DESIGN, MODELING AND CONTROL OF A TWO-WHEEL BALANCING ROBOT DRIVEN BY BLDC MOTORS

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Mechanical Engineering

by

Charles T. Refvem

December 2019

COMMITTEE MEMBERSHIP

TITLE:	Design, Modeling and Control of a Two-wheel Balancing Robot Driven by BLDC Motors
AUTHOR:	Charles T. Refvem
DATE SUBMITTED:	December 2019
COMMITTEE CHAIR:	William R. Murray, Ph.D. Professor of Mechanical Engineering
COMMITTEE MEMBER:	Charles Birdsong, Ph.D. Professor of Mechanical Engineering
COMMITTEE MEMBER:	John Ridgely, Ph.D. Professor of Mechanical Engineering

ABSTRACT

Design, Modeling and Control of a Two-wheel Balancing Robot Driven by BLDC Motors

Charles T. Refvem

The focus of this document is on the design, modeling, and control of a self-balancing two wheel robot, hereafter referred to as the balance bot, driven by independent brushless DC (BLDC) motors. The balance bot frame is composed of stacked layers allowing a lightweight, modular, and rigid mechanical design. The robot is actuated by a pair of brushless DC motors equipped with Hall effect sensors and encoders allowing determination of the angle and angular velocity of each wheel. Absolute orientation measurement is accomplished using a full 9-axis IMU consisting of a 3-axis gyroscope, a 3-axis accelerometer, and a 3-axis magnetometer.

The control algorithm is designed to minimize deviations from a set point specified by an external radio remote control, which allows the remote operator to steer and drive the bot wirelessly while it remains balanced. Multiple dynamic models are proposed in this analysis, and the selected model is used to develop a linear-quadratic regulator based state-feedback controller to perform reference tracking. Controller tracking performance is improved by incorporating a pre-filter stage between the setpoint command from the remote control and the state-feedback controller.

Modeling of the actuator dynamics is considered briefly and is discussed in relation to the control algorithm used to balance the robot. Electrical and software design implementations are also presented with a focus on effective implementation of the proposed control algorithms.

Simulated and physical testing results show that the proposed balance bot and controller design are not only feasible but effective as a means of achieving robust performance under dynamic tracking profiles provided by the remote control.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF LISTINGS

LIST OF SYMBOLS

m_w	Mass of wheel.
m_b	Mass of body.
r_w	Radius of wheel.
r_b	Distance from center of wheel to c.g. of body.
w	Distance between contact points of left and right wheels.
$\mathbb{I}_w,\ I_w,\ I_{w,xx},\ I_{w,yy},\ I_{w,zz}$	Centroidal moment of inertia for wheel.
$\mathbb{I}_b,\ I_b,\ I_{b,xx},\ I_{b,yy},\ I_{b,zz}$	Centroidal moment of inertia for body.
g	Acceleration due to gravity.
x	Horizontal displacement of balance bot.
θ	Pitch angle: angular displacement of body about axle.
ψ	Yaw angle: angular displacement of body about vertical axis.
$\dot{x}$	Horizontal velocity of balance bot.
$\dot{\theta}$	Pitch velocity: angular velocity of body about axle.
$\dot{\psi}$	Yaw velocity: angular velocity of body about vertical axis.
T	Total kinetic energy in the system.
V	Total potential energy in the system.
L	Lagrangian, defined as $T - V$.
q_i	Generalized coordinate associated with the ith degree of freedom.
Q_i	Generalized, non-conservative, force corresponding to the ith degree of freedom.
$\boldsymbol{r}$	Raw reference signal from RC remote.
$\boldsymbol{x}$	State vector.
$\boldsymbol{x}_{des}$	Desired state vector (before prefiltering).
$\boldsymbol{x}_{ref}$	Reference state vector (after prefiltering).
$\hat{\boldsymbol{x}}$	Transformed state vector.
$\hat{\boldsymbol{x}}_{des}$	Transformed desired state vector (before prefiltering).
$\hat{\boldsymbol{x}}_{ref}$	Transformed reference state vector (after prefiltering).

$\boldsymbol{u}$	Model input vector; composed of $\begin{bmatrix} T_m \end{bmatrix}$ for the two degree of freedom model and composed of $\begin{bmatrix} T_{m,r} & T_{m,l} \end{bmatrix}^\top$ for the three degree of freedom model
Q	LQR tuning parameter; associates cost with states.
R	LQR Tuning Parameter; associates cost with inputs.
K	State-feedback gain matrix.
$\hat{K}$	Transformed state-feedback gain matrix.

Chapter 1

AN INTRODUCTION TO BALANCING ROBOTS

Self-balancing robots have existed for years. Most notably the Segway has been prominent in media over the last two decades. The principle is simple: a robot is equipped with one or more wheels allowing it drive forward and backward to compensate as the robot begins to fall. While practical implementations are limited, balance bots are pertinent in the field of control theory, because they are naturally unstable systems. In other words, without a closed-loop feedback controller the balance bot is bound to fall over if given even a slight perturbation from equilibrium. Only through proper controller design can a stable system be achieved. The same theory used in the development of a balance bot can be applied to any system that is naturally unstable, such as a rocket or a spacecraft.

In recent years, more attention has been focused on balancing robots as computers have become more capable and actuators have become more powerful. Several remarkable examples of the recent attention to balancing robots can be found by looking at the line of products produced by Boston Dynamics[1]. Nearly all of their designs require stabilization of otherwise naturally unstable systems. The robot Handle, produced by Boston Dynamics and shown in Figure 1.1, is an excellent example of a wheeled balancing robot. Handle is a package handling robot designed to agilely maneuver in a warehouse moving packages between pallets.

There are many other smaller scale but similar products available, such as the Hoverboard[2], the Sphero[3], and the Onewheel[4]. The Hoverboard and Onewheel even incorporate the human element as part of the control loop.

[1]Boston Dynamics is a robotics company started in 1992 that focuses mainly on walking bipedal and quadrupedal robots. They are most famous for their quadrupedal robot BigDog developed for the US military.

[2]The Hoverboard is a two wheeled balancing scooter that the rider stands on in an upright, forward-facing, position. The rider can then apply forward or backward pressure with his or her feet to drive and steer the scooter.

[3]Sphero is a small spherical robot that moves by rolling its outer shell. Owners can steer and drive the robot wirelessly.

[4]The Onewheel is a single wheel scooter that the rider stands on similar to a skateboard. The rider can lean forward or backward as well as left and right to drive and steer the scooter.

Figure 1.1: The robot Handle, created by Boston Dynamics.

1.1 Prior Work

There has been considerable prior work done in the field of balancing robots, both commercially and academically focused. Since 2018 alone, several thousand scholarly articles regarding two-wheeled balancing robots have been published. While the field is rather saturated, the project remains popular among students studying control theory due to the appealing small scale and considerable depth of study. This section will highlight several notable works over the last two decades.

A researcher at The University of Western Australia School of Mechanical Engineering completed a master's thesis titled *Balancing a Two-Wheeled Autonomous Robot* [14] in 2003. The author proposes a simplified linear model of the balance bot considering only forward motion and tilting motion. Several different control algorithms are proposed, including state feedback regulator design using both LQR and pole placement as well as a tracking controller design using a PID controller.

The paper additionally discusses implementation of a Kalman filter to provide usable data from sensor measurements. The photograph in Figure 1.2 shows the balance bot used in [14].

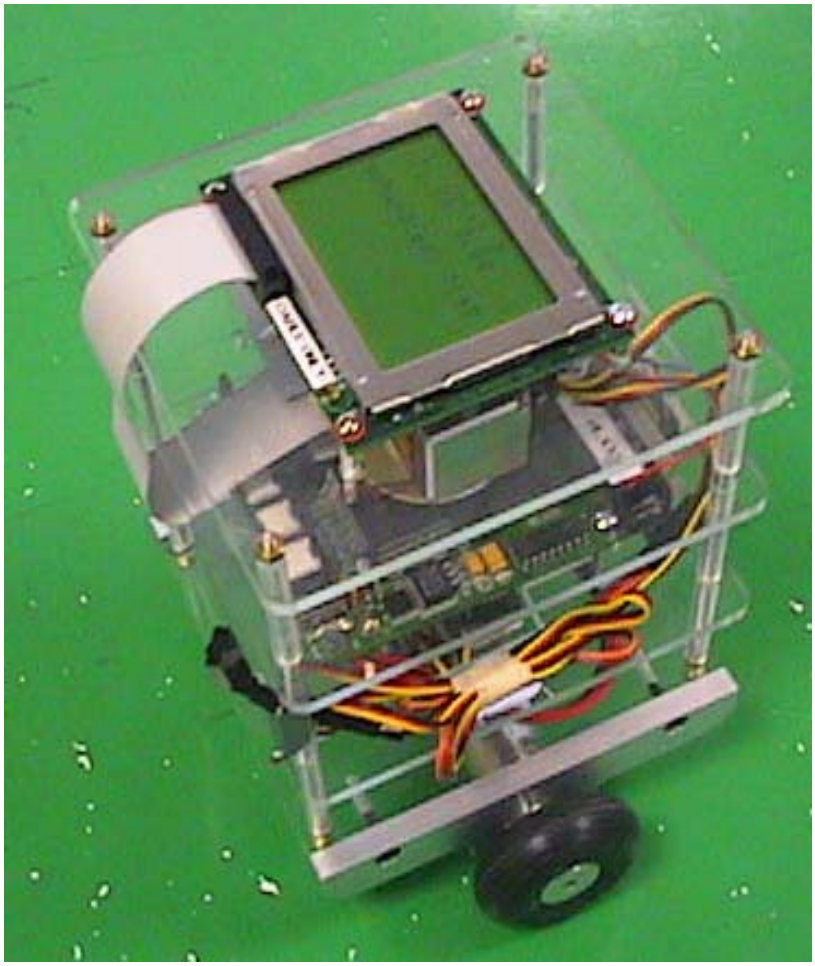


Figure 1.2: A two-wheel balancing robot designed at The University of Western Australia School of Mechanical Engineering[14].

A paper presented at the the 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems proposes a control algorithm using Backstepping Sliding-Mode Control and Fuzzy Basis Function Networks[7]. Although no physical implementation was created, simulated testing shows that the methods proposed in [7] can not only stabilize the balance bot, but also track trajectories in an X-Y plane. The implementation of Sliding-Mode Control reduces the need to have an accurate set of parameters describing the system, thereby offering more robust control.

Researchers at the Department of Electrical and Computer Engineering, Sri Lanka Institute of Information Technology published a paper surveying implementation techniques for two-wheel balancing robots in 2019[11]. The work compares multiple controller and filter designs commonly used in

balancing robots. The testing results found in this effort suggest that a PD controller is ideal for developing stability with minimal oscillation. Figure 1.3 shows the balance bot used for validation in [11].

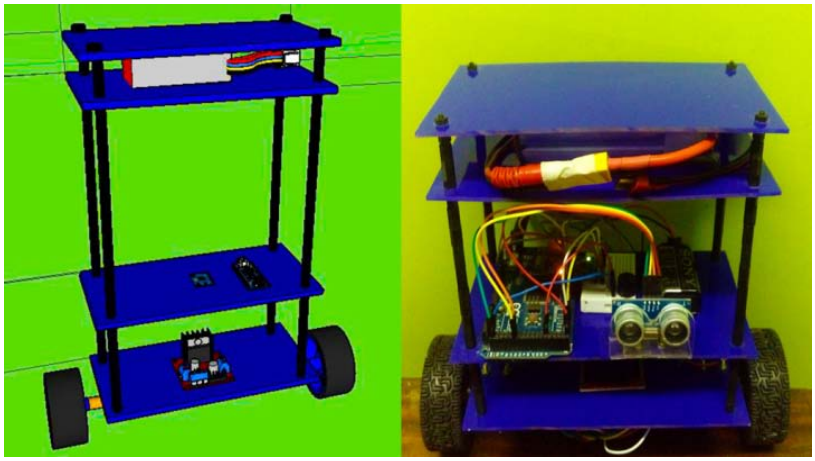


Figure 1.3: A two-wheel balancing robot designed at The Department of Electrical and Computer Engineering, Sri Lanka Institute of Information Technology [11].

Chapter 2

DYNAMIC MODELING

2.1 Modeling Goals

This chapter covers the dynamic model of a self-balancing robot subject to a set of assumptions regarding the physical dynamics. The balance bot is reduced to a set of rigid bodies: the wheels and the body of the robot. The drawings in Figures 2.1 and 2.2 depict schematic representations of the balance bot.

2.1.1 Initial Scope

The initial scope is to model the system assuming the balance bot has only two degrees of freedom, or DOF, as indicated in Figure 2.1. Modeling the system with two DOF provides enough freedom to balance the bot and to drive the bot forward and backward, but without the ability to turn. For this simplified model, there are four states representing the position and velocity for each degree of freedom. With these four states, the dynamics of the actuators are ignored; however, additional states representing the actuator dynamics could be considered allowing the dynamic behavior of the motors to be included in the system model. Actuator dynamics are considered in greater detail in Section 7.5.

2.1.2 Revised Scope

The model is extended to include a third DOF: rotation about a vertical axis. This additional DOF is added to allow the robot to steer while driving and balancing. This extension adds two additional states to the system: position and velocity for the vertical rotary axis. The actuator models could be included in this augmented system as well but are not included in the following analysis.

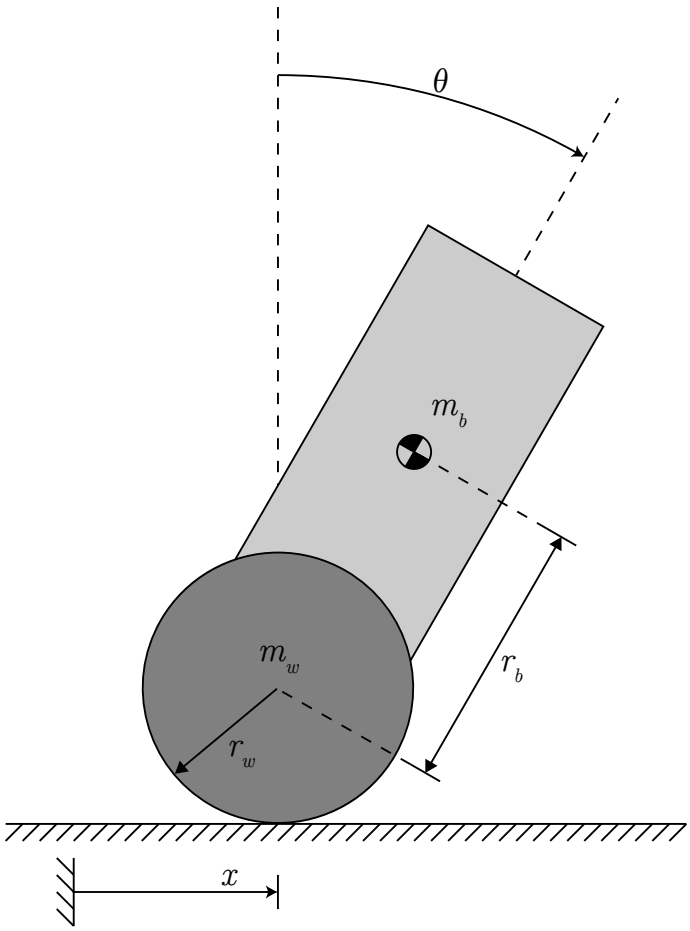


Figure 2.1: Schematic representation of the 2-DOF balance bot model coordinate definitions.

2.2 Coordinate System and Parameter Definitions

2.2.1 Two Degree-of-Freedom Model

The simplified, 2-DOF model of the balance bot considers two rigid bodies: the body and two wheels connected by an imaginary[1] axle. The body, with mass m_b is at an angle θ from the vertical axis; the angle θ is also referred to as the pitch angle. The wheels are lumped together with mass $2m_w$ and roll on what is assumed to be a flat horizontal surface. The entire system is displaced a distance x, measured to the center of the imaginary axle connecting the two wheels.

[1]The 2-DOF model assumes that both wheels are connected by a rigid axle and therefore spin with the same angular velocity. Physically, the two wheels are free to spin independently; however, if both motors are always driven with the same duty-cycle, and the wheels remain in contact with the ground, they will remain nearly synchronized.

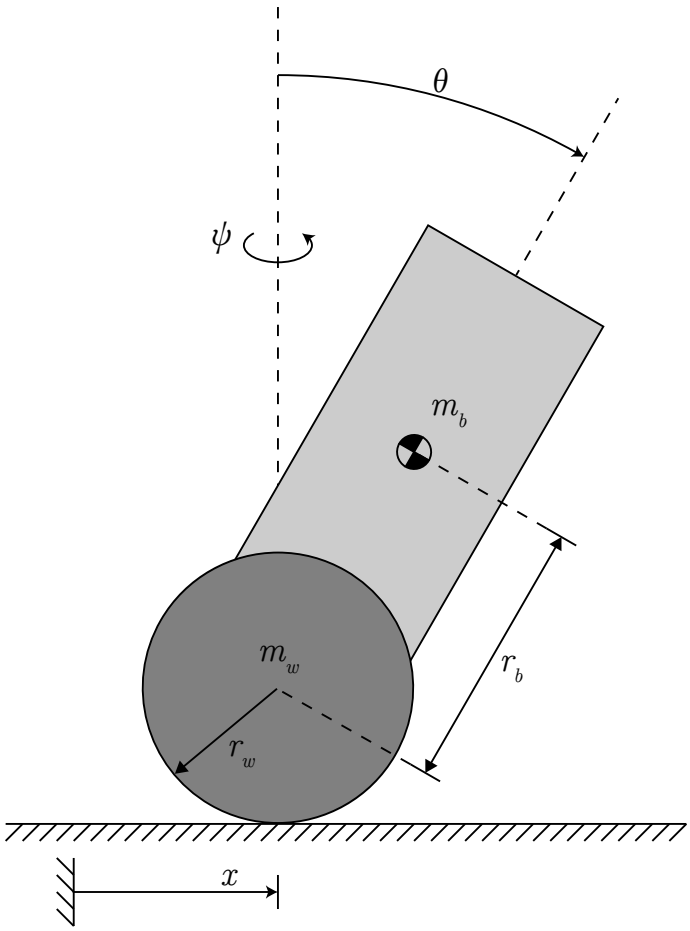


Figure 2.2: Schematic representation of the 3-DOF balance bot model coordinate definitions.

2.2.2 Three Degree-of-Freedom Model

The 3-DOF model of the balance bot considers three rigid bodies: the body and the two wheels considered independently. The body has mass m_b and is at a pitch angle θ from the vertical axis and has rotated by an angle ψ about the vertical axis; the angle ψ is also referred to as the *yaw* angle. The wheels each have mass m_w and roll on a flat horizontal surface. The entire system is displaced a distance x measured to the center of the wheels.

2.3 System Modeling using Energy Methods

The balance bot models considered here have either two or three DOF. Depending on the selected model, 2-DOF or 3-DOF, two or three second-order differential equations are necessary to describe the dynamics of the entire system[2].

Using methods from classical mechanics, such as Newton-Euler methods, would lead to the correct set of equations, but would introduce excess variables and equations in the process; these excess equations would result from the pin forces and reaction forces that show up when summing moments and forces for each body. These extra terms can be eliminated through substitution. Instead, other methods are used in this analysis that do not introduce extra terms in the first place.

Energy methods, such as used in Lagrangian mechanics, provide a solution without consideration of the pin and reaction forces. Lagrange's equation, shown in Equation 2.1, is often a more direct method for finding the system of ODEs constituting the equations of motion.

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = Q_i \qquad i = 1, 2, \ldots \tag{2.1}$$

Where L is a property called the Lagrangian, q_i is a generalized coordinate describing one of the degrees of freedom in the system, and Q_i is a generalized non-conservative force acting on the generalized coordinate q_i. The set of generalized coordinates are the parameters necessary to define the position and orientation of the system and constrain each degree of freedom.

For the 2-DOF model, there are two generalized coordinates: $q_1 = x$, the horizontal displacement, and $q_2 = \theta$, the pitch angle.

