2012

TRAJECTORY CONTROL OF A TWO-WHEELED ROBOT

Michael D. Peltier
University of Rhode Island, michael_peltier2003@yahoo.com

TRAJECTORY CONTROL OF A TWO-WHEELED ROBOT

BY

MICHAEL D. PELTIER

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

ELECTRICAL ENGINEERING

UNIVERSITY OF RHODE ISLAND

2012

MASTER OF SCIENCE THESIS

OF

MICHAEL D. PELTIER

APPROVED:

Thesis Committee:

Major Professor Richard J. Vaccaro

Peter F. Swaszek

William J. Palm

Nasser H. Zawia

DEAN OF THE GRADUATE SCHOOL

UNIVERSITY OF RHODE ISLAND

2012

ABSTRACT

Robots have been used in many applications in the past three decades. One type of robot is a two-wheeled robot that requires control for both balancing and maneuvering. This thesis shows the design of a controller that can both balance and provide trajectory control for a two-wheeled robot. The controller is a digital tracking system that utilizes pole placement for system stability. The thesis provides a detailed description for modeling the plant, design of the controller, simulating the final system and implementation into hardware. Three pole placement methods are analyzed as well as tracking designs that use steady-state or ramp tracking. Each of these controllers have simulation results and the controller that has the best simulation performance is implemented into the hardware. The final controller is implemented into Lego® Mindstorm® hardware using Matlab® and Simulink®. The results of the simulations are compared to the results found in hardware.

ACKNOWLEDGMENTS

I would like to acknowledge Dr. Richard J. Vaccaro for his guidance and advice in the area of control theory. The knowledge he has provided was instrumental in the completion of this thesis. I would also like to thank my wife Dr. Dena L. Peltier for her support and patience.

TABLE OF CONTENTS

Page

Page

LIST OF TABLES

LIST OF FIGURES

Figure **Page**

Figure **Page**

CHAPTER 1

Introduction

The ability to control robots has been in the forefront of control theory in the past three decades. Robots have been used in many applications. They can be used to make everyday living for the handicapped easier while allowing tasks to be performed more efficiently in industry. These robots are becoming more sophisticated to where they can autonomously perform tasks.

The control of each type of robot becomes its own engineering problem. Robots can move around using wheels, tracks or belts. They may even stay in place and perform tasks along an assembly line. The actuators can be electromechanical or use hydraulics. All these differences result in different kinematics and dynamics of the robot.

There are many examples of controlling an inverted pendulum. Seul Jung and Sung Su Kim [1] use a Neural Network to provide tracking control for a single-input-multiple-output (SIMO) two-wheeled robot. The cart is controlled forward and back, tracking a sinusoidal motion. Vaccaro [2] uses digital state feedback tracking to control a SIMO inverted pendulum that uses a cart coupled to a drive screw. In work introduced by S.W. Nawawi et al, [3] a sliding mode controller was used to balance a two-wheeled robot; however, it does not indicate that the controller can be used to control the robot's trajectory.

1.1 Problem Identification

The two-wheeled robot is an inverted pendulum that can also move about in a two dimensional plane. The robot contains sensors that provide feedback for balancing and trajectory tracking. Pole placement will be used in the design of the controller. Therefore, the two-wheeled robot will need to be modeled. Since

pole placement is used to design the controller the method of placement needs to be scrutinized for both stability and performance. Consideration will also need to be taken when implementing the new controller in real hardware.

1.2 Contributions of this Thesis

The two-wheeled robot has some challenges that are not seen in the inverted pendulum problem. In the inverted pendulum problem found in the book by Vaccaro [2], the pendulum is mounted to a cart. The cart is directly coupled to the drive motor, while the pendulum moves freely on a mounted encoder. The two-wheeled robot has the inverted pendulum directly coupled to the drive motor. Also, the inverted pendulum rotates on a single shaft. The two-wheeled robot does not rotate on one shaft, but uses two shafts, one for each wheel. This problem was examined by Yamamoto [4] by controlling the average of the two wheels. Yamamoto was successful in creating a closed-loop control for balancing; however, the controller did not provide closed-loop control for trajectory tracking. Rather than control the average of the wheels (a single control input), the controller in this thesis will send independent coordinated signals to each motor (two control inputs). The design of this system will require the tools of multivariable control theory. Tests with the hardware robot will demonstrate the applicability of multivariable control theory to a real-world system. Modifications may have to be made for the theoretical results to be implemented in hardware.

1.3 Overview of this Thesis

The research for this master's thesis will require a scientific approach. The dynamics of the system need to be carefully scrutinized and also the micro controller architecture, which can introduce delays [5]. A plant model exists [4] and some system parameters will be found directly from measurements of the hardware

system. The final plant will be used to design a controller based on Digital Tracking System Theory. The design will include a comparison of multiple methods used to calculate a feedback matrix for the desired closed-loop pole locations, and a stability analysis to determine robustness. Simulations will be used to observe the response of the system. The controller that provides the best stability margins and performance will be loaded into hardware for testing. Finally, collected data from the hardware and simulations will be compared.

List of References

[1] S. Jung and S. S. Kim, "Control experiment of a wheel-driven mobile inverted pendulum using neural network," *IEEE Transactions On Control Systems Technology*, vol. 16, pp. 297–303, March 2008.

[2] R. J. Vaccaro, *Digital Control A State-Space Approch.* New York, New York, United States of America: McGraw-Hill, Inc, 1995.

[3] S. Nawawi, M. Ahmad, J. Osman, A. Husain, and M. Abdollah, "Controller design for two-wheels inverted pendulum mobile robot using PISMC," *4th Student Conference on Research and Development*, pp. 194 – 199, June 2006.

[4] Y. Yamamoto, "Nxtway-gs model-based design - control of self-balancing two-wheeled robot built with lego mindstorms nxt -," May 2009, unpublished. [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/19147-nxtway-gs-self-balancing-two-wheeled-robot-controller-design?controller=file_infos&download=true

[5] T. Chikamasa, "Embedded coder robot nxt modeling tips," April 2010, unpublished. [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/13399-embedded-coder-robot-nxt-demo?controller=file_infos&download=true

CHAPTER 2

Two-Wheeled Robot Model

The two-wheeled robot model used in this thesis was derived by Yamamoto [1]. Figure 2.1 shows a static model of the two-wheeled robot. This static model is also shown in Figure 2.2 as a side and plane view in order to bring clarity into the derivation of the robot's dynamic motion. Table 2.1 defines the variables and Table 2.2 defines the constants of the two-wheeled robot.

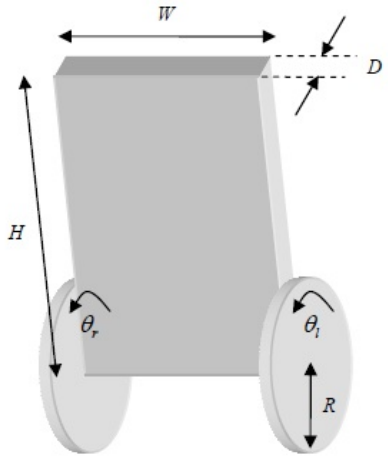

Figure 2.1. Two-wheeled Robot [1]

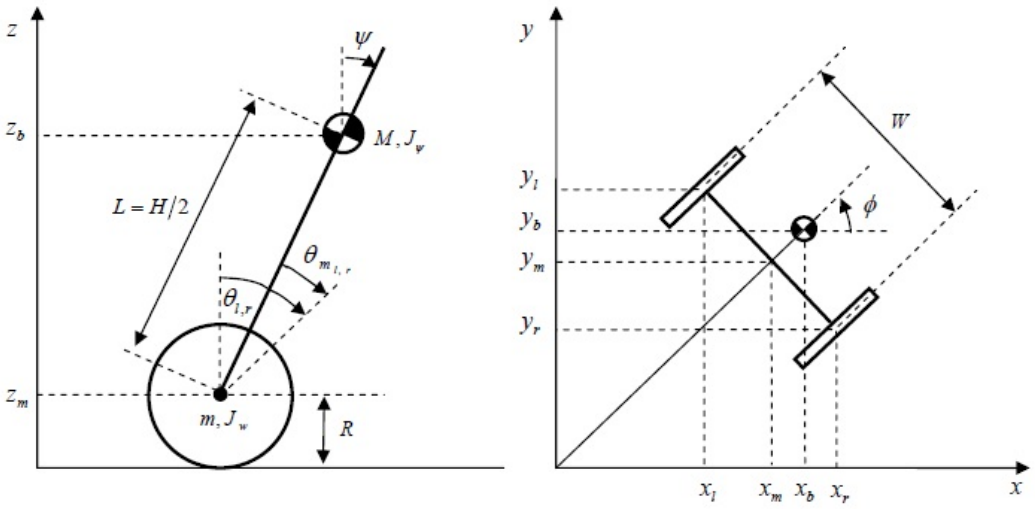

Figure 2.2. Side and Plane View of Two-wheeled Robot [1]

Variable	Units	Description
θ	$[rad]$	Average angle of left and right wheel
Ψ	$[rad]$	Body pitch angle
ϕ	$[rad]$	Body yaw angle
θ_r	$[rad]$	Rotational angle of the right wheel
θ_l	$[rad]$	Rotational angle of the left wheel
θ_{m_r}	$[rad]$	DC motor angle of the right wheel
θ_{m_l}	$[rad]$	DC motor angle of the left wheel
x_m, y_m, z_m	$[m]$	Coordinates of the centerline
x_r, y_r, z_r	$[m]$	Coordinates of the right wheel
x_l, y_l, z_l	$[m]$	Coordinates of the left wheel
x_b, y_b, z_b	$[m]$	Coordinates of the center of gravity
v_l	$[Volts]$	Voltage applied to the left motor
v_r	$[Volts]$	Voltage applied to the right motor

Table 2.1. Two-wheeled Inverted Robot Variables

Constant	Value	Units	Description
g	9.81	$[m/s^2]$	Gravity acceleration
m	0.03	$[kg]$	Wheel weight
J_w	2.4e-5	$[kgm^2]$	Wheel inertia moment
W	0.175	$[m]$	Width of the robot body
D	0.04	[m]	Depth of the robot body
H	0.144	[m]	Height of the robot body
L	0.079	[m]	Distance of the center of mass from the wheel axle
R	0.04	$[m]$	Radius of the wheel
J_Ψ	$\frac{ML^2}{3}$	$[kgm^2]$	Body pitch inertia moment
J_ϕ	$\frac{M\left(W^2+D^2\right)}{12}$	$[kgm^2]$	Body yaw inertia moment
R_m	6.69	$[\Omega]$	DC motor resistance
K_b	0.468	$[Vsec/rad]$	DC motor back EMF constant
K_t	0.317	$[Nm/A]$	DC motor torque constant
f_m	0.0224	N/A	Friction coefficient between body and DC motor

Table 2.2. Two-wheeled Robot Constants

The state-space equation for the two-wheeled robot is based on the coordinate system in Figure 2.2. The equations that define the motion of the two-wheeled robot in the coordinate system are shown in Equations (2.1) through (2.7) and have the initial heading along the positive x-axis at t = 0. These equations show that the final state-space equation only requires the variable θ, Ψ, and ϕ.

$$(\theta_l, \theta_r) = \left(\theta_{m_l} + \Psi, \theta_{m_r} + \Psi\right) \tag{2.1}$$

$$(\theta, \phi) = \left(\frac{1}{2}\left(\theta_l + \theta_r\right), \frac{R}{W}\left(\theta_r - \theta_l\right)\right) \tag{2.2}$$

$$(x_m, y_m, z_m) = \left(\int \dot{x}_m dt, \int \dot{y}_m dt, R\right) \tag{2.3}$$

$$(\dot{x}_m, \dot{y}_m) = \left(R\dot{\theta}\cos\phi, R\dot{\theta}\sin\phi\right) \tag{2.4}$$

$$(x_l, y_l, z_l) = \left(x_m - \frac{W}{2}\sin\phi, y_m + \frac{W}{2}\cos\phi, z_m\right) \tag{2.5}$$

$$(x_r, y_r, z_r) = \left(x_m + \frac{W}{2}\sin\phi, y_m - \frac{W}{2}\cos\phi, z_m\right) \tag{2.6}$$

$$(x_b, y_b, z_b) = (x_m + L\sin\Psi\cos\phi, y_m + L\sin\Psi\sin\phi, z_m + L\cos\Psi) \tag{2.7}$$

2.1 State-Space Equations

The state-space equation is derived from the translational kinetic energy, rotational kinetic energy, potential energy, DC motor torque and the DC motor viscous friction. Equation (2.8) is the motion equation for the two-wheeled robot for the variables θ and Ψ.

$$\mathbf{E}\begin{bmatrix} \ddot{\theta} \\ \ddot{\Psi} \end{bmatrix} + \mathbf{F}\begin{bmatrix} \dot{\theta} \\ \dot{\Psi} \end{bmatrix} + \mathbf{G}\begin{bmatrix} \theta \\ \Psi \end{bmatrix} = \mathbf{H}\begin{bmatrix} v_l \\ v_r \end{bmatrix} \tag{2.8}$$

The matrices **E**,**F**,**G** and **H** are defined in Equation (2.9) and the constants are defined in Equation (2.10).

$$\begin{aligned} \mathbf{E} &= \begin{bmatrix} C1 & C2 \\ C2 & C3 \end{bmatrix} \\ \mathbf{F} &= 2 \times \begin{bmatrix} C4 & -C4 \\ -C4 & C4 \end{bmatrix} \\ \mathbf{G} &= \begin{bmatrix} 0 & 0 \\ 0 & -C5 \end{bmatrix} \\ \mathbf{H} &= \begin{bmatrix} C6 & C6 \\ -C6 & -C6 \end{bmatrix} \end{aligned} \tag{2.9}$$

$$\begin{aligned} C1 &= (2m + M)R^2 + J_w \\ C2 &= MRL \\ C3 &= ML^2 + J_\Psi \\ C4 &= \frac{K_t K_b}{R_m} + f_m \\ C5 &= MgL \\ C6 &= \frac{K_t}{R_m} \end{aligned} \tag{2.10}$$

Equation (2.11) is the motion equation for the two-wheeled robot for the variable ϕ. The constants are defined in Equation (2.12).

$$I\ddot{\phi} + J\dot{\phi} = K(v_r - v_l) \tag{2.11}$$

$$
\begin{aligned}
I &= \frac{1}{2}mW^2 + J_\phi + \frac{W^2}{2R^2}J_w \\
J &= \frac{W^2}{2R^2}\left(\frac{K_t K_b}{R_m} + f_m\right) \\
K &= \frac{W}{2R}\frac{K_t}{R_m}
\end{aligned} \tag{2.12}
$$

The state-space Equation (2.13) can be derived using Equations (2.8) and (2.11).

$$
\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \tag{2.13}
$$

The state vector is defined by $\mathbf{x}$ and the input vector is defined by $\mathbf{u}$.

$$
\mathbf{x} = \begin{bmatrix} \theta \\ \Psi \\ \dot{\theta} \\ \dot{\Psi} \\ \phi \\ \dot{\phi} \end{bmatrix}, \mathbf{u} = \begin{bmatrix} v_l \\ v_r \end{bmatrix} \tag{2.14}
$$

The matrices $\mathbf{A}$ and $\mathbf{B}$ are defined in Equation (2.15) along with the definitions of the constants.

$$
\begin{aligned}
\mathbf{A} &= \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ \mathbf{A1}_{(1,1)} & \mathbf{A1}_{(1,2)} & \mathbf{A2}_{(1,1)} & \mathbf{A2}_{(1,2)} & 0 & 0 \\ \mathbf{A1}_{(2,1)} & \mathbf{A1}_{(2,2)} & \mathbf{A2}_{(2,1)} & \mathbf{A2}_{(2,2)} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & -\frac{J}{I} \end{bmatrix} \\
\mathbf{B} &= \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ B1_{(1,1)} & B1_{(1,2)} \\ B1_{(2,1)} & B1_{(2,2)} \\ 0 & 0 \\ -K/I & K/I \end{bmatrix} \\
\mathbf{A1} &= -\mathbf{E}^{-1}\mathbf{G} \\
\mathbf{A2} &= -\mathbf{E}^{-1}\mathbf{F} \\
\mathbf{B1} &= \mathbf{E}^{-1}\mathbf{H}
\end{aligned} \tag{2.15}
$$

2.2 Experimental Parameters

The model that was provided in reference [1] had all the required parameters defined. However, it was determined that some of the parameters could be more accurate if they were extracted experimentally. The distance to the center of mass L and the friction coefficient between the body and DC motor f_m were two of these parameters. L was chosen since the current model assumed that the body was rectangular. This is clearly not true since the shape of the body is not rectangular. f_m was chosen because the friction can easily change from one DC motor to another.

The first parameter that was extracted was L. This was accomplished by using the natural frequency of the freely hanging robot. By suspending the robot upside down by the axle and not rotating the wheels, Equation (2.8) can be simplified. Since the wheels are not rotating, the angle θ and Ψ are equal and $\mathbf{u}$ is equal to 0. This reduces Equation (2.8) to the state Equation (2.16).

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ \frac{C5}{C2+C3} & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_3 \end{bmatrix} \tag{2.16}$$

The following equation can be used to characterize an inverted pendulum by its natural frequency w_p as stated in Vaccaro [2].

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ {w_p}^2 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_3 \end{bmatrix} \tag{2.17}$$

Using Equations (2.16) and (2.17), the natural frequency of the robot is defined by Equation (2.18).

$$w_p = \sqrt{\frac{C5}{C2+C3}} \tag{2.18}$$

From Equation (2.18) and substituting the constants for C2, C3 and C5 with the constants from Equation (2.10), the value of L can be determined.

$$L = \frac{3\left(g - {w_p}^2 R\right)}{4{w_p}^2} \tag{2.19}$$

The robot was suspended by the axle and the period of oscillation was recorded from the gyro sensor. The measured period of oscillation of the robot was 0.7647 [sec] which results in a natural frequency of 8.2165 [rad/sec]. Solving for Equation (2.19), the distance of the center of mass to the axial, L equals 7.9 [cm].

The second parameter f_m was also extracted experimentally. It was determined that the DC motor poles were fast and would be difficult to measure using feedback over the robot's serial bluetooth interface. Therefore, the experiment was designed to only examine the steady-state transfer function of the DC motors. This experiment required using a speed controller on each wheel. The robot was held upright by a piece of string. The speed controller was then activated, commanding both wheels to the same speed. Once the robot reached steady-state, the voltages were recorded via the bluetooth data link.

Holding the robot upright allows Equation (2.8) to be simplified. When upright, Ψ and $\dot{\Psi}$ equal 0 which results in Equations (2.20) and (2.21).

$$C1\dot{x}_3 + 2(C4)x_3 = C6\left(v_l + v_r\right) \tag{2.20}$$

$$C2\dot{x}_3 - 2(C4)x_3 = -C6\left(v_l + v_r\right) \tag{2.21}$$

Since a speed controller is being used for the experiment, $\dot{x}_3$ in Equations (2.20) and (2.21) drops out once the two wheels have reached a steady-state speed. Also, setting v_l and v_r to u results in the Equation (2.22).

$$C4x_3 = C6u \tag{2.22}$$

The ratio for the measured output to the measured input is defined as α, where

$$\alpha = \frac{x_3}{u} = \frac{C6}{C4} \tag{2.23}$$

Using Equation (2.23) and substituting the constants for C4 and C6 with the constants from Equation (2.10) the value of f_m can be determined.

$$f_m = \frac{K_t}{R_m\alpha}\left(1 - K_b\alpha\right) \tag{2.24}$$

The constant α was experimentally determined to be 1.0622 [rad/(Vsec)]. Solving for Equation (2.24), the friction coefficient between the body and DC motor, f_m equals 0.0224.

List of References

[1] Y. Yamamoto, "Nxtway-gs model-based design - control of self-balancing two-wheeled robot built with lego mindstorms nxt -," May 2009, unpublished. [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/19147-nxtway-gs-self-balancing-two-wheeled-robot-controller-design?controller=file_infos&download=true

[2] R. J. Vaccaro, *Digital Control A State-Space Approch.* New York, New York, United States of America: McGraw-Hill, Inc, 1995.

CHAPTER 3

Digital Tracking System Theory

The two-wheeled robot must balance while maneuvering. Balancing the robot is the well known inverted pendulum problem. However, to maneuver the robot with closed-loop control is not an easy task. The robot as described in Equation (2.15) has six state variables which all contribute to the stability of both maneuvering and balancing of the robot. The task also has an added level of difficulty due to the fact that there are two control actuators.

The design of the controller was based on the digital tracking system in the book by Vaccaro [1] with aid of the digital control toolbox which can be downloaded from http://www.mathworks.com/matlabcentral/fileexchange/2199-digital-control. It was decided that the robot would have position tracking on the state variable θ and heading tracking on the state variable ϕ. This provides the robot the ability to move in straight lines (control θ), change heading (control ϕ) or follow an arc (control θ and ϕ). The robot balances based on state feedback and always tries to force the states Ψ and $\dot{\Psi}$ to zero.

The digital tracking system requires that the plant model be converted into a discrete-time state-space equation or ZOH model as shown in Equation (3.1).

$$\begin{aligned} \mathbf{x}[k+1] &= \mathbf{\Phi x}[k] + \mathbf{\Gamma u}[k] \\ \mathbf{y}[k] &= \mathbf{Cx}[k] + \mathbf{Du}[k] \\ \mathbf{\Phi} &= e^{\mathbf{A}T} \\ \mathbf{\Gamma} &= \int_0^T e^{\mathbf{A}\tau}\mathbf{B}d\tau \end{aligned} \tag{3.1}$$

where T is the sampling interval and k is the discrete-time index.

The digital control toolbox contains the function zohe that will convert the

continuous state-space matrices $\mathbf{A}$ and $\mathbf{B}$ to $\mathbf{\Phi}$ and $\mathbf{\Gamma}$. The function requires $\mathbf{A}$, $\mathbf{B}$ and T as inputs. It uses a single matrix exponential shown in Equation (3.2) that can simultaneously calculate the matrices $\mathbf{\Phi}$ and $\mathbf{\Gamma}$. The method is credited to C. F. Van Loan [2].

$$e^{\mathbf{M}T} = \begin{bmatrix} \mathbf{\Phi} & \mathbf{\Gamma} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \qquad \mathbf{M} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \tag{3.2}$$

3.1 Controllability

The robot must be determined to be controllable prior to the design of the digital tracking system. Vaccaro [1] defines controllability or reachability in the following manner "The state equation $\mathbf{x}[k+1] = \mathbf{\Phi x}[k] + \mathbf{\Gamma u}[k]$ is said to be *controllable* if it is possible to find an input sequence $\mathbf{u}$[k] that takes the system from an arbitrary initial state $\mathbf{x}[0] = \mathbf{z}_1$ to an arbitrary final state $\mathbf{x}$[j] $= \mathbf{z}_2$ for some finite j". This is true if the *controllability matrix* $\mathbf{W}_c$, calculated as in Equation (3.3), has a determinant that is not equal to zero or the rank equals n, where n is the number of states. In a single-input-single-output (SISO) system $\mathbf{\Phi}$ is an n $\times$ n matrix and $\mathbf{\Gamma}$ is a n $\times$ 1 matrix. This means that $\mathbf{W}_c$ is an n $\times$ n square matrix and the determinant can be calculated. The two-wheeled robot is a multiple-input-multiple-output (MIMO) system, therefore, $\mathbf{\Gamma}$ is a n $\times$ p matrix and p is defined as the number of inputs. In this case controllability of the ZOH models can only be determined by the rank not the determinant of the matrix $\mathbf{W}_c$.

$$\mathbf{W}_c = \begin{bmatrix} \mathbf{\Gamma} & \mathbf{\Phi\Gamma} & \dots & \mathbf{\Phi}^{n-1}\mathbf{\Gamma} \end{bmatrix} \tag{3.3}$$

The ZOH model must also be controllable. The first criteria is to ensure that the continuous-time plant is controllable. The continuous-time plant is controllable if and only if the system plant transfer function does not contain common roots

in the numerator or denominator. In other words, the continuous-time plant can not contain pole zero cancellation. The second criteria is determined by examining the imaginary part of the poles of the **A** matrix. The imaginary part of the pole with the largest magnitude is defined as β_{max}. The ZOH model is *controllable* if Equation (3.4) is satisfied and the continuous-time plant is controllable. If all the poles are on the real axis that is $\beta_{max} = 0$ and the continuous-time plant is controllable, then the ZOH model is controllable for *any* value of T.

$$T < \frac{\pi}{\beta_{max}} \tag{3.4}$$

3.2 Tracking System

The control of the two-wheeled robot is accomplished by using a digital tracking system as shown in Figure 3.1. The digital tracking system is comprised of the plant, a feedback matrix and additional dynamics. The advantage of this controller is that it is effective in dealing with transient disturbances while also tracking a reference with zero steady-state error. The ability to track a reference with zero steady-state error makes the controller robust because it is not sensitive to model inaccuracies. Finally, the controller will not only be effective in dealing with transient disturbances but also in dealing with constant disturbances. The constant disturbance rejection and reference tracking is dependent on the additional dynamics.

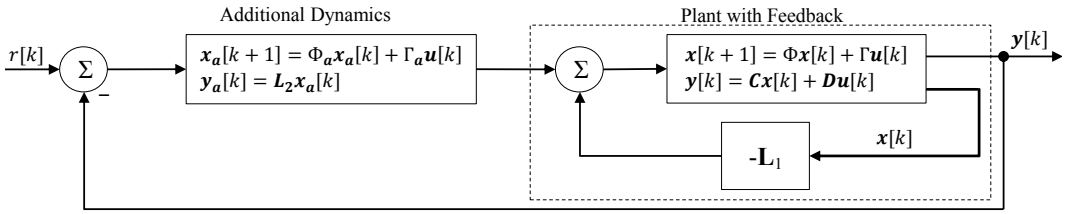

Figure 3.1. Digital Tracking System

The design of the digital tracking system starts by obtaining a ZOH model as in Equation (3.1). Then the additional dynamics matrices $\mathbf{\Phi}_a$ and $\mathbf{\Gamma}_a$ need to be determined. The additional dynamics, ZOH plant model and desired closed-loop poles are then used to calculate the feedback matrix which consist of $\mathbf{L}_1$ and $\mathbf{L}_2$.

3.3 Additional Dynamics

The additional dynamics are required to have a tracking system. The additional dynamics in a digital tracking system are implemented as a state-space equation as seen in Figure 3.1. The inputs to the additional dynamics are the difference of the measured outputs from the plant and the commanded reference. The outputs of the additional dynamics are based on the feedback matrix $\mathbf{L}_2$. The additional dynamics drive the system to zero steady-state error by adjusting the output. This can be implemented to remove steady-state error only (single integrator) or remove error to a ramp (double integrator).

In hardware the output of a controller can saturate. At the point of saturation, or just before saturation, the additional dynamics need to be disabled. Disabling the additional dynamics prevents the integrators from running away, or "winding up".

3.4 Feedback Matrix

The feedback matrix is used to place the closed-loop poles. The selection of the closed-loop poles is performed for the continuous-time system and later mapped to the z-plane. The location of the closed-loop poles are based on the desired settling time Ts. The first poles selected are plant poles that are sufficiently damped. Sufficiently damped plant poles are poles which have real parts that are less than -4.62/Ts. Complex poles that are not sufficiently damped have their real parts changed to -4.62/Ts, while maintaining the original imaginary parts. This will add

damping to the complex poles. The remaining closed-loop poles may be placed using the tabulated values of normalized Bessel poles scaled by the desired settling time Ts [1]. Once the closed-loop continuous-time domain poles are determined then they must be mapped to the discrete-time domain using

$$\mathbf{z} = e^{\mathbf{s}T} \tag{3.5}$$

where z is the discrete-time domain pole location, s is the continuous-time domain pole and T is the sampling rate.

