sec.'], 'Color', 'r', 'FontSize', 12, 'FontWeight', 'b');
xlabel('x [m]', 'FontWeight', 'b')
ylabel('y [m]', 'FontWeight', 'b')
axis equal;
grid;
drawnow;
end
Appendix E

Upright Maneuver Simulation

%%%Nonlinear Equations of Motion
%%%Up(A, B, C, K, L)

function [] = Up(a, b, c, k, l)

close all;

global A B C K L L1 L2 Lb Lt r m1 m2 x0

A = a;
B = b;
C = c;
K = k;
L = l;

L1 = 0.20136242; % meters
L2 = 0.09091696; % meters
Lb = 0.23177500; % meters
Lt = 0.16660000; % meters
r = 0.03050835; % meters
m1 = 1.09601699; % kilograms
m2 = 0.82729460; % kilograms

t_final = 8;
x0 = [0, 0, 0, pi / 2, pi / 2, pi];
x = [0, 0, 0, pi, pi, 2 * pi]’;
x_hat = [0, 0, 0, pi, pi, 2 * pi]’;
x_star = [x; x_hat];
%Simulation Time Step Size
    h = .001;
    disp('Simulating nonlinear model...')

%Initialize Elapsed Time Vector
    t = zeros(t_final / h + 1, 1);
    n = 0;
    p_aalpha_d = zeros(t_final / h, 1);
    p_phi_d = zeros(t_final / h, 1);
    p_theta_d = zeros(t_final / h, 1);
    p_aalpha = zeros(t_final / h, 1);
    p_phi = zeros(t_final / h, 1);
    p_theta = zeros(t_final / h, 1);
    p_aalpha_d_hat = zeros(t_final / h, 1);
    p_phi_d_hat = zeros(t_final / h, 1);
    p_theta_d_hat = zeros(t_final / h, 1);
    p_aalpha_hat = zeros(t_final / h, 1);
    p_phi_hat = zeros(t_final / h, 1);
    p_theta_hat = zeros(t_final / h, 1);

%Run Nonlinear Simulation
while t(n + 1) < t_final

    %Update RK4 Iteration Counter
    n = n + 1;

    %RK4: evaluate slope at four points from z_n to z_n + 1, to get
    %k1, k2, k3, k4, then solve for z_n + 1
    k1 = f_of_x(x_star);
    k2 = f_of_x(x_star + (h / 2) * k1);
    k3 = f_of_x(x_star + (h / 2) * k2);
    k4 = f_of_x(x_star + h * k3);

    %Calculate z at new interval using linear combination of slopes
    %k1 - k4 to draw a line from zn to zn + 1
    x_star = x_star + (h / 6) * k1 + (h / 3) * (k2 + k3) + (h / 6)... * k4;

    x = x_star(1 : 6);
    x_hat = x_star(7 : 12);

    %Update Elapsed Time Vector
\[ t(n + 1) = t(n) + h; \]
\[ t(n) \]

\% Parse state vector, \( x \), into named state variables
\[
\begin{align*}
    \text{p}_a\alpha_d(n) &= x(1); \\
    \text{p}_\phi_d(n) &= x(2); \\
    \text{p}_\theta_d(n) &= x(3); \\
    \text{p}_a\alpha(n) &= x(4); \\
    \text{p}_\phi(n) &= x(5); \\
    \text{p}_\theta(n) &= x(6); \\
\end{align*}
\]

\[
\begin{align*}
    \text{p}_a\alpha_d\hat{}(n) &= x_\hat{}(1); \\
    \text{p}_\phi_d\hat{}(n) &= x_\hat{}(2); \\
    \text{p}_\theta_d\hat{}(n) &= x_\hat{}(3); \\
    \text{p}_a\alpha\hat{}(n) &= x_\hat{}(4); \\
    \text{p}_\phi\hat{}(n) &= x_\hat{}(5); \\
    \text{p}_\theta\hat{}(n) &= x_\hat{}(6); \\
\end{align*}
\]

end

\% Remove last extra element in time vector, \( t \)
\[
\begin{align*}
    \text{temp} &= t; \\
    \text{clear} \ t; \\
    \text{t} &= \text{temp}(1 : \text{end} - 1); \\
    \text{clear} \ \text{temp}; \\
\end{align*}
\]

plot\_switchblade(t, \( \text{p}_a\alpha \), \( \text{p}_\phi \), \( \text{p}_\theta \));
figure;
plot(t, [\( \text{p}_a\alpha \) \( \text{p}_\phi \) \( \text{p}_\theta \)], '-', t,...
    [\( \text{p}_a\alpha\hat{} \) \( \text{p}_\phi\hat{} \) \( \text{p}_\theta\hat{} \)], '--', 'linewidth', 2);
set(gca, 'FontSize', 12, 'FontName', 'Californian FB');
xlabel('Time [sec]', 'FontWeight', 'b');
ylabel('Angular Position [rad]', 'FontWeight', 'b');
set(legend('$\alpha$', '$\phi$', '$\theta$', '$\hat{\alpha}$',
    '$\hat{\phi}$', '$\hat{\theta}$'), 'Interpreter', 'LaTeX');
figure;
plot(t, [\( \text{p}_a\alpha_d \) \( \text{p}_\phi_d \) \( \text{p}_\theta_d \)], '-', t,...
    [\( \text{p}_a\alpha_d\hat{} \) \( \text{p}_\phi_d\hat{} \) \( \text{p}_\theta_d\hat{} \)], '--', 'linewidth', 2);
set(gca, 'FontSize', 12, 'FontName', 'Californian FB');
xlabel('Time [sec]', 'FontWeight', 'b');
ylabel('Angular Velocity [rad/sec]', 'FontWeight', 'b');
set(legend('$\dot{\alpha}$', '$\dot{\phi}$', '$\dot{\theta}$',
    '$\dot{\hat{\alpha}}$', '$\dot{\hat{\phi}}$',
    '$\dot{\hat{\theta}}$'), 'Interpreter', 'LaTeX');
\[
\dot{\hat{\theta}}
\]