For the 3-DOF model, there are three generalized coordinates: $q_1 = x$, the horizontal displacement, $q_2 = \theta$, the pitch angle, and $q_3 = \psi$, the yaw angle.

The generalized forces represent the non-conservative forces that can do work on the system due to displacement along each generalized coordinate. For the balance bot, the generalized forces Q_1, Q_2, and Q_3 come solely from the torque applied by the motors. The proposed model neglects rolling resistance and bearing friction; if these passive dissipative forces were to be included as part of the

[2]Two or three second-order differential equations are necessary if the actuator dynamics are ignored. Additional differential equations are required if the motor modeling is to be incorporated into the complete system model.

model, terms representing the friction forces and rolling resistance would appear in Q_1, Q_2, and Q_3 as well.

A systematic way to validate the selection of generalized coordinates is to conceptually freeze all but one of the selected coordinate axes in succession, while considering the range of motion of the remaining unfrozen axis. After considering each axis one by one, if each remaining axis is able to change freely while the others are frozen, then the set of generalized coordinates is valid[3].

For the 2-DOF model, if the wheels of the balance bot were frozen to the ground, then the horizontal displacement, x, would be constrained, but the pitch angle, θ, would be free to change; if the angle of the body were frozen, then θ would be constrained, but x would free to change. Thus, each axis is able to move independently of each other axis. Similar conclusions can be made for the 3-DOF model.

The parameter L is a property of the system called the Lagrangian, which describes the difference between kinetic energy, T, and potential energy, V.

$$L = T - V \tag{2.2}$$

The integral of L with respect to time is called the action. According to the principle of least action, all physical systems must minimize action over any period of time[9]. Lagrange's equation, shown above as Equation 2.1, defines a set of differential equations describing how motion about each degree of freedom can change with respect to time such that the principle of least action is satisfied. Lagrange's equation can be derived using the calculus of variations[9].

2.4 Two Degree-of-Freedom Model

This section covers the derivation of the equations of motion for the 2-DOF model of the balance bot using the Lagrangian and Lagrange's equation.

[3]This validation comes from the idea of holonomic constraints. For a valid set of generalized coordinates the system constraints must be holonomic.

2.4.1 Potential Energy

The potential energy in the system is purely due to the gravitational potential energy of the body and is defined relative to a datum at the center of the wheel. The potential energy of the wheels could be considered as well, but the model assumes that the wheels will never change elevation, so the potential energy of the wheels always remains constant.

$$V = m_b r_b \cos\theta \, g \tag{2.3}$$

2.4.2 Kinetic Energy

The kinetic energy in the system is due to the linear and angular velocities of the body and both wheels. The linear and angular velocities must be expressed in terms of the coordinate systems that will be used in the Lagrangian so that the kinetic energy T is defined with respect to the generalized coordinates $q_1 \dots q_n$.

$$T = \frac{1}{2} m_x \dot{x}^2 + m_b r_b \cos\theta \, \dot{x}\dot{\theta} + \frac{1}{2} I_\theta \dot{\theta}^2 \tag{2.4}$$

Where m_x is a constant describing the effective composite mass for motion along the x-axis and I_θ is a constant describing the effective composite inertia for motion about the pitch axis. A full derivation of Equation 2.4 can be found in Appendix A.

$$m_x = m_b + 2m_w + 2\frac{I_{w,yy}}{r_w^2} \quad \text{and} \quad I_\theta = m_b r_b^2 + I_{b,yy}$$

2.4.3 Lagrangian

In this section the Lagrangian is used with Lagrange's equation to develop the system of ODEs describing the equations of motion. The Lagrangian is defined as the difference between kinetic and potential energy.

$$L = T - V \tag{2.5}$$

$$L = \frac{1}{2} m_x \dot{x}^2 + m_b r_b \cos\theta \, \dot{x}\dot{\theta} + \frac{1}{2} I_\theta \dot{\theta}^2 - m_b r_b \cos\theta \, g \tag{2.6}$$

Plugging L into Lagrange's equation results in a system of two second-order differential equations.

$$m_x\ddot{x} + m_b r_b \cos\theta\,\ddot{\theta} = -2\frac{1}{r_w}T_m + m_b r_b \sin\theta\,\dot{\theta}^2 \tag{2.7}$$

$$m_b r_b \cos\theta\,\ddot{x} + I_\theta\ddot{\theta} = 2T_m + m_b r_b \sin\theta\,g \tag{2.8}$$

A derivation for Equations 2.7 and 2.8 can also be found in Appendix A. These two second-order differential equations are coupled; that is, the highest derivatives, $\ddot{x}$ and $\ddot{\theta}$, appear in both equations. The two coupled equations can be described together in a single matrix equation, which can then be decoupled using matrix inversion techniques.

$$\begin{bmatrix} m_x & m_b r_b \cos\theta \\ m_b r_b \cos\theta & I_\theta \end{bmatrix} \begin{bmatrix} \ddot{x} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} -2\frac{1}{r_w}T_m + m_b r_b \sin\theta\,\dot{\theta}^2 \\ 2T_m + m_b r_b \sin\theta\,g \end{bmatrix} \tag{2.9}$$

2.5 Three Degree-of-Freedom Model

This section repeats the analysis determining the equations of motion, but for the 3-DOF model of the balance bot.

2.5.1 Potential Energy

The potential energy in the system does not change with the additional degree of freedom.

$$V = m_b r_b \cos\theta\,g \tag{2.10}$$

2.5.2 Kinetic Energy

The kinetic energy in the system changes considerably with the addition of the third degree of freedom and becomes significantly more complicated. As above, the linear and angular velocities must be determined in terms of the generalized coordinates before the kinetic energy can be expressed in terms of these coordinates.

Defining m_x as the effective mass for the x-axis, I_θ as the effective inertia for the pitch axis, and I_ψ as the effective inertia for the yaw axis,

$$T = \frac{1}{2}m_x\dot{x}^2 + m_b r_b \cos\theta\,\dot{x}\dot{\theta} + \frac{1}{2}I_\theta\dot{\theta}^2 + \frac{1}{2}\left(m_b r_b^2 \sin^2\theta + I_{b,xx}\sin^2\theta + I_{b,zz}\cos^2\theta + I_\psi\right)\dot{\psi}^2. \quad (2.11)$$

Where,

$$\begin{aligned} m_x &= m_b + 2m_w + 2\frac{I_{w,yy}}{r_w^2}, \\ I_\theta &= m_b r_b^2 + I_{b,yy}, \\ I_\psi &= m_w\frac{w^2}{2} + \frac{I_{w,yy}}{r_w^2}\frac{w^2}{2} + 2I_{w,zz}. \end{aligned}$$

A full derivation of Equation 2.11 can be found in Appendix A.

2.5.3 Lagrangian

The Lagrangian is used with Lagrange's equation to develop the system of ODEs describing the equations of motion.

$$m_x\ddot{x} + m_b r_b\cos\theta\,\ddot{\theta} = -\frac{1}{r_w}T_{m,r} + \frac{1}{r_w}T_{m,l} + m_b r_b\sin\theta\,\dot{\theta}^2 \quad (2.12)$$

$$\begin{aligned} m_b r_b\cos\theta\,\ddot{x} + I_\theta\ddot{\theta} &= T_{m,r} - T_{m,l} + m_b r_b\sin\theta\,g \\ &\quad + \left(m_b r_b^2 + I_{b,xx} - I_{b,zz}\right)\sin\theta\cos\theta\,\dot{\psi}^2 \end{aligned} \quad (2.13)$$

$$\begin{aligned} \left(m_b r_b^2\sin^2\theta + I_{b,xx}\sin^2\theta + I_{b,zz}\cos^2\theta + I_\psi\right)\ddot{\psi} &= -\frac{w}{2r_w}T_{m,r} - \frac{w}{2r_w}T_{m,l} \\ &\quad - 2\left(m_b r_b^2 + I_{b,xx} - I_{b,zz}\right)\sin\theta\cos\theta\,\dot{\theta}\dot{\psi} \end{aligned} \quad (2.14)$$

A derivation for Equations 2.12, 2.13, and 2.14 can also be found in Appendix A. These three equations are coupled together and can be combined into one matrix equation.

$$\begin{bmatrix} m_x & m_b r_b \cos\theta & 0 \\ m_b r_b \cos\theta & I_\theta & 0 \\ 0 & 0 & m_b r_b^2 \sin^2\theta + I_{b,xx} \sin^2\theta + I_{b,zz} \cos^2\theta + I_\psi \end{bmatrix} \begin{bmatrix} \ddot{x} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix} = \begin{bmatrix} f_1(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u}) \\ f_2(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u}) \\ f_3(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u}) \end{bmatrix} \tag{2.15}$$

Where f_1, f_2, and f_3 are functions of lower-order terms and input torques.

$$f_1 = -\frac{1}{r_w} T_{m,r} + \frac{1}{r_w} T_{m,l} + m_b r_b \sin\theta\, \dot{\theta}^2 \tag{2.16}$$

$$f_2 = T_{m,r} - T_{m,l} + m_b r_b \sin\theta\, g + \left(m_b r_b^2 + I_{b,xx} - I_{b,zz}\right) \sin\theta \cos\theta\, \dot{\psi}^2 \tag{2.17}$$

$$f_3 = -\frac{w}{2r_w} T_{m,r} - \frac{w}{2r_w} T_{m,l} - 2\left(m_b r_b^2 + I_{b,xx} - I_{b,zz}\right) \sin\theta \cos\theta\, \dot{\theta}\dot{\psi} \tag{2.18}$$

2.6 Linearized Equations of Motion

The equations derived in the previous sections for the two and three DOF models are in the inconvenient form $M(\boldsymbol{q})\ddot{\boldsymbol{q}} = \boldsymbol{f}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u})$ which naturally results from Lagrange's equation. Here the vector parameter $\boldsymbol{u}$ defines the external inputs to the system. The equations can be decoupled by multiplying each side of the equation by $M(\boldsymbol{q})^{-1}$. Any system of ODEs determined by Lagrange's equation can be decoupled in this manner because the matrix $M(\boldsymbol{q})$ is, in general, invertible[4] for systems with non-zero mass associated with each axis.

In order to design a controller for the system, it is desired to rearrange the equations to the form $\dot{\boldsymbol{x}} = \boldsymbol{a}(\boldsymbol{x}) + b(\boldsymbol{x})\boldsymbol{u}$ which is a set of first order differential equations that are nonlinear with respect to the state variables but are linear with respect to the inputs[5]. This decoupling can be accomplished with the following substitutions if the function $\boldsymbol{f}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u})$ is linear with respect to $\boldsymbol{u}$, which is the case for both balance bot models.

[4]It is straightforward to prove that $M(\boldsymbol{q})$ is invertible by considering an alternate representation of the kinetic energy T. Consider the quadratic form $T = \frac{1}{2}\dot{\boldsymbol{q}}^\top M(\boldsymbol{q})\dot{\boldsymbol{q}}$ relating the time rate-of-change of the generalized coordinates to the kinetic energy in the system. By definition, kinetic energy is always greater than or equal to zero; further, the kinetic energy in the system can only be zero when there is no motion. That is, $T = 0$ if and only if $\dot{\boldsymbol{q}} = \boldsymbol{0}$. Therefore the matrix $M(\boldsymbol{q})$ must be positive definite. Positive definite matrices are invertible, therefore the mass matrix must also be invertible.

[5]Representing the equations of motion in the form $\dot{\boldsymbol{x}} = \boldsymbol{a}(\boldsymbol{x}) + b(\boldsymbol{x})\boldsymbol{u}$ sets the stage for linearization in order to develop the standard linear time-invariant state-space representation $\dot{\boldsymbol{x}} = A\boldsymbol{x} + B\boldsymbol{u}$.

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{q} \\ \dot{\boldsymbol{q}} \end{bmatrix} \tag{2.19}$$

$$\boldsymbol{a}(\boldsymbol{x}) = \begin{bmatrix} \dot{\boldsymbol{q}}(\boldsymbol{x}) \\ M(\boldsymbol{q}(\boldsymbol{x}))^{-1}\boldsymbol{f}(\boldsymbol{q}(\boldsymbol{x}), \dot{\boldsymbol{q}}(\boldsymbol{x}), \boldsymbol{0}) \end{bmatrix} \tag{2.20}$$

$$b(\boldsymbol{x}) = \begin{bmatrix} 0 \\ M(\boldsymbol{q}(\boldsymbol{x}))^{-1}\left(\frac{\partial \boldsymbol{f}(\boldsymbol{q}(\boldsymbol{x}), \dot{\boldsymbol{q}}(\boldsymbol{x}), \boldsymbol{u})}{\partial \boldsymbol{u}}\right) \end{bmatrix} \tag{2.21}$$

With the equations of motion decoupled, it is straightforward to linearize the system about an operating point using the Jacobian matrix $\frac{\partial \boldsymbol{a}(\boldsymbol{x})}{\partial \boldsymbol{x}}$. In order to develop control laws using linear control theory, a linearized model of the system is required. In operation, the balance bot does not remain fixed at its operating point, but with proper control does remain near enough to the operating point that the linearized model still represents the dynamics of the system.

It is generally required that the selected operating point is also an equilibrium point for the system. The equilibrium point requirement is because the linearization method proposed here relies on a Taylor series expansion, where the Jacobian matrix represents the first-order component of the expansion. In order for the dynamics to rely fundamentally on the first-order component, the zeroth-order components must be negligible, which is only the case when the operating point is very near to an equilibrium point.

$$A = \left.\frac{\partial \boldsymbol{a}(\boldsymbol{x})}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}=\boldsymbol{x}_0} \tag{2.22}$$

$$B = \left. b(\boldsymbol{x})\right|_{\boldsymbol{x}=\boldsymbol{x}_0} \tag{2.23}$$

If $\boldsymbol{x}_0$ is selected as an equilibrium point, the matrices A and B satisfy $\dot{\boldsymbol{x}} = A\boldsymbol{x} + B\boldsymbol{u}$ to $O((\boldsymbol{x} - \boldsymbol{x}_0)^2)$.

These transformations can be excessively tedious to solve symbolically, particularly due to the inverse of the large matrix M. Accordingly, these transformations can be completed numerically using the measured system properties determined from the final design implementation.

Chapter 3

CONTROLLING THE BOT

One of the first things to do in designing a control system is to choose the structure or topology of the controller. The two principal forms of controller are classical controllers and state-space controllers. Classical controllers, such as PID[1] controllers, are designed for single-input, single-output (SISO) systems. However, the balance bot is inherently a multi-input, multi-output (MIMO) system[2], for which controllers are more easily designed using state-space approaches.

The focus of this chapter is state-feedback controller design; however, for completeness, PID control is covered briefly.

3.1 Cascaded PID Control

An effective workaround to control SIMO systems with classical controllers is to split the system model into subsystems, and apply a SISO controller to each subsystem in a cascaded fashion. An example of a simple cascaded controller design for the 2-DOF balance bot model is shown in Figure 3.1. In this controller design, the inner loop is tuned to balance the robot by regulating the pitch angle, θ. The outer loop is tuned to control the displacement of the robot by tracking a position set point using the inner loop as if it were an actuator. The position of the balance bot can be controlled by changing the setpoint of the inner loop such that the bot will try to drive toward the disturbance in order to regain balance.

Special care must be taken when tuning cascaded feedback controllers to guarantee that the inner and outer loops work together as intended. In most cases the inner loop should be tuned more aggressively than the outer loop so that the inner loop dynamics occur faster than the outer loop dynamics[4]. In the case of the balance bot, an aggressive inner loop means that the robot will favor balancing over driving, thus maintaining an upright posture as the outer loop tries to command the inner loop.

[1]The term PID controller refers to a proportional-integral-derivative controller, a very common structure for controlling single-input, single-output systems.

[2]The 2-DOF model is actually a single-input, multi-output system because both motors are driven with the same duty-cycle.

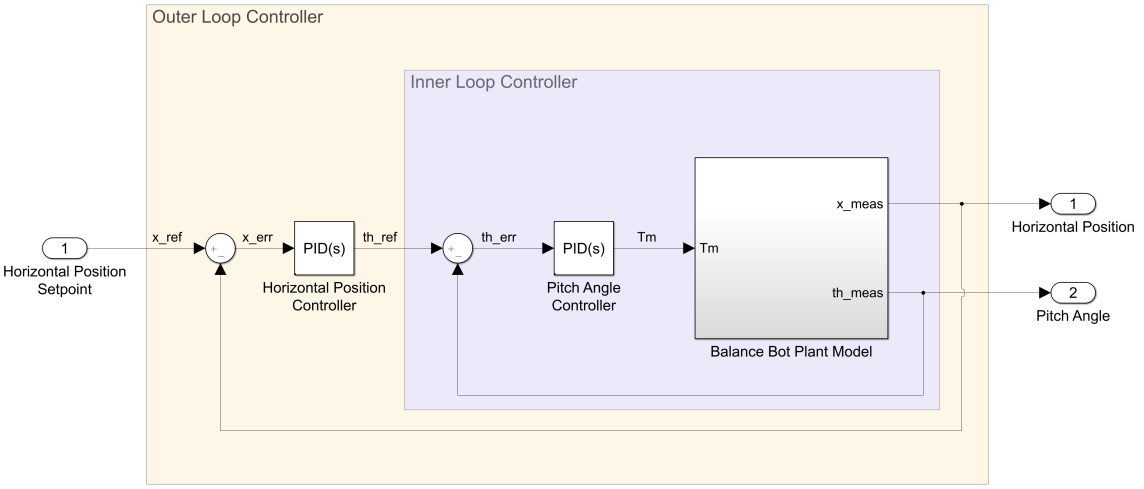


Figure 3.1: Pitch controller and position controller in cascaded configuration.

Consider the previous example of a cascaded control structure for the 2-DOF balance bot model in the case where both the inner loop and outer loop are configured as PD controllers. In this configuration, the feedback parameters represent all system states. If all of the state variables are measured and fed back to the controller, then the cascaded PID is equivalent to a state feedback controller[4].

3.2 State Feedback Control

Controller design should start with a simple approach and only grow in complexity as the need arises. As such, a simple feedback controller is a logical starting place for a MIMO system such as the balance bot. Additionally, linear control methods should be investigated before more advanced nonlinear methods are pursued.

For a system in the linear state-space form,

$$\dot{\boldsymbol{x}} = A\boldsymbol{x} + B\boldsymbol{u}, \tag{3.1}$$

an input can be defined that will modify the dynamics of the closed-loop system to produce the desired performance. Such an input can be defined as

$$\boldsymbol{u} = -K\boldsymbol{x}, \tag{3.2}$$

where K is a feedback gain matrix used to determine the input $\boldsymbol{u}$. If Equation 3.2 is substituted into Equation 3.1, a new homogeneous linear system is created.

$$\dot{\boldsymbol{x}} = A\boldsymbol{x} - BK\boldsymbol{x} \tag{3.3}$$

$$\dot{\boldsymbol{x}} = (A - BK)\boldsymbol{x} \tag{3.4}$$

3.2.1 Pole Placement

The poles of the new system can be selected by choosing the appropriate values for the elements composing K such that the matrix $A - BK$ has desirable time-domain or frequency-domain characteristics. Many methods exist for finding gains that will produced desired pole locations. Some examples may be: Ackerman's formula[6], conversion to canonical controllable form[6], characteristic polynomial matching[6], or by directly solving the eigenvalue problem on the matrix $A - BK$[6].

3.2.1.1 Pole Locations

The locations of the system poles for the linearized model dictate the dynamic response of the system near the operating point. One method of controller design is to select desired pole locations and then to determine the gain required to achieve these pole locations. It is challenging to select pole locations for systems with order higher than two, such as the fourth-order system representing the 2-DOF balance bot. Instead, it may be useful to approximate the system as second-order in an effort to select dominant poles that dictate the driving dynamics of the system.