The command **fbg** from the controls toolbox can now be used to calculate the feedback matrix. There are also other tools provided by Matlab® such as **place** and **acker** that can be used to generate the feedback matrix. The command **acker** will not be examined in this thesis. It only produces results for a single input system, and the two-wheeled robot is a multiple input system. There will also be another recently developed algorithm called **TFBG** [3] that will be compared to other methods. Each of the algorithms require an open-loop ZOH model and the desired closed-loop poles. In a digital tracking system, the open-loop ZOH model as well as the number of closed-loop poles are determined by both the additional dynamics and the plant model. The functions **place**, **fbg** and **TFBG** require the open-loop state space model of the additional dynamics and the plant $\mathbf{\Phi}_d$, $\mathbf{\Gamma}_d$ and discrete pole location z as the inputs and will return $\mathbf{L}$.

$$\mathbf{\Phi}_d = \begin{bmatrix} \mathbf{\Phi} & \mathbf{0} \\ \mathbf{\Gamma}_a\mathbf{C} & \mathbf{\Phi}_a \end{bmatrix} \tag{3.6}$$

$$\mathbf{\Gamma}_d = \begin{bmatrix} \mathbf{\Gamma} \\ \mathbf{0} \end{bmatrix} \tag{3.7}$$

$$\mathbf{L} = \begin{bmatrix} \mathbf{L}_1 & \mathbf{L}_2 \end{bmatrix} \tag{3.8}$$

The closed-loop system matrix $\mathbf{\Phi}_d - \mathbf{\Gamma}_d\mathbf{L}$ has poles λ_i and eigenvectors $\mathbf{\Psi}_i$ that satisfy

$$\left(\mathbf{\Phi}_d - \mathbf{\Gamma}_d\mathbf{L}\right)\mathbf{\Psi}_i = \lambda_i\mathbf{\Psi}_i \tag{3.9}$$

This equation can be rewritten as

$$\mathbf{P}(\lambda_i) \begin{bmatrix} \mathbf{\Psi}_i \\ \mathbf{L}\mathbf{\Psi}_i \end{bmatrix} = \mathbf{0} \tag{3.10}$$

where

$$\mathbf{P}(\lambda_i) \stackrel{def}{=} \begin{bmatrix} (\lambda_i \mathbf{I} - \mathbf{\Phi}_d) & \mathbf{\Gamma}_d \end{bmatrix} \tag{3.11}$$

The dimensions of the $\mathbf{P}(\lambda_i)$ are $n \times (n+p)$ where n is the number of states and p is the number of inputs. This means that the null-space has a dimension p which is partitioned into an n-vector $\mathbf{\Psi}_i$ and a p-vector $\mathbf{L}\mathbf{\Psi}_i$. There are up to p linearly independent vectors that can be chosen for each pole location in the closed-loop system. The number of vectors chosen for each pole is equal to the multiplicity of the pole.

To find the value of $\mathbf{L}$, each null-space vector for the desired pole location is partitioned into a "top part" $\mathbf{t}_i$ and "bottom part" $\mathbf{b}_i$. Using Equation (3.10) then $\mathbf{t}_i$ can be used to solve for $\mathbf{L}$.

$$\mathbf{b}_i = \mathbf{L}\mathbf{t}_i \tag{3.12}$$

Since n is equal to the number of desired poles, this is repeated n times, grouping all the top and bottom null-space vectors. This results in a top matrix $\mathbf{T}$ which is $n \times n$ and bottom matrix $\mathbf{B}$ which is $n \times p$ with the equation

$$\mathbf{B} = \mathbf{L}\mathbf{T} \tag{3.13}$$

which a unique $\mathbf{L}$ can be solved for by

$$\mathbf{L} = \mathbf{B}\mathbf{T}^{-1} \tag{3.14}$$

Although this provides a unique solution for $\mathbf{L}$, the construction of $\mathbf{B}$ and $\mathbf{T}$ are not *unique*. Different constructions of $\mathbf{B}$ and $\mathbf{T}$ will result in the desired closed-loop poles, however may not produce the optimal performance for the closed-loop

system [4]. It should also be noted that to avoid $\mathbf{L}$ having complex values, complex poles need to include the complex conjugate pole location.

The algorithms **fbg** and Matlab's **place** are based on methods introduced by Kautsky, Nichols, and Van Dooren [5]. This method calculates eigenvectors that are as orthogonal as possible, while achieving the desired closed-loop poles. This is accomplished by using an iterative method. The matrix $\mathbf{T}$ is first found by initializing the matrix using the method above. Then, one at a time, each eigenvector in $\mathbf{T}$ is projected to be as orthogonal as possible to all other eigenvectors, which produces the optimal eigenvector. Before calculating the next eigenvector in $\mathbf{T}$, the new eigenvector replaces the old one. Once all the eigenvectors of $\mathbf{T}$ are calculated, the process is repeated. The iteration that results in the nonsingular matrix $\mathbf{T}$ with the best condition number is then used to calculate $\mathbf{B}$. In order to calculate the matrix $\mathbf{B}$ a vector $\boldsymbol{\alpha}$ is introduced where

$$\begin{bmatrix} \mathbf{t} \\ \mathbf{b} \end{bmatrix} = \begin{bmatrix} \mathbf{T}(\lambda_i)\,\boldsymbol{\alpha} \\ \mathbf{B}(\lambda_i)\,\boldsymbol{\alpha} \end{bmatrix} \tag{3.15}$$

The vector $\boldsymbol{\alpha}$ can be found with the knowledge of $\mathbf{T}(\lambda_i)$ and $\boldsymbol{\Psi}_i$ by using Equation (3.16).

$$\mathbf{t}_i = \mathbf{T}(\lambda_i)\,\boldsymbol{\alpha}_i = \boldsymbol{\Psi}_i \tag{3.16}$$

The knowledge of $\boldsymbol{\alpha}_i$ allows $\mathbf{b}_i$ to be determined using the the following equation

$$\mathbf{b}_i = \mathbf{B}(\lambda_i)\,\boldsymbol{\alpha}_i \tag{3.17}$$

and repeating this n times to form the matrices $\mathbf{T}$ and $\mathbf{B}$. Then Equation (3.14) can be used to find $\mathbf{L}$.

There are slight differences in both **fbg** and Matlab's **place** that will be seen later numerically. The most likely place where they may differ is the criteria that is used to choose the initial null-space vectors.

The new method of calculating $\mathbf{L}$ is based on optimizing a stability robustness norm, which is described in Burl [6]. The calculation of $\mathbf{L}$ is performed in the function **TFBG**. **TFBG** is initialized using the same method as **fbg**, which is the only commonality between the two algorithms. The new algorithm searches for *stability robustness* in a system with a MIMO plant. The stability robustness indicates how big $\Delta(z)$, the error in the plant, can be before the system goes unstable, where $\mathbf{I} + \Delta(z)$ is a MIMO transfer function cascaded with the plant. The inputs and outputs of $\Delta(z)$ are $\mathbf{y}_d$ and $\mathbf{w}_d$ respectively, and if $\Delta(z) = \mathbf{0}$ then the control system is using the nominal plant model. The condition for stability of the control system is

$$\|\Delta(z)N(z)\|_\infty < 1 \tag{3.18}$$

where $N(z)$ is the transfer function from $\mathbf{w}_d$ to $\mathbf{y}_d$. Using the relationship

$$\|\Delta(z)N(z)\|_\infty \leq \|\Delta(z)\|_\infty \|N(z)\|_\infty < 1 \tag{3.19}$$

the robustness norm δ_{max} can be found by

$$\|\Delta(z)N(z)\|_\infty < \frac{1}{\|N(z)\|_\infty} \tag{3.20}$$
$$\delta_{max} = \frac{1}{\|N(z)\|_\infty}$$

The objective of **TFBG** is to find $\mathbf{L}$ that results in the largest possible δ_{max}. The two-wheeled robot requires another constraint, where the rows of $\mathbf{L}$ have the same magnitude and certain columns have opposite signs.

3.5 Stability Margins

The value of δ_{max} is directly related to the gain and phase margin of the control system. In a SISO system the stability margins can be found using q, which represents gain and phase uncertainty of the plant in classical control. In a MIMO system q could be used; however the stability margins are only representative of the

individual plant inputs without any consideration of simultaneous errors on each plant input. Therefore, there would be stability margins only for each individual input. Assuming δ_{max} has been calculated for a given control system and $\Delta(z)$ is assumed to be a complex number C, the corresponding classical gain margins can be found by using

$$q = 1 + C \tag{3.21}$$

$$q_{max} = 1 + \delta_{max} \tag{3.22}$$

$$q_{min} = 1 - \delta_{max} \tag{3.23}$$

where the upper gain margin (UGM) and lower gain margin (LGM) are

$$UGM = 20 \log_{10} q_{max} \tag{3.24}$$

$$LGM = 20 \log_{10} q_{min} \tag{3.25}$$

The UGM and LGM are to be greater than 3db and less than -3db, which requires δ_{max} to be greater than or equal to 0.4.

In order to find the phase margin, the equation $q = e^{-j\phi} = 1 + C$ is used, where 1 is the center of the circle and C is a disk with the radius δ_{max}. Therefore, the phase margin (PM) is

$$\phi_{max} = 2 \sin^{-1} \left(\frac{\delta_{max}}{2} \right) \tag{3.26}$$

To have a PM of at least 30°, δ_{max} needs to be greater than or equal to 0.5. Therefore, a δ_{max} that has good PM will always have good gain margins.

List of References

[1] R. J. Vaccaro, *Digital Control A State-Space Approch.* New York, New York, United States of America: McGraw-Hill, Inc, 1995.

[2] C. F. V. Loan, "Computing integrals involving the matrix exponential," *IEEE Transactions on Automatic Control*, vol. AC-23, pp. 395–404, June 1978.

[3] R. J. Vaccaro, Department of Electrical, Computer, and Biomedical Engineering, University of Rhode Island, Kingston RI 02881, United States of America, October 2011, personal communication.

[4] V. Sima, A. L. Tits, and Y. Yang, "Computational experience with robust pole assignment algorithms," *Conference on Computer Aided Control Systems Design*, vol. WeA01.6, pp. 36–41, October 2006.

[5] J. Kautsky, N. Nichols, and P. V. Dooren, "Robust pole assignment in linear state feedback," *International Journal of Control*, vol. 41, pp. 1129–1155, 1985.

[6] J. B. Burl, *Linear Optimal Control*. Reading, Massachusetts, United States of America: Addison-Wesley, 1998.

CHAPTER 4

Hardware

The two-wheeled robot is built using the LEGO® Mindstorm® NXT system Figure 4.1.

Figure 4.1. LEGO Two-wheeled Robot

4.1 Architecture

The Lego Mindstorm NXT has a main processor ARM7(AMTEL AT91SAM7S256) for computations and access to digital I/O, and a co-processor (ATMEL AVR) that handles analog sensors and pulse width modulation (PWM) outputs [1]. The co-processor communicates to the main processor via an I2C serial bus. The architecture of the robot can be seen in Figure 4.2.

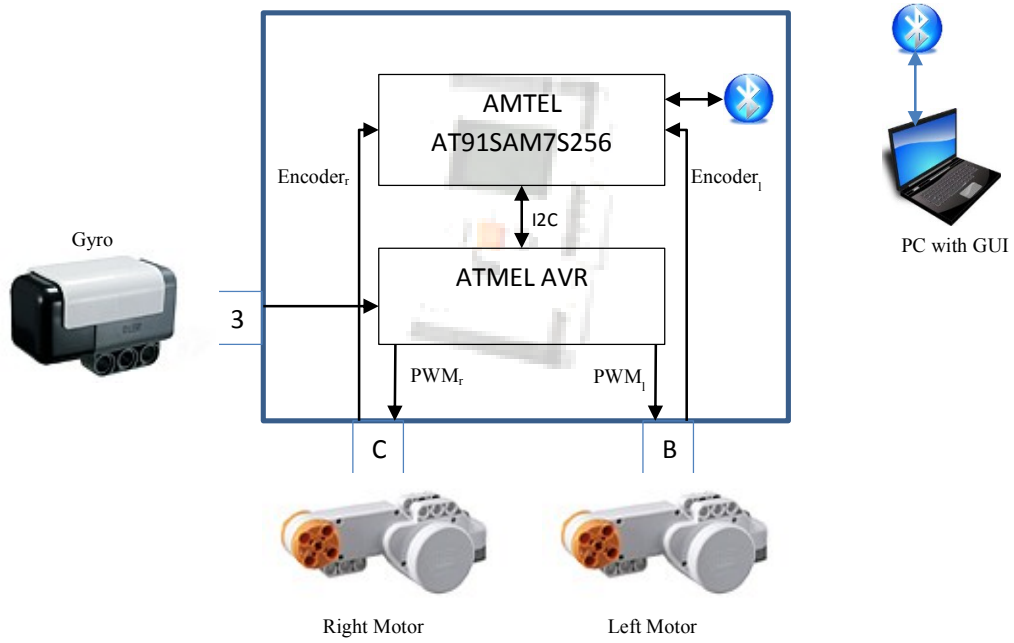

Figure 4.2. Hardware Architecture

There are three sensors and two actuators that are used in the system. The motors receive their PWM commands from the co-processor and the encoder feedback is sent directly to the main processor. The voltages that are applied to each motor range from plus or minus the battery voltage. The encoders for each motor contain 360 counts per revolution and have a resolution of 0.0175 rad/count. The HiTechnic® Gyro Sensor is a single axis gyroscopic sensor. The axis of measurement is shown in Figure 4.3. It can detect angular speed at a resolution of 0.0175 rad/sec. The Gyro is an analog sensor and so it utilizes the co-processor. The data from the AVR is accessed every 2 ms. The PWM commands are sent for 1 ms and the sensor feedback is received the next 1 ms.

Figure 4.3. Gyro Sensor

4.2 Design Tools

The controller for the robot is designed using Matlab and Simulink® and is loaded onto the robot by utilizing Real-Time Workshop®. The interface for Real-Time Workshop to the robot's processor is a compilation of third party software which was coordinated by a method developed by Villanova University [2]. This method allows a Simulink model to be compiled and downloaded directly to the robot's processor. Villanova University developed a Simulink library called VU-LRT Blockset, which allows the controller to interface with all the I/O.

4.3 Bluetooth Interface

The bluetooth interface provides a communications link from a bluetooth enabled device to the Lego Mindstorm NXT main processor. The link contains the information shown in Table 4.1. A PC was chosen as the bluetooth enabled device to communicate to the robot. The VU-LRT Blockset contains a block to allow the controller to interface to the bluetooth connection. The inputs and outputs of the block are vectors where the elements of the vector are 8 bit unsigned integers. The 16 bit float output data needed to be converted into two 8 bit unsigned integers. The 32 bit float commands, from the PC, needed to be reassemble from 8 bit unsigned integers. The PC on the other end also needed to perform the data

conversion in the reverse direction.

Signal Name	Data Type	Data Source
Balance Enable	8 uint	PC
θ_{cmd}	float 32	PC
ϕ_{cmd}	float 32	PC
V_l	float 16	NXT
V_r	float 16	NXT
θ	float 16	NXT
Ψ	float 16	NXT
$\dot{\theta}$	float 16	NXT
$\dot{\Psi}$	float 16	NXT
ϕ	float 16	NXT
$\dot{\phi}$	float 16	NXT
θ_{cmd}	float 16	NXT
ϕ_{cmd}	float 16	NXT

Table 4.1. Bluetooth Data Link

4.4 GUI

The PC has a graphical user interface (GUI) that was designed for this application and can be seen in Figure 4.4. This custom interface provides a method to select and open a communications port. It also provides a command to balance, a command for θ, a command for ϕ, and a method to record data.

The first step in the operation of the GUI is to make a connection through the bluetooth communications port. This requires the Lego Mindstorm NXT to be paired with the computer by performing the following steps adapted from Chikamasa [3].

1. On the robot, in the main NXT screen scroll to the bluetooth symbol.
2. In the bluetooth menu scroll to the ON/OFF and ensure that it is on.
3. Return to the bluetooth menu.
4. Place the PC's bluetooth into Discovery mode.

5. In the NXT bluetooth menu scroll and select search, then select the PC that the NXT should be paired to.

6. The NXT will provide a pairing number and once that number is accepted it will attempt to connect to the PC.

7. The PC should indicate that the NXT is trying to connect and requires the pairing number. Provide the number and finalize the connection.

8. Then to determine what communications port will be used, go to the bluetooth settings on th PC and locate the port associated with "outgoing NXT 'Dev B"'.

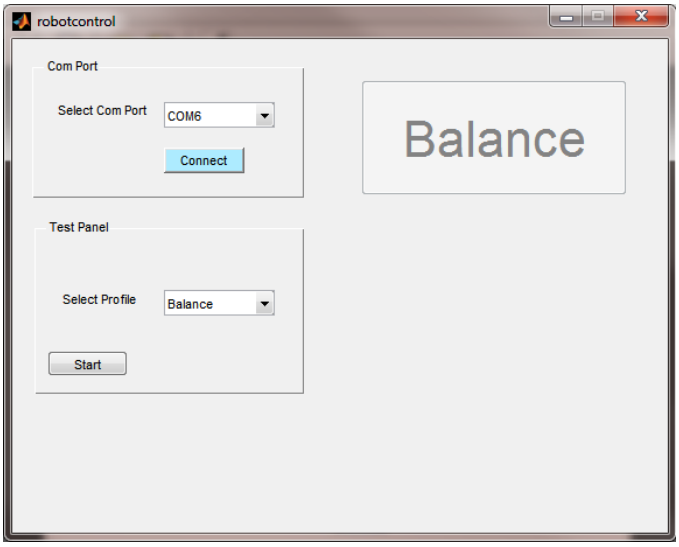

Figure 4.4. Control GUI

The communications port that was found using the pairing method used above is the port selection used in the GUI. This port will remain the link to the Lego Mindstorm NXT until the device is removed from the PC. Now that the bluetooth is configured, the GUI can be used to control the robot. The following steps are taken to run the robot and assume that the controller with the bluetooth interface block has been preloaded.

1. Turn on the Lego Mindstorm NXT and press the orange button until the nxtOSEK screen flashes.

2. Select "Run" by pressing the right arrow. The robot will beep once the calibration of the Gyro is complete. The robot should not be moved prior to the beep since this will result in an abnormal gyro calibration.

3. Run the GUI in Matlab.

4. In the drop down menu, select the communications port for the bluetooth interface.

5. Click on "Connect". Once a connection is made the "Balance" button will activate.

6. Place the robot in an upright position on a surface and press the "Balance" button. Once the robot starts to balance, let go and let the controller do the work.

7. In the test panel, select from the four tests that can be performed.

8. Clicking "Start" will send the profile and collect data. Once the profile is finished, the data is available in Trial.mat.

4.5 Controller Design Considerations

The main processor is a 32 bit fixed point processor. The Real-Time Workshop allows floating point calculations to to be performed on the 32 bit fixed point processor. These operations will consume more processing time. The controller will be implemented in floating point; therefore, the extra computation is a factor. The data that will be sent back and forth over bluetooth communications also requires processing time. These factors determined that a 10ms sampling interval

should be used. The high sampling interval is also much higher than the interval at which the I2C communications are handled with the AVR. Therefore, the I2C delays will not be a factor in the design and analysis of the controller.

The hardware will also affect the design of the controller since the resolution of the sensors is low. This requires the controller to have a slower setting time so that the controller does not respond to the bit changes. This affects the system most when it is trying to balance in place and a bit change occurs causing a large derivative. A slower settling time means the gains are not tuned so high that the large derivative cause the voltage inputs to saturate. There are other effects such as the sloppy gears that produce dead bands and the saturation point of the output at the battery voltage that requires slower settling times.

List of References

[1] T. Chikamasa, "Embedded coder robot nxt modeling tips," April 2010, unpublished. [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/13399-embedded-coder-robot-nxt-demo?controller=file_infos&download=true

[2] J. P. Jones, C. McArthur, and T. Young, *VU-LEGO Real Time Target: User's Guide*, 1st ed., Center for Nonlinear Dynamics and Control, Villanova University, Villanova PA 19085, United States of America, February 2011.

[3] T. Chikamasa, "Nxt gamepad," January 2009, unpublished. [Online]. Available: http://lejos-osek.sourceforge.net/nxtgamepad.htm

CHAPTER 5

Controller Design

The controller for the robot uses the control theory in Chapter 3. The first step is to obtain the continuous plant model based on Equation (2.15) and Table 2.2. The continuous state-space model is shown in Equation (5.1).

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & -459.8438 & -320.5946 & 320.5946 & 0 & 0 \\ 0 & 267.8199 & 139.6514 & -139.6514 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & -185.2118 \end{bmatrix} \quad (5.1)$$

$$\mathbf{B} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 170.2666 & 170.2666 \\ -74.1683 & -74.1683 \\ 0 & 0 \\ -45.0603 & 45.0603 \end{bmatrix}$$

The poles of the plant are 0,0,-460.7253,7.0979,-6.6185,-182.04. The continuous-time model is controllable and all poles are on the real axis, therefore the ZOH model is controllable for any sampling interval as stated in section 3.1. The next step is to calculate the ZOH model using Equation (3.2) by calling the function zohe. The sample interval of 10 ms was chosen based on the hardware limitations.

The calculated $\mathbf{\Phi}$ and $\mathbf{\Gamma}$ are shown in Equation (5.2).

$$\begin{aligned}
\mathbf{\Phi} &= \begin{bmatrix} 1 & -0.0063 & 0.0045 & 0.0054 & 0 & 0 \\ 0 & 1.0061 & 0.0024 & 0.0076 & 0 & 0 \\ 0 & -0.6237 & 0.3089 & 0.6848 & 0 & 0 \\ 0 & 0.9484 & 0.3017 & 0.7044 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0.0045 \\ 0 & 0 & 0 & 0 & 0 & 0.1563 \end{bmatrix} \\
\mathbf{\Gamma} &= \begin{bmatrix} 0.0029 & 0.0029 \\ -0.0013 & -0.0013 \\ 0.3670 & 0.3670 \\ -0.1602 & -0.1602 \\ -0.0013 & 0.0013 \\ -0.2048 & 0.2048 \end{bmatrix}
\end{aligned} \tag{5.2}$$

The plant is then examined and determined to be controllable by obtaining $\mathbf{W}_c$ from Equation (3.3) and verifying that the rank is equal to 6.

5.1 Additional Dynamics

The next step is to determine the additional dynamics. The ability to track steady-state or ramps in θ and ϕ were both examined. The additional dynamics for a steady-state tracking are shown in Equation (5.3) for steady-state tracking and Equation (5.4) for ramp tracking.

$$\mathbf{\Phi}_a = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \mathbf{\Gamma}_a = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{5.3}$$

$$\mathbf{\Phi}_a = \begin{bmatrix} 1 & 0.01 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.01 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \mathbf{\Gamma}_a = \begin{bmatrix} 5e^{-5} & 0 \\ 0.01 & 0 \\ 0 & 5e^{-5} \\ 0 & 0.01 \end{bmatrix} \tag{5.4}$$

The closed-loop poles need to be determined before the feedback matrix can be calculated. The two tracking designs, steady-state and ramp, will have their own set of closed-loop pole locations. This is due to the fact that step tracking requires an 8th order closed-loop system and ramp tracking requires a 10th order closed-loop system. In both ramp and step tracking, three of the closed-loop poles from the plant are used. Table 5.1 shows the location of the continuous time

closed-loop poles that were chosen based on the information in Section 3.4 and simulation. In addition to the three plant poles, the step controller has two Bessel poles with a settling time of 1 [sec] and three Bessel poles with a settling time of 3 [sec]. The additional poles for the ramp are placed with two Bessel poles at a settling time of 5 [sec] and five Bessel poles with a settling time of 3 [sec].

Steady-State	Ramp
-460.725	-460.725
-6.619	-6.619
-182.04	-182.04
-4.053 + 2.340j	-0.8106 - 0.468j
-4.053 - 2.340j	-0.8106 + 0.468j
-1.322 + 1.261j	-1.3701 - 2.1047j
-1.322 - 1.261j	-1.3701 + 2.1047j
-1.670	-2.1493
	-1.9756 - 1.0271j
	-1.9756 + 1.0271j

Table 5.1. Closed-Loop Poles

5.2 Feedback Matrix

The design of the controller can utilize one of three algorithms for calculating the feedback matrix. These gains were calculated for both the steady-state and ramp tracking. Therefore, six sets of gains were determined in all. The stability margins are also calculated for the six controller designs. The performance of each of these designs is later examined in the Matlab/Simulink Model.

	L1						L2	
fbg	-1.63	-38.80	-1.86	-4.34	-16.61	-0.08	-0.01	-0.22
	-1.63	-38.80	-1.86	-4.34	16.61	0.08	-0.01	0.22
place	5.19	-36.40	-1.38	-3.28	-9.41	-0.04	0.06	-0.12
	-11.38	-68.14	-3.33	-7.71	10.05	0.04	-0.11	0.12
TFBG	-5.05	-62.95	-2.89	-6.74	-5.44	-0.02	-0.04	-0.07
	-5.05	-62.95	-2.89	-6.74	5.44	0.02	-0.04	0.07

Table 5.2. Steady-State Tracking Feedback Matrices

	L1						L2			
fbg	2.99	-9.12	-0.31	-0.56	-10.71	-0.02	-1.21	3.39	-15.25	-20.92
	-7.20	-77.35	-3.53	-8.40	10.68	0.09	-0.70	-7.97	15.34	20.46
place	-2.52	-38.54	-1.90	-3.85	-11.35	-0.35	1.60	-2.58	-11.18	-19.62
	-1.75	-39.80	-1.82	-4.02	11.65	0.42	-4.17	-2.78	10.87	20.00
TFBG	-1.77	-43.14	-1.90	-4.45	-10.14	-0.04	-0.54	-1.52	-27.06	-24.77
	-1.77	-43.14	-1.90	-4.45	10.14	0.04	-0.54	-1.52	27.06	24.77

Table 5.3. Ramp Tracking Feedback Matrices

5.3 Stability Margins

The stability margins for each of the closed-loop systems were calculated using the equations in Section 3.5. The system N(z) to calculate δ_{max} is the state-space model in Equation (5.5).

$$\begin{aligned} \mathbf{\Phi}_N &= \mathbf{\Phi_d} - \mathbf{\Gamma_d L} \\ \mathbf{\Gamma}_N &= \mathbf{\Gamma_d} \\ \mathbf{C}_N &= \mathbf{L} \\ \mathbf{D}_N &= \mathbf{0} \end{aligned} \tag{5.5}$$

		δ_{max}	UGM [db]	LGM [db]	PM [Deg]
fbg	Steady-State	0.34	2.52	-3.57	19.40
	Ramp	0.25	1.90	-2.44	14.08
place	Steady-State	0.34	2.53	-3.59	19.48
	Ramp	0.27	2.10	-2.77	15.68
TFBG	Steady-State	0.49	3.44	-5.79	28.16
	Ramp	0.35	2.59	-3.71	20.01

Table 5.4. Stability Margins

The stability margins shown in the Table 5.4 do not indicate a controller that has the desired stability margins of 3dB UGM, -3dB LGM or 30 degree PM. The values in Table 5.4 do indicate that using **TFBG** will provide the most robust controller. These controllers are examined further for performance in Chapter 6.