%F_OF_X
function \[dx_{\text{dt}}\] = f_of_x(x_star)

global A B C K L x0

x = x_star(1 : 6);
x_hat = x_star(7 : 12);

% \(x = [aalpha_d; phi_d; theta_d; aalpha; phi; theta]\)
aalpha_d = x(1);
phi_d = x(2);
theta_d = x(3);
aalpha = x(4);
% phi = x(5);
theta = x(6);

u = - K * (x_hat - x0');

for j = 1 : length(u)
    if abs(u(j)) > 12
        u(j) = 12 * sign(u(j));
        disp(['STALL ' num2str(j)])
    end
end

u1 = u(1);
u2 = u(2);

% u1 = 0;
% u2 = 0;

E_N = [...;
...;
...;
aalpha_d;
phi_d;
theta_d];

% E_N = E_N + [100 * (rand - 0.5); zeros(5, 1)];

dx_hat_{\text{dt}} = (A - L * C) * (x_hat - x0') + B * u + L * C * (x - x0');
dx_dt = [E_N; dx_hat_dt];

%Plot Simulation Results
function [] = plot_switchblade(t, aalpha, phi, theta)

global L1 L2 Lb Lt r m1 m2

x3 = - r * (aalpha - phi);
y3 = x3 - x3 + r;

x2 = x3 + Lt * cos(aalpha);
y2 = y3 + Lt * sin(aalpha);

x1 = x2 - Lb * cos(theta - aalpha);
y1 = y2 + Lb * sin(theta - aalpha);

xc = x3 + (m2 * L2 * cos(aalpha) + m1 * (Lt * cos(aalpha) - L1 *...)
    cos(theta - aalpha)) / (m1 + m2);
yc = y3 + (m2 * L2 * sin(aalpha) + m1 * (Lt * sin(aalpha) + L1 *...)
    sin(theta - aalpha)) / (m1 + m2);

%Animate Simulation Results
for j = 1 : round((1 + length(t)) / 4) : length(t)
    figure;
    boom = [x1(j) y1(j); x2(j) y2(j)];
    tank1 = [x2(j) + r * cos(aalpha(j) + pi / 2), y2(j) + r *...]
        sin(aalpha(j) + pi / 2);
    tank1 = [x3(j) + r * cos(aalpha(j) + pi / 2), y3(j) + r *...]
        sin(aalpha(j) + pi / 2)];
    tank2 = [x2(j) - r * cos(aalpha(j) + pi / 2), y2(j) - r *...]
        sin(aalpha(j) + pi / 2);
    tank2 = [x3(j) - r * cos(aalpha(j) + pi / 2), y3(j) - r *...]
        sin(aalpha(j) + pi / 2)];
    wheel1 = [x1(j) + r * cos(0 : .1 : 2 * pi)', y1(j) + r *...]
        sin(0 : .1 : 2 * pi)'];
    wheel1 = [x2(j) + r * cos(0 : .1 : 2 * pi) - pi / 2 +...]
        aalpha(j) - phi(j)']));
    wheel12 = [x2(j) + r * sin(0 : .1 : 2 * pi)... - pi / 2 + aalpha(j) - phi(j))]';

```
wheel2(size(wheel2, 1) + 1, :) = wheel2(1, :);
wheel3 = [x3(j) + r * cos((0 : 1 : 2 * pi) - pi / 2 + ...
apalpha(j) - phi(j))', y3(j) + r * sin((0 : 1 : 2 * pi) -...
pi / 2 + aalpha(j) - phi(j))]';
wheel3(size(wheel3, 1) + 1, :) = wheel3(1, :);
hold off;
plot(boom(:, 1), boom(:, 2), 'linewidth', 2, 'color', 'k');
hold on;
plot(tank1(:, 1), tank1(:, 2), 'linewidth', 2, 'color', 'g');
plot(tank2(:, 1), tank2(:, 2), 'linewidth', 2, 'color', 'g');
plot(wheel1(:, 1), wheel1(:, 2), 'linewidth', 2, 'color', 'k');
plot(wheel2(:, 1), wheel2(:, 2), 'linewidth', 2, 'color', 'g');
plot(wheel2(1, 1), wheel2(1, 2), 'y.');
plot(wheel3(:, 1), wheel3(:, 2), 'linewidth', 2, 'color', 'g');
plot(wheel3(1, 1), wheel3(1, 2), 'y.');
plot(xc(1:j), yc(1:j), 'linewidth', 2, 'color', 'b');
plot(xc(j), yc(j), 'b.');
plot(xc(j), yc(j), 'bo');
axis([-0.3, 0.3, 0, 0.6]);
set(gca, 'FontSize', 12, 'FontName', 'Californian FB');
text(0, .55, ['t = ', num2str(floor(10 * t(j)) / 10),...
' sec.'], 'Color', 'r', 'FontSize', 12, 'FontWeight', 'b');
xlabel('x [m]', 'FontWeight', 'b')
ylabel('y [m]', 'FontWeight', 'b')
axis equal;
grid;
drawnow;
end
Bibliography