3.2.1.2 Dominant Pole Approximation

Consider the characteristic equation for a second-order system parameterized in terms of natural frequency, ω_n, and damping ratio, ζ. Compare the parameterized characteristic equation to the characteristic equation for a factored second-order system.

$$(s - \lambda_1)(s - \lambda_2) = 0 \tag{3.5}$$

$$s^2 - (\lambda_1 + \lambda_2)s + \lambda_1\lambda_2 = 0 \tag{3.6}$$

$$s^2 + 2\zeta\omega_n s + \omega_n^2 = 0 \tag{3.7}$$

By equating the coefficients for each order of s in Equations 3.6 and 3.7 and solving for λ_1 and λ_2, the dominant pole locations can be determined as functions of the desired natural frequency, ω_n, and damping ratio, ζ, for the approximated second-order system.

$$\lambda_1 = -\omega_n \left(\zeta - \sqrt{\zeta^2 - 1}\right) \tag{3.8}$$

$$\lambda_2 = -\omega_n \left(\zeta + \sqrt{\zeta^2 - 1}\right) \tag{3.9}$$

There exist many metrics for selecting a natural frequency and a damping ratio such as the time-domain characteristics rise time, t_r, peak time, t_p, settling time, t_s, and maximum overshoot M_P.

$$t_r = \frac{1}{\omega_n \sqrt{1-\zeta^2}} \left(\pi - \arctan\left(\frac{\sqrt{1-\zeta^2}}{\zeta}\right)\right) \tag{3.10}$$

$$t_p = \frac{\pi}{\omega_n \sqrt{1-\zeta^2}} \tag{3.11}$$

$$t_s = \frac{4}{\zeta \omega_n} \quad \text{(within 2\% of final value)} \tag{3.12}$$

$$M_P = \exp\left(-\frac{\zeta\pi}{\sqrt{1-\zeta^2}}\right) \tag{3.13}$$

The remaining pole locations can be made non-dominant by choosing them to be negative and have significantly larger magnitude than the real parts of λ_1 and λ_2. Selecting large values for the non-dominant poles will force the dynamics associated with the non-dominant poles to decay rapidly compared to the dynamics of the dominant poles.

$$\|\lambda_3\| \gg \|\max(Re(\lambda_1), Re(\lambda_2))\| \tag{3.14}$$

$$\|\lambda_4\| \gg \|\max(Re(\lambda_1), Re(\lambda_2))\| \tag{3.15}$$

The feedback gain, K, is dependent on the matrices A and B and the desired pole locations λ_1 through λ_4. The most straight forward method for computing K is to equate the characteristic polynomial of the system to the desired characteristic polynomial. For a system with four states like the 2-DOF balance bot model, this process should result in four equations with four unknowns, K_1 through K_4.

There also exist functions in MATLAB® that perform pole placement such as `place()` and `acker()`. In practice, it is often more convenient to use numerical methods provided by software like MATLAB®

than to perform pole-placement analytically. Numerical methods such as implemented by the `place()` command in MATLAB® are designed to be numerically stable and allow the designer to rapididly iterate while the controller is being tuned.

3.2.2 Linear Quadratic Regulators

Another method for determining the feedback gain K is through design of a linear quadratic regulator (LQR). Consider a quadratic cost function that, when minimized over an interval of time $[0\ t_f]$, will yield the desired system performance.

$$J = \boldsymbol{x}(t_f)^\top F\, \boldsymbol{x}(t_f) + \int_0^{t_f} \left(\boldsymbol{x}^\top Q\boldsymbol{x} + \boldsymbol{u}^\top R\boldsymbol{u} + 2\boldsymbol{x}^\top N\boldsymbol{u}\right) dt \tag{3.16}$$

J is a cost function that can be adjusted by the user to select the desired performance criteria. There are four main parameters contributing to the cost function.

- F penalizes error in the states at the final time t_f.
- Q penalizes error in the states during the transient response.
- R penalizes controller effort, often referred to as fuel consumption.
- N penalizes the product of states and controller input.

In many applications it is useful to consider the "infinite horizon" case; that is, $t_f \to \infty$. Typically the first term $\boldsymbol{x}(t_f)^\top F\boldsymbol{x}(t_f)$ is dropped in the infinite horizon case because the second term $\boldsymbol{x}^\top Q\boldsymbol{x}$ naturally includes penalty for steady state error as $t \to t_f$. The feedback gain K can be determined by solving the continuous-time algebraic Ricatti equation (CARE) for a matrix P, and then using P to determine K. One expression for the CARE is:

$$A^\top P + PA - (PB + N)R^{-1}(B^\top P + N^\top) + Q = 0 \tag{3.17}$$

With:

$$K = R^{-1}(B^\top P + N^\top). \tag{3.18}$$

The derivation for the CARE is not included in this document for conciseness but can be found in [6]. The CARE can be solved analytically, using the Hamiltonian, or in programs like MATLAB® using the command `care(A,B,Q,R,N)`. It is also possible to get the gain K directly from MATLAB® using the command `lqr(A,B,Q,R,N)`.

Chapter 4

MECHANICAL DESIGN

4.1 Design Overview

The balance bot hardware consists of multiple tiers connected together using standoffs. Consider Figure 4.1 which depicts a layout of the balance bot mechanical design with major components labeled. The stacked design allows the separation and number of tiers in the design to be easily adjusted during the design process.

Each tier in the design is made from a printed circuit board, usually referred to as a PCB. Modern manufacturers can produce PCBs with positional tolerances of less than 5 $[mil]$ at very low cost and are typically made from a type of fiberglass called FR4. Fiberglass is considerably more rigid than other viable materials such as polycarbonate or acrylic. The low cost, high level of manufacturability, and large stiffness make PCBs the ideal choice for each tier. Furthermore, the electronics can be mounted directly onto the PCBs, further reducing the overall cost and number of components.

The mechanical hardware is designed to be as small as possible without limiting component availability or introducing problems during manufacturing and assembly. The balance bot measures 200 $[mm]$ in height[1], 180 $[mm]$ in width, and 90 $[mm]$ in depth. The wheel diameter of 90 $[mm]$ is selected to be as large as possible while fitting the scale of the other components. The entire system has a mass of 960 $[g]$. The small scale design reduces the component cost and facilitates convenient storage when the device is not in use.

4.2 Motor Selection

Motor selection for the balance bot design is critical for good performance. There are three major requirements that constrain the available motor choices in addition to several non-critical requirements.

[1]The stated height of 200 $[mm]$ does not include the antenna which extends an additional 300 $[mm]$ from the top of the balance bot.

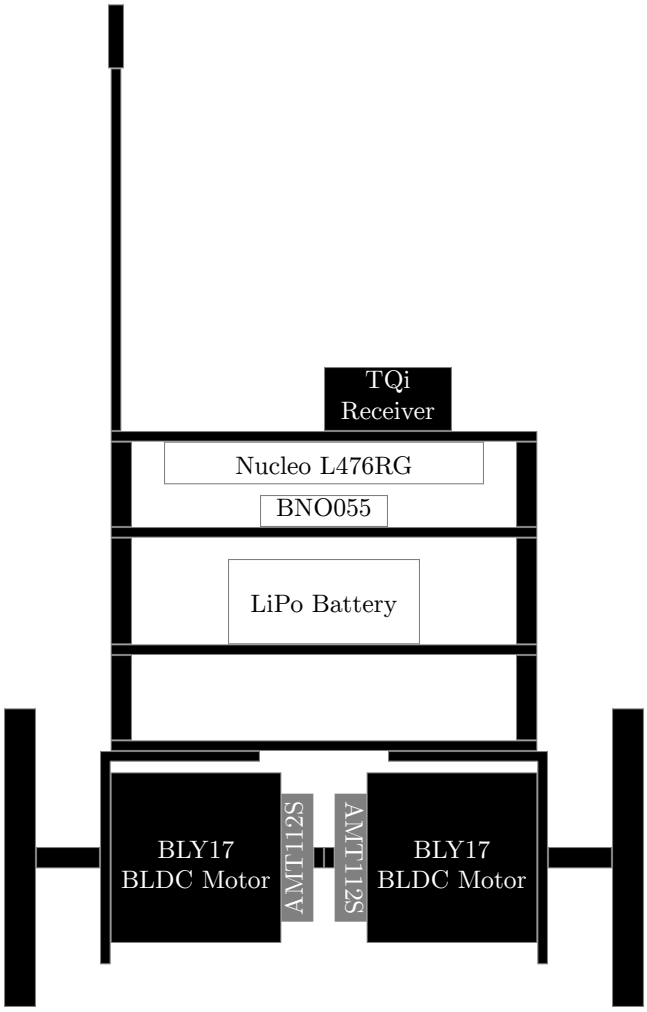


Figure 4.1: Layout of the balance bot mechanical hardware with main components labeled.

- The first critical constraint is the torque requirement for the bot to balance properly. Simulation results suggest that a combined maximum peak torque of 0.4 $[N \cdot m]$ is required from both drive motors acting together; this peak torque requirement suggests that motors with stall torque specifications exceeding 0.2 $[N \cdot m]$ or 200 $[mN \cdot m]$ would be viable choices. Simulation results also indicate that a continuous torque of around 50 $[mN \cdot m]$ would be necessary for the balance bot to perform over an extended period of time without the motor overheating.

- The second critical constraint comes from the need for the motor output torque to change direction repeatedly as the bot attempts to maintain orientation. In order to eliminate play in the mechanical design, it is necessary to select a direct-drive setup so that there is no backlash between the motor and the wheels. In practice, a small amount of backlash could be accounted

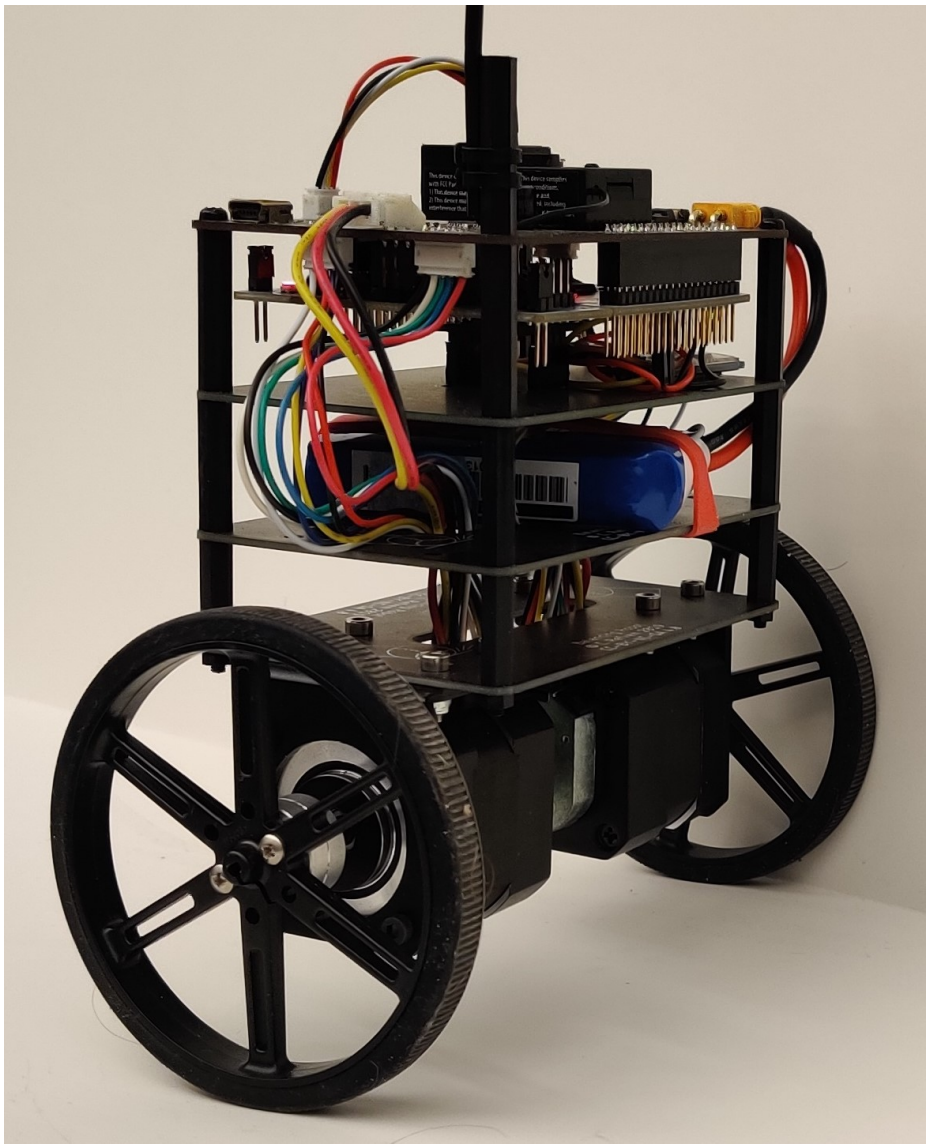


Figure 4.2: Photograph of the completed balance bot hardware; front view shown.

for by the control algorithm; however, to promote increased performance and stability, a direct-drive setup is selected nonetheless. In a larger-scale design it may be infeasible to select motors that perform as desired in direct-drive.

- The final critical constraint is the size and form-factor of each motor. A maximum envelope of $50 \times 50 \times 75$ $[mm]$ per motor is sufficient to keep the design compact and the motors small in comparison to the remaining components.

- The motor must be able to interface with a position sensor such as a set of Hall effect sensors or a quadrature encoder. Position sensors are needed to perform feedback control and, in the

case of brushless DC motors, provide information to commutate the motor properly. To satisfy the position sensor requirement, the motor must be internally equipped with a position sensor or allow external mounting of a position sensor through a secondary output shaft extending from the rear of the motor.

- For this particular application, speed was not a driving constraint because nearly all brushed and brushless DC motors operating in direct drive will have sufficient velocity for the bot to drive as desired. Simulation results show that a speed of several hundred RPM is sufficient to drive the balance bot at reasonable speeds.

- For convenience and safety, a nominal voltage range of $8 - 24$ $[V]$ is selected as an additional requirement. This nominal voltage range allows the usage of batteries with few cells in series, reducing the physical size and cost of the batteries. Furthermore, DC voltages less than ≈ 40 $[V]$ are safe for human interaction and have low risk of electric shock.

Table 4.1 summarizes the motor requirements.

Table 4.1: Motor Requirements.

Stall Torque (Minimum)	250 $[mN \cdot m]$
Rated Torque (Minimum)	50 $[mN \cdot m]$
Gear Reduction	Direct drive
Rated Speed (Minimum)	200 $[RPM]$
Dimensions (Maximum)	$50 \times 50 \times 75$ $[mm]$
Cost Ea. (Maximum)	\$100
Sensor	Hall effect or quad. encoder

From these requirements, several motor choices were considered. Initial research pointed toward brushless DC (BLDC) motors due to the large torque-to-size ratio needed to balance the bot properly. Even so, brushed DC (BDC) motors were considered as well. Table 4.2 shows a list of viable choices that were considered.

The ECi 30 or ECi 40, both available from Maxon, would be excellent choices as well, but come at a much higher cost making them infeasible for this project.

The bottom two motors in the list, manufactured by Turnigy, are both designed for multirotor applications[2]. Accordingly, the motors are out-runners, meaning that the permanent magnets are on the outside of the motor rather than the inside. Out-runners typically have larger torque and

[2]A multirotor or multicopter is any rotorcraft with more than two rotors. Quadcopters are the most common example of multirotors.

Table 4.2: Viable motor choices that meet most or all requirements.

Manufacturer	Model	Stall Torque $[mN \cdot m]$	Rated Torque $[mN \cdot m]$	Speed $[RPM]$	Price	Dimensions $[mm]$
Maxon Motors	DCX26L	384	52.3	4600	\$224.30	$\varnothing 26 \times 57$
Maxon Motors	EC32	355	47.2	9450	\$324.71	$\varnothing 32 \times 60$
Maxon Motors	EC Max 30	458	60.7	8040	\$201.22	$\varnothing 30 \times 64$
Maxon Motors	EC Max 40	497	89.6	6250	\$275.92	$\varnothing 40 \times 58$
Maxon Motors	EC 4Pole 30	1270	68.8	16300	\$622.40	$\varnothing 30 \times 47$
Maxon Motors	ECi 30	811	63.8	7030	\$203.13	$\varnothing 30 \times 42$
Maxon Motors	ECi 40	858	64.6	6520	\$232.87	$\varnothing 40 \times 26$
Maxon Motors	EC 45 Flat	253	54.8	2940	\$80.95	$\varnothing 43.2 \times 16.5$
Anaheim Automation	**BLY171D-24V-1400**	**225**	**67.1**	**1400**	**\$65.00**	$\mathbf{42 \times 42 \times 42}$
Trinamic	QBL4208-41-04-006	467	62.5	4000	\$86.06	$42 \times 42 \times 42$
NMB	36P16A	338	56.0	4355	\$154.48	$\varnothing 36.2 \times 33.6$
Turnigy	Multistar Elite 3508	1626	81.0	6432	\$35.30	$\varnothing 41.2 \times 32$
Turnigy	Multistar 4014	4163	208.0	7680	\$51.09	$\varnothing 45 \times 32$

lower speed due to a longer moment arm within the motor. However, multirotor motors do not come with internal Hall effect sensors making them practical only in sensorless[3] applications.

Additionally, multirotor motors are wound "hot", meaning that they are intentionally designed to draw a lot of current in order to produce a lot of power. Hot windings are ideal for multirotor applications because during flight there is forced convection cooling the motors due to the attached propellers. However, in an application such as the balance bot, there will be no forced cooling and the motors will pull more current than desired. Finally, multirotor motors come with significant cogging torque[4], which would interfere with the balance of the bot.

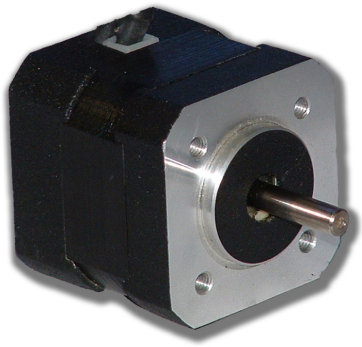


Figure 4.3: BLY17 Brushless DC motor from Anaheim Automation.

[3]BLDC motor commutation is referred to as sensored or sensorless depending on the means of determining rotor orientation. If a Hall effect sensor or an encoder is used to determine rotor orientation then the commutation is sensored. If the back-EMF of the motor is used to determine the rotor orientation then the commutation is sensorless.

[4]Cogging torque is caused by the reluctance in the iron core within the BLDC motor. The "teeth" inside the motor, around which the windings are formed, are attracted to the permanent magnets even when no current is applied to the windings. This attraction causes the rotor to align with the stator in an orientation that minimizes reluctance such that the magnetic flux from the permanent magnets is minimally impeded. High quality motors have cogging reduction measures such as skewed stator laminations to reduce this undesirable effect.

The BLY171D-24V-1400 is a satisfactory choice for the balance bot as it meets all of the minimum requirements and comes at a low cost. Figure 4.3 shows the BLY17 motor selected for use in the balance bot. The BLY17 is not the smallest motor in the list, but it is one of the lowest cost choices. The selected motor also comes with a dual shaft allowing an external encoder to be easily mounted. The BLY17 comes in a variety of different winding configurations; the 1D-24V-1400 suffix specifies a 24 $[V]$ nominal voltage and a 1400 $[RPM]$ rated speed, which is the configuration that provides the most torque per amp. Additionally, the motor comes with internal Hall effect sensors that can be used to measure the orientation of the rotor for electronic commutation[5].

[5]BLDC motors are not truly DC motors. BLDC Motors require electronic commutation due to the lack of mechanical commutation through brushes.

Chapter 5

ELECTRONICS DESIGN

5.1 Component Selection and Circuit Design

This section describes the component selection process for the electronic portion of the design.

5.1.1 Microcontroller and Development Board

The microcontroller selection is constrained based on compatibility with MicroPython and easy interface with the hardware used in the senior level mechatronics course at Cal Poly. The compatibility constraints all but force the design to use the Nucleo L476RG[16] development board produced by ST Microelectronics. The Nucleo is a breakout board designed for development with STM32 microcontrollers. The L476RG variant of the Nucleo has an STM32L476RG[17] microcontroller which runs at 80 $[MHz]$ and includes 64 pins, barely sufficient for this design implementation.