5.4 Plant Inputs and Outputs

In order to implement the controller, the feedback and commands of the robot need to be transformed into the states that are shown in Equation (2.14). The feedback signals from the robot are θ_{m_l}, θ_{m_r} and $\dot{\Psi}$ which are converted from degrees to radians and then used to find the required states. θ and ϕ are determined by using Equations (2.1) and (2.2). Ψ is calculated by using a forward Euler integration of $\dot{\Psi}$.

$$\Psi[k+1] = \Psi[k] + T\dot{\Psi}[k] \tag{5.6}$$

The state variables $\dot{\phi}$ and $\dot{\theta}$ require the derivative of θ and ϕ. The derivative was implemented as a second order filter with a specific settling time in order to reject any frequencies that are higher than the system response. The filter is designed using Bessel roots with a settling time of 0.025 [sec] which results in the transfer function in Equation(5.7).

$$H(s) = \frac{35044s}{s^2 + 324.24s + 35044} \tag{5.7}$$

Converting the transfer function to a continuous state-space model and then using the zohe command with a sampling time of 0.01 [sec] generates the discrete state-space Equation (5.8) for the filtered derivative.

$$\begin{aligned} \mathbf{x}[k+1] &= \mathbf{\Phi}_f \mathbf{x}[k] + \mathbf{\Gamma}_f u[k] \\ y[k] &= \mathbf{C}_f \mathbf{x}[k] \\ \mathbf{\Phi}_f &= \begin{bmatrix} 0.3928 & 0.0017 \\ -59.588 & -0.1585 \end{bmatrix} \\ \mathbf{\Gamma}_f &= \begin{bmatrix} 0.607 \\ 59.59 \end{bmatrix} \\ \mathbf{C}_f &= \begin{bmatrix} 0 & 1 \end{bmatrix} \end{aligned} \tag{5.8}$$

Estimates of the state variables $\dot{\phi}$ and $\dot{\theta}$ could have been implemented using a linear observer system. However, modeling has shown that non-linearities due to quantization make it difficult to use the linear observer accurately.

The outputs of the plant require scaling. Since the DC motor voltage is PWM based, the voltage from the controller is scaled and limited by the battery voltage ($\pm V_{bat}$). The commands sent to the motors are $\text{PWM}_{l,r} = 100\left(\frac{V_{l,r}}{V_{bat}}\right)$. The presence of the limiters require that the additional dynamics, which are integrators, be held in order to keep them from rapidly growing while in the limits. The integrators are not held exactly at $\pm V_{bat}$ but at a voltage slightly lower. Limiting the integrators at a lower voltage than the battery allows the controller to respond as a regulator for the remaining voltage head room. The regulator is the main component used to maintain the robot's balance, giving a priority to the system, which is first balancing and second maneuvering.

CHAPTER 6

Modeling and Simulations

The controller design is first implemented in a computer model utilizing Matlab and Simulink. The model used can be seen in Figure 6.1.

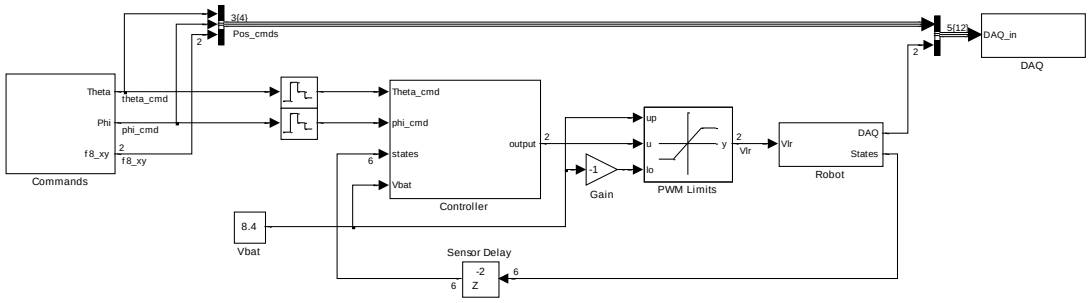

Figure 6.1. Simulink Model

The main components of the model are the commands, controller and robot block. The commands block is used to simulate the profiles for θ and ϕ that are sent over bluetooth. It contains profiles which cause the robot to balance in place, move back and forth, spin in place and follow a figure eight. The commands are in continuous time, and so the zero order hold block is used to convert the output to discrete-time commands. The robot block contains a continuous-time state-space model of the robot using Equation (2.15) with full state feedback. This differs from the hardware which would return θ_{m_l}, θ_{m_r} and $\dot{\Psi}$ and would then calculate all the states. The states are fed back to the controller through a sensor delay block. This block performs both a zero order hold and also delays the signal by 2 ms. This delay is to account for the delay of 1ms to send the voltage commands and 1ms to receive the measurements as stated in Chapter 4. The inputs to the robot are the voltages to each of the motors. These voltages are outputs of the controller and limited by the battery voltage prior to being applied to the robot. The robot

voltage inputs and states along with the commands are sent to the DAQ block so that data can be collected and analyzed.

The final block is the controller. This block contains most of the controller that was designed in Chapter 5 and is shown in Figure 6.2.

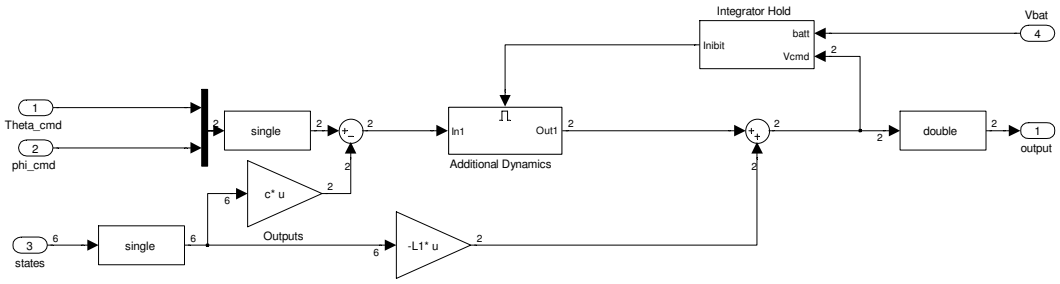

Figure 6.2. Controller

The additional dynamics block is implemented using a discrete state-space equation with the values calculated in Chapter 5 for $\mathbf{\Phi_a}$, $\mathbf{\Gamma_a}$ and $\mathbf{L_2}$. The input to the additional dynamics block is the error of the commanded and measured θ and ϕ. The additional dynamics block is only enabled when the motor voltage commands are between $\pm$ 95% of V_{bat}. The output of the controller is the subtraction of the voltages of state feedback gain $\mathbf{L_1}$ and the voltages of the additional dynamics.

The parameters needed for both the simulation and the control design are produced in files Parameters.m and Controller.m included in the Appendix. The model will automatically call the files when the model is opened and every time the model is started.

6.1 Performance Testing

The model was used to determine the performance of the controller design for each of the feedback matrix calculation algorithms to both steady-state and ramp tracking. In Chapter 5 the stability margins for each feedback matrix calculation algorithm with steady-state and ramp tracking were calculated. The stability

margins are a good indication of how the design will perform. However, even though the stability margins may show the system to be more robust, the dynamics may still not produce the best performance.

Steady-State Tracking

Testing was performed using the steady-state tracking designs utilizing **place**, **fbg**, and **TFBG**. The following figures were generated from data collected from the model. The inputs to the model, in the first test are a ramping profile in ϕ and a constant zero reference to θ. Figure 6.3 shows the voltage commanded to both wheels as expected. The voltages are equal in magnitude and have opposite direction which results in the robot turning. The three methods of calculating the feedback matrix all reach zero steady state error and have close to the same amount of error to a ramp as seen in Figure 6.5. Figure 6.4 shows that the **place** algorithm allows θ to be affected by a command in ϕ, while the other algorithms hold to the reference. In all cases balancing is achieved; however, Figure 6.6 shows that **place** introduces disturbances into Ψ. This is an example of where **place** had higher stability margins than **fbg** as seen in Table 5.4, and does not have better performance.

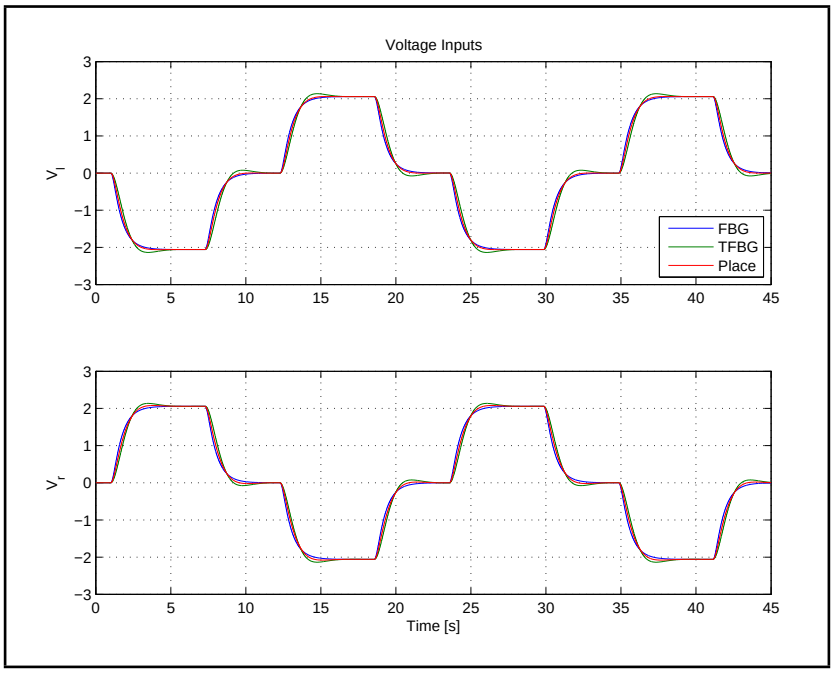

Figure 6.3. Steady-State ϕ Tracking Voltage Commands

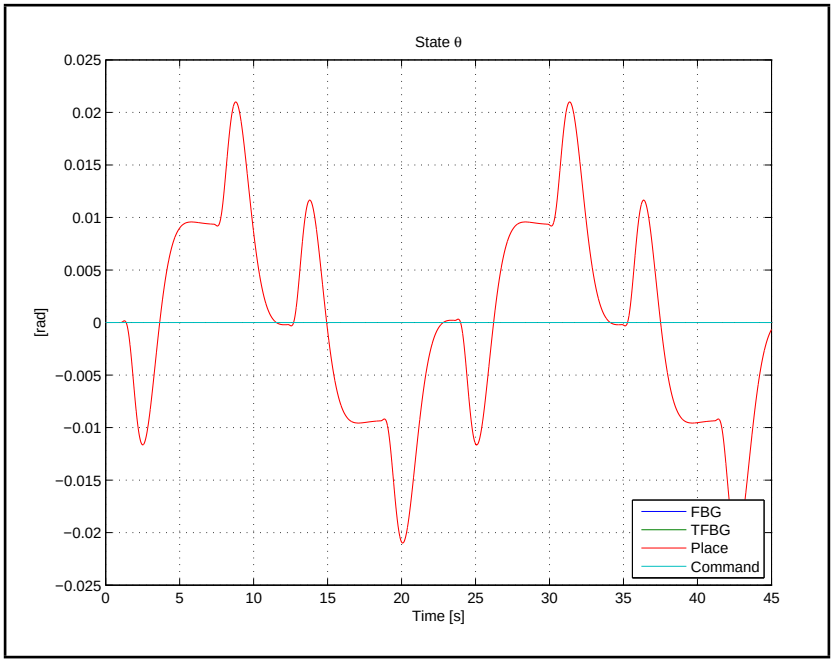

Figure 6.4. Steady-State ϕ Tracking Measured θ State

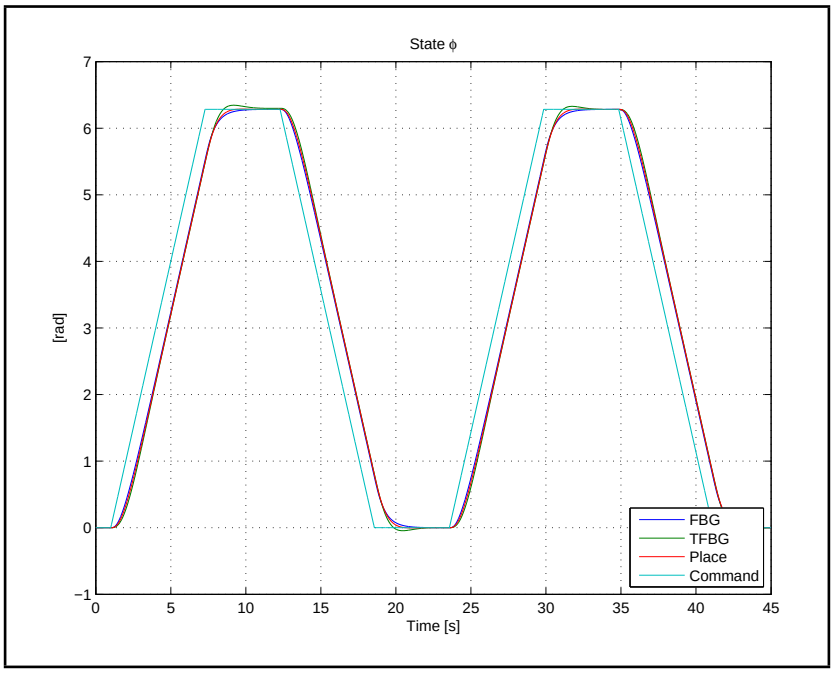

Figure 6.5. Steady-State ϕ Tracking

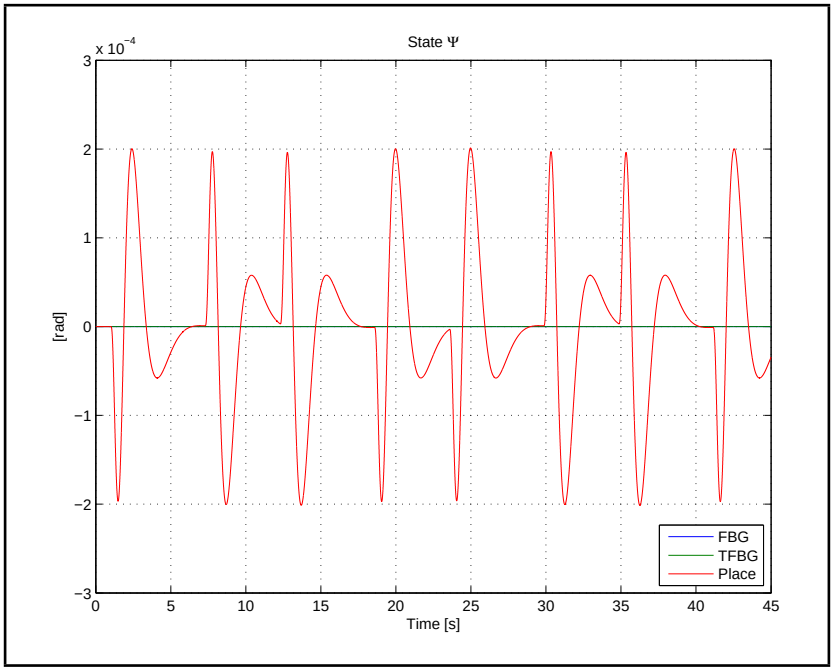

Figure 6.6. Steady-State ϕ Tracking Measured Ψ State

The model was also executed with a ramping profile in θ and a constant zero reference to ϕ. The voltages in Figure 6.7 show that both wheels are provided

commands in the same direction which causes the robot to move forward and back. However, it is evident that the left wheel has much more damping than the right wheel with the **place** algorithm. Figure 6.8 show that all three of the algorithms produce tracking system with zero steady-state error and all have approximately the same error to a ramp. Figure 6.9 shows that the **place** algorithm allows ϕ to be affected by a command in θ, while the other algorithms hold to the reference. In all cases, balancing is achieved and a disturbance is introduced in Ψ as shown in Figure 6.10.

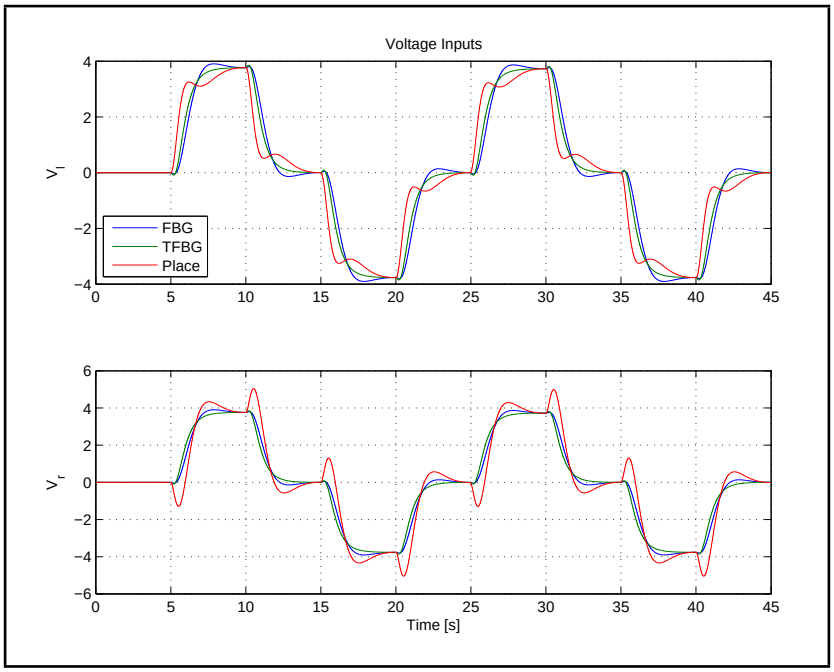

Figure 6.7. Steady-State θ Tracking Voltage Commands

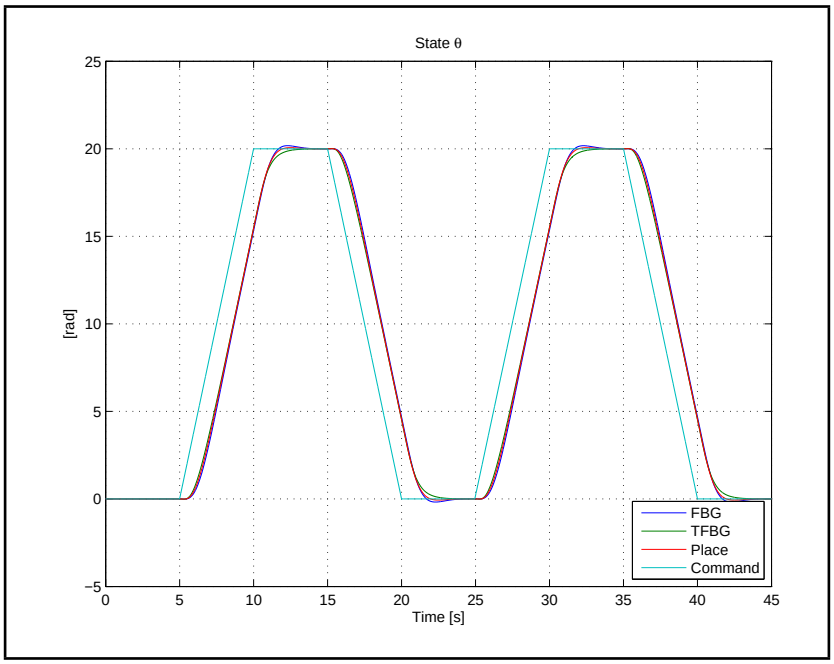

Figure 6.8. Steady-State θ Tracking

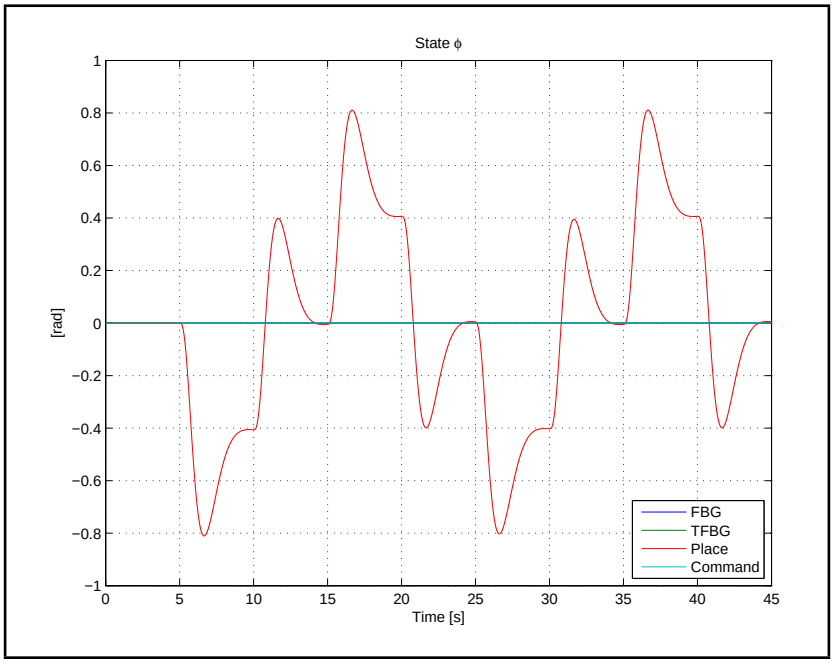

Figure 6.9. Steady-State θ Tracking Measured ϕ State

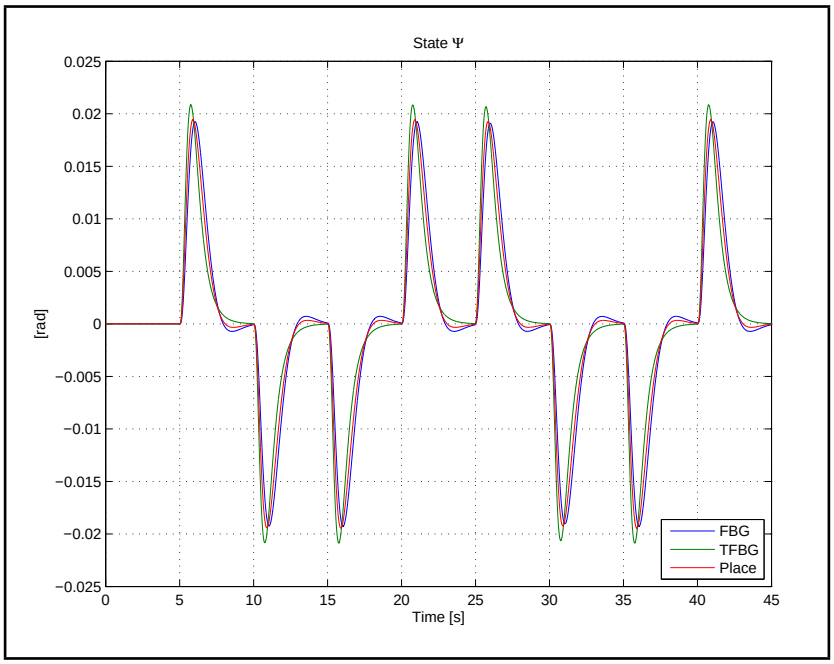

Figure 6.10. Steady-State θ Tracking Measured Ψ State

The steady-state tracking modeling shows that **place** had the worst performance. **TFBG** seems to show marginally better performance than **fbg**. Using **place** would cause the robot to move very sloppily because the states are highly coupled to each other. This would make it very difficult to coordinate commands in θ and ϕ that would result in a desired trajectory. In all cases steady-state tracking was achieved. While it is not expected that any of these designs track a ramp perfectly since they were designed for steady-state tracking, it would be desirable that they track with the least amount of error as possible.

Ramp Tracking

Testing was performed using the ramp tracking designs utilizing **place**, **fbg**, and **TFBG**. The following figures were generated from data collected from the model. The inputs to the model, in the first test were a ramping profile in ϕ and a constant zero reference to θ. Figure 6.11 shows the voltage commanded to both wheels, and as expected, the voltages are equal in magnitude and with opposite

direction, which results in the robot turning. The voltage commands show that the **TFBG** algorithm has a faster response, which can also be seen in Figure 6.13 where **TFBG** has less overshoot. Figure 6.12 shows that the **place** and **fbg** algorithms allow θ to be affected by a command in ϕ, while the **TFBG** algorithm hold to the reference. In all cases balancing is achieved. However, Figure 6.14 shows that **place** and **fbg** introduces disturbances into Ψ. In this case **TFBG** from Table 5.4 has the best performance and stability margins.