The Nucleo development board facilitates very easy programming and debugging. The MicroPython firmware uses USB to implement a flash storage device on any PC or laptop computer. The flash device makes loading of code as simple as dragging the python files onto the enumerated flash storage device on the PC. The USB interface also emulates a standard serial port, allowing debugging and real-time interaction through a text based serial terminal such as PuTTY. The Nucleo provides power management for the microcontroller and supports hot-swap between an external $7 - 12$ $[V]$ supply and USB power.

The STM32L476RG microcontroller is a valid selection for this project because it provides sufficient clock speed, native floating point support, and enough timers to interface with all of the hardware peripherals in the design. The microcontroller is limited by pin count however, so some additional functionality desired for future revisions will be prohibited without rearranging pin selection or choosing a new microcontroller.

An important consideration for microcontroller selection is the number of timers available. The balance bot design requires seven different multi-channel timer modules: one timer per motor for

PWM, one timer per motor for Hall sensor input capture, one timer per motor for quadrature encoder counting, and one additional timer for input capture to interface with the radio receiver.

An additional consideration is the need for native I^2C communication with the inertial measurement unit (IMU) used for orientation sensing.

The final constraint on microcontroller selection is the need for free ADC pins available to read data from six current sensors, one for each motor phase.

5.1.2 Motor Driver

Preliminary design efforts focused on motor "pre-drivers" which require external MOSFETs to build the three half-bridges necessary to drive each BLDC motor. This configuration is standard for high-power motors, but was eliminated early once the motor current was deemed low enough to select a fully integrated monolithic motor driver IC.

The DRV8313[18], shown in Figure 5.1, is a 2.5 $[A]$ triple half-bridge motor driver IC with integrated gate drivers and fault protection. The DRV8313 also performs in a sufficient range of voltages to accommodate a single power source for the Nucleo and Motors. This motor drive IC allows independent control of each half-bridge with PWM frequencies up to 250 $[KHz]$, allowing any style of commutation. The DRV8313 also comes in a conveniently small package with a relatively large thermal pad on the bottom of the IC allowing it to cool effectively even under full load; the enhanced thermal package eliminates the need for external heat sinking.

5.1.3 Current Sensor

Current sensors are implemented on each phase of each motor allowing independent measurement of bidirectional current through each winding individually. Current sensing allows the possibility of performing torque control on the BLDC motors if actuator dynamics were added to the system model. Even without extending the system model to include actuator dynamics, information about the motor current is useful for diagnostics and fault prevention.

In theory, only two current sensors are required per motor, as the three phase currents must sum to zero. However, the additional current sensor reduces software complexity and provides additional,

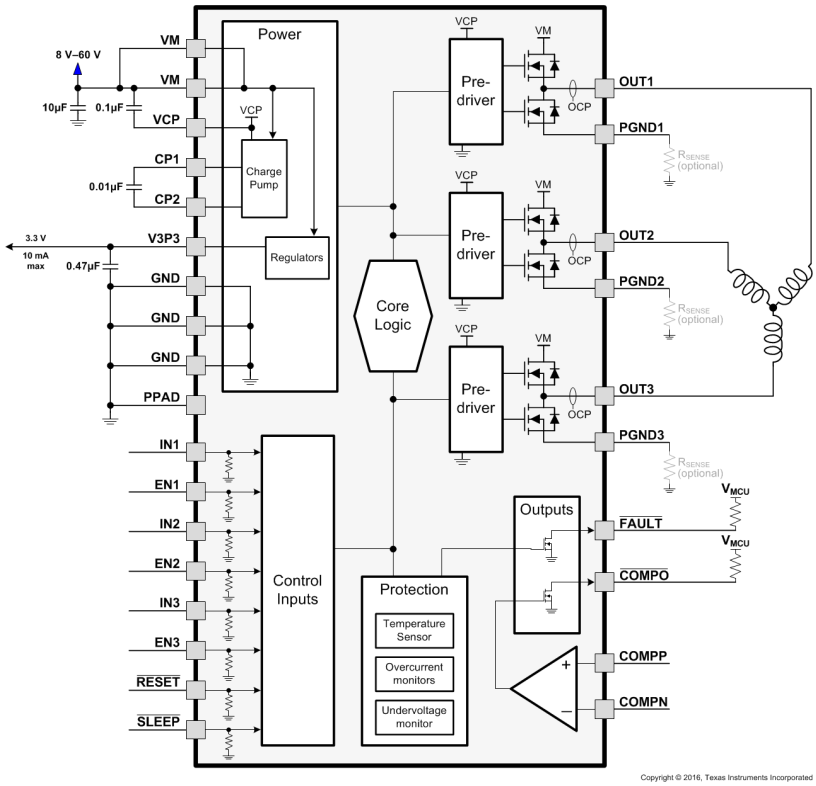


Figure 5.1: Functional block diagram for DRV8313 BLDC motor driver provided by TI[18].

if redundant, data. Redundant data could be useful for improved fault detection in future software revisions.

The ACS70331EESATR-2P5B3[2] is chosen as it provides a satisfactory measurement range of ± 2.5 $[A]$ and has the bandwidth needed to measure the AC waveforms required to commutate the motor at high speeds. The ACS70331, shown in Figure 5.2, is a GMR[1] based current sensor that provides a ratiometric voltage output proportional to the bidirectional current flowing through the sensor. Six sensors are used, one directly in series with each phase of each motor. The ACS70331 datasheet claims a nominal primary conductor resistance of 1.1 $[m\Omega]$, which is sufficiently small to prevent burdening the motors or motor drivers. Under full load of 2.5 $[A]$ there is only a burden voltage of 2.75 $[mV]$ per sensor, which is inconsequential compared to the nominal supply voltage of 12 $[V]$.

[1]GMR stands for "giant magnetoresistive"; GMR based senors work similarly to Hall effect based senors allowing electrically isolated current measurement.

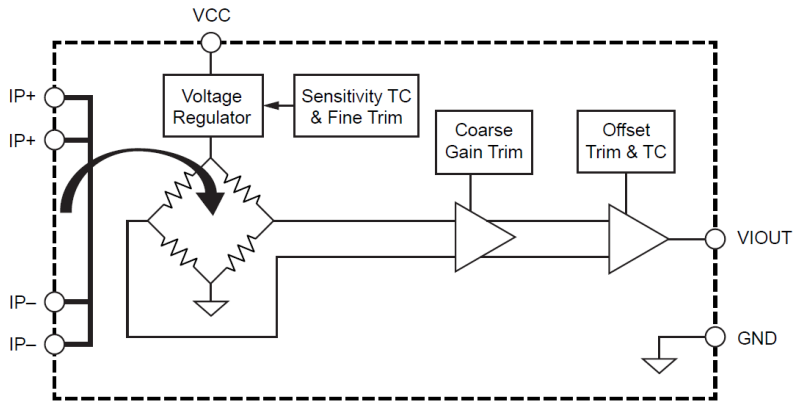


Figure 5.2: Functional block diagram for ACS70331 BLDC motor driver provided by Allegro Microsystems[2]

5.1.4 Inertial Measurement Unit

For this design, the main motivation in IMU selection is the ability to off-load the task of data-fusion to the IMU itself. Most modern IMUs provide 9-axis measurement data measured with a 3-axis accelerometer, a 3-axis gyroscope, and a 3-axis magnetometer. While it would be a valid academic exercise to implement a data fusion algorithm to produce properly filtered orientation data, it is well outside the scope of this project. Additionally, it would require more computing power than available, potentially overburdening the STM32 microcontroller[2].

The BNO055[5] produced by Bosch is a MEMS based 9-axis IMU with its own integrated ARM Cortex M0 microcontroller running a proprietary data fusion algorithm[3]. Additionally, the BNO055 provides a register based I^2C communication protocol that is easily handled in Micropython.

The BNO055 is sold on a breakout board by Adafruit Industries; Figure 5.3 shows a photograph of the breakout board. This breakout board is relocatable allowing adjustment of the location of the sensor relative to the balance bot. The sensor should ideally be mounted in the axis of the wheels of the balance bot, so that the acceleration measured by the IMU is as close as possible to gravitational acceleration with minimal effects due to the motion of the balance bot. A larger acceleration will be

[2]The IMU data fusion algorithms need to run around 100 $[Hz]$ and require real-time trigonometric calculations. This may be feasible to implement in C or C++ but in Micropython this would be unnecessarily computationally expensive for the selected Nucleo.

[3]Kalman filters and particle filters are likely choices for a data fusion algorithm as they can re-tune on the fly as the quality of each measurement changes during operation

measured due to the dynamic movement of the balance bot when the IMU is mounted far from the wheel axis. However, rotating magnetic fields produced by the BLDC motors also have a negative effect on the magnetometer within the IMU, causing oscillatory disturbances which can lead to instability in the heading measurement. To accommodate these competing location constraints, the sensor is mounted near the wheel axis, but elevated slightly to increase the air-gap between the sensor and the motors.

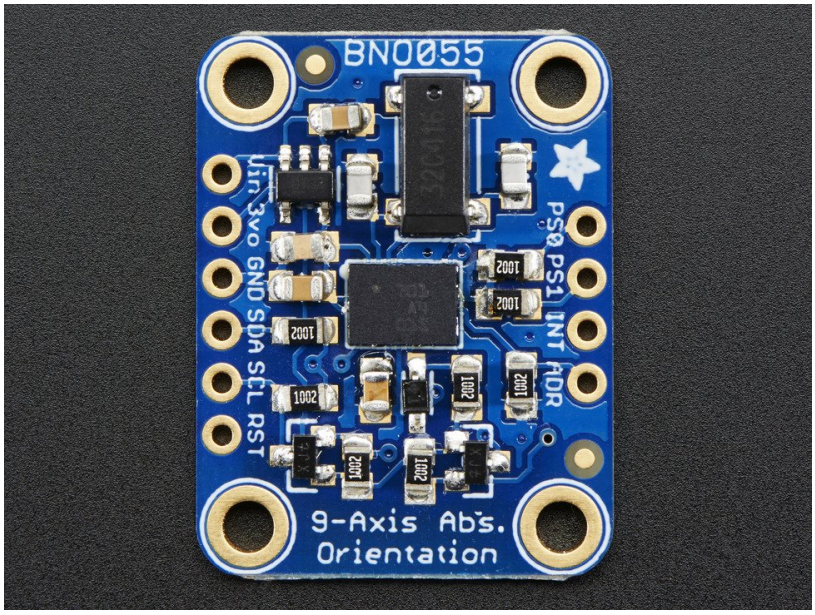


Figure 5.3: Breakout board housing the Bosch BNO055 produced by Adafruit Industries.

The BNO055 allows a variety of operation modes using varying subsets of the nine available axes. The nine-degree-of-freedom (NDOF) fusion mode is selected, as it provides the most robust output and allows absolute orientation measurement using the magnetometer.

5.1.5 Battery and Protection Circuitry

For this project it is important that the battery be sufficiently small to avoid adding unnecessary mass to the body of the balance bot. However, it is also import that the battery be able to supply the necessary voltage, current, and run-time for the balance bot to operate. To accommodate all of these concerns, a 700 $[mA \cdot h]$ three cell lithium polymer battery is selected. Lithium polymer batteries, like the one shown in Figure 5.4 are typically referred to as LiPo batteries and have comparatively

large power and energy density, especially with respect to alkaline batteries. Additionally, they come in a variety of form factors, cell configurations, and capacities.

The selected battery provides 12.6 $[V]$ when fully charged and 11.1 $[V]$ when discharged. It has a small form factor with dimensions of $73 \times 35 \times 18$ $[mm]$ and comes with the popular XT30 polarized plug. With a nominal capacity of 700 $[mA \cdot h]$ the battery provides around an hour of continuous operation before needing to be recharged[4]. The battery is also equipped with a balance charging plug that is required to maintain uniform cell voltages during recharging.

Figure 5.4: Turnigy 3S 700mAh lithium polymer battery.

There are three layers of battery protection implemented in the PCB design. The first layer is a fast-acting fuse rated for 5 $[A]$ directly in series with the positive battery terminal. This fuse is only intended to prevent catastrophic battery failure in the case of a short circuit or stalled rotor condition. The LiPo batteries selected for the design can output hundreds of amps in short circuit conditions, creating a potential fire hazard.

The second layer of battery protection is a reverse polarity protection circuit implemented with a P-Channel MOSFET (PN AOD417[3]) in series with the positive battery terminal. When the battery is plugged in with the correct orientation, the MOSFET will switch on and conduct with an added

[4]A 700 $[mA \cdot h]$ battery can supply 700 $[mA]$ continuously for 1 hour. However, in operation the battery will pull more or less current than that depending on the throttle and steering commands from the radio module. The balance bot require less than one Ampere to balance, but can pull multiple Amperes during periods of highly dynamic operation.

resistance of only 27 $[m\Omega]$. However, when the battery is plugged in with incorrect orientation, the MOSFET will not conduct and will prevent a reverse polarity condition.

The third layer of battery protection is a resistor network used to measure the battery voltage to estimate state of charge. The nominal 12 $[V]$ battery voltage is divided to provide a nominal 3 $[V]$ signal to an available ADC pin on the STM microcontroller. Measuring the state of charge allows the software to prevent the battery from overdischarging, which is important for LiPo battery health.

The schematic snippet shown in Figure 5.5 includes all three layers of battery protection. The battery connects to the PCB through the polarized connector CN8. Fuse F1 is in series with the positive battery terminal. The P-Channel MOSFET Q2 then prevents reverse polarity. To guarantee that the MOSFET switches on and off properly, the gate is tied to ground through a 10 $[K\Omega]$ resistor. Finally, the resistor network composed of R3, R6, and R9 divides the battery voltage from 12 $[V]$ to 3 $[V]$ to be read by the microcontroller.

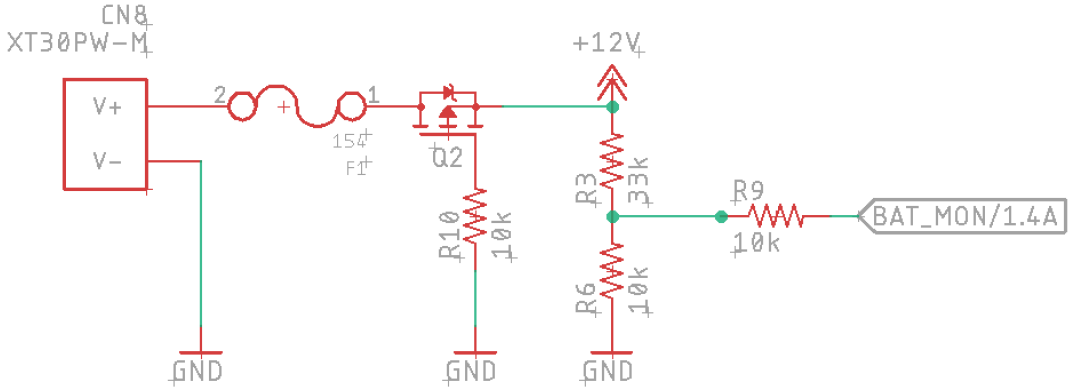


Figure 5.5: Battery protection schematic snippet.

There is no functionality to prevent a battery with incorrect voltage from being connected to the PCB through CN8. Therefore, it is the operator's responsibility to connect only batteries with a nominal voltage in the range of $8 - 12$ $[V]$.

5.1.6 Radio Receiver

For simplicity, a two channel radio receiver from a remote control car is selected. The TQi radio transmitter and receiver, shown in Figure 5.6, from Traxxas[19] are selected due to availability and ease of use.

Figure 5.6: TQi RC radio receiver from Traxxas[19].

RC receivers are simple to interface because they output a pulse width modulated signal. The PWM signal can be read easily using input capture with one of the timer modules present in the STM32L476RG microcontroller. Consider the signals in Figure 5.7, which depicts the convention for one channel of PWM output from the TQi receiver. The PWM signal is of constant[5] 20 $[ms]$ period with a pulse width ranging from 1 $[ms]$ to 2 $[ms]$. The throttle and steering commands are encoded in the pulse widths, not in the duty-cycle of the PWM signal.

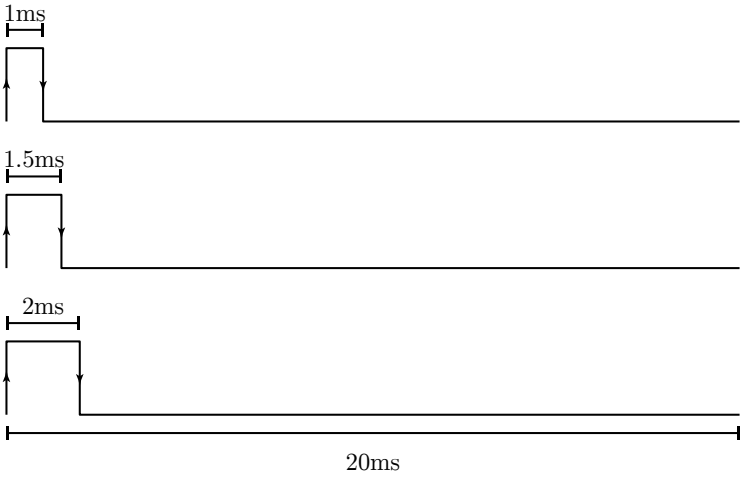


Figure 5.7: RC radio PWM output: full reverse (top); neutral (center); full forward (bottom).

[5]A period of 20 $[ms]$ is most common but some radios follow the same protocol with a faster update rate as low as 2.08 $[ms]$ (a of 480 $[Hz]$).

The signal indicates zero when the pulse width is 1.5 $[ms]$; longer or shorter pulses represent positive or negative steering or throttle command. The duration of the pulse can be measured by triggering input capture interrupts on the rising and falling edge of the signal. The difference in timer count between rising and falling edges can be used to determine the duration of the pulse. It is often useful for the data to be converted from timer count units to a range from -1 to 1. One convenient way to do this conversion is to select an appropriate prescaler for the timer and then apply a bit of arithmetic. The STM32L476RG has a base clock of 80 $[MHz]$, so selecting a prescaler of 80 results in a time-base of 1 $[^{\mu s}/_{tick}]$. This time-base allows the normalized signal to be computed as

$$r = \frac{(t_{fall} - t_{rise}) - 1500\ [\mu s]}{500\ [\mu s]} \tag{5.1}$$

Where r represents the dimensionless input signal from either the throttle or steering channel on the radio and t_{rise} and t_{fall} represent the timer count as measured on the rising and falling edges respectively. It is necessary to perform modular arithmetic when finding the difference $t_{fall} - t_{rise}$ in case the timer count has overflowed and reset.

5.1.7 Quadrature Encoder

In order to precisely measure the speed and angle of the motor shafts, each shaft has been equipped with a quadrature encoder. The selected encoder is the AMT112S shown on the right in Figure 5.8. The AMT11 is a capacitive encoder produced by CUI that produces incremental quadrature output. The AMT11 series is selected because it can be reconfigured in software to produce a variety of different resolutions ranging from 48 to 4096 pulses per revolution per channel. Additionally, the encoder has a configurable index channel that can be relocated with a serial command, which allows the index channel of the encoder to be aligned with a particular orientation if desired[6].

[6]In some situations it is useful to align the index channel on the encoder with one of the Hall effect sensor transitions. This feature allows the encoder count to be used for commutating the motor.

Figure 5.8: CUI AMT11 modular incremental encoder.

5.1.8 Hall Effect Sensors

The BLDC motors selected for this project come equipped with Hall sensors mounted in a standard 120° configuration. Hall sensors are used to detect the sector[7] of the motor to use for commutation. Hall effect sensors are typically in an open-collector configuration and therefore require pull-up resistors to generate a proper square wave. The STM32 microcontroller has built-in internal pull-up resistors that can be enabled in software. In testing, these internal pull-ups have proven to be sufficient to create stable waveforms from the Hall sensors.