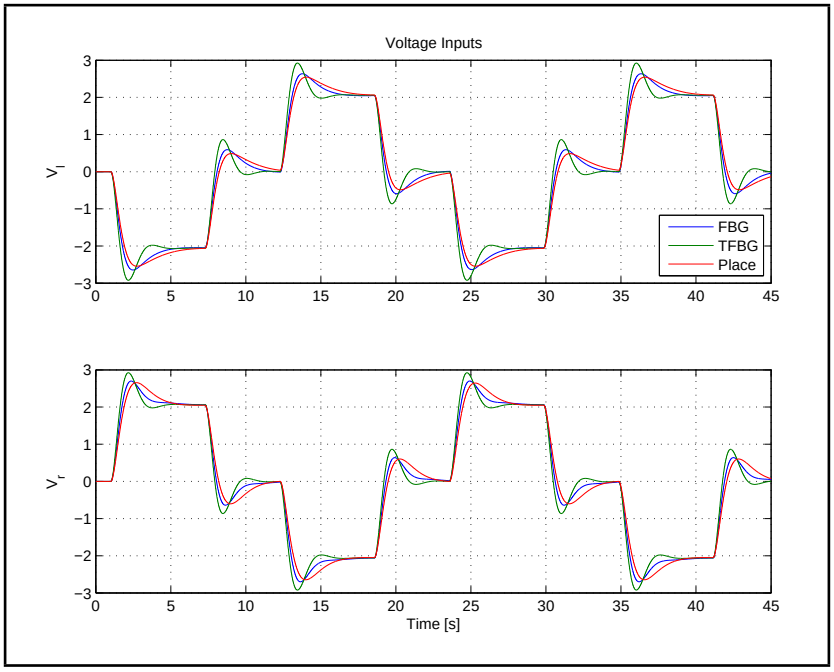

Figure 6.11. Ramping ϕ Tracking Voltage Commands

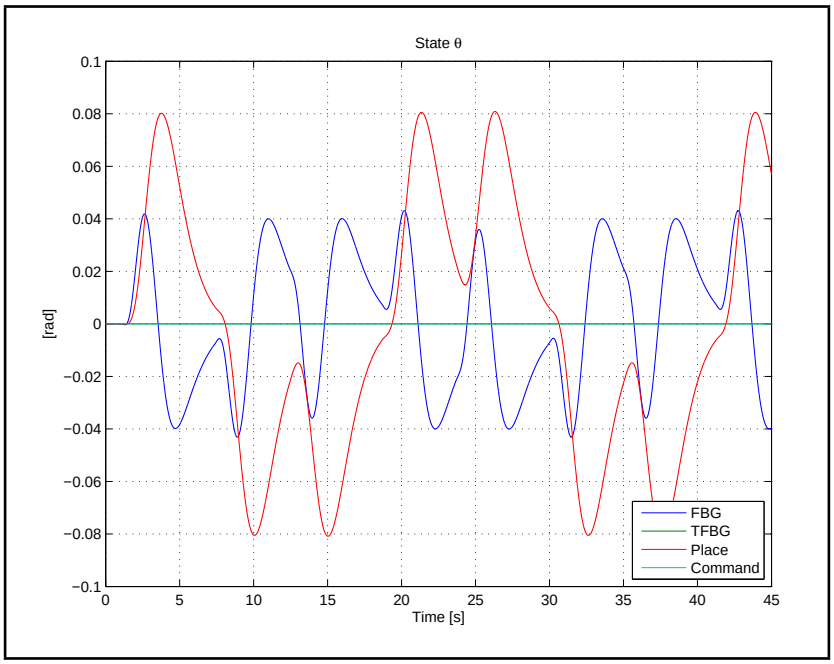

Figure 6.12. Ramping ϕ Tracking Measured θ State

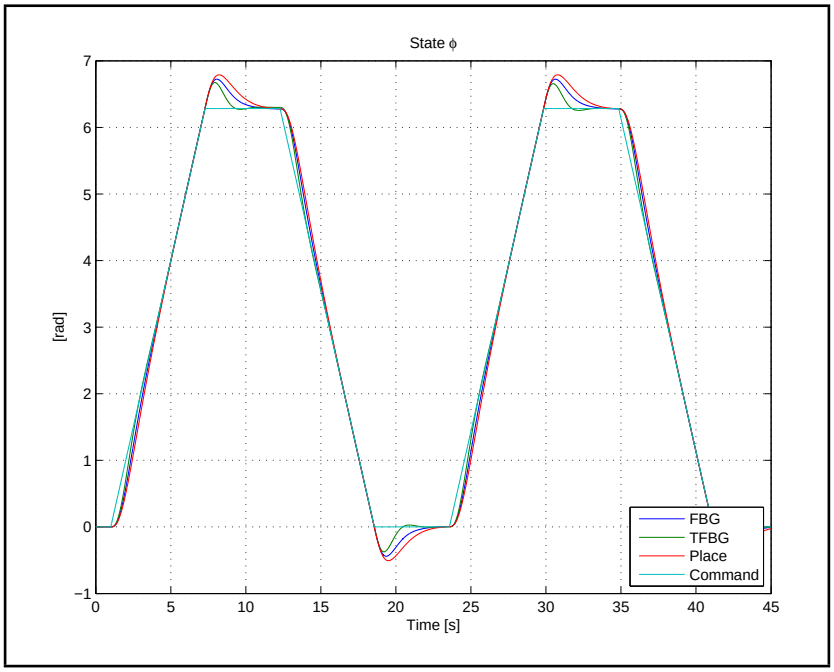

Figure 6.13. Ramping ϕ Tracking

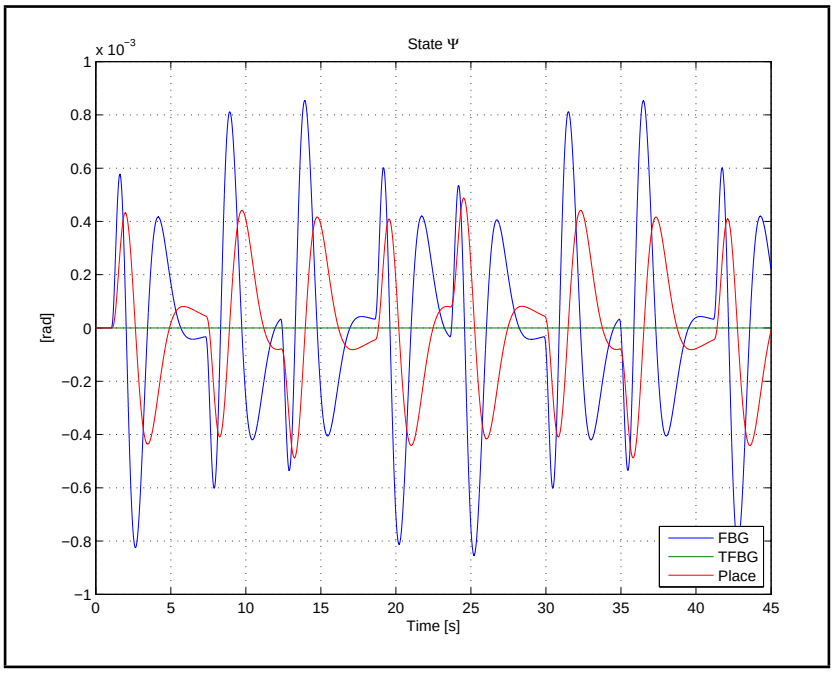

Figure 6.14. Ramping ϕ Tracking Measured Ψ State

The model was also executed with a ramping profile in θ and a constant zero reference to ϕ. It is expected that the voltages should have commands with equal magnitude and direction, as in Figure 6.15, for the robot to move forward and back. However, it is evident that the right wheel has much more damping than the left wheel with the **place** algorithm. It can also be seen that **TFBG** has a slower response to commands in θ, and therefore has the most overshoot to a commanded θ as seen in Figure 6.16. Figure 6.17 shows that the **place** and **fbg** algorithms allow ϕ to be affected by a command in θ, while **TFBG** algorithm holds to the reference. The state Ψ shown in Figure 6.18 indicates that during maneuvering the robot remains balanced.

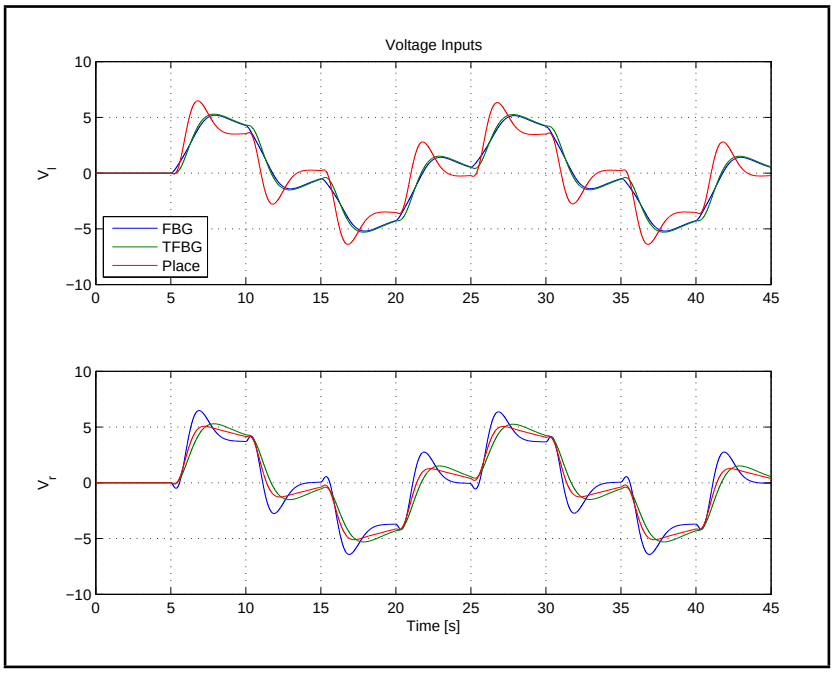

Figure 6.15. Ramping θ Tracking Voltage Commands

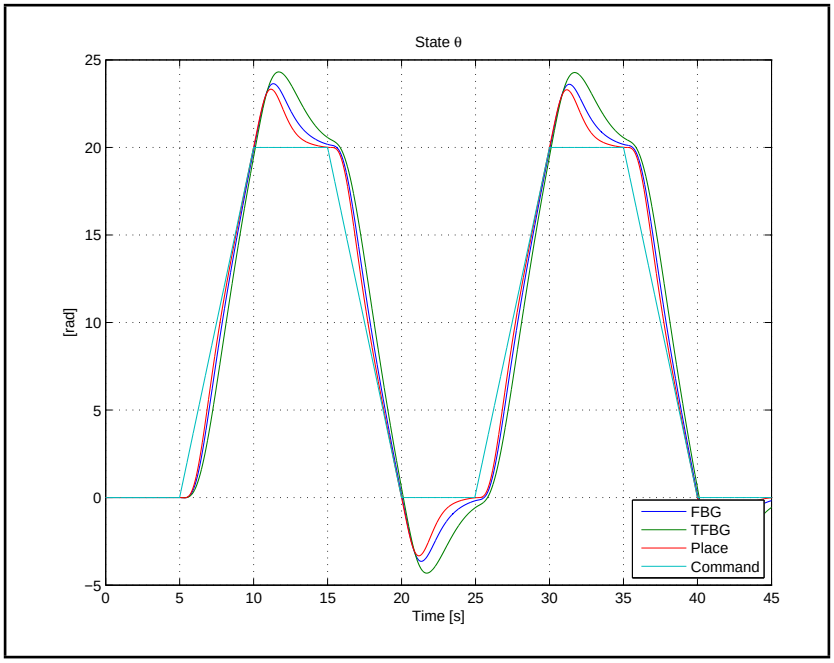

Figure 6.16. Ramping θ Tracking

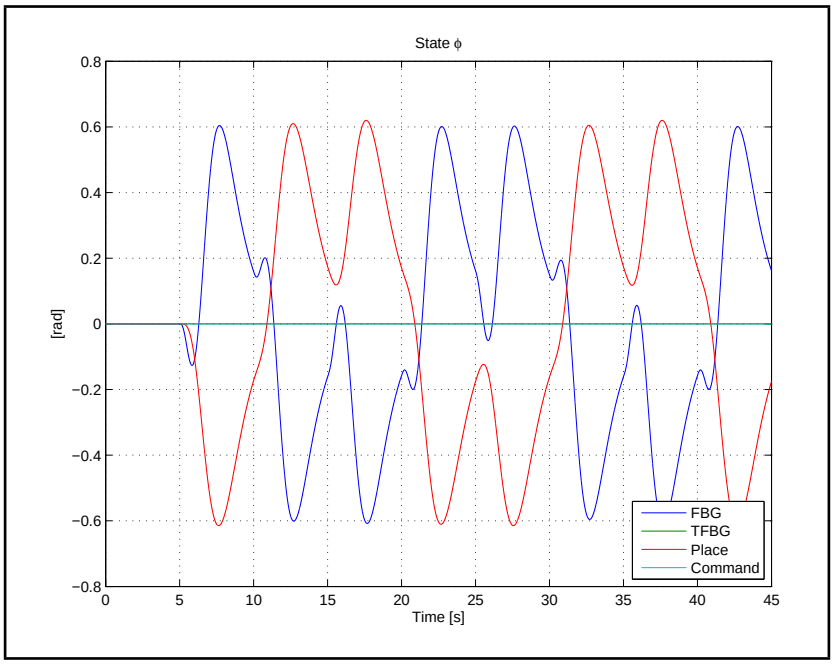

Figure 6.17. Ramping θ Tracking Measured ϕ State

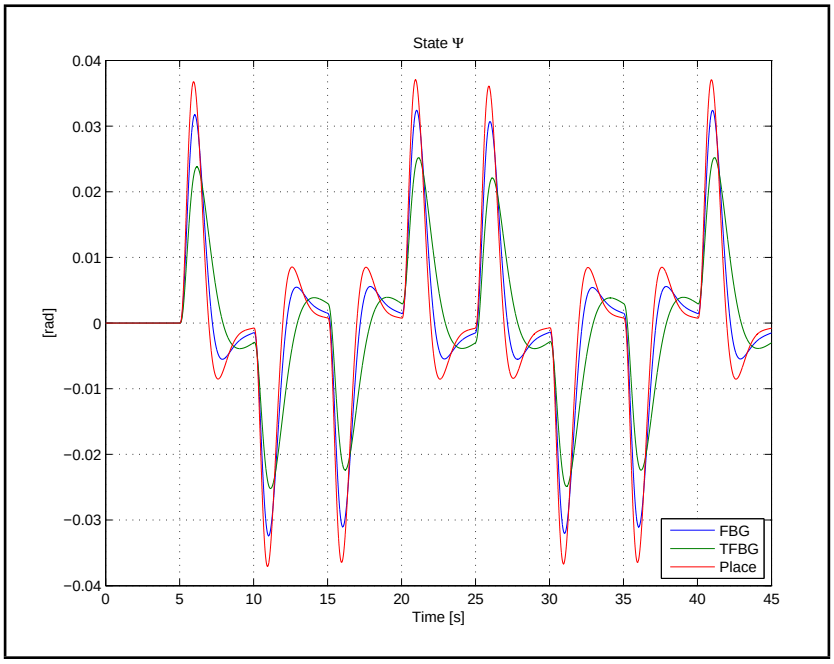

Figure 6.18. Ramping θ Tracking Measured Ψ State

6.2 Figure Eight Tracking

Tracking a figure eight will test the controller's ability to track both ϕ and θ at the same time. During this testing only the design that uses **TFBG** will be used. The performance testing in the previous section shows that using **place** would cause the system response to be very sloppy. It also shows that **fbg** is sloppy when performing ramp tracking even with the ramp tracking design. **TFBG** has better performance due to the fact that it not only optimizes phase and gain margin, it also ensures that the feedback matrix has a certain symmetry which can be seen in Tables 5.2 and 5.3. The **fbg** algorithm had shown this symmetry in the steady-state tracking design, which had good performance. In the ramp tracking design, **fbg** did not have this symmetry. This explains why **fbg** performed better in steady-state tracking than in ramp tracking. Therefore, since neither **fbg** or **place** guarantee the symmetry needed to get the best performance, **TFBG** will be used in the controller design for figure eight tracking. The figure eight will help to determine if steady-state or ramp tracking is better for hardware implementation.

Figure Eight Profile Generation

The figure eight profile generation is calculated in the commands block of the model. A command for θ and ϕ are calculated from the desired x_m and y_m. The desired x_m and y_m are calculated using the following equations

$$x_m(t) = A_x \sin \frac{2t}{T} \tag{6.1}$$

$$y_m(t) = A_y \sin \frac{t}{T} \tag{6.2}$$

where A_x is the amplitude of x_m, A_y is the amplitude of y_m and T is the period of the figure eight. This means that a figure eight can be commanded to remain in an area of $A_x \times A_y$ and complete the figure eight in $2\pi T$ seconds. In the following tests T is set to 10, A_x to 50 cm and A_y to 100 cm.

The locations x_m and y_m are transformed to θ and ϕ using Equation (2.4) which results in the following equations

$$\theta_{cmd} = \int_0^t R\sqrt{\left(\frac{2A_x}{T}\cos\left(\frac{2\tau}{T}\right)\right)^2 + \left(\frac{A_y}{T}\cos\left(\frac{\tau}{T}\right)\right)^2}\,d\tau \tag{6.3}$$

$$\phi_{cmd} = \tan^{-1}\frac{A_y\cos\left(\frac{t}{T}\right)}{2A_x\cos\left(\frac{2t}{T}\right)} \tag{6.4}$$

The $\tan^{-1}$ is calculated by using atan2 in order for all quadrants to be properly calculated. The atan2 function is bound by $\pm\frac{\pi}{2}$ so an "unwrapper" is used so that the discontinuities in the atan2 function are removed. This is needed because the robot expects absolute position, not relative position. The 2π wrap causes the robot to get a large step in rotational position that would try to spin the robot 1 revolution very rapidly, saturating the voltage commands.

The initial angle ϕ is a large step. The large command takes time to settle out, so the command generator applies the initial angle ϕ and then waits for the system to settle before starting the figure eight commands.

Steady-State Tracking

The following plots show how the system responds to the commands to a figure eight with the **TFBG** steady-state design. Figures 6.20 and 6.21 clearly show that there is a delay in both θ and ϕ. This delay results in a figure eight that is shifted as seen in Figure 6.23. The figure eight is clearly out of the limits of $\pm$ 50 cm on the x axis and slightly out of $\pm$ 100 cm in the y axis. Figure 6.23 shows the extent of the error in both x and y. The trend seems to show that as time passes the figure eight would continue to shift to the negative x and y axis. Figure 6.22 shows that the robot will remain balanced while maneuvering the figure eight.

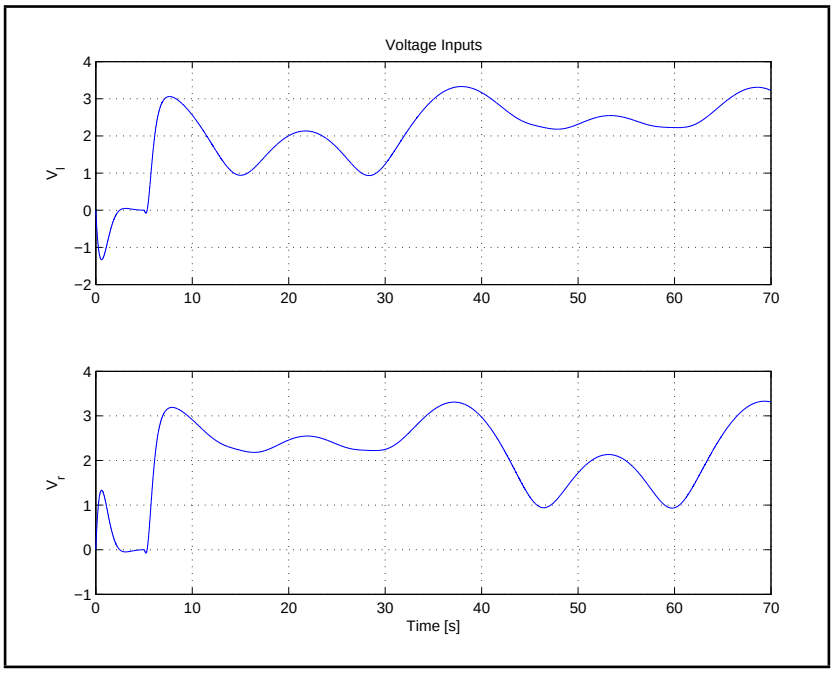

Figure 6.19. Steady-State Figure Eight Tracking Voltage Commands

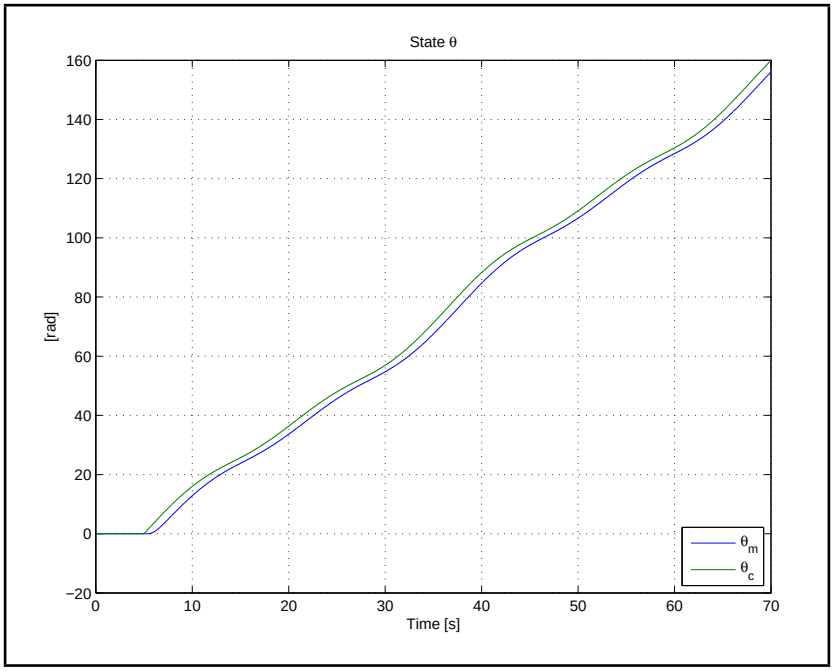

Figure 6.20. Steady-State Figure Eight Tracking θ

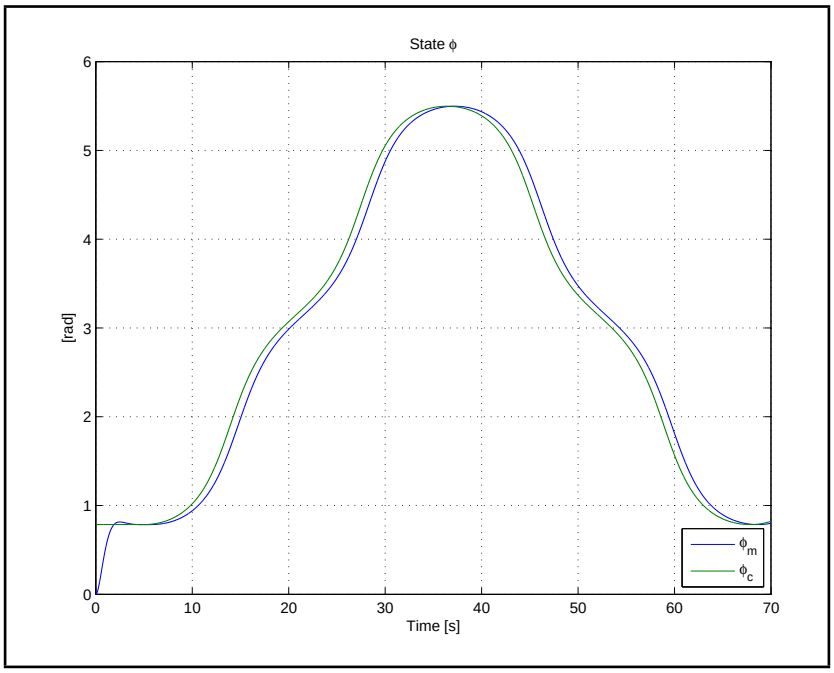

Figure 6.21. Steady-State Figure Eight Tracking ϕ

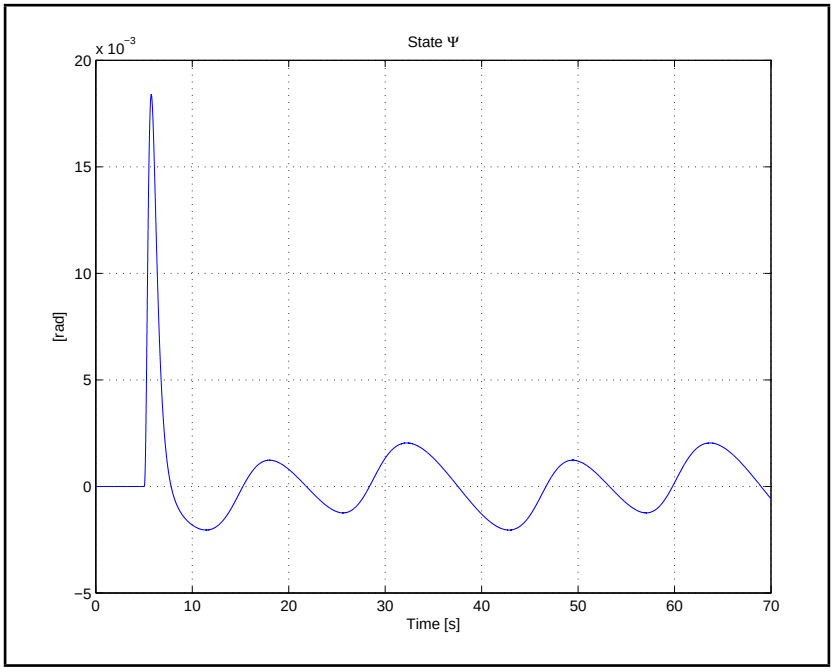

Figure 6.22. Steady-State Figure Eight Tracking Ψ

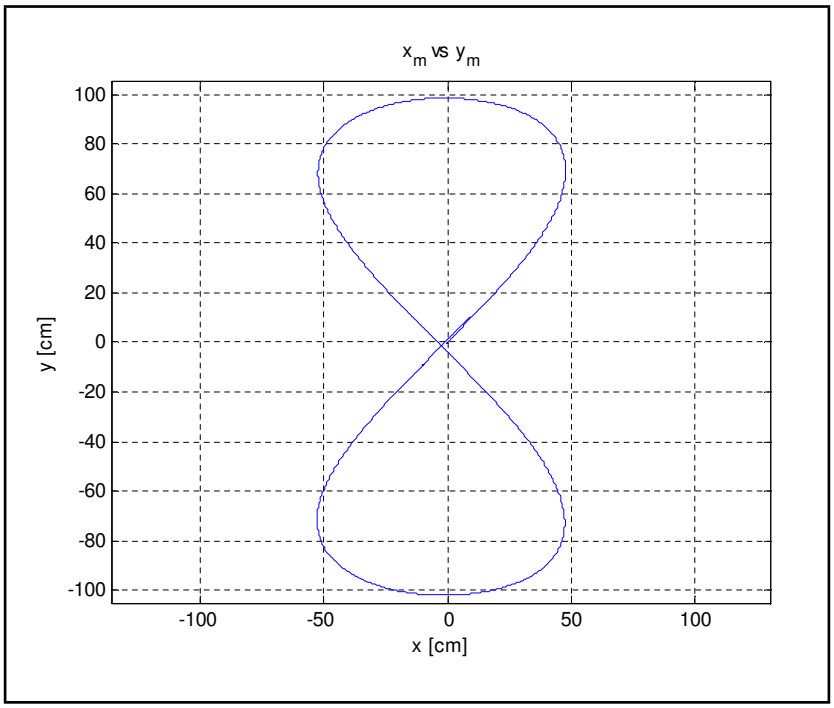

Figure 6.23. Steady-State Figure Eight Tracking x_m vs y_m

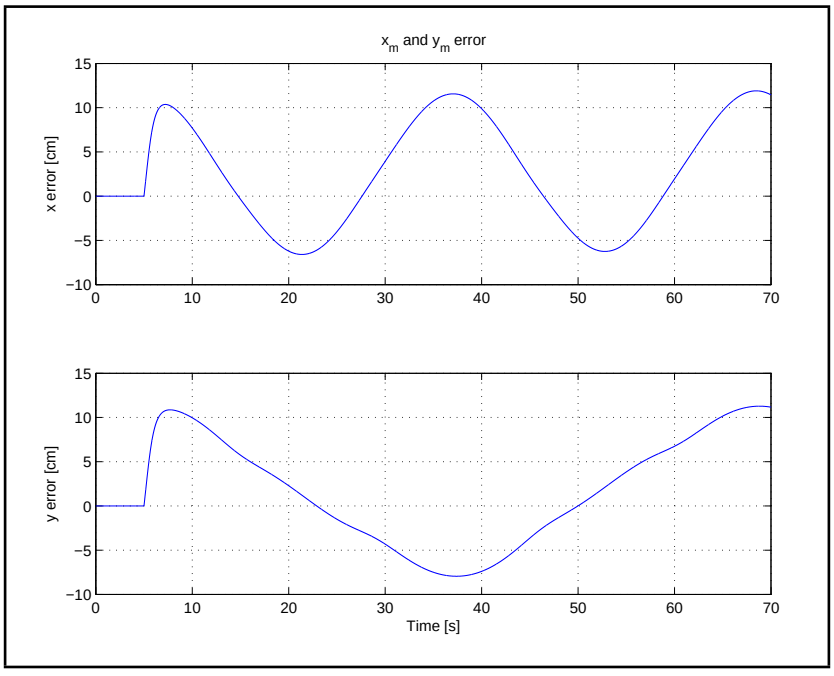

Figure 6.24. Steady-State Figure Eight Tracking x_m and y_m Error

Ramp Tracking

The following plots show how the system responds to the commands to a figure eight with the **TFBG** ramp tracking design. Figures 6.26 and 6.27 show that there is very little delay in both θ and ϕ. There is, however, a start-up delay in θ which does cause a misalignment in time between ϕ and θ. This results in a slightly malformed figure eight as seen in Figure 6.29. The figure eight is out of the limits of $\pm$ 50 cm on the x axis and slightly short of the $\pm$ 100 cm in the y axis. The figure eight does have less error than the steady-state design which can be seen in Figure 6.30. This design also results in very stable balancing which can be seen in Figure 6.28.

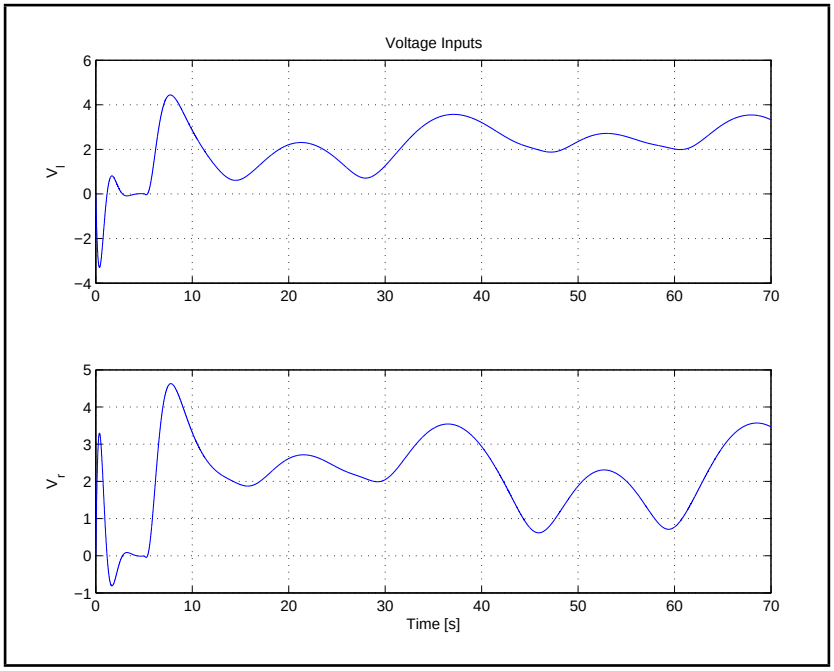

Figure 6.25. Ramping Figure Eight Tracking Voltage Commands

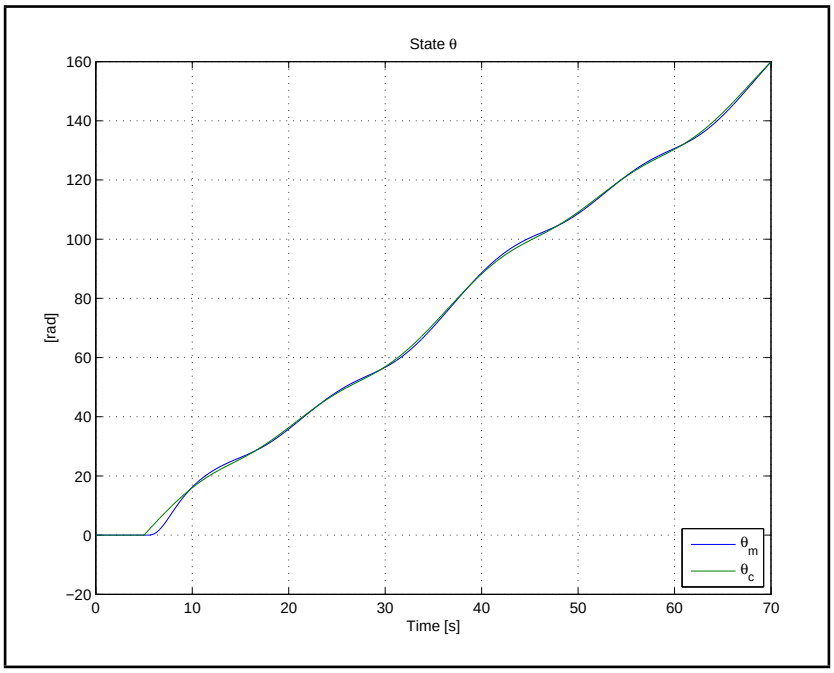

Figure 6.26. Ramping Figure Eight Tracking θ

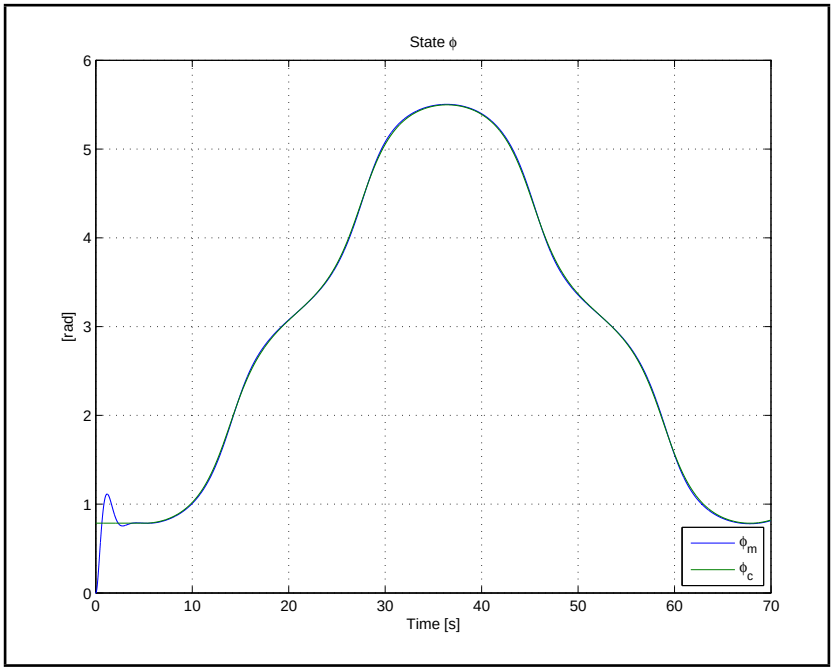

Figure 6.27. Ramping Figure Eight Tracking ϕ

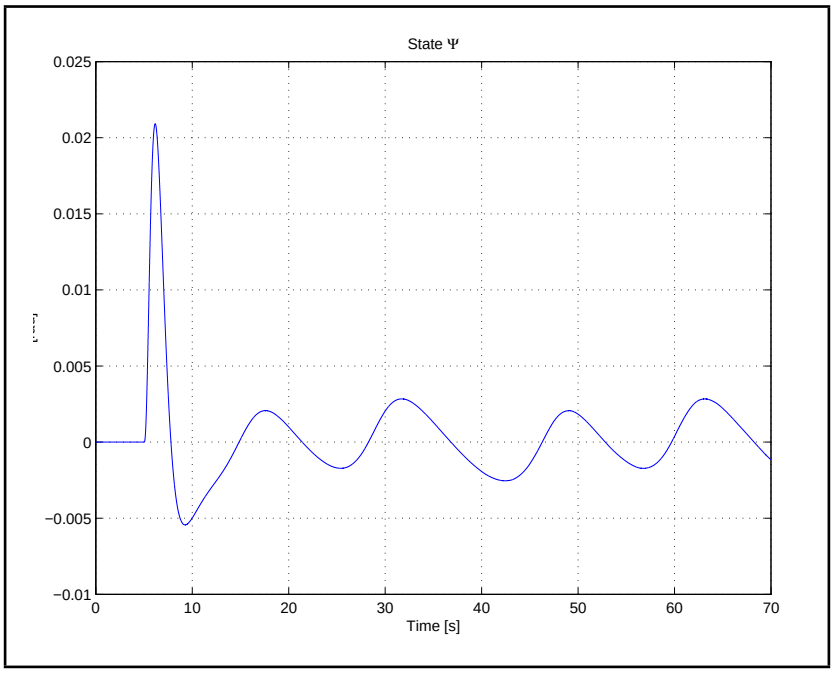

Figure 6.28. Ramping Figure Eight Tracking Ψ

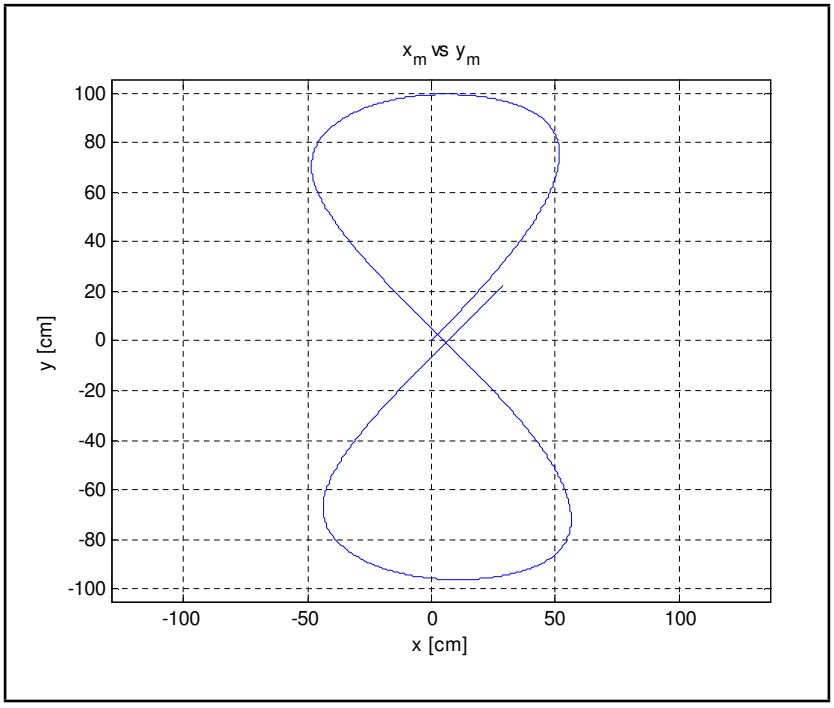

Figure 6.29. Ramping Figure Eight Tracking x_m vs y_m

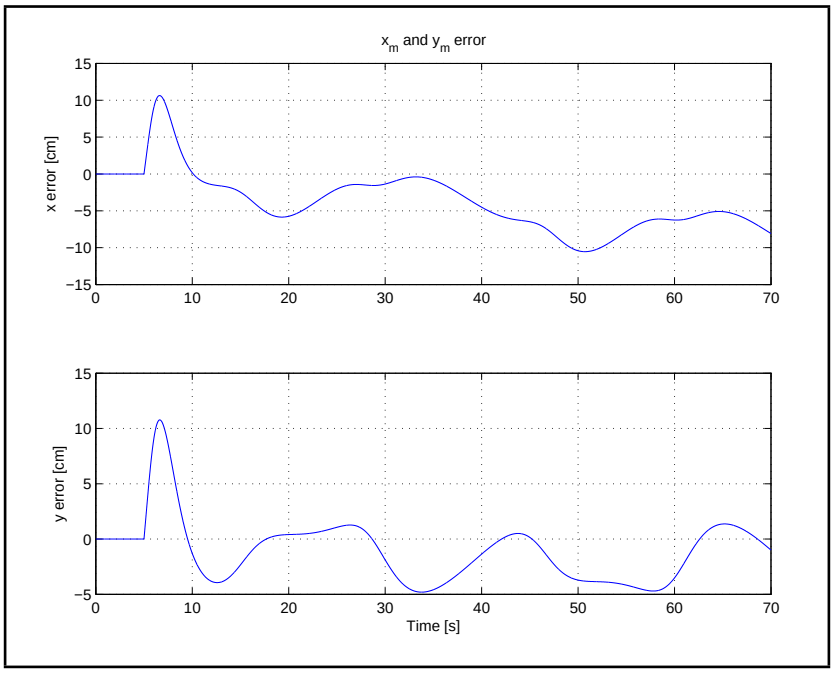

Figure 6.30. Ramping Figure Eight Tracking x_m and y_m Error

The figure eight was also tested with a longer period of T = 20 while keeping same boundary conditions as the previous test. This test shows that the x and y position error, see Figure 6.31, is reduced when the figure eight period is increased. This result can also be seen by the nearly perfect figure eight shown in Figure 6.32.

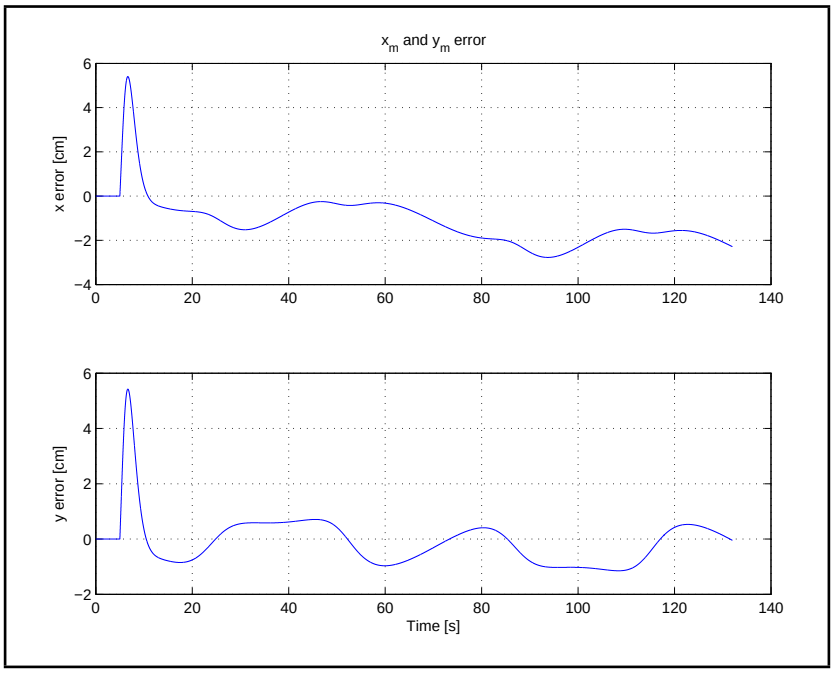

Figure 6.31. Slow Figure Eight Tracking x_m and y_m Error

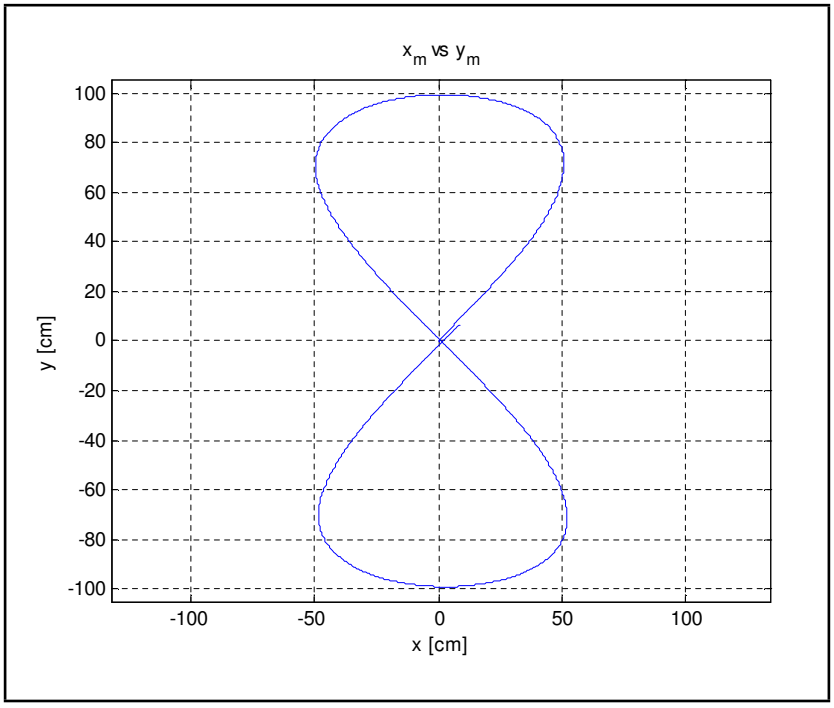

Figure 6.32. Slow Figure Eight Tracking x_m vs y_m

CHAPTER 7

Hardware Testing

The **TFBG** steady-state tracking design had better performance in simulation and best stability margins. During the hardware experiments the high gains associated with the **TFBG** steady-state tracking design made it difficult to implement in hardware. Increasing the settling time for the **TFBG** steady-state tracking design may have resulted in a working hardware system; however, this would have decreased the performance. Therefore, the **TFBG** ramp tracking design was implemented in hardware. The controller is implemented in Simulink and the top level design is shown in Figure 7.1.

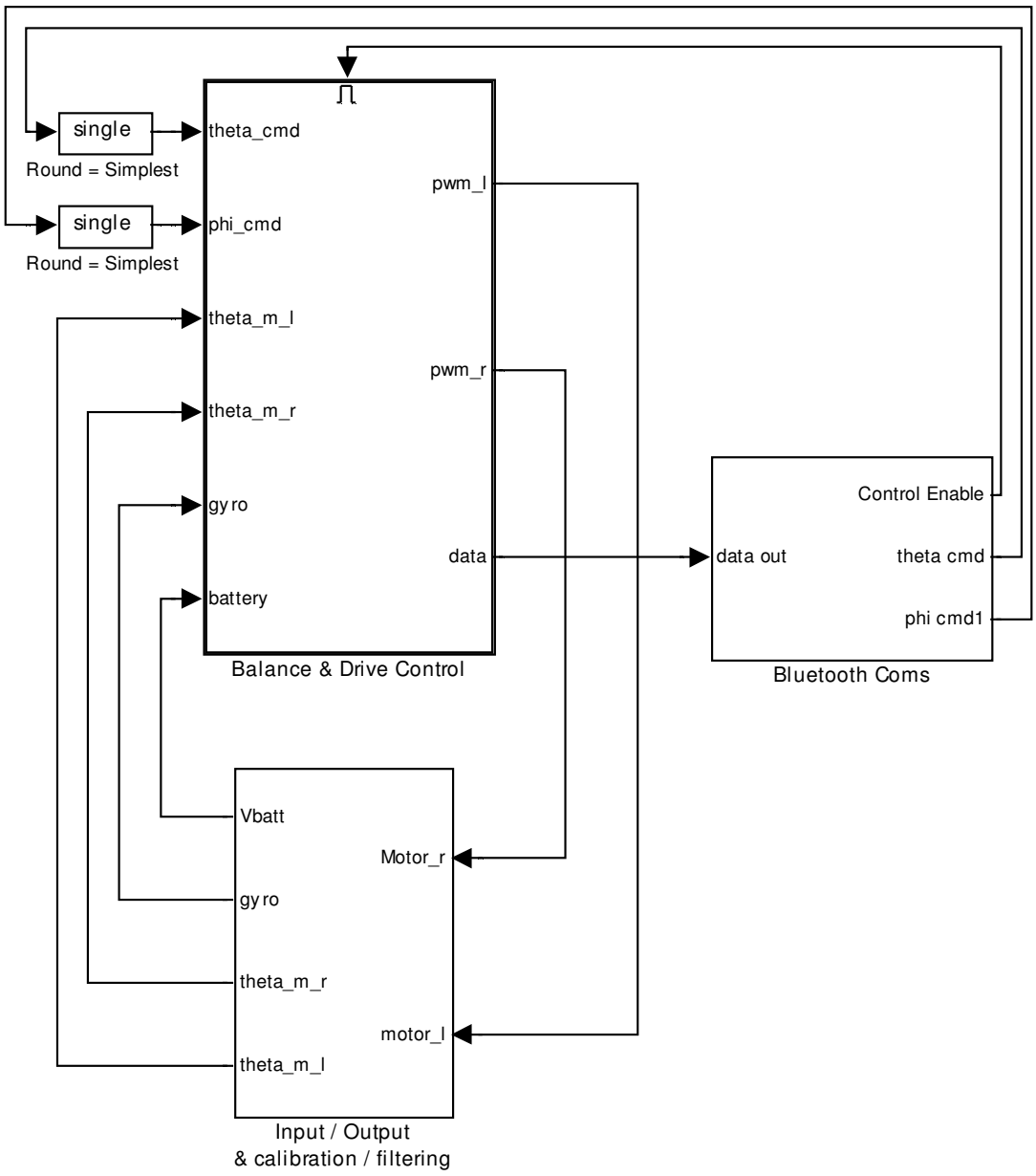

Figure 7.1. Robot Software

There are three main blocks used in the hardware controller. The sensor inputs and actuator outputs are found in the "Input/Output & Calibration / Filtering" block shown in Figure 7.2. The block receives the encoder inputs from each motor in degrees, the battery voltage in mV, time in ms, and the angular velocity from the gyro in deg/sec. The battery voltage is filtered with a 10 Hz low pass filter and

then converted to volts with a calibrated gain and offset. The gyro sensor has an offset that needs to be removed. This is accomplished by using a 10 Hz low pass filter for 1 second at the start of the program. The output of the filter is the gyro offset provided that the robot has been stationary. The filter is disabled after 1 second which holds and outputs the offset value.

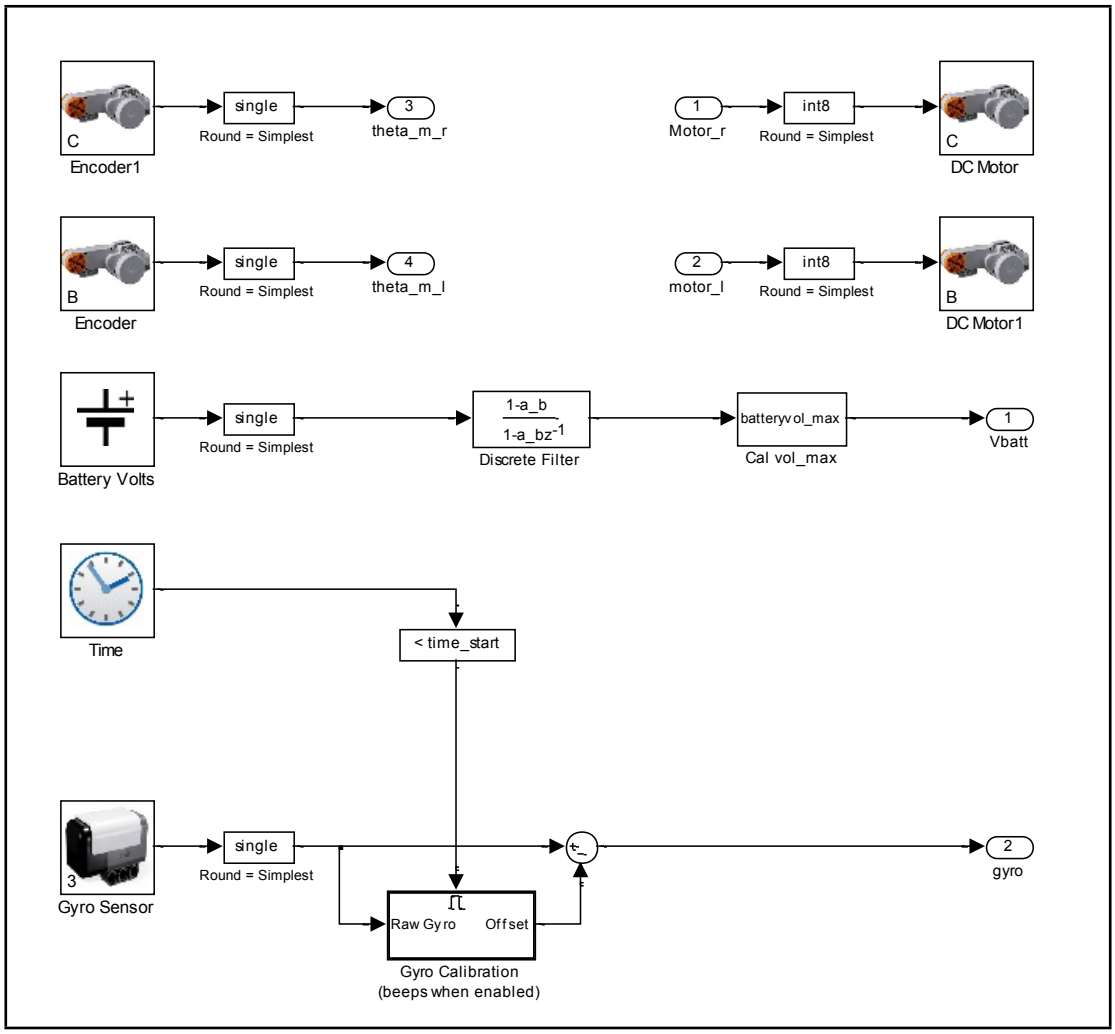

Figure 7.2. Input/Output & Calibration / Filtering

The "Bluetooth Coms" block shown in Figure 7.3 is the block that handles the bluetooth communications with the PC. The output data, which is single floating point precision, is parsed into 8 bit unsigned integers and sent to the PC. The

input data also requires reconstruction from 8 bit unsigned integers from the PC. The enable signal is converted from an 8 bit unsigned integer to boolean where a 1 enables the tracking system. θ and ϕ commands are converted from four 8 bit unsigned integers to 32 bit floats.

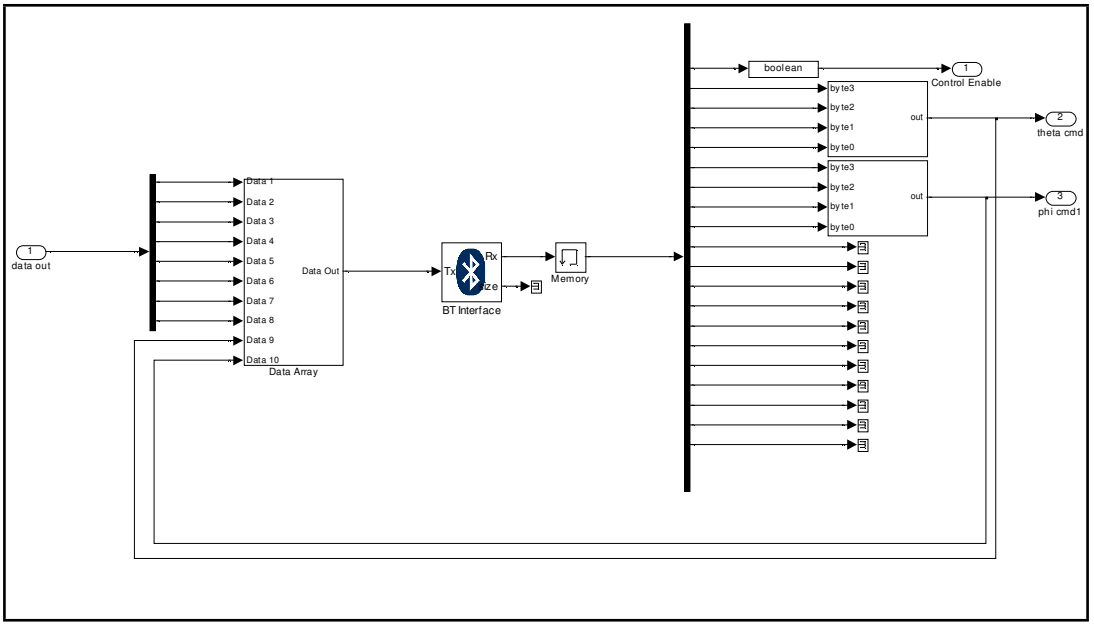

Figure 7.3. Bluetooth Coms

The final block "Balance & Drive Control" shown in Figure 7.4 contains the controller, implemented with three subblocks and is enabled through the bluetooth communications. The "States" block is shown in Figure 7.5 and is used to calculate the states from the three sensor inputs. In this block, all sensor inputs are converted from degrees to radians. The "Drift Comp" contains a 0.04 Hz low pass filter of the gyro signal to remove any DC offset that may accumulate in the gyro. The equations used in the "States" block are discussed in Section 5.4. The "Balance & Drive Control" block also contains the PWM generations shown in Figure 7.6. This block contains the limiters and the equations for the outputs as specified in Section 5.4.