5.2 Printed Circuit Board Design

During the development of the balance bot many revisions were designed, assembled, and tested. A detailed revision history can be found in Appendix C. The full circuit schematic is also available in Appendix D.

[7] In the context of BLDC motors, sector refers to the orientation of the rotor relative to the stator and indicates which of the three motor phases should source or sink current in a given orientation.

Chapter 6

MODELING AND SIMULATION

This chapter covers the model used to design the control laws and to simulate the system in MATLAB®. Both the two degree-of-freedom and three degree-of-freedom models are utilized further in this document and are discussed below.

Recall that both the 2-DOF and 3-DOF models are originally of the form $M(\boldsymbol{q})\ddot{\boldsymbol{q}} = \boldsymbol{f}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u})$. Each model can be expressed in the form $\dot{\boldsymbol{x}} = \boldsymbol{a}(\boldsymbol{x}) + b(\boldsymbol{x})\boldsymbol{u}$ because the models are nonlinear with respect to the states but linear with respect to the inputs.

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{q} \\ \\ \dot{\boldsymbol{q}} \end{bmatrix} \tag{6.1}$$

$$\boldsymbol{a}(\boldsymbol{x}) = \begin{bmatrix} \dot{\boldsymbol{q}}(\boldsymbol{x}) \\ \\ M(\boldsymbol{q}(\boldsymbol{x}))^{-1}\boldsymbol{f}(\boldsymbol{q}(\boldsymbol{x}), \dot{\boldsymbol{q}}(\boldsymbol{x}), \boldsymbol{0}) \end{bmatrix} \tag{6.2}$$

$$b(\boldsymbol{x}) = \begin{bmatrix} 0 \\ \\ M(\boldsymbol{q}(\boldsymbol{x}))^{-1}\left(\frac{\partial \boldsymbol{f}(\boldsymbol{q}(\boldsymbol{x}), \dot{\boldsymbol{q}}(\boldsymbol{x}), \boldsymbol{u})}{\partial \boldsymbol{u}}\right) \end{bmatrix} \tag{6.3}$$

In order to compute the feedback gain, K, and to simulate the system in MATLAB®, it is necessary to have numerical values for each of the parameters defining the model. These parameters are listed in Table 6.1 shown below. Each of these parameters comes from a measurement of the completed hardware design.

Table 6.1: Numerical mass and geometric properties used to model the balance bot.

Parameter	Value
r_w	45×10^{-3} $[m]$
r_b	25×10^{-3} $[m]$
w	120×10^{-3} $[m]$
m_w	80×10^{-3} $[kg]$
m_b	800×10^{-3} $[kg]$
$I_{w,xx}$	1.25×10^{-6} $[kg \cdot m^2]$
$I_{w,yy}$ (I_w)	2.50×10^{-6} $[kg \cdot m^2]$
$I_{w,zz}$	1.25×10^{-6} $[kg \cdot m^2]$
$I_{b,xx}$	5.80×10^{-3} $[kg \cdot m^2]$
$I_{b,yy}$ (I_b)	5.80×10^{-3} $[kg \cdot m^2]$
$I_{b,xx}$	2.90×10^{-3} $[kg \cdot m^2]$
g	9.81 $[m/s^2]$

6.1 Two Degree-of-freedom Model

The two degree of freedom model results in a fourth-order system. Recall the result from Lagrange's equation as derived in the balance bot modeling chapter.

$$\boldsymbol{q} = \begin{bmatrix} x \\ \\ \theta \end{bmatrix} \tag{6.4}$$

$$M(\boldsymbol{q}) = \begin{bmatrix} m_x & m_b r_b \cos\theta \\ \\ m_b r_b \cos\theta & I_\theta \end{bmatrix} \tag{6.5}$$

$$\boldsymbol{f}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u}) = \begin{bmatrix} -2\frac{1}{r_w} T_m + m_b r_b \sin\theta\, \dot{\theta}^2 \\ \\ 2T_m + m_b r_b \sin\theta\, g \end{bmatrix} \tag{6.6}$$

6.1.1 Decoupled EOMs

These parameters are now used to find the state-vector and the time rate-of-change of the state-vector $\boldsymbol{x}$ and $\dot{\boldsymbol{x}}$ as well as the non-linear state-space model parameters $\boldsymbol{a}(\boldsymbol{x})$, and $b(\boldsymbol{x})$. First, $\boldsymbol{x}$ and $\dot{\boldsymbol{x}}$ are found.

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{q} \\ \dot{\boldsymbol{q}} \end{bmatrix} = \begin{bmatrix} x \\ \theta \\ \dot{x} \\ \dot{\theta} \end{bmatrix} \qquad \dot{\boldsymbol{x}} = \begin{bmatrix} \dot{x} \\ \dot{\theta} \\ \ddot{x} \\ \ddot{\theta} \end{bmatrix}$$

And then $\boldsymbol{a}(\boldsymbol{x})$ is found.

$$\boldsymbol{a}(\boldsymbol{x}) = \begin{bmatrix} \dot{\boldsymbol{q}}(\boldsymbol{x}) \\ M(\boldsymbol{q}(\boldsymbol{x}))^{-1}\boldsymbol{f}(\boldsymbol{q}(\boldsymbol{x}), \dot{\boldsymbol{q}}(\boldsymbol{x}), \boldsymbol{0}) \end{bmatrix} \tag{6.7}$$

$$\boldsymbol{a}(\boldsymbol{x}) = \begin{bmatrix} \dot{x} \\ \dot{\theta} \\ \frac{1}{m_x I_\theta - m_b^2 r_b^2 \cos\theta^2}\left(I_\theta m_b r_b \sin\theta\,\dot{\theta}^2 - m_b^2 r_b^2 \sin\theta\cos\theta\, g\right) \\ \frac{1}{m_x I_\theta - m_b^2 r_b^2 \cos\theta^2}\left(-m_b^2 r_b^2 \sin\theta\cos\theta\,\dot{\theta}^2 + m_x m_b r_b \sin\theta\, g\right) \end{bmatrix} \tag{6.8}$$

And finally, $b(\boldsymbol{x})$ is found.

$$b(\boldsymbol{x}) = \begin{bmatrix} 0 \\ M(\boldsymbol{q}(\boldsymbol{x}))^{-1}\left(\frac{\partial \boldsymbol{f}(\boldsymbol{q}(\boldsymbol{x}), \dot{\boldsymbol{q}}(\boldsymbol{x}), \boldsymbol{u})}{\partial \boldsymbol{u}}\right) \end{bmatrix} \tag{6.9}$$

$$b(\boldsymbol{x}) = \begin{bmatrix} 0 \\ 0 \\ \frac{1}{m_x I_\theta - m_b^2 r_b^2 \cos\theta^2}\left(-2\frac{I_\theta}{r_w} - 2m_b r_b \cos\theta\right) \\ \frac{1}{m_x I_\theta - m_b^2 r_b^2 \cos\theta^2}\left(2\frac{m_b r_b \cos\theta}{r_w} - 2m_x\right) \end{bmatrix} \tag{6.10}$$

6.1.2 Linearization

Now that $\boldsymbol{a}(\boldsymbol{x})$ and $b(\boldsymbol{x})$ have been found, it is possible to linearize the system to find A and B. The operating point used for linearization is $\boldsymbol{x}_0 = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}^\top$; that is the operating point is defined by the bot balancing upright and motionless. The selected operating point is an equilibrium point, but an unstable equilibrium point.

With the operating point selected, A is found by linearizing $\boldsymbol{a}(\boldsymbol{x})$.

$$A = \left.\frac{\partial \boldsymbol{a}(\boldsymbol{x})}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}=\boldsymbol{x}_0} \tag{6.11}$$

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & A_{3,2} & 0 & 0 \\ 0 & A_{4,2} & 0 & 0 \end{bmatrix} \tag{6.12}$$

Where, $A_{3,2}$ and $A_{4,2}$ are coefficients too large to neatly display in the A matrix as shown below.

$$A_{3,2} = \frac{-m_b^2 r_b^2 g}{m_x I_\theta - m_b^2 r_b^2} \tag{6.13}$$

$$A_{4,2} = \frac{m_x m_b r_b g}{m_x I_\theta - m_b^2 r_b^2} \tag{6.14}$$

And then B is found.

$$B = b(\boldsymbol{x})|_{\boldsymbol{x}=\boldsymbol{x}_0} \tag{6.15}$$

$$B = \begin{bmatrix} 0 \\ 0 \\ \frac{1}{m_x I_\theta - m_b^2 r_b^2}\left(-2\frac{I_\theta}{r_w} - 2m_b r_b\right) \\ \frac{1}{m_x I_\theta - m_b^2 r_b^2}\left(2\frac{m_b r_b}{r_w} + 2m_x\right) \end{bmatrix} \tag{6.16}$$

6.1.3 Output Equations

In the preceding analysis, only the system model was discussed without consideration of the output equations relating the system states and input parameters to the measurable system outputs. In the derivation of the equations of motion for the 2-DOF model, the first generalized coordinate was selected as $q_1 = x$. This coordinate was a logical selection during modeling; but, in implementation, sensors will measure the motor angle, θ_m, which is the angle between the wheel and the body. The output is defined such that the output vector $\boldsymbol{y}$ contains the measurable system parameters; the output vector is defined as

$$\boldsymbol{y} = \begin{bmatrix} \theta_m \\ \theta_b \\ \dot{\theta}_m \\ \dot{\theta}_b \end{bmatrix}, \tag{6.17}$$

where θ_m is the motor shaft angle and $\theta_b = \theta$ is the pitch angle of the balance bot. The specified output can be defined as a linear combination of the system states and inputs using an output equation of the form $\boldsymbol{y} = C\boldsymbol{x} + D\boldsymbol{u}$.

$$\begin{bmatrix} \theta_m \\ \theta_b \\ \dot{\theta}_m \\ \dot{\theta}_b \end{bmatrix} = \begin{bmatrix} \frac{1}{r_w} & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{r_w} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ \theta \\ \dot{x} \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} T_m \end{bmatrix} \tag{6.18}$$

For the balance bot model, the output $\boldsymbol{y}$ is only a function of the state variables and there is no direct feed-through component from the input $\boldsymbol{u}$.

6.1.4 Controllability and Observability

The controllability of the system is evaluated about the operating point using the A and B matrices by forming the controllabilty Gramian $\mathcal{C}$ and computing its rank. The controllability of the system is a measure of how well the input can affect the states of the system. A fully controllable system satisfies $\text{rank}(\mathcal{C}) = \text{rank}(A)$, indicating that the input $\boldsymbol{u}$ is able to affect each and every state of the system.

$$\mathcal{C} = \begin{bmatrix} B & AB & A^2B & A^3B \end{bmatrix} \tag{6.19}$$

$$\mathcal{C} = \begin{bmatrix} 0 & B_3 & 0 & A_{3,2}B_4 \\ 0 & B_4 & 0 & A_{3,2}B_4 \\ B_3 & 0 & A_{3,2}B_4 & 0 \\ B_4 & 0 & A_{4,2}B_4 & 0 \end{bmatrix} \tag{6.20}$$

$$\text{rank}(\mathcal{C}) = 4 \tag{6.21}$$

The rank of the controllability Gramian determines whether or not the system is fully controllable. For unique non-zero coefficients $A_{3,2}$, $A_{4,2}$, B_3, and B_4, the Gramian is full rank for the 2-DOF balance bot; therefore, the model is considered fully controllable near its operating point.

The observability Gramian, $\mathcal{O}$, can also be considered, but is nearly trivial in this implementation because the matrix C is full rank; that is, the observability Gramian must be full rank because a full rank matrix composes the top four rows.

$$\mathcal{O} = \begin{bmatrix} C \\ CA \\ CA^2 \\ CA^3 \end{bmatrix} \tag{6.22}$$

$$\text{rank}(\mathcal{O}) = 4 \tag{6.23}$$

The observability matrix $\mathcal{O}$ is cumbersome to display within the body of this text and can be found fully expanded in Appendix B.

6.2 Three Degree-of-freedom Model

This section repeats the decoupling and linearization steps for the 3-DOF model.

$$\boldsymbol{q} = \begin{bmatrix} x \\ \theta \\ \psi \end{bmatrix} \tag{6.24}$$

$$M(\boldsymbol{q}) = \begin{bmatrix} m_x & m_b r_b \cos\theta & 0 \\ m_b r_b \cos\theta & I_\theta & 0 \\ 0 & 0 & m_b r_b^2 \sin^2\theta + I_{b,xx} \sin^2\theta + I_{b,zz} \cos^2\theta + I_\psi \end{bmatrix} \tag{6.25}$$

$$\boldsymbol{f}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u}) = \begin{bmatrix} -\frac{1}{r_w} T_{m,r} + \frac{1}{r_w} T_{m,l} + m_b r_b \sin\theta\, \dot{\theta}^2 \\ T_{m,r} - T_{m,l} + m_b r_b \sin\theta\, g + m_b r_b^2 \sin\theta \cos\theta\, \dot{\psi}^2 + I_{b,xx} \sin\theta \cos\theta\, \dot{\psi}^2 - I_{b,zz} \sin\theta \cos\theta\, \dot{\psi}^2 \\ -\frac{w}{2r_w} T_{m,r} - \frac{w}{2r_w} T_{m,l} - \left(2 m_b r_b^2 \sin\theta \cos\theta\, \dot{\theta} + 2 I_{b,xx} \sin\theta \cos\theta\, \dot{\theta} - 2 I_{b,zz} \sin\theta \cos\theta\, \dot{\theta} \right) \dot{\psi} \end{bmatrix} \tag{6.26}$$

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{q} \\ \dot{\boldsymbol{q}} \end{bmatrix} = \begin{bmatrix} x \\ \theta \\ \psi \\ \dot{x} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \qquad \dot{\boldsymbol{x}} = \begin{bmatrix} \dot{x} \\ \dot{\theta} \\ \dot{\psi} \\ \ddot{x} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix}$$

$$\boldsymbol{a}(\boldsymbol{x}) = \begin{bmatrix} \dot{\boldsymbol{q}}(\boldsymbol{x}) \\ M(\boldsymbol{q}(\boldsymbol{x}))^{-1}\boldsymbol{f}(\boldsymbol{q}(\boldsymbol{x}), \dot{\boldsymbol{q}}(\boldsymbol{x}), \boldsymbol{0}) \end{bmatrix} \tag{6.27}$$

$$\boldsymbol{a}(\boldsymbol{x}) = \begin{bmatrix} \dot{x} \\ \dot{\theta} \\ \dot{\psi} \\ \frac{1}{m_x I_\theta - m_b^2 r_b^2 \cos\theta^2}\left(I_\theta m_b r_b \sin\theta\,\dot{\theta}^2 - m_b^2 r_b^2 \sin\theta\cos\theta\, g + \left(-m_b^2 r_b^3 - m_b r_b I_{b,xx} + m_b r_b I_{b,zz}\right)\sin\theta\cos\theta\,\dot{\psi}^2\right) \\ \frac{1}{m_x I_\theta - m_b^2 r_b^2 \cos\theta^2}\left(m_x m_b r_b \sin\theta\, g - m_b^2 r_b^2 \sin\theta\cos\theta\,\dot{\theta}^2 + \left(m_x m_b r_b^2 + m_x I_{b,xx} - m_x I_{b,zz}\right)\sin\theta\cos\theta\,\dot{\psi}^2\right) \\ \frac{1}{m_b r_b^2 \sin\theta^2 + I_{b,xx}\sin\theta^2 + I_{b,zz}\cos\theta^2 + I_\psi}\left(-2\left(m_b r_b^2 + I_{b,xx} - I_{b,zz}\right)\sin\theta\cos\theta\,\dot{\theta}\dot{\psi}\right) \end{bmatrix} \tag{6.28}$$

$$b(\boldsymbol{x}) = \begin{bmatrix} 0 \\ M(\boldsymbol{q}(\boldsymbol{x}))^{-1}\left(\frac{\partial \boldsymbol{f}(\boldsymbol{q}(\boldsymbol{x}), \dot{\boldsymbol{q}}(\boldsymbol{x}), \boldsymbol{u})}{\partial \boldsymbol{u}}\right) \end{bmatrix} \tag{6.29}$$

$$b(\boldsymbol{x}) = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \frac{1}{m_x I_\theta - m_b^2 r_b^2 \cos\theta^2}\left(\frac{-I_\theta}{r_w} - m_b r_b \cos\theta\right) & \frac{1}{m_x I_\theta - m_b^2 r_b^2 \cos\theta^2}\left(\frac{I_\theta}{r_w} + m_b r_b \cos\theta\right) \\ \frac{1}{m_x I_\theta - m_b^2 r_b^2 \cos\theta^2}\left(\frac{m_b r_b \cos\theta}{r_w} + m_x\right) & \frac{1}{m_x I_\theta - m_b^2 r_b^2 \cos\theta^2}\left(\frac{-m_b r_b \cos\theta}{r_w} - m_x\right) \\ \frac{1}{m_b r_b^2 \sin\theta^2 + I_{b,xx}\sin\theta^2 + I_{b,zz}\cos\theta^2 + I_\psi}\left(\frac{-w}{2r_w}\right) & \frac{1}{m_b r_b^2 \sin\theta^2 + I_{b,xx}\sin\theta^2 + I_{b,zz}\cos\theta^2 + I_\psi}\left(\frac{-w}{2r_w}\right) \end{bmatrix} \tag{6.30}$$

6.2.1 Linearization

Now that $\boldsymbol{a}(\boldsymbol{x})$ and $b(\boldsymbol{x})$ have been found, it is possible to linearize the system to find A and B. The operating point used for linearization is $\boldsymbol{x}_0 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^\top$; that is the operating point is defined by the bot balancing upright and motionless. First A can be found.

$$A = \left.\frac{\partial \boldsymbol{a}(\boldsymbol{x})}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}=\boldsymbol{x}_0} \tag{6.31}$$

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & A_{4,2} & 0 & 0 & 0 & 0 \\ 0 & A_{5,2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{6.32}$$

Where, $A_{4,2}$ and $A_{5,2}$ are coefficients too large to display in the A matrix as shown below.

$$A_{4,2} = \frac{-m_b^2 r_b^2 g}{m_x I_\theta - m_b^2 r_b^2} \tag{6.33}$$

$$A_{5,2} = \frac{m_x m_b r_b g}{m_x I_\theta - m_b^2 r_b^2} \tag{6.34}$$

And then B can be found.

$$B = b(\boldsymbol{x})|_{\boldsymbol{x}=\boldsymbol{x}_0} \tag{6.35}$$

$$B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \frac{1}{m_x I_\theta - m_b^2 r_b^2}\left(\frac{-I_\theta}{r_w} - m_b r_b\right) & \frac{1}{m_x I_\theta - m_b^2 r_b^2}\left(\frac{I_\theta}{r_w} + m_b r_b\right) \\ \frac{1}{m_x I_\theta - m_b^2 r_b^2}\left(\frac{m_b r_b}{r_w} + m_x\right) & \frac{1}{m_x I_\theta - m_b^2 r_b^2}\left(\frac{-m_b r_b}{r_w} - m_x\right) \\ \frac{1}{I_{b,zz} + I_\psi}\left(\frac{-w}{2r_w}\right) & \frac{1}{I_{b,zz} + I_\psi}\left(\frac{-w}{2r_w}\right) \end{bmatrix} \tag{6.36}$$

6.2.2 Output Equations

In the preceding analysis, only the system model was discussed without consideration of the output equations relating the system states and input parameters to the measurable system outputs. In the derivation of the equations of motion for the 3-DOF model, the first generalized coordinate was

selected as $q_1 = x$. This coordinate was a logical selection during modeling; but, in implementation, sensors will measure the motor angle, θ_m, which is the angle between the wheel and the body. The output is defined such that the output vector $\boldsymbol{y}$ contains the measurable system parameters; the output vector is defined as

$$\boldsymbol{y} = \begin{bmatrix} \theta_m \\ \theta_b \\ \psi \\ \dot{\theta}_m \\ \dot{\theta}_b \\ \dot{\psi} \end{bmatrix}. \tag{6.37}$$

where θ_m is the motor shaft angle, $\theta_b = \theta$ is the pitch angle of the balance bot, and ψ is the yaw angle of the balance bot. The specified output can be defined as a linear combination of the system states and inputs using an output equation of the form $\boldsymbol{y} = C\boldsymbol{x} + D\boldsymbol{u}$.

$$\begin{bmatrix} \theta_m \\ \theta_b \\ \psi \\ \dot{\theta}_m \\ \dot{\theta}_b \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} \frac{1}{r_w} & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{r_w} & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ \theta \\ \psi \\ \dot{x} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} T_{m,r} \\ T_{m,l} \end{bmatrix} \tag{6.38}$$

For the balance bot model, the output $\boldsymbol{y}$ is only a function of the state variables and there is no direct feed-through component from the input $\boldsymbol{u}$.