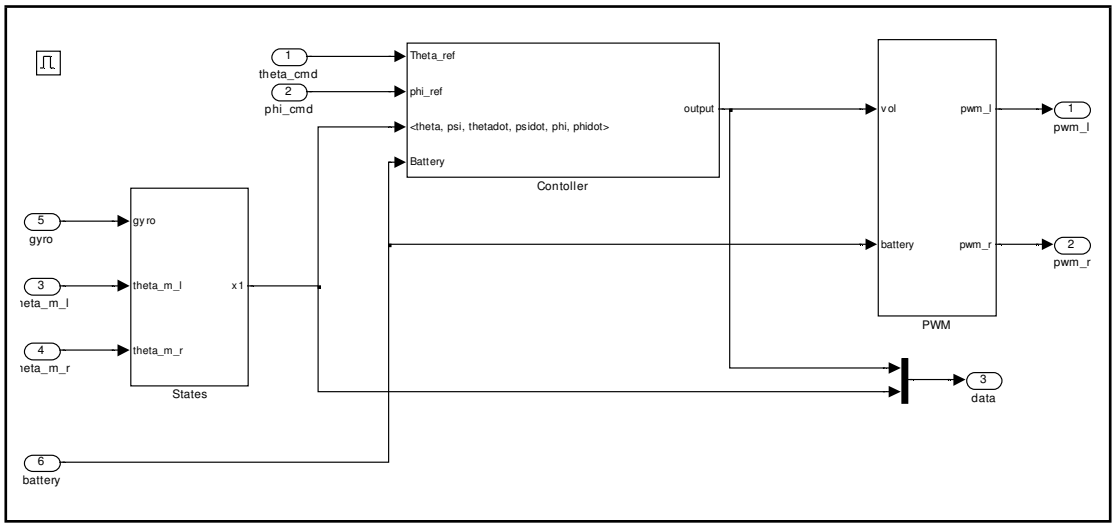

Figure 7.4. Balance & Drive Control

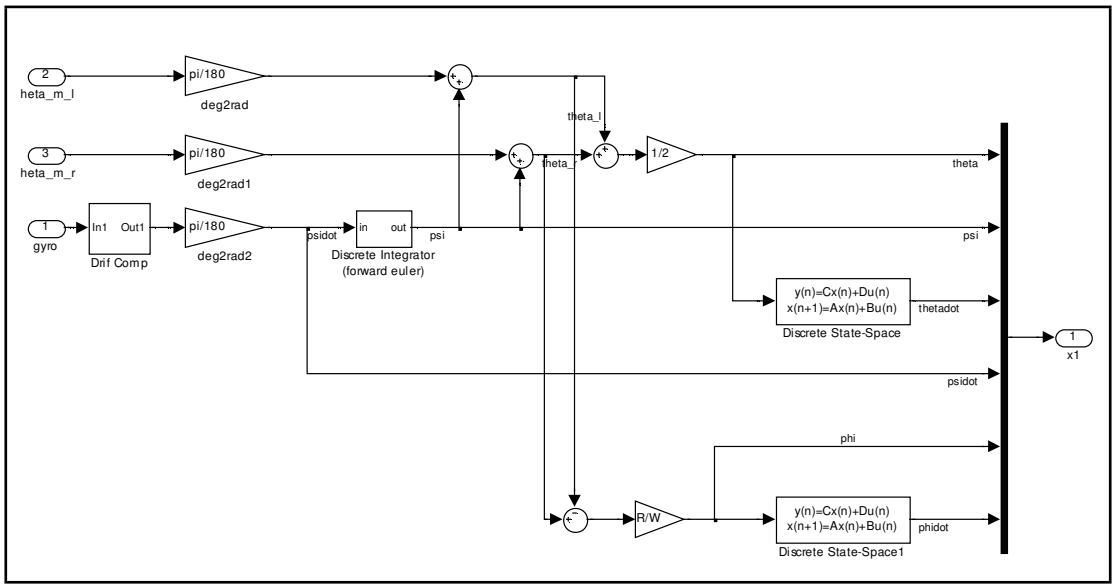

Figure 7.5. States

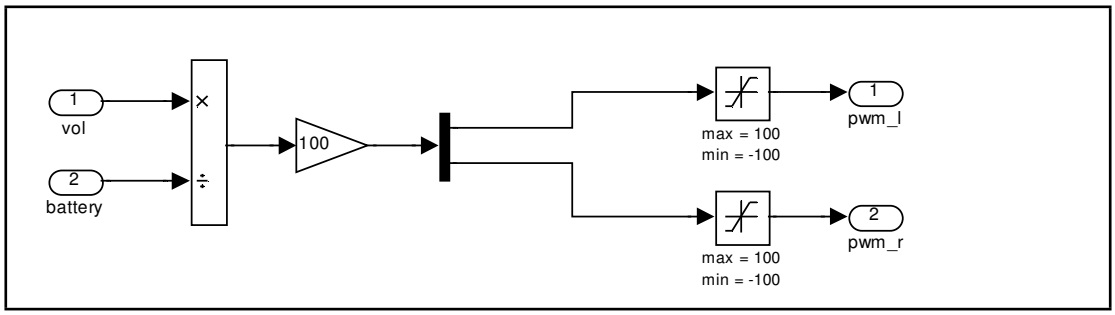

Figure 7.6. PWM Calculation

The final and most most important part of the "Balance & Drive Control" is the "Controller" block shown in Figure 7.7. This is where the actual controller resides. The block contains the state-space for the additional dynamics that is enabled when the output is in range. When the additional dynamics block is disabled, it holds the last output value until it is enabled once again. The enabling of the subsystem is handled by the "Integrator Logic" which ensures that the magnitude of the voltage commands are less than the battery voltage. The block will have the robot generate a beep whenever the output voltage command is out of range. Also, the "Controller" block generates the error for θ and ϕ and adds the output of the $-\mathbf{L_1}$ gain to the output of the additional dynamics.

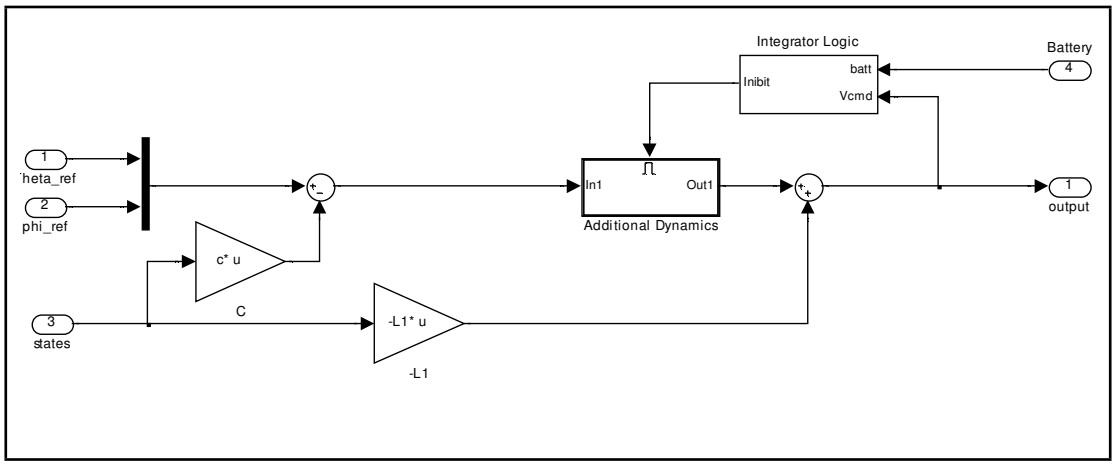

Figure 7.7. Robot Controller

The controller was exercised under four cases. These cases are *Balance*, *Ramping* θ, *Ramping* ϕ and a *figure eight*. The commands in these tests were duplicated from the simulations so that the results can be compared.

7.1 Balancing

This test was conducted to see how well the the robot would balance in place. Figure 7.8 shows the voltages that are sent out to the robot from the controller. It can be seen that there are large spikes in the voltage. These spikes are due

to the system trying to respond to the non-linear dead-band that is introduced by friction and mechanical slop in the motor gears. This is also compounded by the low resolution of the sensors. These effects are what dictated using a slower settling time for the controller. This also made it difficult to place an observer as part of the controller because of the large voltage spikes that would be fed into a linear observer. The states and their responses are shown in Figures 7.9 - 7.14.

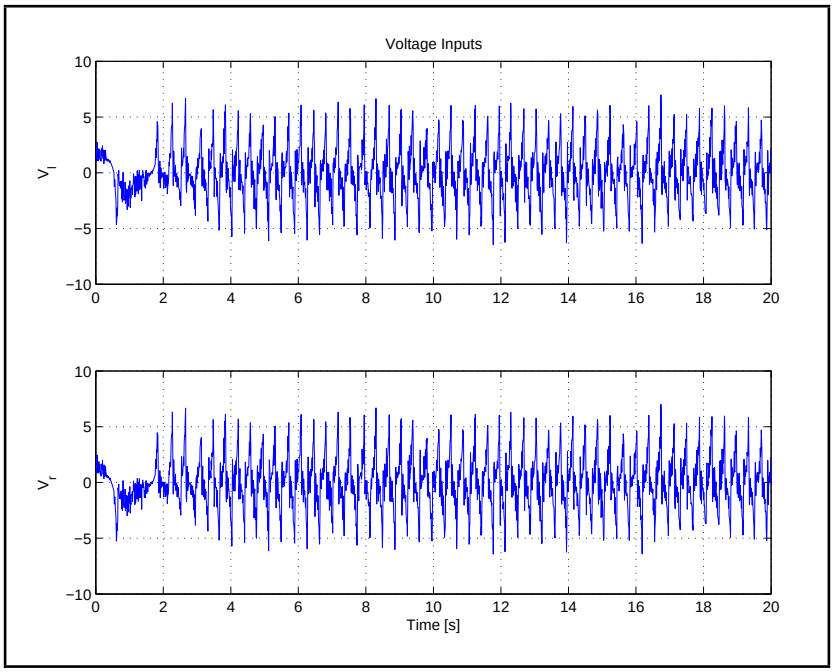

Figure 7.8. Balancing Voltage Commands

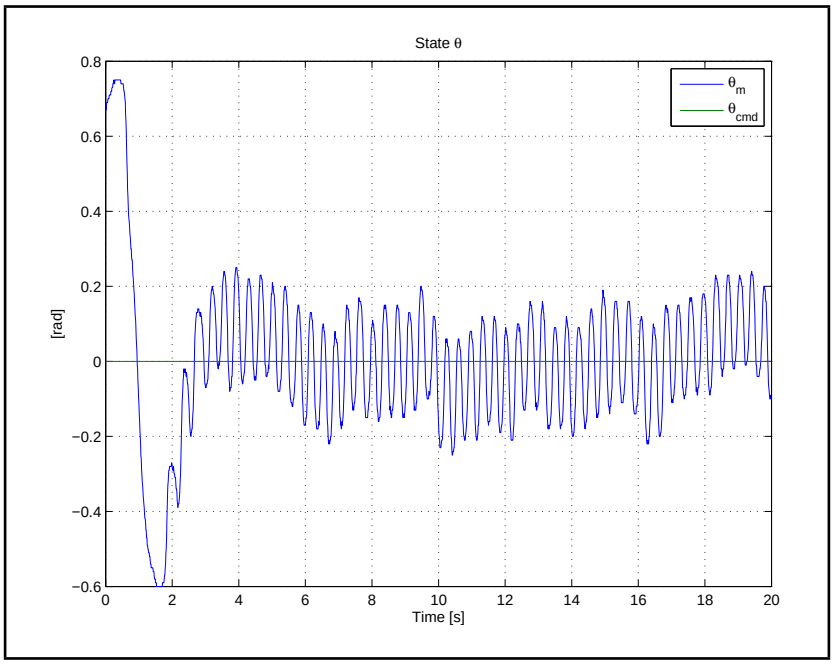

Figure 7.9. Balancing Measured θ State

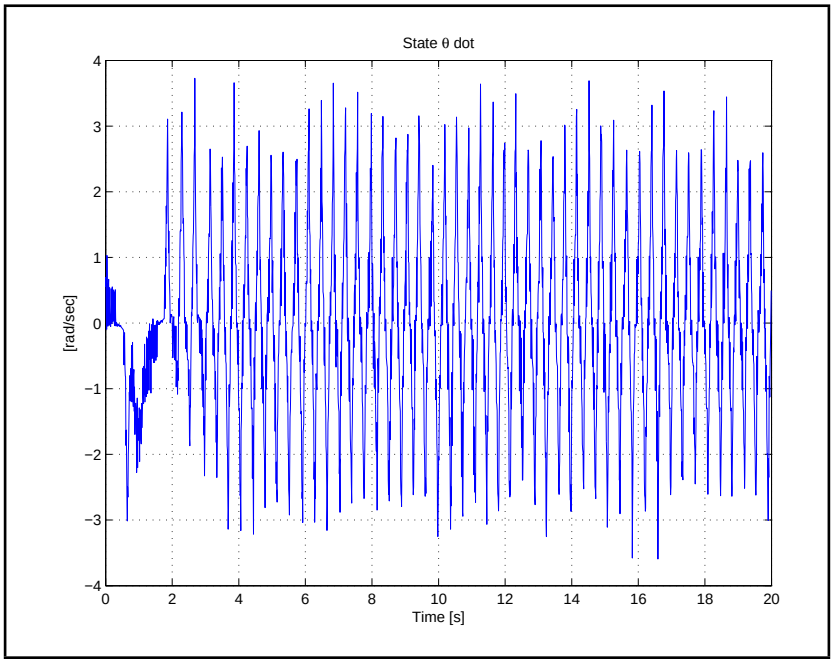

Figure 7.10. Balancing Measured $\dot{\theta}$ State

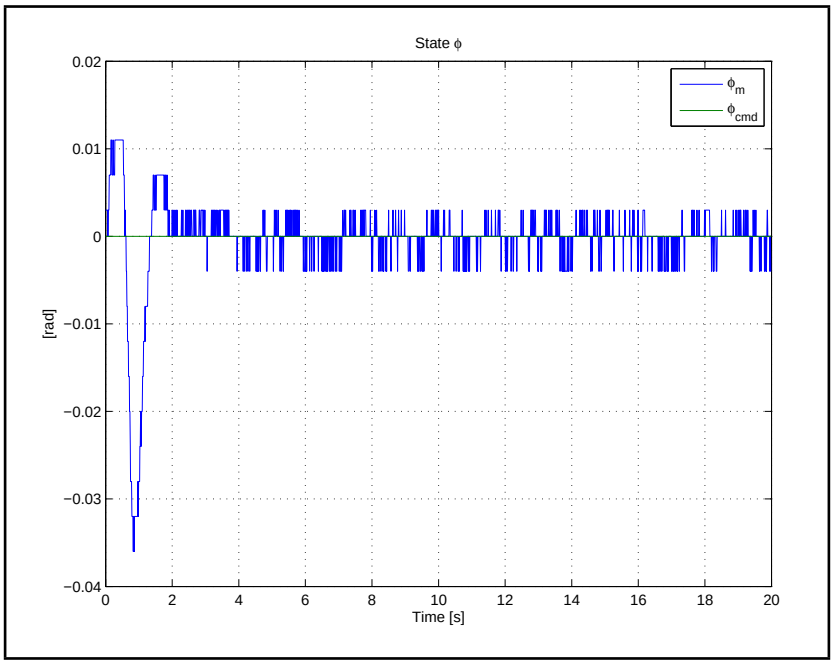

Figure 7.11. Balancing Measured ϕ State

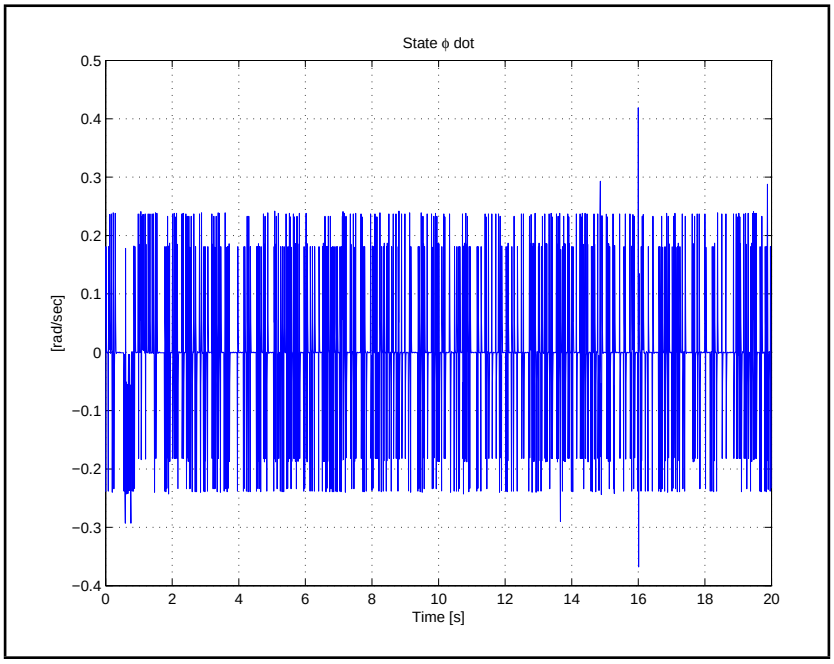

Figure 7.12. Balancing Measured $\dot{\phi}$ State

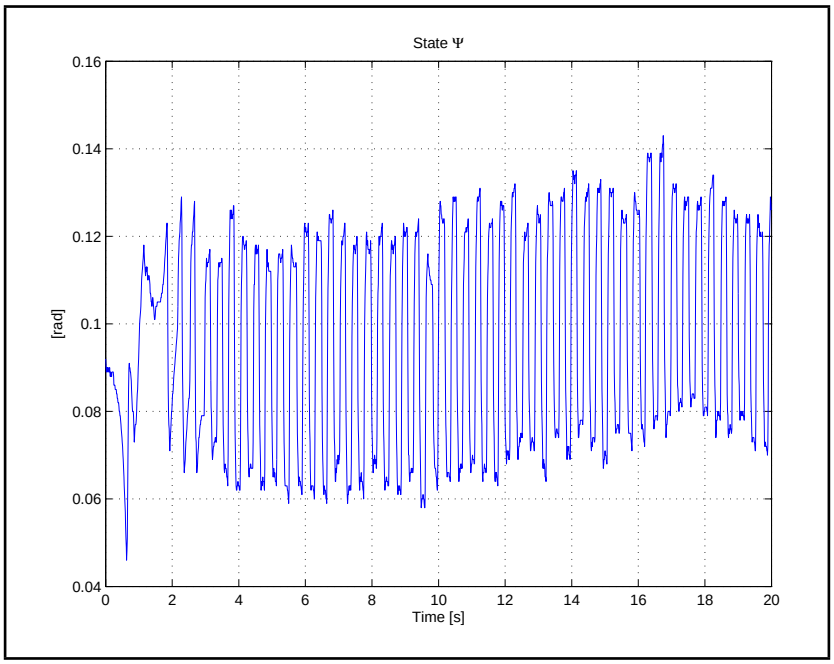

Figure 7.13. Balancing Measured Ψ State

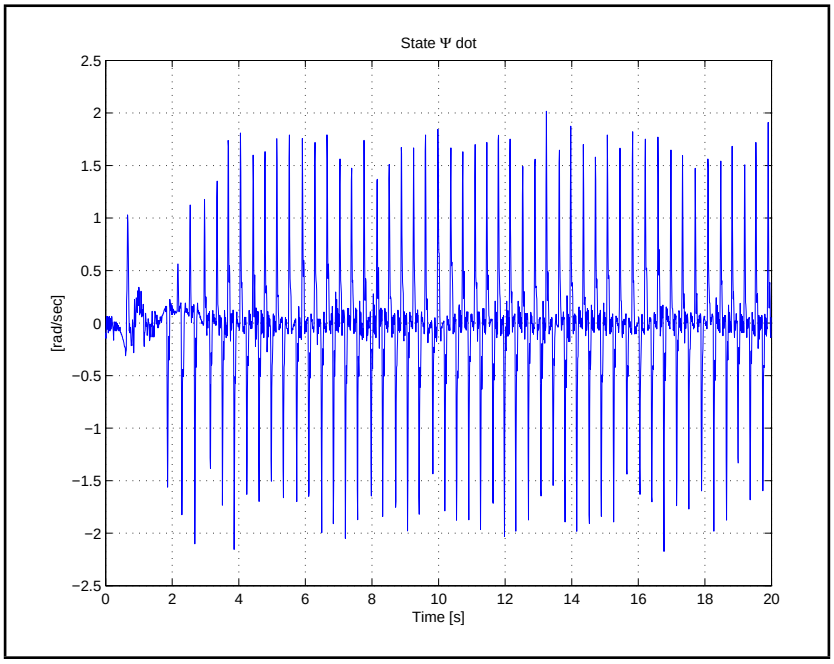

Figure 7.14. Balancing Measured $\dot{\Psi}$ State

7.2 Theta Ramping

In the θ ramping tests, the voltages in Figure 7.15 generally behave as expected where they have the same direction and magnitude for each wheel. The right wheel does show extra dynamics that the left wheel does not. The voltages are comparable to the voltages that can be seen in Figure 6.15 from simulation, which is a good indication that the model and the real hardware respond very similarly. This can also be seen in Figure 7.16 where the state variable θ has an overshoot of approximately 4 radians as in Figure 6.16 of the simulation. The response of ϕ to a ramping θ as shown in Figure 7.18 is not ideal since ϕ looks to be perturbed, unlike Figure 6.17 of the simulation which holds solid. This is most likely due to the fact that both motors have slightly different properties, while the model assumes that they have equal parameters. This variation can be seen in the wobbly movement in the XY plot Figure 7.22. The robot did remain well balanced throughout the maneuver where the measured state Ψ in Figure 7.20 has a very small variance.

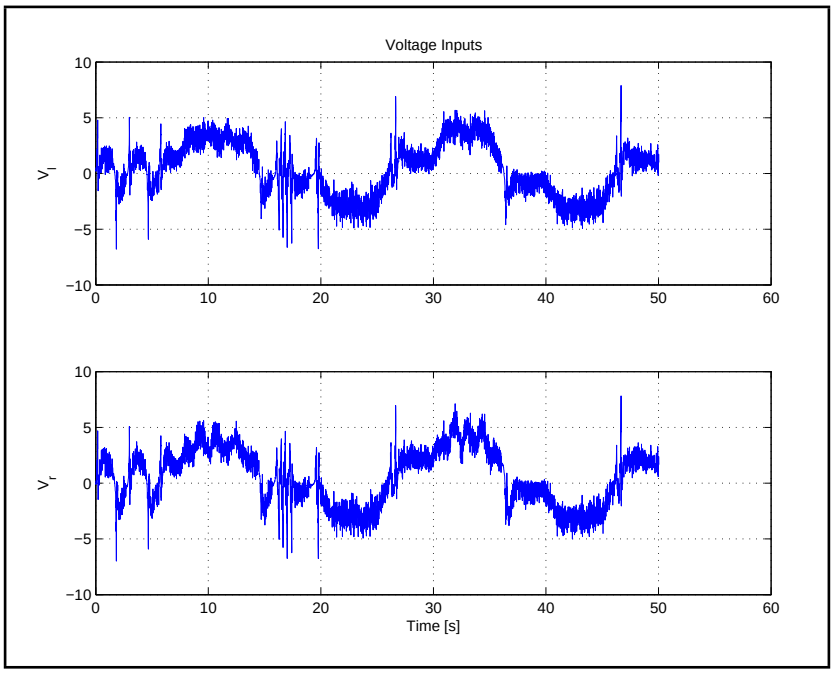

Figure 7.15. θ Ramping Voltage Commands

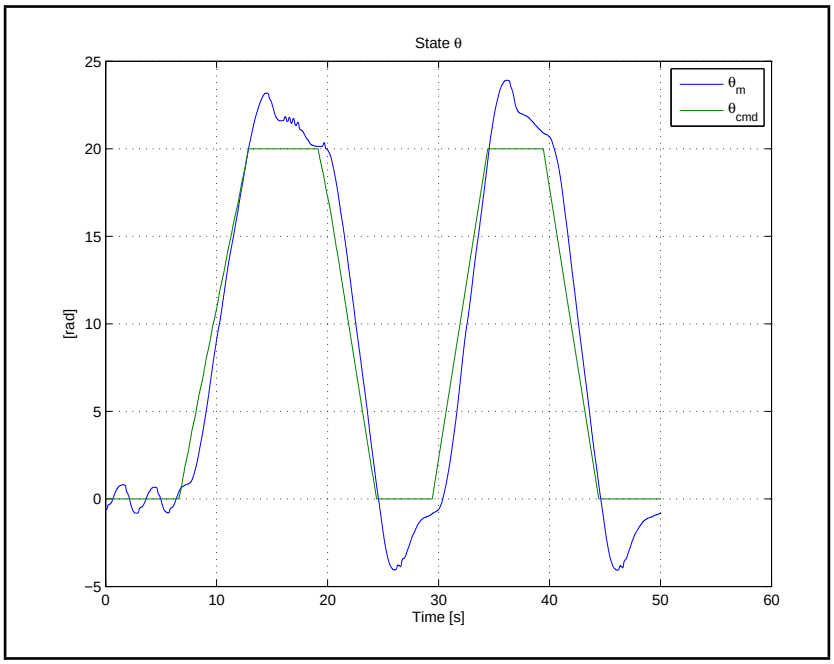

Figure 7.16. θ Ramping Measured θ State

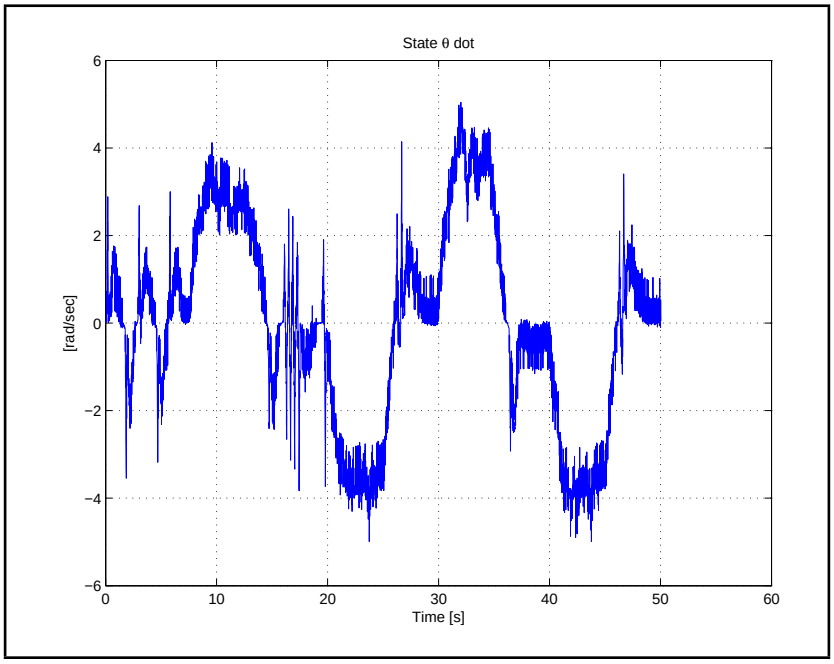

Figure 7.17. θ Ramping Measured $\dot{\theta}$ State

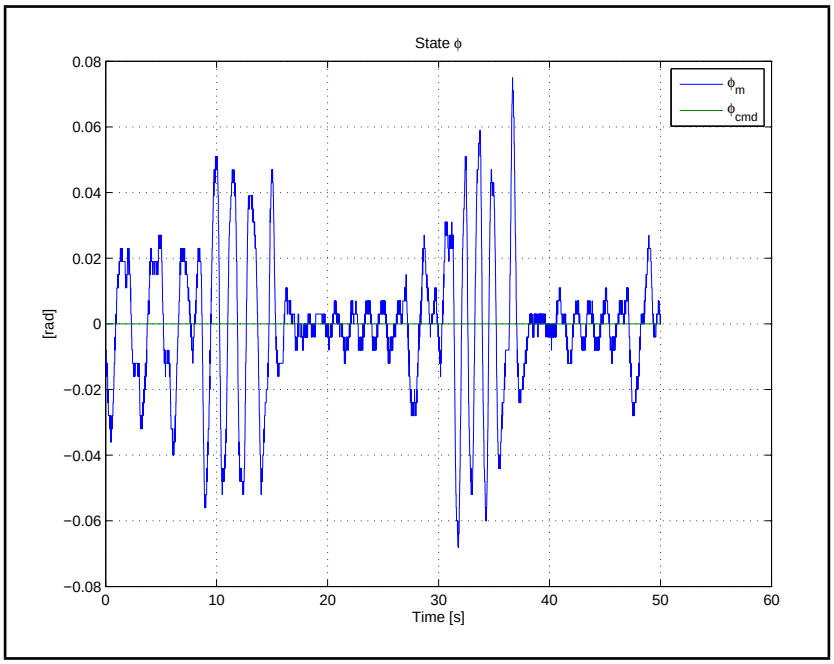

Figure 7.18. θ Ramping Measured ϕ State

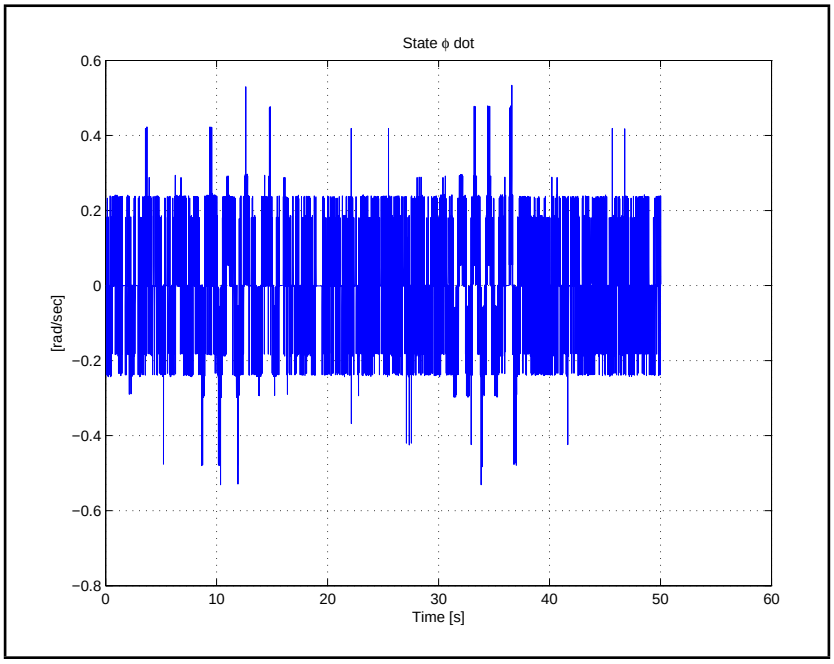

Figure 7.19. θ Ramping Measured $\dot{\phi}$ State

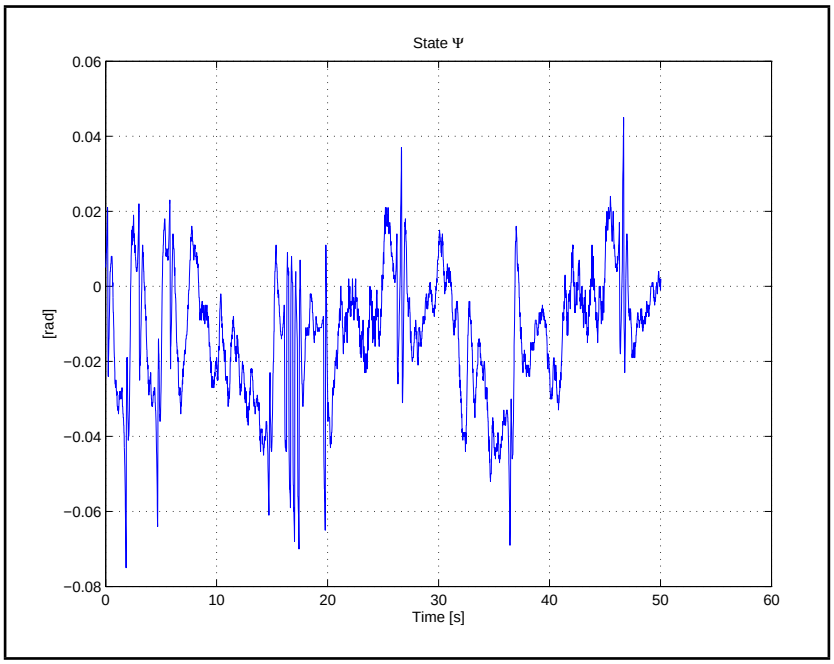

Figure 7.20. θ Ramping Measured Ψ State

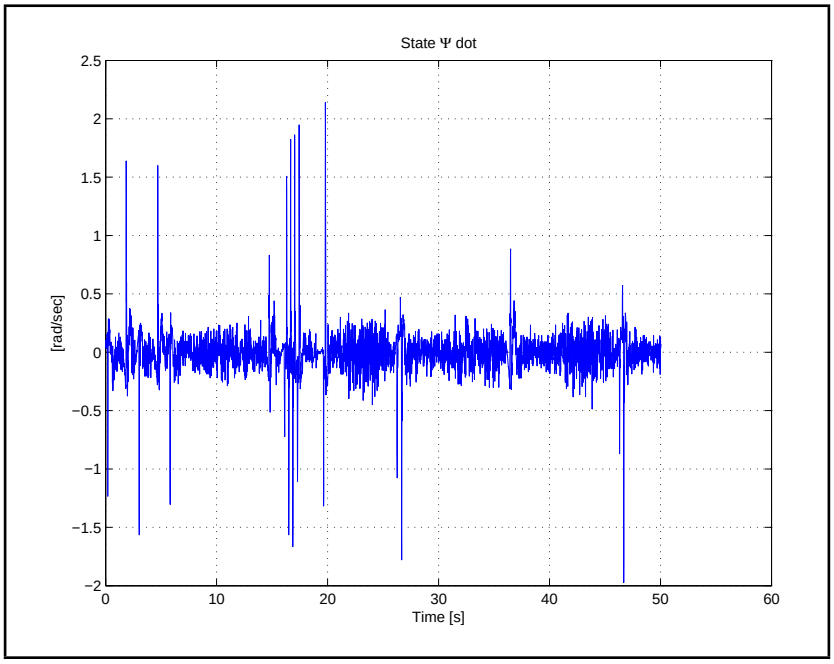

Figure 7.21. θ Ramping Measured $\dot{\Psi}$ State

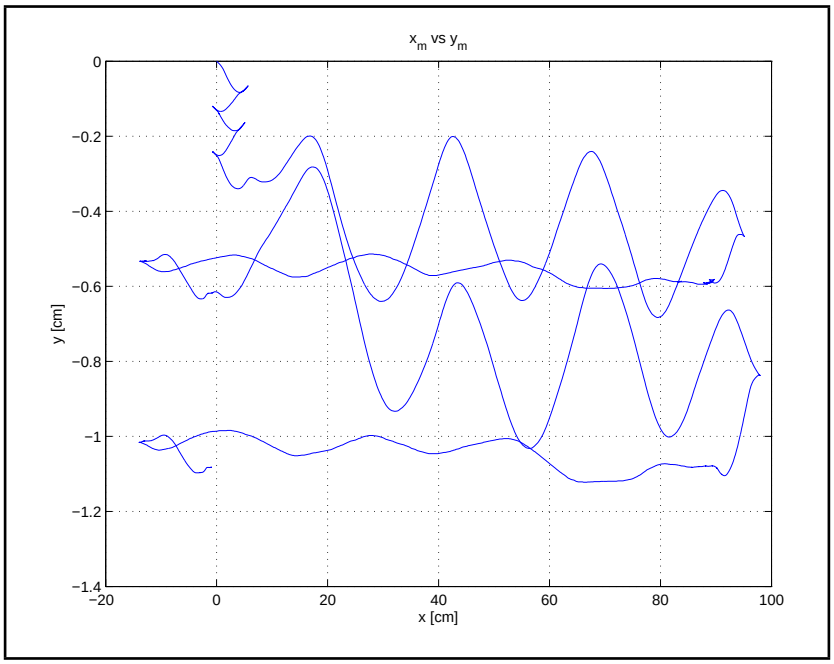

Figure 7.22. θ Ramping XY Plot

7.3 Phi Ramping

In the ϕ ramping tests, the voltages in Figure 7.23 are comparable to the voltages that can be seen in Figure 6.11. The voltages are opposite in direction and have equal magnitude for each wheel. Figure 7.25 has an overshoot of approximately 0.5 radians as in Figure 6.13 of the simulation. The response of θ to a ramping ϕ as shown in Figure 7.26 is not ideal since θ is perturbed, unlike Figure 6.12 of the simulation which holds solid. This variation can be seen in the wobbly movement in the XY plot Figure 7.30. The robot remained well balanced throughout the maneuver as shown in the measured state Ψ in Figure 7.28 .

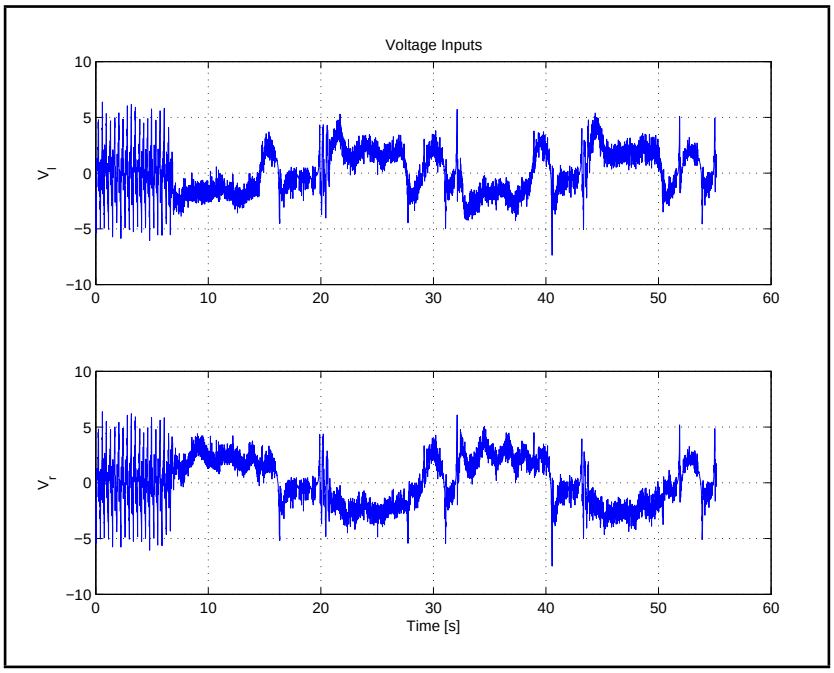

Figure 7.23. ϕ Ramping Voltage Commands

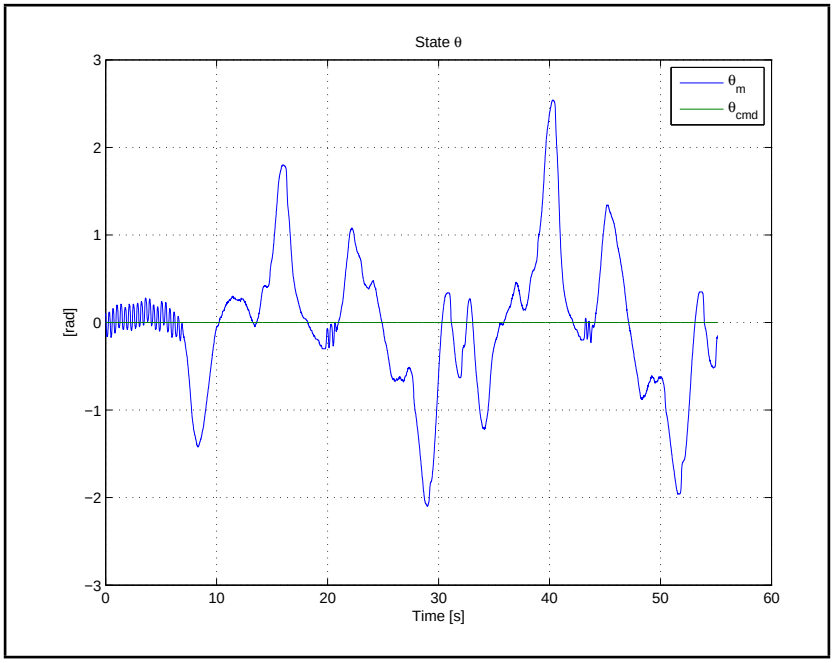

Figure 7.24. ϕ Ramping Measured θ State

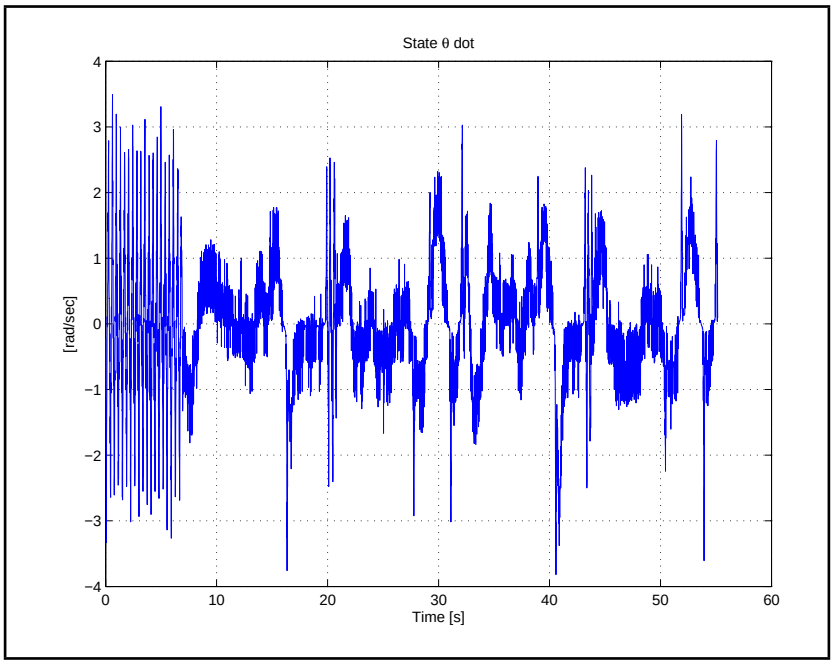

Figure 7.25. ϕ Ramping Measured $\dot{\theta}$ State

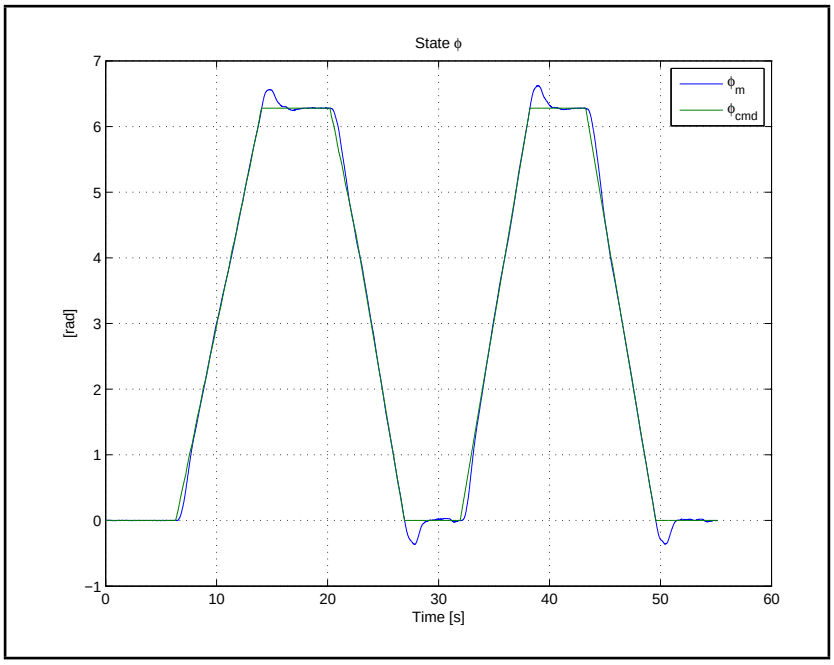

Figure 7.26. ϕ Ramping Measured ϕ State

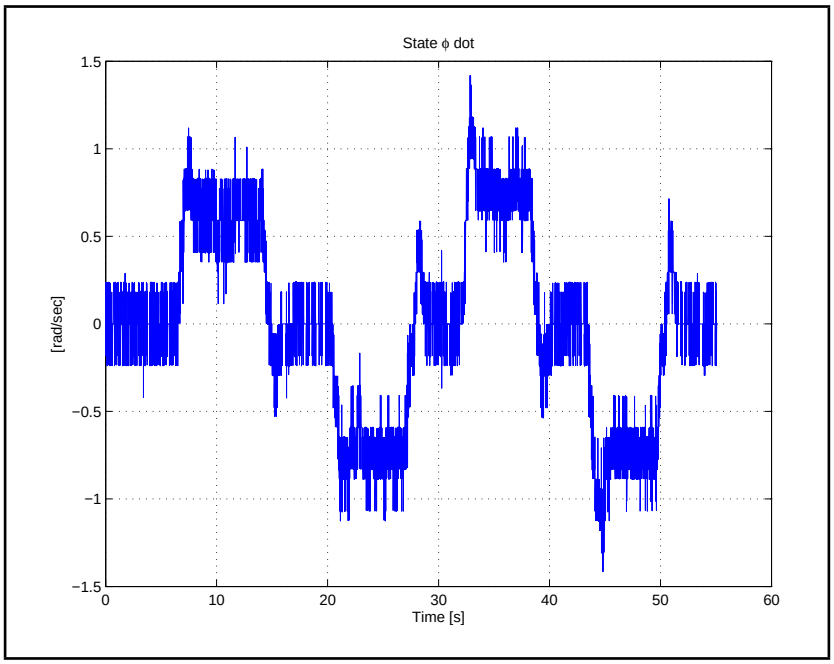

Figure 7.27. ϕ Ramping Measured $\dot{\phi}$ State

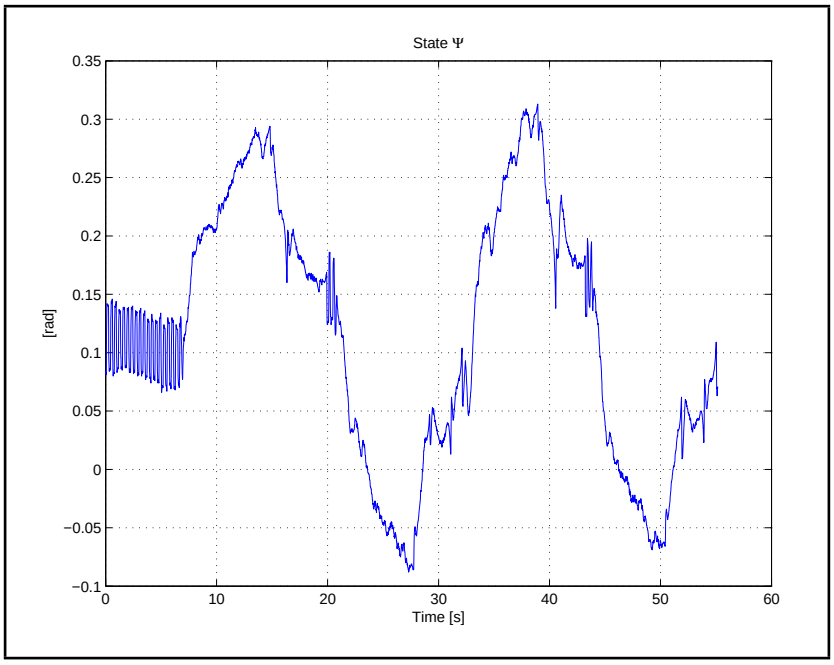

Figure 7.28. ϕ Ramping Measured Ψ State

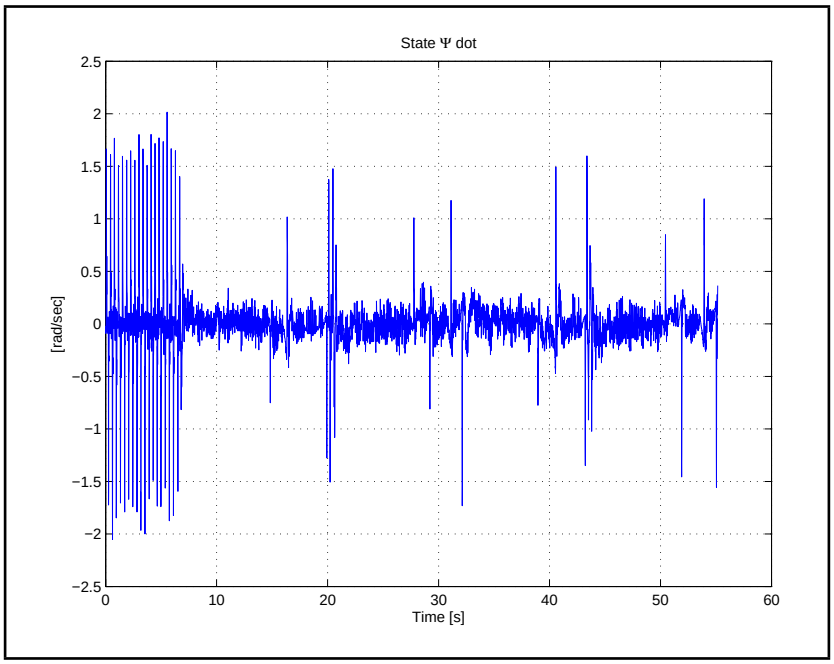

Figure 7.29. ϕ Ramping Measured $\dot{\Psi}$ State

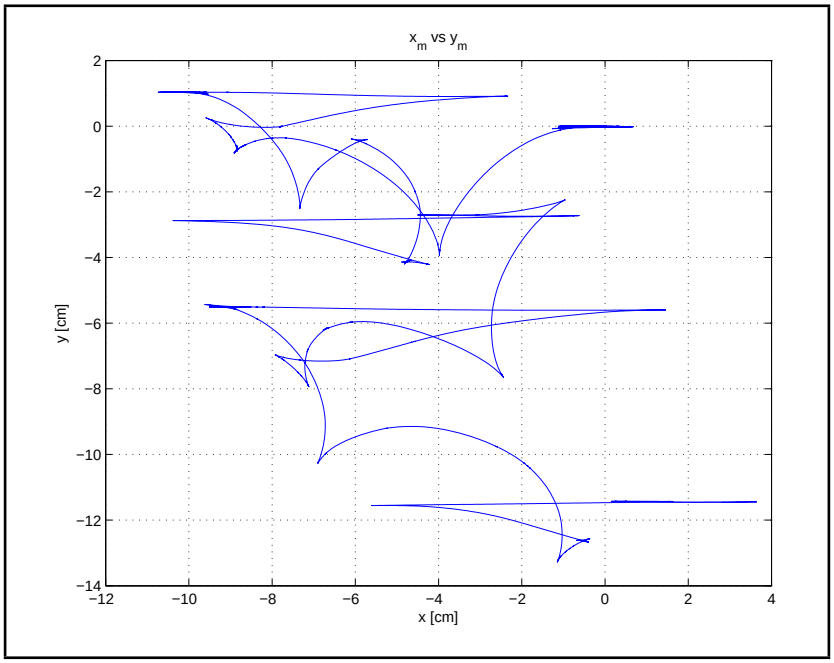

Figure 7.30. ϕ Ramping XY Plot

7.4 Figure Eight

The figure eight in this test was commanded with an $A_x = 50$ cm, $A_y = 100$ cm and T = 10. The voltages in Figure 7.31 are comparable to the voltages that can be seen in Figure 6.25 of the simulations, except they are much noisier. Figure 7.32 shows that the commanded θ is tracked very well and is comparable to the simulation results in Figure 6.26. This is also seen in the tracking of ϕ when comparing Figure 7.34 to Figure 6.27 of the simulations. Figure 7.36 shows that Ψ has very little movement so the robot is well balanced. Finally, the XY plot Figure 7.38 shows how the robot moved in the XY plane. The figure eight is slightly deformed but it's deformity can be compared to Figure 6.29. This error is due to how well the states θ and ϕ match in time. The biggest issue is the start-up of θ which does not track for the first 3 seconds. This causes θ and ϕ to always be misaligned. The robot does track ramps in both θ and ϕ very well.