6.2.3 Controllability and Observability

The controllability of the system can be evaluated about the operating point using the A and B matrices by forming the controllabilty Gramian $\mathcal{C}$ and computing its rank.

$$\mathcal{C} = \begin{bmatrix} B & AB & A^2B & A^3B & A^4B & A^5B \end{bmatrix} \tag{6.39}$$

$$\text{rank}(\mathcal{C}) = 6 \tag{6.40}$$

The rank of the controllability Gramian determines whether or not the system is fully controllable. For the 3-DOF system model the controllability Gramian is a $[6 \times 12]$ matrix too large to fit neatly in the body of this document. Please refer to Appendix B for the fully expanded controllability Gramian. The Gramian is full rank for the 3-DOF system model indicating that the balance bot can be considered fully controllable at its operating point.

The observability Gramian is also considered, but is nearly trivial in this implementation because the state-to-output coupling matrix C composing the top six rows of $\mathcal{O}$ is full rank. It is clear that the observability Gramian must be full rank because the observability Gramian contains a full rank matrix as the top six rows.

$$\mathcal{O} = \begin{bmatrix} C \\ CA \\ CA^2 \\ CA^3 \\ CA^4 \\ CA^5 \end{bmatrix} \tag{6.41}$$

$$\text{rank}(\mathcal{O}) = 6 \tag{6.42}$$

The fully expanded observability Gramian is shown in Appendix B.

Chapter 7

CONTROLLER DESIGN

This chapter focuses on the control architecture for the entire system. The LQR controller presented in Chapter 3 is modified slightly to improve tracking performance. Additionally, the commutation scheme for the selected motors is discussed.

7.1 Overall Controller Architecture

This section covers the high-level controller architecture as implemented on the balance bot. The architecture shown in Figure 7.1 is able to successfully control the desired states of the balance bot based on the throttle and steering commands from the RC car remote.

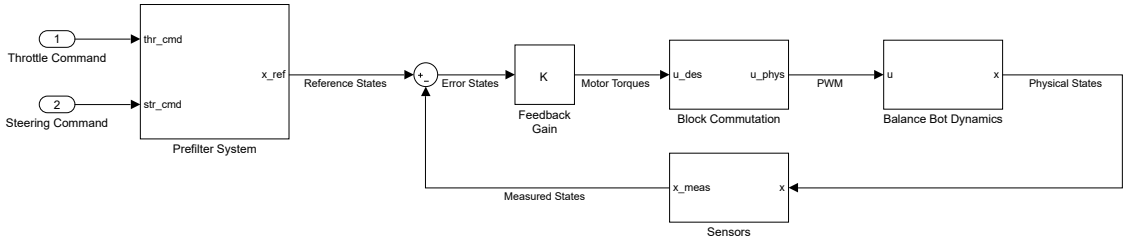


Figure 7.1: High-level balance bot controller architecture.

The throttle and steering inputs from the remote control are first filtered and augmented to produce reference values for all of the state variables. The reference states are then used with the measured state values to determine error states for the system. The error states are then fed through the linear feedback gain matrix, K, to determine the desired torques for each motor. The torque values are then used to determine the appropriate duty cycle to be applied to each phase of each motor. Finally, the state of the system is measured through the sensors and scaled in software to represent appropriate units.

7.2 Prefilter

The prefilter stage of the controller has two purposes. First, the prefilter applies a low-pass filter to the throttle and steering input commands, so that the controller does not have to react to instanta-

neous changes in command values. Second, the prefilter converts the filtered throttle and steering commands to desired states using an approximated inverse model of the balance bot equations.

The prefilter stage is implemented using a discrete state-space model; however, the model is developed as a continuous model and then discretized mathematically using the sample time of the filter.

The inputs to the prefilter are the reference commands from the remote, r_{thr} and r_{str}, which each range from -1 to 1. The throttle and steering reference commands are scaled by α_{thr} and α_{str} respectively to find the desired values for the linear velocity, $\dot{x}_{des}$, and pitch rate, $\dot{\psi}_{des}$.

$$\dot{x}_{des} = \alpha_{thr} r_{thr} \tag{7.1}$$

$$\dot{\psi}_{des} = \alpha_{str} r_{str} \tag{7.2}$$

The desired values for the horizontal velocity and yaw rate are filtered using a first-order low-pass filter. The gains β_x and β_ψ tune the response time of the low-pass filter. The values of β_x and β_ψ are tuned through testing after initially selecting β_x and β_ψ as the desired cutoff frequencies for the throttle and steering filters, respectively.

$$\frac{d}{dt}\dot{x}_{ref} = \beta_x \left(\dot{x}_{des} - \dot{x}_{ref}\right) \tag{7.3}$$

$$\frac{d}{dt}\dot{\psi}_{ref} = \beta_\psi \left(\dot{\psi}_{des} - \dot{\psi}_{ref}\right) \tag{7.4}$$

The reference horizontal displacement and yaw angle are then found by integrating the reference horizontal velocity and yaw rate.

$$\frac{d}{dt}x_{ref} = \dot{x}_{ref} \tag{7.5}$$

$$\frac{d}{dt}\psi_{ref} = \dot{\psi}_{ref} \tag{7.6}$$

The four ODEs in Equations 7.3 through 7.6 compose a fourth-order linear system. To estimate the remaining two reference states, the pitch angle and pitch velocity, the model of the balance bot is manipulated to algebraically relate the reference linear acceleration to the reference pitch angle and pitch velocity. This step requires a significant number of assumptions, and is only meant to approximate the reference pitch angle and pitch rate.

The prefilter causes the reference values for the pitch and pitch rate to change in correspondence with the desired linear acceleration so that the bot is not trying to remain balanced during dynamic motion. In other words, this part of the prefilter is intended to cause the balance bot to tilt in the direction of the desired acceleration to allow a better response to changing input.

To determine the relationship between θ_{ref} and $\ddot{x}_{ref}$, the model for the 2-DOF system is manipulated assuming that $\ddot{\theta}_{ref} = 0$, $\dot{\theta}_{ref} = 0$, and $\theta_{ref} \approx 0$ leading to a small angle approximation.

The 2-DOF system model is repeated below.

$$\begin{bmatrix} m_x & m_b r_b \cos\theta \\ m_b r_b \cos\theta & I_\theta \end{bmatrix} \begin{bmatrix} \ddot{x} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} -2\frac{1}{r_w} T_m + m_b r_b \sin\theta\, \dot{\theta}^2 \\ 2T_m + m_b r_b \sin\theta\, g \end{bmatrix} \tag{7.7}$$

After applying the aforementioned assumptions, a relationship between $\ddot{x}_{ref}$ and θ_{ref} is determined by solving for the motor torque T_m in both equations and then eliminating T_m.

$$(m_x r_w + m_b r_b)\, \ddot{x}_{ref} = m_b r_b g \theta_{ref} \tag{7.8}$$

$$\theta_{ref} = \frac{m_x r_w + m_b r_b}{m_b r_b g} \ddot{x}_{ref} \tag{7.9}$$

$$\theta_{ref} = \frac{m_x r_w + m_b r_b}{m_b r_b g} \beta_x \left(\dot{x}_{des} - \dot{x}_{ref}\right) \tag{7.10}$$

An estimate for $\dot{\theta}_{ref}$ is found by differentiationg θ_{ref} and assuming $\ddot{x}_{des} = 0$.

$$\frac{d}{dt}\theta_{ref} = \frac{d}{dt}\left(\frac{m_x r_w + m_b r_b}{m_b r_b g}\beta_x \left(\dot{x}_{des} - \dot{x}_{ref}\right)\right) \tag{7.11}$$

$$\dot{\theta}_{ref} = \frac{m_x r_w + m_b r_b}{m_b r_b g}\beta_x \left(\ddot{x}_{des} - \ddot{x}_{ref}\right) \tag{7.12}$$

$$\dot{\theta}_{ref} = -\frac{m_x r_w + m_b r_b}{m_b r_b g}\beta_x \ddot{x}_{ref} \tag{7.13}$$

$$\dot{\theta}_{ref} = -\frac{m_x r_w + m_b r_b}{m_b r_b g}\beta_x^2 \left(\dot{x}_{des} - \dot{x}_{ref}\right) \tag{7.14}$$

For the sake of notation, an approximate parameter, σ, is defined as

$$\sigma = \frac{m_x r_w + m_b r_b}{m_b r_b g}. \tag{7.15}$$

All aspects of the prefilter are combined into a single continuous state-space representation.

$$\frac{d}{dt}\begin{bmatrix} x_{ref} \\ \psi_{ref} \\ \dot{x}_{ref} \\ \dot{\psi}_{ref} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -\beta_x & 0 \\ 0 & 0 & 0 & -\beta_\psi \end{bmatrix} \begin{bmatrix} x_{ref} \\ \psi_{ref} \\ \dot{x}_{ref} \\ \dot{\psi}_{ref} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \beta_x \alpha_{thr} & 0 \\ 0 & \beta_\psi \alpha_{str} \end{bmatrix} \begin{bmatrix} r_{thr} \\ r_{str} \end{bmatrix} \tag{7.16}$$

$$\begin{bmatrix} x_{ref} \\ \theta_{ref} \\ \psi_{ref} \\ \dot{x}_{ref} \\ \dot{\theta}_{ref} \\ \dot{\psi}_{ref} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -\sigma\beta_x & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \sigma\beta_x^2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{ref} \\ \psi_{ref} \\ \dot{x}_{ref} \\ \dot{\psi}_{ref} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ \sigma\beta_x\alpha_{thr} & 0 \\ 0 & 0 \\ 0 & 0 \\ -\sigma\beta_x^2\alpha_{thr} & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} r_{thr} \\ r_{str} \end{bmatrix} \tag{7.17}$$

This continuous state-space representation can also be expressed symbolically.

$$\dot{\boldsymbol{x}}_{ref} = A_{ref}\boldsymbol{x}_{ref} + B_{ref}\boldsymbol{r} \tag{7.18}$$

$$\boldsymbol{y}_{ref} = C_{ref}\boldsymbol{x}_{ref} + D_{ref}\boldsymbol{r} \tag{7.19}$$

The system is ready to be discretized using the sampling period t_s, equal to 0.025 $[s]$ for the balance bot, to be of the form $\boldsymbol{x}_{k+1} = A_d\boldsymbol{x}_k + B_d\boldsymbol{u}_k$ and $\boldsymbol{y}_k = C\boldsymbol{x}_k + D\boldsymbol{u}_k$. Discretization can be achieved by considering the response of the system when a zero-order hold is applied to the input parameter $\boldsymbol{u}$. The discretized matrices, A_d and B_d, are found by making use of the matrix exponential function:

$$A_d = e^{At_s}, \tag{7.20}$$

and

$$B_d = \left(e^{At_s} - I\right) A^{-1}B. \tag{7.21}$$

A derivation for this discretization process can be found in [6].

Discretization is important in the proposed controller implementation for the balance bot due to the slow loop closure rate of 40 $[Hz]$. The controller is too slow to assume that the system will behave like the proposed continuous model. Discretization allows a better performing controller to be designed using slower hardware.

As discussed briefly in the modeling chapter of this document, the sensors on the balance bot measure the motor angle, θ_m, and not the horizontal displacement, x. In order to use the prefilter defined above, it is necessary to apply a transformation T to the matrices C_{ref} and D_{ref} in order for the reference states to match the measured outputs. This transformation is exactly what the output equation $\boldsymbol{y} = C\boldsymbol{x} + D\boldsymbol{u}$ does in the linearized model; therefore, $T = C$, where C refers to the state-to-output coupling matrix for the 3-DOF system model. The transformation T is applied to the prefilter output equation $\boldsymbol{y}_{ref} = C_{ref}\boldsymbol{x}_{ref} + D_{ref}\boldsymbol{r}$ to represent a new output, $\hat{\boldsymbol{y}}_{ref}$ in terms of the reference states $\boldsymbol{x}_{ref}$ and command inputs $\boldsymbol{r}$.

$$\hat{\boldsymbol{y}}_{ref} = T\boldsymbol{y}_{ref} \tag{7.22}$$

$$\hat{\boldsymbol{y}}_{ref} = T\left(C_{ref}\boldsymbol{x}_{ref} + D_{ref}\boldsymbol{r}\right) \tag{7.23}$$

$$\hat{\boldsymbol{y}}_{ref} = TC_{ref}\boldsymbol{x}_{ref} + TD_{ref}\boldsymbol{r} \tag{7.24}$$

$$\hat{\boldsymbol{y}}_{ref} = \hat{C}_{ref}\boldsymbol{x}_{ref} + \hat{D}_{ref}\boldsymbol{r} \tag{7.25}$$

The transformed output $\hat{\boldsymbol{y}}_{ref}$ can be computed directly from the un-transformed state, $\boldsymbol{x}_{ref}$, and command input, $\boldsymbol{r}$, if the transformation is applied to the C_{ref} and D_{ref} matrices to get $\hat{C}_{ref}$ and $\hat{D}_{ref}$.

$$\hat{C}_{ref} = TC_{ref} \tag{7.26}$$

$$\hat{C}_{ref} = \begin{bmatrix} \frac{1}{r_w} & 0 & \sigma\beta_x & 0 \\ 0 & 0 & -\sigma\beta_x & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{r_w} & 0 \\ 0 & 0 & -\sigma\beta_x^2 & 0 \\ 0 & 0 & \sigma\beta_x^2 & 1 \end{bmatrix} \tag{7.27}$$

$$\hat{D}_{ref} = TD_{ref} \tag{7.28}$$

$$\hat{D}_{ref} = \begin{bmatrix} -\sigma\beta_x\alpha_{thr} & 0 \\ \sigma\beta_x\alpha_{thr} & 0 \\ 0 & 0 \\ \sigma\beta_x^2\alpha_{thr} & 0 \\ -\sigma\beta_x^2\alpha_{thr} & 0 \\ 0 & 0 \end{bmatrix} \tag{7.29}$$

7.3 Feedback Gain

The feedback gain K is found by iterating on gains initially computed through a linear quadratic regulator design. Bryson's rule is used to select Q and R matrices for use in the LQR design. The LQR gain is only an initial starting point; the system must also be tuned manually to improve performance.

The loop-closure rate of the controller is only 40 $[Hz]$ in implementation, so gains for a discretized controller are found by use of the `lqrd` command in MATLAB®. This command designs a discrete LQR for a continuous linear system, accounting for the sample time of the controller.

The LQR designed was an infinite horizon discrete time linear-quadratic regulator. The most general cost function for a controller of this type is

$$J = \sum_{k=0}^{\infty} \left(\boldsymbol{x}_k^\top Q \boldsymbol{x}_k + \boldsymbol{u}_k^\top R \boldsymbol{u}_k + 2 \boldsymbol{x}_k^\top N \boldsymbol{u}_k \right) \tag{7.30}$$

Where Q is a matrix that penalizes non-zero state variables, R is a matrix that penalizes non-zero inputs, and N is a matrix that penalizes the interaction between the states and inputs. For many LQR designs N is set to zero and Q and R are selected as diagonal matrices.

7.3.1 Bryson's Rule

Bryson's Rule is one method for selecting the Q and R matrices to create a satisfactory controller. With this method, the Q and R matrices are selected as diagonal matrices where each element on the diagonal represents the inverse square of the maximum acceptable value for the corresponding state or input. Consider the matrices below for the three degree-of-freedom model with six states, $x_1 \dots x_6$ and two inputs, u_1 and u_2.

$$Q = \begin{bmatrix} \frac{1}{x_{1,max}^2} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{x_{2,max}^2} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{x_{3,max}^2} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{x_{4,max}^2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{x_{5,max}^2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{x_{6,max}^2} \end{bmatrix} \tag{7.31}$$

$$R = \begin{bmatrix} \frac{1}{u_{1,max}^2} & 0 \\ 0 & \frac{1}{u_{2,max}^2} \end{bmatrix} \tag{7.32}$$

These Q and R matrices are intended to normalize the states based on their maximum acceptable values so that the LQR will effectively regulate all of them with similar priority.

7.3.2 Discretization and the Discrete-time Algebraic Ricatti Equation

Discretization is mentioned in the preceding discussion of the prefilter, but must be applied to the controller as well. The behavior of a digital control loop implemented with a microcontroller is fundamentally discrete in nature. Furthermore, at the onset of controller design it may not be apparent how quickly the controller will run in hardware implementation[1]. A general rule-of-thumb is that if the control loop runs at least an order of magnitude faster than the desired closed-loop system dynamics, the controller can be assumed to behave like a continuous controller; however, even with a fast controller, treating the controller as continuous is an approximation. Furthermore, when designing an LQR, the designer does not directly select the system pole locations, making it challenging to estimate the speed of the closed-loop dynamics.

Therefore, the controller is first developed as a continuous-time LQR, but in implementation the controller runs discretely. For a discrete-time controller, the gain K can be determined from the Q and R matrices by solving the Discrete-time Algebraic Ricatti Equation (DARE) rather than the Continuous-time Algebraic Ricatti Equation (CARE). The DARE, shown in Equation 7.33, is an

[1] In hardware implementation the minimal loop period is found to be 25 $[ms]$ during data collection, but as low as 10 $[ms]$ if data collection is disabled.

algebraic matrix equation that can be solved for the matrix X, sometimes called P, which can in turn be used to determine the gain K. A derivation for the DARE can be found in [6] or [12].

$$A^\top XA - X - \left(A^\top XB\right)\left(B^\top XB\right)^{-1}\left(A^\top XB\right) + Q = 0 \tag{7.33}$$

One approach for finding X is to define the symplectic matrix Z and use the stable Eigenvectors of Z to determine X. Once X is determined, the feedback gain K can be found by manipulating the matrix X found from the DARE[12].

$$K = \left(R + B^\top XB\right)^{-1} B^\top XA \tag{7.34}$$

7.3.3 Gain Calculation with `lqrd`

For large systems with many states and inputs it quickly becomes impractical or infeasible to solve the DARE manually. Further, to design a discrete controller for a continuous plant, the continuous system must first be discretized before the LQR can be designed. The `lqrd` function in MATLAB® first does a continuous-to-discrete-time conversion on a state-space system and then computes the LQR gains for the discretized systems according to the specified Q and R matrices.

7.3.4 Gain Transformation

In implementation it is convenient to feedback the measured signals through a gain matrix rather than manipulating the measured signals to represent state variables only to feed those back through a matrix. In essence, this implies that $\boldsymbol{u} = -\hat{K}\boldsymbol{y}$, where $\hat{K}$ is a transformed gain matrix that simultaneously transforms the output into state variables, and then computes the input vector $\boldsymbol{u}$ in one step. That is,

$$\boldsymbol{u} = -\hat{K}\boldsymbol{y} \tag{7.35}$$

$$\boldsymbol{u} = -\hat{K}C\boldsymbol{x} \tag{7.36}$$

$$\boldsymbol{u} = -\left(\hat{K}C\right)\boldsymbol{x}, \tag{7.37}$$

which implies that

$$\hat{K}C = K \tag{7.38}$$

$$\hat{K} = C^{-1}K. \tag{7.39}$$

The transformed gains are more useful in firmware, because the transformed gains apply directly to the measured parameters coming from the sensors. Application of $\hat{K}$ requires only one step whereas state transformation followed by application of the original gain K would require multiple steps.