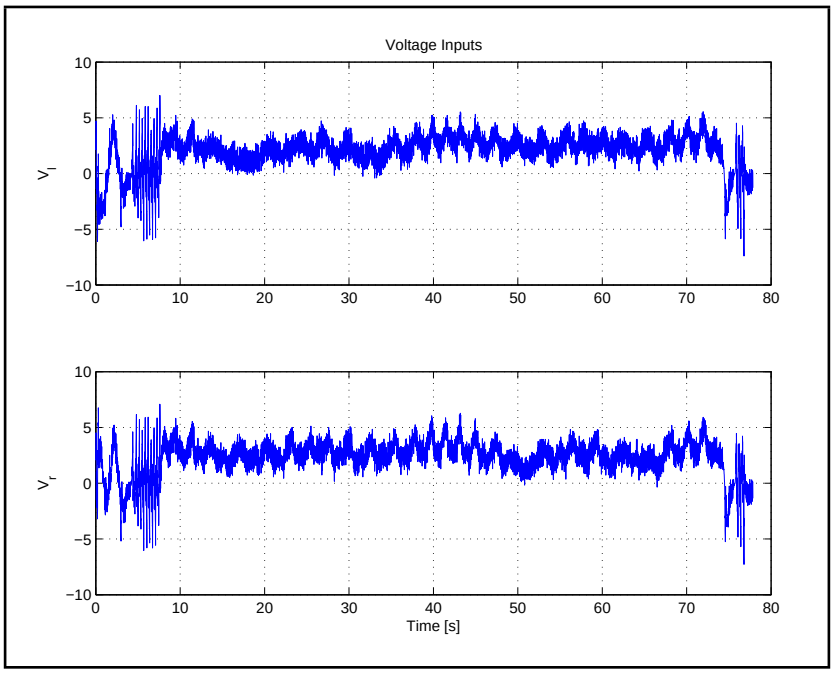

Figure 7.31. Figure Eight Voltage Commands

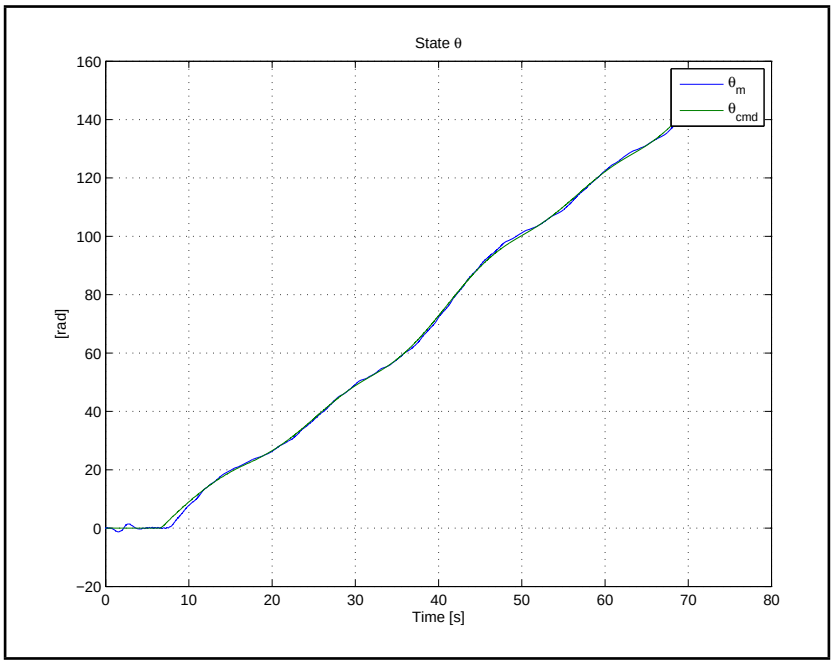

Figure 7.32. Figure Eight Measured θ State

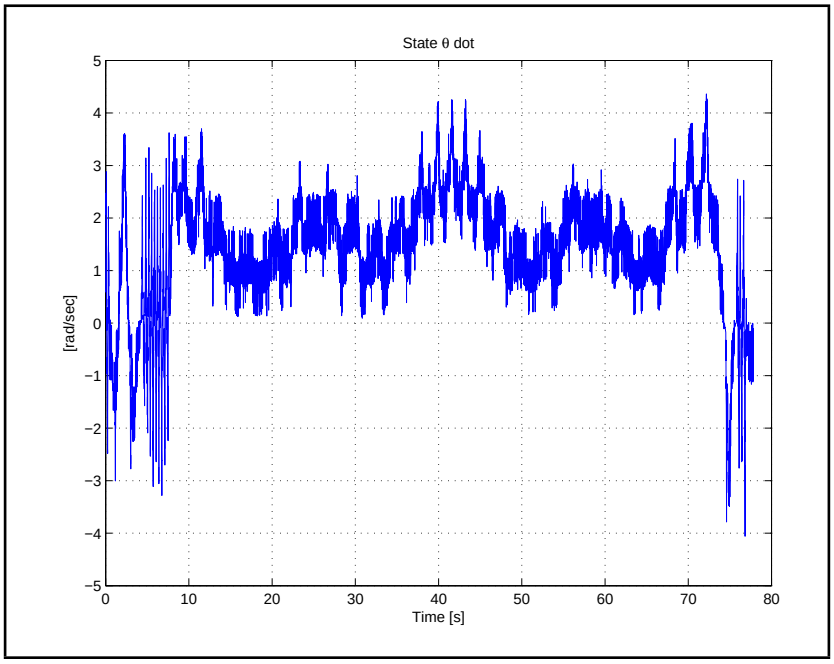

Figure 7.33. Figure Eight Measured $\dot{\theta}$ State

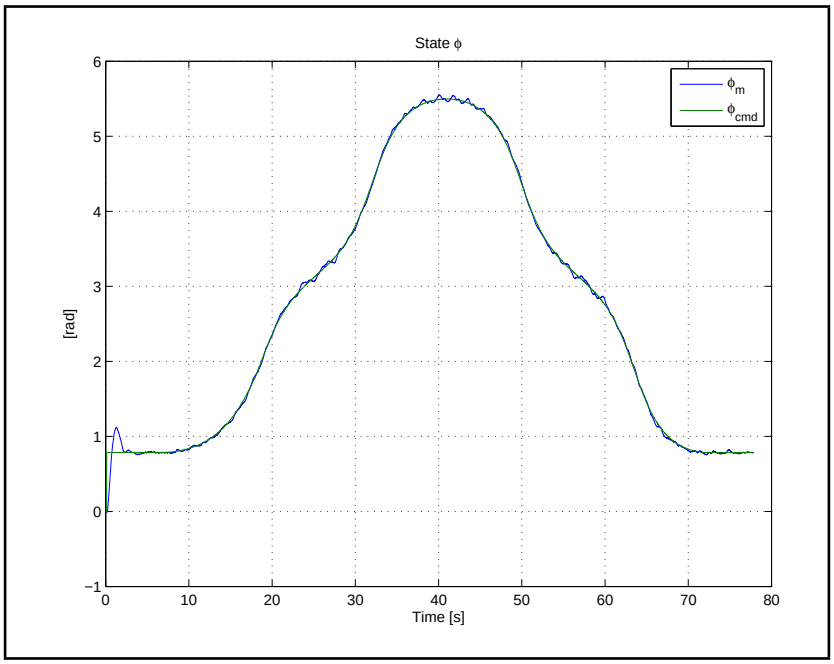

Figure 7.34. Figure Eight Measured ϕ State

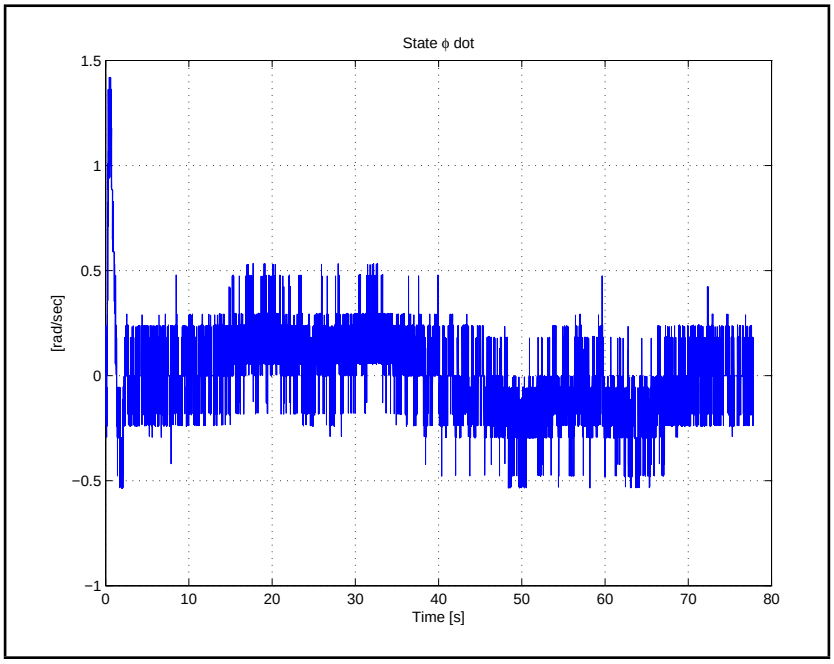

Figure 7.35. Figure Eight Measured $\dot{\phi}$ State

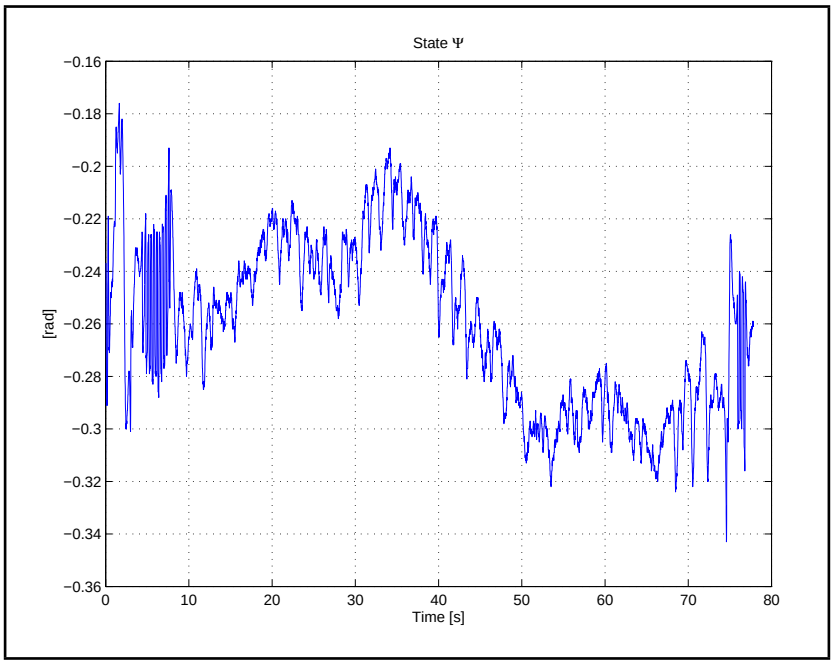

Figure 7.36. Figure Eight Measured Ψ State

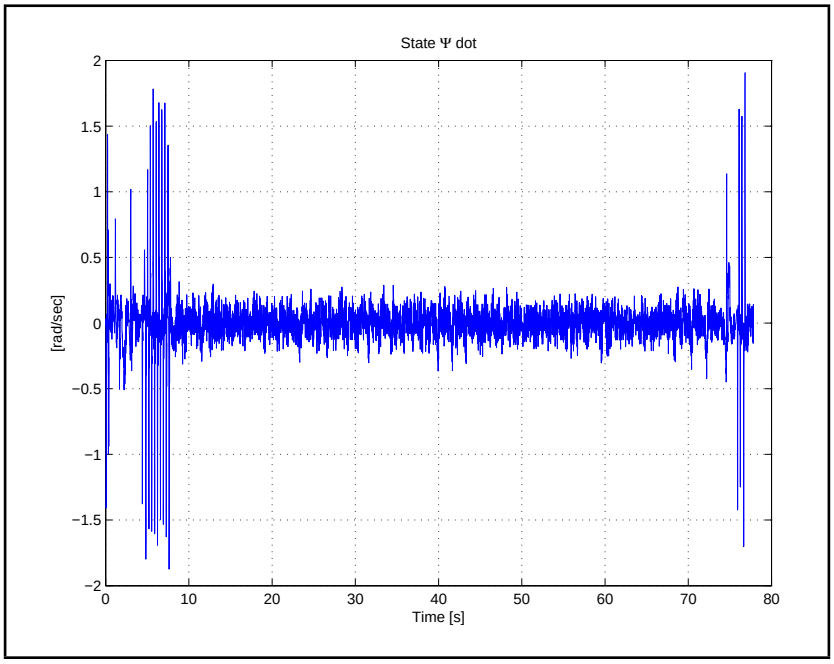

Figure 7.37. Figure Eight Measured $\dot{\Psi}$ State

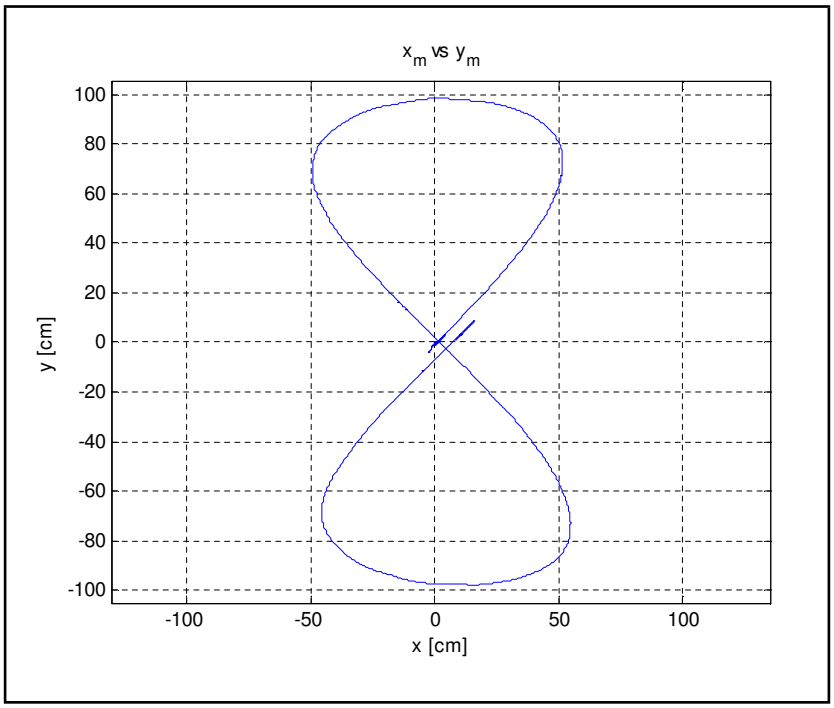

Figure 7.38. Figure Eight XY Plot

The figure eight was also ran with $A_x = 50$ cm, $A_y = 100$ cm and T = 20. Increasing the period of the figure eight resulted in a much cleaner path seen in the XY plot Figure 7.39. The path was improved due to the controllers ability to more accurately track the slower commands in ϕ and θ with less error. These results are similar to the ones seen in Figure 6.32 in simulation.

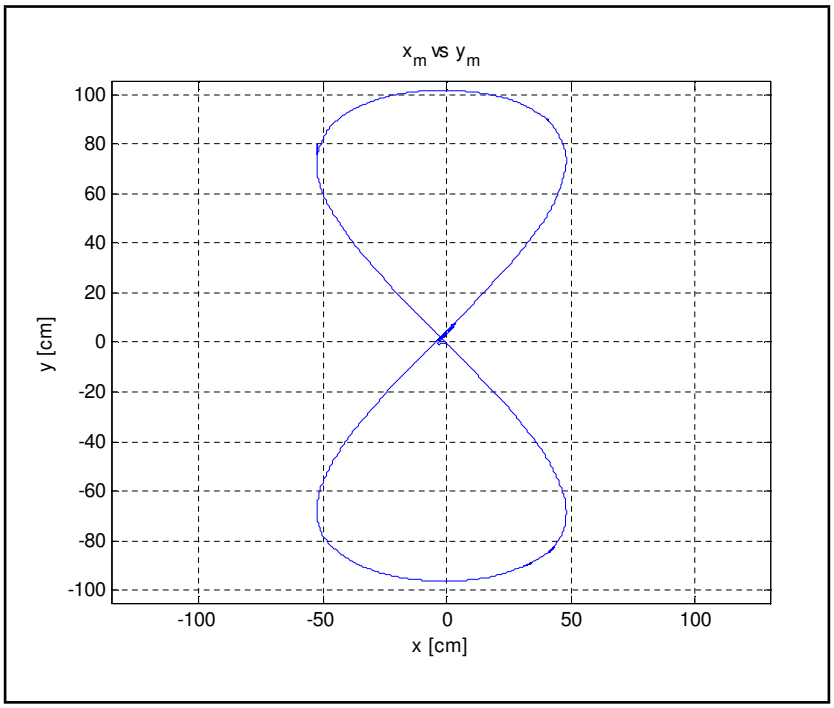

Figure 7.39. Slow Figure Eight XY Plot

CHAPTER 8

Conclusions

The robot is a MIMO system with two inputs and six state variables. In this thesis the state variables θ and ϕ are the tracked outputs and all state variables were regulated. The design of the controller was based on pole placement. The placement of the poles determined the settling time of the system. Due to non-linearities in the motors and quantization in the system, the settling time needed to be increased, which reduced the performance of the controller.

The design of the tracking system placed consideration on hardware constraints. Some of these constraints were voltage saturation, quantization and the dead band in the motors. These effects were predominant while balancing in place. Quantization introduced a derivative action at low speeds which, if the gains were too high, would cause the system to be driven into saturation. Decreasing the settling time also introduced problems with dead band. The dead band forced the integrators to wind up and again, if the gains were too high, the output would saturate. To compensate for these non-ideal conditions the settling time was increased. The performance of the digital tracking system as seen in the hardware and simulation testing was reduced with a higher settling time. The result was that the state variable θ had a high level of overshoot and the figure eight was malformed. It was shown that a figure eight with a longer period could be better tracked. Although it was not shown in this thesis, decreasing the settling time would result in perfect θ and ϕ ramp tracking with very little overshoot in simulations. However, decreasing the settling time does not produce zero error to tracking x_m and y_m. Although the error is reduced, the figure eight may still drift due to start up delay in θ.

This thesis provides a basis for future design for a more accurate two-wheeled robot trajectory tracking. The current design can accurately move back and forth, and turn under closed-loop control. This would allow movement through a maze that contains straight paths with turns at the end of the paths. The controller would have problems with paths that have arcs. These issues are due to the non-linear kinematic characteristics and non-holonomic constraints of a wheeled mobile robot [1]. The controller introduced in this thesis relies on θ and ϕ having zero error at every instant in time to track complex paths such as a figure eight. When state variables θ and ϕ are not synchronized in time the two-wheeled robot will go slightly off course. One source of this error it that the closed-loop system may have different delays for θ and ϕ. Future designs may incorporate an inverse kinematic transform to compensate for these system dynamics. The inverse kinematic transform would have the desired trajectory as inputs and produce θ and ϕ commands that are compensated for by the system dynamics. Another source of error are disturbances in the state variable Ψ that results in a disturbance in θ and ϕ in order to balance. This could be remedied by having another slower outer-loop controller that corrects for error in x_m and y_m.

The feedback control algorithms **place**, **fbg** and **TFBG** were all examined. Analysis and simulations showed that even though a feedback matrix may have better stability margins, it may not have the best performance. These algorithms were all compared to one another in both steady-state tracking and in ramp tracking. The **place** algorithm did not produce very good results, which has been the case in other studies [2]. **TFBG** showed the best results. The algorithm searches for the best stability margins for both inputs. It also searches for a desired symmetry in the feedback matrix.

The controller was able to track θ and ϕ, which was demonstrated in both

simulation and in the hardware. The hardware and simulation showed difficulty in tracking a figure eight at high a high speed. In both simulation and hardware, slowing down the commands by changing the parameter T from 10 to 20 allowed the system to track much better. The result was a very slow moving system that tracked.

List of References

[1] M. Nilulescu, "Controlling a mobile robot along planned trajectories," *CEAI*, vol. 7, no. 20, pp. 18–24, 2005.

[2] S. Nawawi, M. Ahmad, J. Osman, A. Husain, and M. Abdollah, "Controller design for two-wheels inverted pendulum mobile robot using PISMC," *4th Student Conference on Research and Development*, pp. 194 – 199, June 2006.

APPENDIX

Matlab Code

```
%Parameters.m
%
%Lego Two-Wheeled Inverted Pendulum State-Space Model
% clear
% clc

g = 9.81;          %Acceleration due to gravity [m/s^2]

Tp = 0.7647;       %Period of the pendulum found experimentaly in Gyro Test
wp = (1/Tp)*2*pi;  %Natural frequency of the penulum in [rad/sec]
H = 0.144;
D = 0.04;
W = 0.175;
M  = 0.6;          %mass of robot with out wheels in [Kg] need to measure
R  = 0.04;         %Radius of the wheels [m]
m  = 0.03;         %mass of one wheel
Jw = m*R^2/2;      %Wheel moment of inertia
alpha = 2*pi/5.9153;  %ouput/input of motor at steady state [rad/(V*sec)]
Jphi = M*(W^2 + D^2)/12;
Kt = 0.317;
Kb = 0.468;
Rm = 6.69;

Mc = (M*g-wp^2*M*R)/((4/3)*wp^2*M);
fm = (Kt/Rm - alpha*Kt*Kb/Rm)/alpha;

C1 = (2*m + M)*R^2 + 2*Jw;
C2 = M*R*Mc;
C5 = M*g*Mc;
C3 = 4/3*M*Mc^2;
C4 = (Kt*Kb/Rm + fm);
C6 = Kt/Rm;

E = [C1 C2
    C2 C3];

F = 2*[C4 -C4
      -C4 C4];

G = [0 0
     0 -C5];

H = [C6 C6
    -C6 -C6];

I = 1/2*m*W^2+Jphi+W^2/(2*R^2)*Jw;

J = W^2/(2*R^2)*C4;

K = [-W/(2*R) *C6, W/(2*R) * C6];

A1 = -inv(E)*G;
A2 = -inv(E)*F;
A3 = -J/I;

B1 = inv(E)*H;
```

```
57 B2 = K/I;
58
59 %Full Plant A Matrix
60 A = zeros(6,6);
61 A(1,3) = 1;
62 A(2,4) = 1;
63 A(3,1) = A1(1,1);
64 A(3,2) = A1(1,2);
65 A(3,3) = A2(1,1);
66 A(3,4) = A2(1,2);
67 A(4,1) = A1(2,1);
68 A(4,2) = A1(2,2);
69 A(4,3) = A2(2,1);
70 A(4,4) = A2(2,2);
71 A(5,6) = 1;
72 A(6,6) = A3;
73
74 %Full Plant B Matrix
75 B = [
76     0 0
77     0 0
78     B1
79     0 0
80     B2
81     ];
82 C = eye(6);
83
84 D = zeros(6,2);
```

```
%Controller.m

%This file defines the controller for the Lego Robot

%load bessel roots
load sroots

%If Tracking equals 1 the controller will perfrom ramp tracking otherwise
%steady-state tracking
TRACKING = 1;

%Vector Feedback Algorithm
%1 = RFBG, 2 = FBG, 3 = Place
FBA = 1;

A = A;                          % A Matrix from Plant model
b = B;                          % B Matrix from Plant model
T = 0.01;                       % Sampling Time
c = [1 0 0 0 0 0; 0 0 0 0 1 0]; % States that are tracked

%Calculate the ZOH model
[phi,gamma] = zohe(A,b,T);

%Calculate the controllabilty matrix
Wc = [gamma phi*gamma phi^2*gamma phi^3*gamma phi^4*gamma phi^5*gamma];

if TRACKING == 1    %Ramp Tracking Designs
    %Determine the settling time
    Ts = 3;
    Ts_2 = 5;

    %Calculate the closed loop poles
    spoles = [-460.7253, -6.6185, -182.0352, s2/Ts_2,s5/Ts];
    zpoles = exp(spoles*T);

    % Calculate the Additional Dynamics for Ramp Tracking
    Aint = [0 1; 0 0];
    Bint = [0;1];

    [phia, gammaa] = zohe(Aint,Bint,T);

    phia = [phia zeros(length(phia));zeros(length(phia)) phia];
    gammaa = [gammaa zeros(length(gammaa),1);zeros(length(gammaa),1)...
        gammaa];

    %Calculate the open loop state-space Model of the plant and additional
    %dynamics
    phid=[phi zeros(length(phi),length(phia)); gammaa*c phia];
    gammad=[gamma; zeros(size(gammaa))];

    %Calculate TFBG Gains
    L=TFBG_ramp(phid,gammad,zpoles,T,0.05);
    L11=L(:,1:length(phi));
    L21=L(:,length(phi)+1:length(phia)+length(phi));

    %Calculate FBG Gains
```

```
    L=fbg(phid,gammad,zpoles);
    L12=L(:,1:length(phi));
    L22=L(:,length(phi)+1:length(phia)+length(phi));

    %Calculate Place Gains
    L=place(phid,gammad,zpoles);
    L13=L(:,1:length(phi));
    L23=L(:,length(phi)+1:length(phia)+length(phi));

else %Steady-State Tracking Designs

    %Determine the settling time
    Ts = 3;
    Ts_2 = 1;

    %Calculate the closed loop poles
    spoles = [-460.7253, -6.6185, -182.0352, s2/Ts_2, s3/Ts];
    zpoles = exp(spoles*T);

    %Additional Dynamics for Steady-State Tracking
    phia = [1 0; 0 1];
    gammaa = [1 0; 0 1];

    %Calculate the open loop state-space Model of the plant and additional
    %dynamics
    phid=[phi zeros(length(phi),length(phia)); gammaa*c phia];
    gammad=[gamma; zeros(size(gammaa))];

    %Calculate TFBG Gains
    L=TFBG_ss(phid,gammad,zpoles,T,0.001);
    L11=L(:,1:length(phi));
    L21=L(:,length(phi)+1:length(phia)+length(phi));

    %Calculate FBG Gains
    L=fbg(phid,gammad,zpoles);
    L12=L(:,1:length(phi));
    L22=L(:,length(phi)+1:length(phia)+length(phi));

    %Calculate Place Gains
    L=place(phid,gammad,zpoles);
    L13=L(:,1:length(phi));
    L23=L(:,length(phi)+1:length(phia)+length(phi));

end

%Select which pole placement algorithm to implement
if FBA == 1
    L1 = L11;
    L2 = L21;
elseif  FBA == 2
    L1 = L12;
    L2 = L22;
elseif  FBA == 3
    L1 = L13;
    L2 = L23;
end
```

```
113
114
115 %filtered derivative
116 ts_f = 0.025;
117 b = poly(s2/ts_f);
118 fil1 = tf([b(3) 0],b);
119 [fil1n fil1d] = tfdata(c2d(fil1,T));
120
121 Af = [0 1; -b(3)/b(1) -b(2)/b(1)];
122 Bf = [0; b(3)/b(1)];
123 Cf = [0 1];
124 Df = 0;
125
126 [phif gammaf] = zohe(Af,Bf,T);
127
128 %Addtional Dynamics Hold Parameter
129 int_inhibit_Bat = 0.95;
130
131 % Task Sample Rates
132 ts1 = T;                        % ts1 sample time [sec] main control
133 ts3 = 0.1;                      % ts3 sample time [sec] battery voltage
134
135 time_start = 1000;              % gyro calibration time
136
137 % Low Pass Filter Coefficients
138 a_b = exp(-10*(2*pi)*ts1);          % average battery value
139 a_gc = exp(-10*(2*pi)*ts1);         % initial calibrate gyro offset
140 a_gd = exp(-0.04*(2*pi)*ts1);       % Drifting gyro offset;
141
142 % User Setting Values
143 sound_freq = 440;                   % sound frequency [Hz]
144
145
146 %% Analyis Calculate Stability Margins
147
148 %TFBG
149 L1m = [L11, L21];
150 RFBDDelta = msm(phid,gammad,L1m,T);
151 RFBDUGM = 20*log10(1+RFBDDelta);
152 RFBDLGM = 20*log10(1-RFBDDelta);
153 RFBDPM = 2*180/pi*asin(RFBDDelta/2);
154
155 %FBG
156 L2m = [L12, L22];
157 FBDDelta = msm(phid,gammad,L2m,T);
158 FBDUGM = 20*log10(1+FBDDelta);
159 FBDLGM = 20*log10(1-FBDDelta);
160 FBDPM = 2*180/pi*asin(FBDDelta/2);
161
162 %Place
163 L3m = [L13, L23];
164 PlaceDelta = msm(phid,gammad,L3m,T);
165 PlaceUGM = 20*log10(1+PlaceDelta);
166 PlaceLGM = 20*log10(1-PlaceDelta);
167 PlacePM = 2*180/pi*asin(PlaceDelta/2);
```

BIBLIOGRAPHY

Burl, J. B., *Linear Optimal Control*. Reading, Massachusetts, United States of America: Addison-Wesley, 1998.

Chikamasa, T., "Nxt gamepad," January 2009, unpublished. [Online]. Available: http://lejos-osek.sourceforge.net/nxtgamepad.htm

Chikamasa, T., "Embedded coder robot nxt modeling tips," April 2010, unpublished. [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/13399-embedded-coder-robot-nxt-demo?controller=file_infos&download=true

Jones, J. P., McArthur, C., and Young, T., *VU-LEGO Real Time Target: User's Guide*, 1st ed., Center for Nonlinear Dynamics and Control, Villanova University, Villanova PA 19085, United States of America, February 2011.

Jung, S. and Kim, S. S., "Control experiment of a wheel-driven mobile inverted pendulum using neural network," *IEEE Transactions On Control Systems Technology*, vol. 16, pp. 297–303, March 2008.

Kautsky, J., Nichols, N., and Dooren, P. V., "Robust pole assignment in linear state feedback," *International Journal of Control*, vol. 41, pp. 1129–1155, 1985.

Loan, C. F. V., "Computing integrals involving the matrix exponential," *IEEE Transactions on Automatic Control*, vol. AC-23, pp. 395–404, June 1978.

Nawawi, S., Ahmad, M., Osman, J., Husain, A., and Abdollah, M., "Controller design for two-wheels inverted pendulum mobile robot using PISMC," *4th Student Conference on Research and Development*, pp. 194 – 199, June 2006.

Nilulescu, M., "Controlling a mobile robot along planned trajectories," *CEAI*, vol. 7, no. 20, pp. 18–24, 2005.

Sima, V., Tits, A. L., and Yang, Y., "Computational experience with robust pole assignment algorithms," *Conference on Computer Aided Control Systems Design*, vol. WeA01.6, pp. 36–41, October 2006.

Vaccaro, R. J., *Digital Control A State-Space Approch*. New York, New York, United States of America: McGraw-Hill, Inc, 1995.

Vaccaro, R. J., Department of Electrical, Computer, and Biomedical Engineering, University of Rhode Island, Kingston RI 02881, United States of America, October 2011, personal communication.

Yamamoto, Y., "Nxtway-gs model-based design - control of self-balancing two-wheeled robot built with lego mindstorms nxt -," May 2009, unpublished. [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/19147-nxtway-gs-self-balancing-two-wheeled-robot-controller-design?controller=file_infos&download=true