7.4 Tracking Control with LQR

A traditional linear-quadratic regulator is meant to reduce the value of each state to zero. A tracking controller is meant to force each state to match its corresponding reference state. To reconcile these notions, the LQR must be made to act on *error* states rather than the system states directly. That is, rather than defining the input as $\boldsymbol{u} = -K\boldsymbol{x}$, the input is instead defined as

$$\boldsymbol{u} = -K\boldsymbol{x}_{err} \tag{7.40}$$

$$\boldsymbol{u} = -K\left(\boldsymbol{x} - \boldsymbol{x}_{ref}\right) \tag{7.41}$$

$$\boldsymbol{u} = K\left(\boldsymbol{x}_{ref} - \boldsymbol{x}\right), \tag{7.42}$$

which drives the error in each state to zero instead of the states themselves. An intuitive way to consider the change to error states to is to consider the vector space representing the state variables. Using error states in the gain calculation is equivalent to translating the origin of the state-space to the desired state value. Thus, the LQR will regulate the state to the desired state value rather than to zero.

7.5 Block Commutation and Actuator Modeling

The balance bot is designed to include hardware allowing any kind of commutation of the BLDC motors used for balancing. In practice it is evident that simple block commutation, also called trapezoidal commutation, is more than sufficient for the balance bot application. No further commutation techniques are investigated, but the hardware is able to support other commutation techniques if further work aims to investigate field-oriented control[2] using sinusoidal or space-vector commutation.

The controller assumes that the torque on each wheel is the true input to the open loop system and can be instantaneously and arbitrarily selected. Naturally, this assumption is false, but it allows a much simpler model. To motivate this assumption consider the electrical time constant of the BLDC motor. The line-to-line resistance and inductance of the motor windings are $R = 8.00$ [Ω] and $L = 10.50$ [mH] respectively. The resulting time constant, $\tau_{RL} = {}^{L}/_{R}$, is only 1.3 [mS]. The largest Eigenvalue in the open-loop system is 5.77 [$^{rad}/_{s}$]. This Eigenvalue can be used to approximate an effective time constant for the open-loop system dynamics. $\tau_{OL} \approx \frac{2\pi}{\lambda_{max}}$. Using this approximation, the time constant for the system is on the order of 1.0 [s]. Comparison of $\tau_{RL} \approx 1.3$ [mS] and $\tau_{OL} \approx 1.0$ [s] illustrates that the dynamics associated with building the magnetic field in the motor windings are many times faster than the dynamics associated with the balance bot. Therefore it is valid to assume that the actuator dynamics do not need to be included in the system model used to develop a closed-loop controller.

[2]Field-oriented control (FOC) was developed to improve torque control of BLDC and PMS motors. FOC allows the phase currents to be controlled directly so that the magnetic field generated by the stator windings always interacts with the magnetic field moving with the rotor to produce optimal torque. While FOC does not produce more torque than standard block commutation, it does eliminate some of the torque ripple associated with block commutation.

Chapter 8

FIRMWARE DESIGN

8.1 Tasks

The firmware design is very simple and can be considered as a single task implemented as a finite-state machine with five states. The focus of this project was on balancing the bot and performing proper commutation of the two BLDC motors. Accordingly, the firmware design is simple and requires little multitasking. Future implementations may require more sophisticated, task-based, design; but, for the purpose of this project a single task was sufficient.

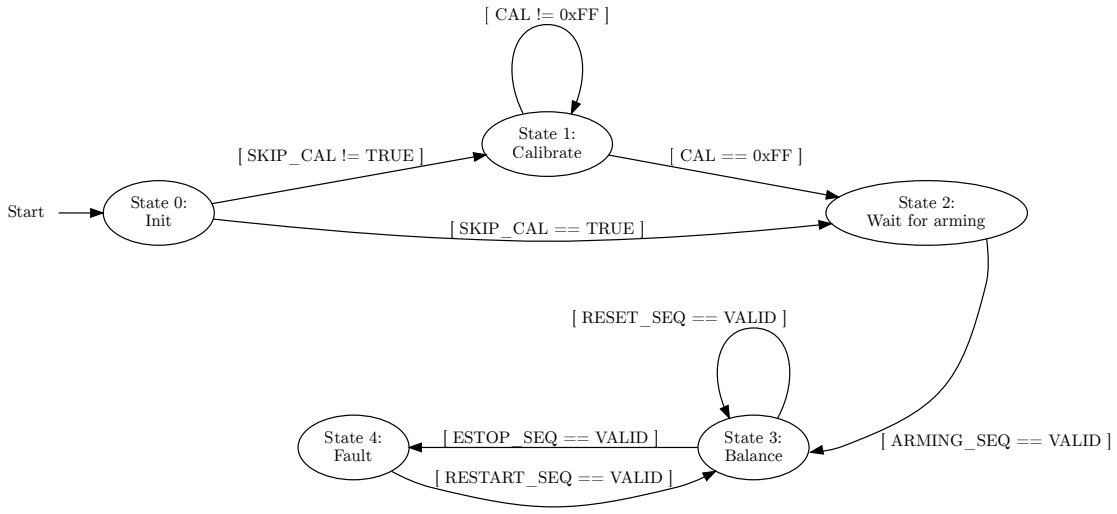


Figure 8.1: State-transition diagram for balance bot controller task.

State 0: Initialization

The Initialization state is responsible for configuring all of the firmware drivers and starting several interrupt driven services that will take place behind the scenes while the main task is running. The interrupt driven services will be discussed in a section below.

State 1: Calibrate

The Calibrate state is responsible for calibration of the BNO055 inertial measurement unit. The IMU runs a sophisticated and proprietary algorithm to fuse data from a 3-axis accelerometer, a 3-axis gyroscope, and a 3-axis magnetometer to determine the absolute orientation of

the sensor. As part of the algorithm, the IMU is continuously recalibrated. However, the calibration requires manipulation of the IMU in all three dimensions that may not occur during typical operation.

In the Calibrate state, the firmware keeps all motors off and waits for the user to physically manipulate the balance bot until the IMU indicates that calibration has been successful. Only after this calibration occurs can the output data of the sensor be used reliably. If the data is used without calibration then the data is prone to drift, primarily due to the inherent drift problems associated with MEMS gyroscopes.

Once calibration is complete, it is possible to store the calibration coefficients in non-volatile storage on the main microcontroller. This allows the user to bypass the calibration step and instead write the calibration values directly to the IMU on startup. Implementation of the calibration bypass was necessary to facilitate rapid development.

State 2: Wait for arming

An arming sequence has been defined using the throttle input to the RC radio receiver; this sequence is depicted in Figure 8.2. After the calibration is complete or the calibration has been bypassed, the system will wait for a particular sequence from the throttle trigger on the RC remote. The user must pull the trigger to full throttle and then full brake to arm the system. Once this sequence has been performed, 500 $[mS]$ later, the device begins operation and transitions to the Balance state.

Figure 8.2: Arming sequence on RC remote controller.

State 3: Balance

The task spends the majority of the time in the Balance state. In this state, the system runs the state-feedback controller previously described in the Controller Implementation section at a rate of 40 $[Hz]$. This loop closure rate was found to provide an optimal balance

between performance and reliability. A faster controller could, in theory, perform better, but with sacrificed reliability due to firmware speed limitations

The timing of this state was dictated by a simple scheme similar to how output-compare is performed in timer peripherals. The following snippet of python shows how the timekeeping works.

Listing 8.1: Task timing code snippet.

```
while True:
    cur_ticks = utime.ticks_us()
    if  cur_ticks >= next_ticks :

        # Control loop code here

    next_ticks = utime.ticks_add(next_ticks, ticks_period)
```

The variable `cur_ticks` contains the number of microseconds elapsed since the microcontroller was last reset. This parameter is updated continuously such that it can be used as a free-running counter. The variable `next_ticks` indicates the timestamp for the next iteration of the Balance state. As soon as `cur_ticks` exceeds `next_ticks`, the content of the state is executed and `next_ticks` is incremented by the period of the control loop, 25 $[ms]$.

Setting up the timing this way guarantees that the duration of the task is not added to the desired period for the task; that is, the period specifies the time between the *start* of each loop and not between the end of one loop and the start of the next.

Due to the nature of MicroPython and the number of interrupt service routines running in the background, it is not guaranteed that the Balance state will always take the same amount of time to execute. This time-keeping method has been shown to work reliably, running the task at the scheduled time with a standard deviation in loop period of less than 2 $[ms]$. The time-keeping is benchmarked by collecting time-stamped data over a test period of 120 $[s]$. Consider Figure 8.3, which shows a histogram of the recorded loop period over the 120 $[s]$ interval. The standard deviation over the 120 $[s]$ data window is 1.65 $[ms]$.

During each iteration of the Balance state, the system performs the following actions:

1. Query the sensors or retrieve the buffered sensor data waiting from the most recent ISR.
2. "Unwrap" any sensor values that represent angles. This prevents issues caused by turning the bot more than one full revolution. For example, the heading output of the IMU is used as a measurement of the yaw angle of the balance bot. The heading is bounded between $-180°$ and $180°$ and wraps around when the heading exceeds these bounds. The

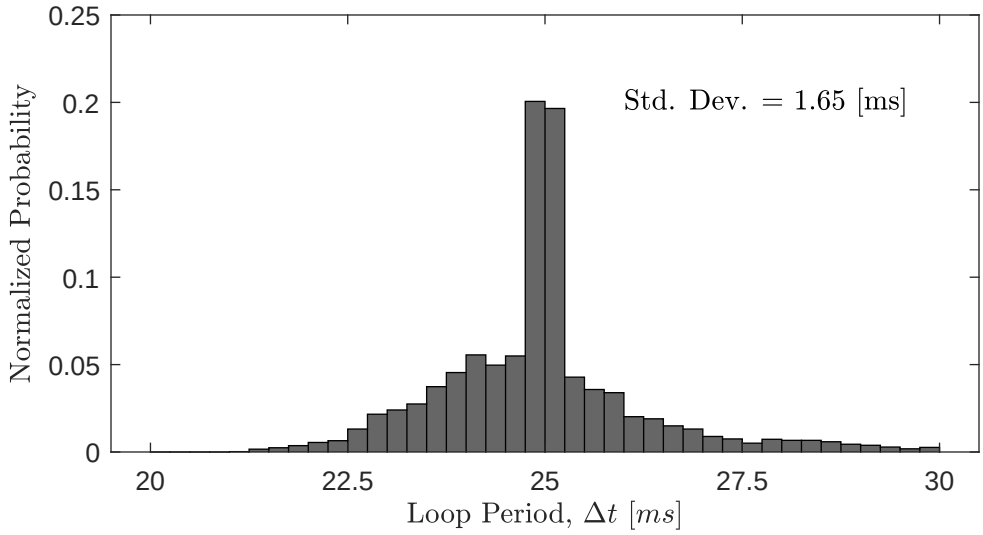


Figure 8.3: Histogram showing variance in loop period, Δt, over 120 $[s]$ interval of testing.

firmware accounts for the heading wrap by checking if the change in heading exceeds 180°. Consider the following snippet of Python code:

Listing 8.2: Heading unwrapping code snippet.

```
# Get the raw heading from the IMU
raw_heading = -imu.get_heading()

# Find the change in heading since the last measurement
delta_heading = raw_heading - last_heading

# If the change exceeds half the available range assume that the
# heading has wrapped
if delta_heading > 180:
    delta_heading -= 360
elif delta_heading < -180:
    delta_heading += 360

# Accumulate the change in heading to find the unwrapped heading
heading += delta_heading

# Record the last raw value so that the change can be computed
# next iteration
last_heading = raw_heading
```

3. Scale the sensor data to represent state variables for use in the control loop by performing appopriate unit conversions.
4. Retrieve the most recently buffered input from the RC remote and normalize to ±1 to be used as input to the prefilter.
5. Apply the prefilter to the normalized input values from the RC remote.
6. Saturate the output of the prefilter to guarantee that the reference values for the tracking controller are in bounds. Saturation prevents the throttle or steering commands from requesting unattainable accelerations or velocities from the balance bot.

7. Compute the duty-cycle associated with each BLDC motor using the output of the state-feedback controller.
8. Saturate the duty-cycles to remain in bounds.
9. Update the duty-cycle command associated with each BLDC motor.
10. Transmit a packet of information over Bluetooth to a host PC for data collection.

State 4: Fault

The Fault state has been implemented to handle the case when something goes wrong, particularly, when the bot falls over. The system may enter this state due to multiple causes. In the case the pitch angle, θ, ever exceeds $\pm\frac{3\pi}{8}$ $[rad]$, the bot will automatically enter the Fault state until the angle is returned to the acceptable range. This allows the operator to manually restore the orientation of the balance bot and for the bot to then automatically resume balancing.

Most often the Fault state is entered when triggered by the RC remote controller by the operator if the robot goes out-of-control or if a test is being performed. The user can trigger the Fault state deliberately by pushing the throttle to full brake. In this case, the Fault state is entered temporarily and the system returns to the Balance state as soon as the trigger is released. If the bot is in the Fault state and the user turns the steering knob all the way to the right, the bot will perform a soft-reset when exiting the Fault state, zeroing the system states used for the control loop. Alternately, if the bot is in the Fault state and the user turns the steering knob on the remote all the way to the left, the bot will latch the fault state permanently until a full system reset occurs. This is useful because it allows the user to cease data collection after performing a test. All three trigger commands are shown graphically in Figure 8.4.

Figure 8.4: Fault trigger commands: temporary fault (left); temporary fault with soft reset (center); latched fault (right).

8.2 Interrupt Service Routines

The firmware design for the balance bot relies on several interrupt driven services to interface with the motors and to receive low latency sensor data.

8.2.1 Hall Sensor Interrupts

Each BLDC motor has a set of three Hall sensors internally mounted to determine rotor orientation during operation. The input-capture feature of the timer module in the STM32L476 is used to trigger interrupts when any of the six Hall sensors change state. The main purpose of this interrupt is to decode which sector a given motor is in when a Hall effect sensor changes state. There are eight possible combinations for each set of Hall sensors; these eight combinations are listed in table 8.1.

Table 8.1: Hall-effect sensor sector decoding.

Hall A	Hall B	Hall C	Sector
0	0	0	Invalid Sector
0	0	1	Sector 4
0	1	0	Sector 6
0	1	1	Sector 5
1	0	0	Sector 2
1	0	1	Sector 3
1	1	0	Sector 1
1	1	1	Invalid Sector

Of the eight possible combinations, two are invalid and indicate a hardware fault. The other six combinations each indicate that a certain sector has been reached. Once the sector has been determined, if the sector is valid the Hall sensor interrupt determines if a valid transition has occurred. Between any two Hall sensor interrupts, only one sensor should change state. If more than one Hall sensor changes state, then a sector has been skipped. If either an invalid combination is found, or an invalid change in sector is determined, the motor is disabled and an exception is thrown.

When a valid sector transition is determined, the new sector dictates which phases of the motor will receive PWM. This is implemented in Python using three object references representing the timer channels associated with the inactive, positive, and negative phases for a given sector. The Hall sensor interrupts update the references to point at the appropriate timer channel objects, so that when the control loop computes a new duty cycle, the `set_duty()` function does not need to decode the sector to determine which phases to commutate. This effectively reduces overhead, because in standard operation, the control loop updates faster than the Hall sensor interrupts occur. Therefore,

the sectors are decoded exactly one time for each state transition, instead of redundantly when the `set_duty` function is called.

The final operation that occurs within each Hall sensor interrupt is an estimation of the motor speed. The six sector transitions per revolution can be used to determine relative displacement. However, unlike most encoders, the resolution is too poor to use effectively with normal speed determination methods. With typical encoders, the number of transitions per unit of time is measured to get an approximation of speed over that interval of time. For low resolution encoders, or BLDC Hall effect sensors, it is instead necessary to compute the amount of time between adjacent transitions. This method allows the measurement to have much higher resolution, but causes other problems, namely with the timer required to measure the duration between edges. With standard 16-bit timers it can be challenging to find a proper prescaler that will result in good resolution at high speeds, without causing timer overflow at low speeds.

A detail that further complicates speed estimation is misalignment of the Hall sensors. The set of three Hall sensors should be evenly spaced, such that when the motor is spinning at constant velocity, a phase shift of exactly 120° would be expected. Unfortunately, some motors have misaligned Hall sensors that cause phase misalignment. Further complications occur when the Hall sensors do not generate square waves of 50% duty cycle, which is also typical of common BLDC motors. The workaround for these issues is to measure the amount of time between like transitions, for instance between rising edges on Hall channel A. This comes at a penalty though, as it increases the latency of the measurement by a factor of six.Latency is a small issue at high speeds or at constant speeds, but at low dynamic speeds the measurement can be considerably late.

Fortunately, the addition of quadrature encoders to the design eliminates the need to use the Hall sensor speed estimate for control purposes. However, in a reduced cost version of this project, it may be feasible to use the Hall sensors nonetheless.

8.3 RC Radio Interrupts

The command inputs to the balance bot come from an RC radio receiver attached to the STM32L476 microcontroller. The PWM output of the receiver is discussed in detail in Section 5.1.6. To interpret the PWM output properly, input capture interrupts are used. A driver class called `RX` was written, that allows the user to easily interface with a radio receiver with up to four channels.

The following code snippet illustrates the logic used within the callback function for channel 1 to determine the pulse width of the signal coming from the receiver.

Listing 8.3: Timer callback code snippet.

```
def cb1(self,tim):
    self.cur_ticks1 = tim.channel(1).capture()
    if not self.ch1.value():
        # Falling edge
        self.width1 = self.cur_ticks1 - self.last_ticks1
        if self.width1 < 0:
            self.width1 += 0x10000
    self.last_ticks1 = self.cur_ticks1
```

Each time an interrupt occurs, the current tick value from the free-running counter is stored in a variable `cur_ticksn`, where n is 1, 2, 3, or 4 depending on the channel. Then, the ISR callback determines if the ISR trigger was a rising or falling edge. Whenever a falling edge is found, the difference between the current tick count and the previous tick count is computed. The variable `widthn` contains the duration of the pulse-width in microseconds

8.4 Hardware Drivers

Each hardware peripheral has its own associated driver. Drivers are implemented for the following sets of components:

- BLDC motor with Hall sensors (class `BLDC_Hall`)
- Quadrature Encoder (class `QuadEncoder`)
- BNO055 Inertial Measurement Unit (class `BNO055`)
- Radio Receiver with PWM output (class `RX`)

Separation of the hardware drivers into separate classes allows the designer to easily interface with the hardware in `main` without too much clutter, and also allows code reuse for devices such as the BLDC motors and encoders.

8.5 Firmware Documentation

The preceding discussion is meant only to cover the general operation of the firmware. All of the code is documented thoroughly through a utility called Doxygen; detailed firmware documentation can be found in Appendix F.

Chapter 9

SIMULATION RESULTS

Preliminary testing was performed through simulation using MATLAB® and Simulink®. Simulated testing was broken down into four main simulations. These four simulations verify that the model behaves properly, clearing the way for physical hardware testing. For each simulation mentioned below, a full set of plots is available in Appendix E.

Simulation #1: 2-DOF open-loop response

The first simulation aims to validate the open-loop 2-DOF model of the system, as derived in Chapter 2. This test shows that the simulated response of the system matches the physical behavior expected from the 2-DOF model when there is no control action.

The initial conditions for simulation #1 represent a stationary balance bot, nearly upright, with a small deviation in pitch angle. The initial conditions are defined as

$$\boldsymbol{x}_0 = \begin{bmatrix} 0 \\ \frac{\pi}{24} \\ 0 \\ 0 \end{bmatrix}.$$

Consider Figure 9.1 which depicts the pitch angle from the 2-DOF open-loop response of simulation #1. In open-loop, the balance bot behaves similar to an inverted pendulum, resulting in steady oscillation about a mean pitch angle of π $[rad]$; the steady oscillations are visible in Figure 9.1. The system model does not include any damping or friction, so there is no energy dissipation and therefore no reduction in amplitude as the balance bot oscillates.

See Section E.1 for a complete set of results for simulation #1.

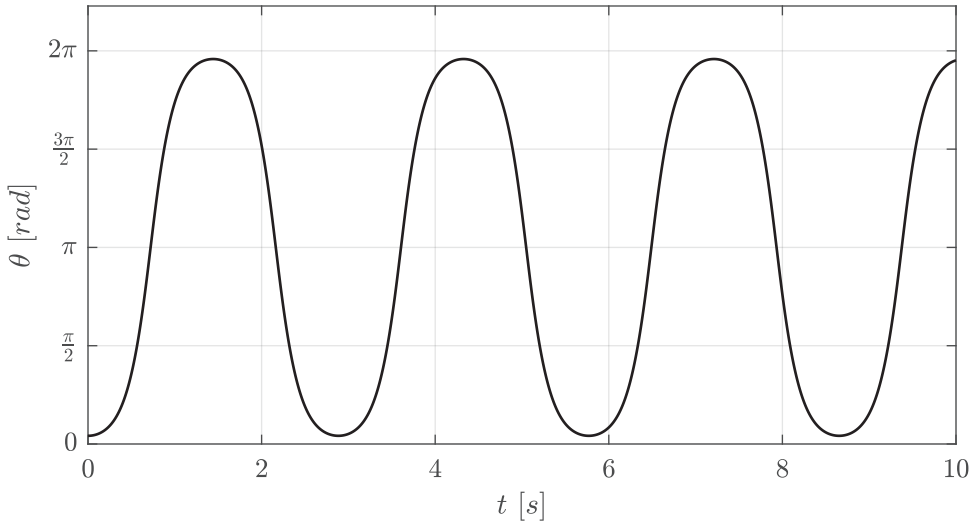


Figure 9.1: Open-loop response for 2-DOF model: pitch angle, θ.

Simulation #2: 2-DOF closed-loop response

The second simulation aims to validate the 2-DOF closed-loop model of the system, as presented in Chapter 3. This test shows that the 2-DOF model can be regulated effectively using a linear-quadratic regulator. This test additionally shows that even though the linearized model is used to calculate controller gains, the LQR is still effective when applied to the nonlinear model of the balance bot when the system state is near the equilibrium point used for linearization.

The initial conditions for simulation #2 are equal to those of simulation #1 and are defined as

$$\boldsymbol{x}_0 = \begin{bmatrix} 0 \\ \frac{\pi}{24} \\ 0 \\ 0 \end{bmatrix}.$$

Consider Figure 9.2 which depicts the pitch angle of the balance bot from the 2-DOF closed-loop response of simulation #2. In closed-loop, the balance bot does not oscillate; instead, the state-feedback controller regulates the states of the 2-DOF model, reducing the states to zero as time progresses.

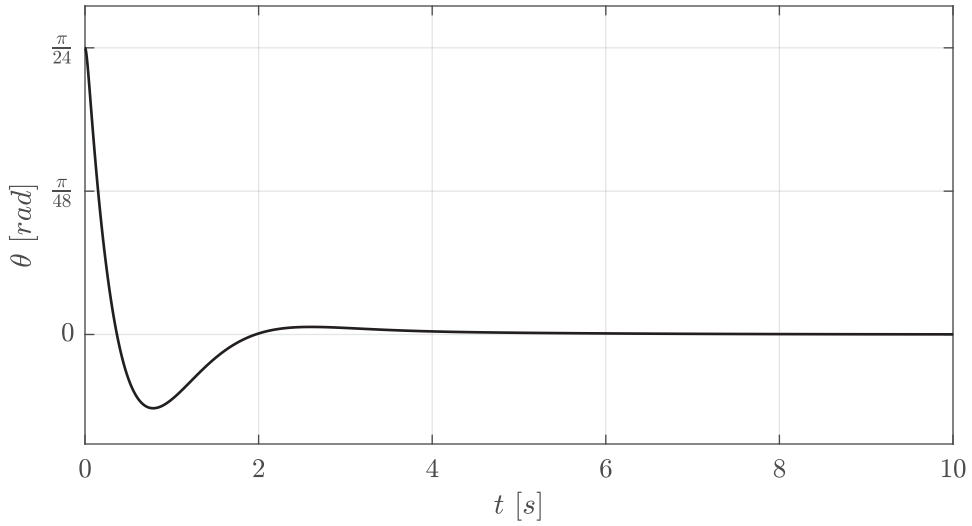


Figure 9.2: Closed-loop response for 2-DOF model: pitch angle, θ.

See Section E.2 for a complete set of results for simulation #2.

Simulation #3: 3-DOF closed-loop response

The third test aims to confirm the equivalence of the 2-DOF and 3-DOF models in closed-loop. That is, the test shows that when only the horizontal and pitch axes are considered, the 3-DOF model reduces to the 2-DOF model. This test additionally illustrates the independence of the yaw axis from the horizontal and pitch axes.

The initial conditions for simulation #3 are equivalent to those of simulations #1 and #2, but are augmented to represent all six states present in the 3-DOF model. The initial conditions are defined as

$$\boldsymbol{x}_0 = \begin{bmatrix} 0 \\ \frac{\pi}{24} \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

Consider Figure 9.3 which depicts the pitch angle of the balance bot from both the 2-DOF and the 3-DOF closed-loop responses of simulation #3. It is clearly evident from this plot that the results from the 2-DOF and 3-DOF models are consistent.

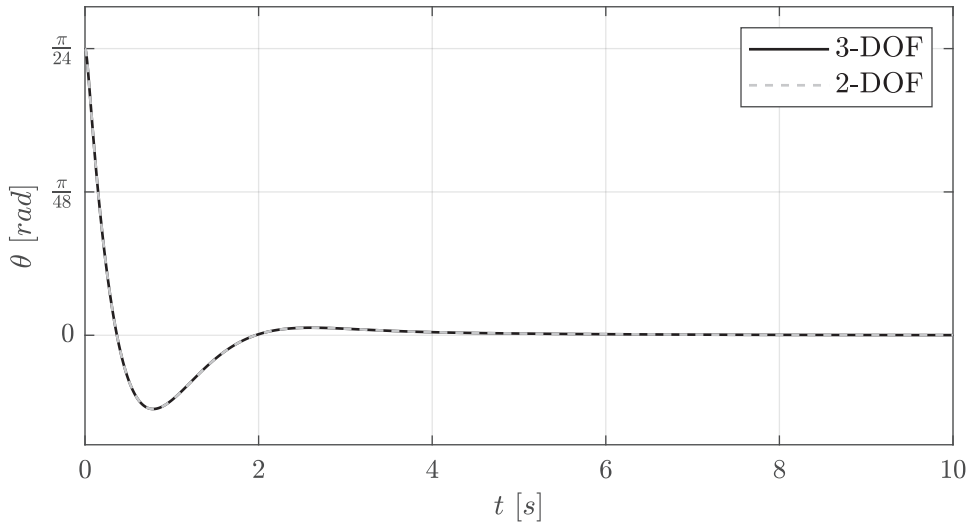


Figure 9.3: Closed-loop response for 2-DOF and 3-DOF models: pitch angle, θ.

See Section E.3 for a complete set of results for simulation #3.

Simulation #4: 3-DOF tracking response

The final simulation aims to show that the prefilter is effective. The prefilter does two things: apply a low-pass filter to the input signals from the throttle and steering commands, and manipulate the filtered throttle and steering commands to produce a reference state corresponding to each state variable.

The initial conditions for simulation #4 are all zero. That is, the initial conditions are defined as

$$\boldsymbol{x}_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

Consider Figure 9.4 and Figure 9.5. Figure 9.4 depicts the throttle command used as the tracking profile for simulation #4 and Figure 9.5 depicts the filtered throttle command and the horizontal velocity response from simulation #4.

The prefilter dramatically attenuates the high-frequency components of the throttle command. To follow the filtered throttle command, the balance bot requires comparatively less acceleration and therefore less torque, than would be required to follow the unfiltered throttle command. The reduction in required torque allows the balance bot to more easily achieve the desired reference profile.

See Section E.4 for a complete set of simulation results.

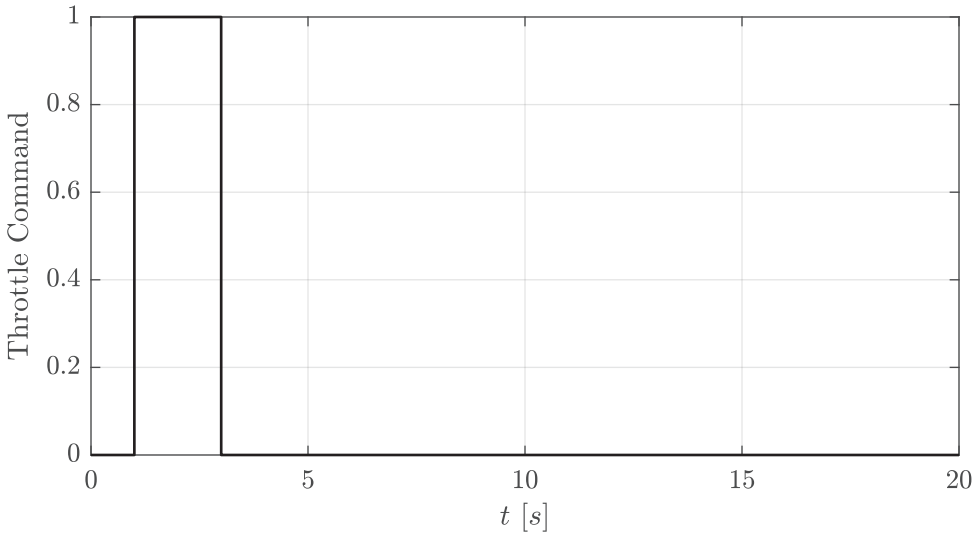


Figure 9.4: Throttle command for 3-DOF model in tracking configuration.

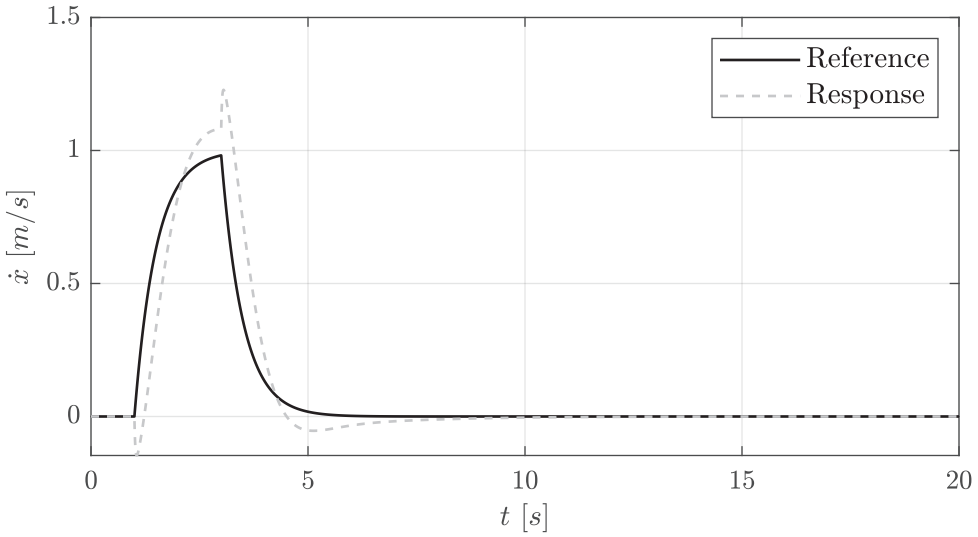


Figure 9.5: Tracking response for 3-DOF model: horizontal displacement, $\dot{x}$.

9.1 Animating the Balance Bot

Considerable intuition can be gained by creating an animation to portray the time-dependent simulation results. The response from each simulation is used to produce an animation showing the real-time motion of the balance bot. These animations are produced by using the advanced plot

functions in MATLAB® within a loop. A single frame of one animation is shown in Figure 9.6. The animation is an effective way to qualitatively and visually evaluate the simulation results[1].

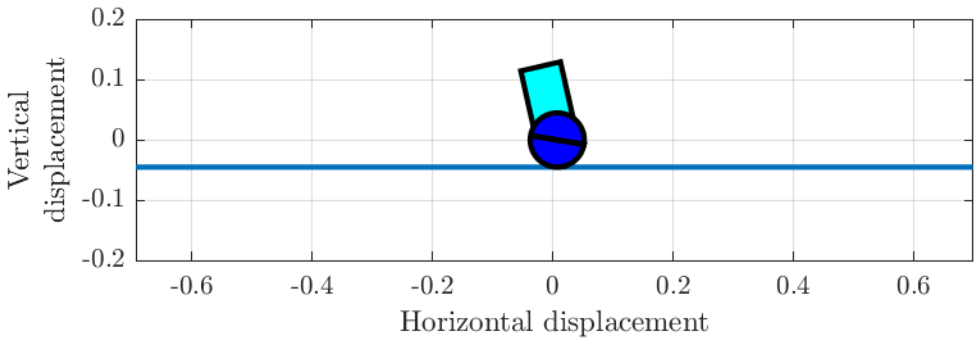


Figure 9.6: Still frame from animation of open-loop response of 2-DOF model.

[1]The animation results help confirm intuition. Of course this supposed confirmation can be misleading because sometimes real physical systems behave in counter-intuitive ways. When rapidly iterating through modeling changes, the animation can be an easy visual way to see the overall change in behavior.

Chapter 10

HARDWARE TESTING RESULTS

This final chapter covers the hardware testing performed with the fully-assembled balance bot. To facilitate data collection, an external Bluetooth module was added to the balance bot to perform serial transmission of data back to a host PC. During data collection, the host PC received the serial data through a terminal window in PuTTY. Each set of data collected on the PC was then saved as a time-stamped CSV file. Each CSV file was loaded into MATLAB® to generate plots of the system response. This method of data collection was used throughout the tuning process to rapidly iterate on gain selection while viewing the system response plots on the host PC.

Note: The data presented in this chapter represents the *transformed* states. That is, the plots show the values of $\hat{x}_1 \dots \hat{x}_6$, which are the state variables as measured by the physical hardware. Refer to Section 7.2 for additional details on the transformed state vector $\hat{\boldsymbol{x}}$.

Consider the testing results shown in Figures 10.1 through 10.6. These figures show 100 $[s]$ of collected data during remotely controlled operation. During the remotely operated testing, the balance bot was subject to external influences and disturbances such as uneven and dirty flooring. Even with the external disturbances, the key tracking results are visible in Figure 10.1 and Figure 10.3. These two figures depict the tracking results for the motor angle and yaw angle, respectively. Examination of these plots clearly shows that the controller successfully tracks the desired throttle and steering commands provided by the RC remote control.

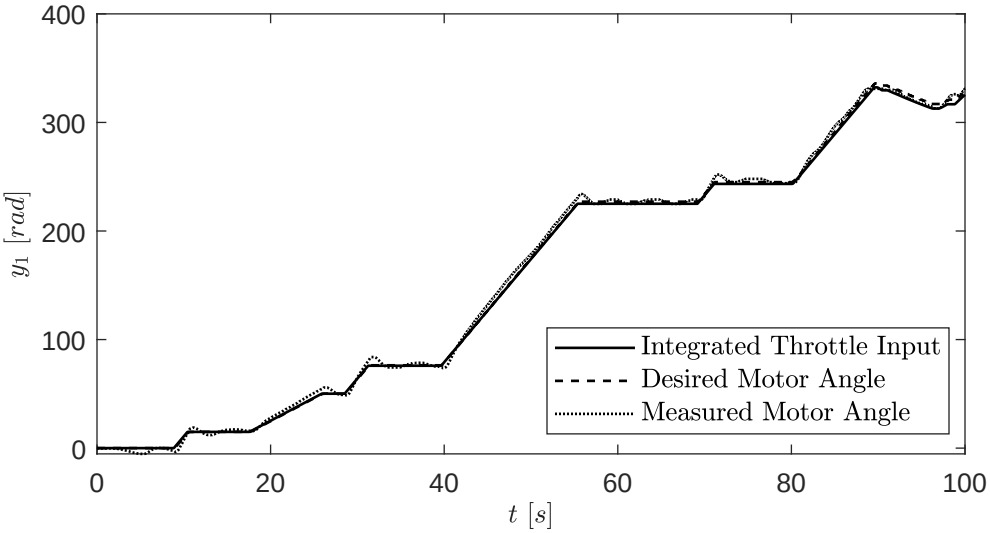


Figure 10.1: Measured system response: motor shaft angle, $\hat{x}_1$.

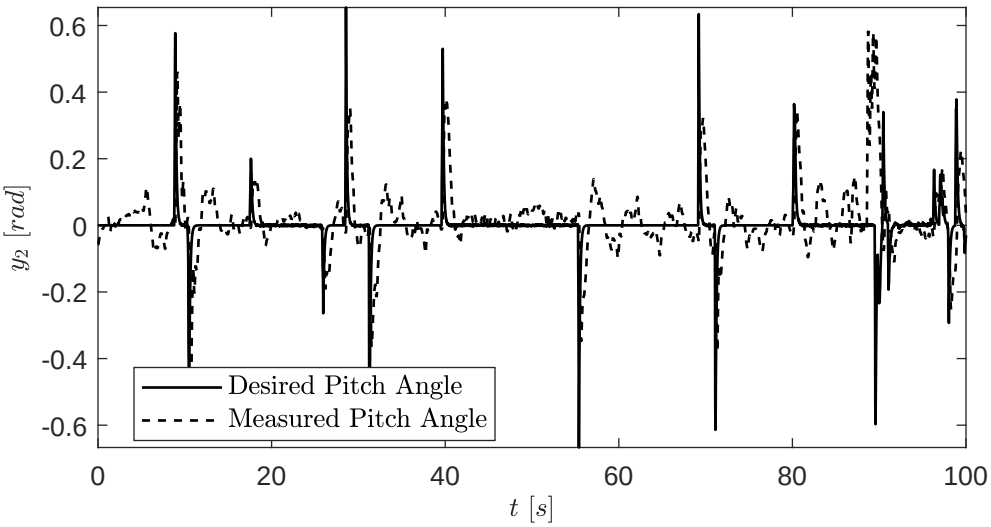


Figure 10.2: Measured system response: pitch angle, $\hat{x}_2$.

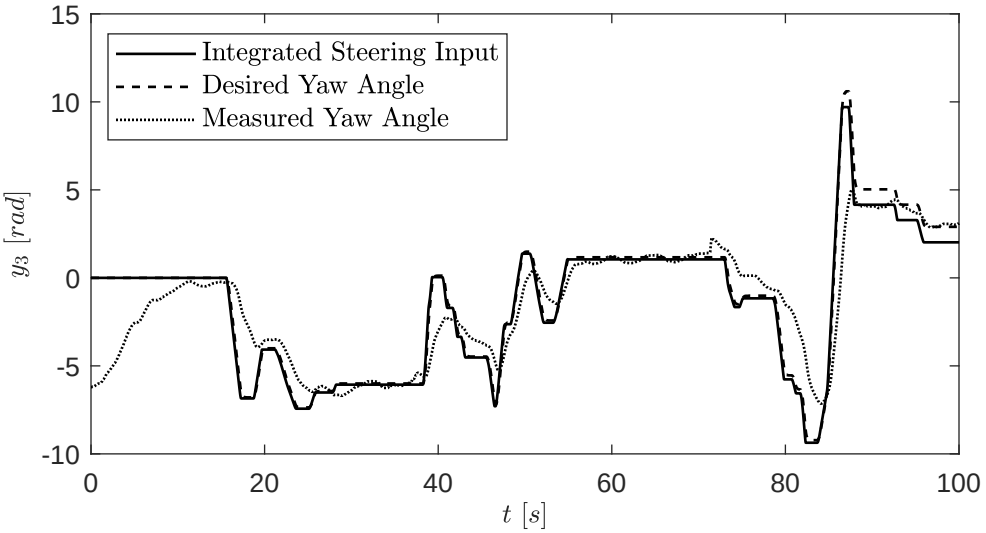


Figure 10.3: Measured system response: yaw angle, $\hat{x}_3$.

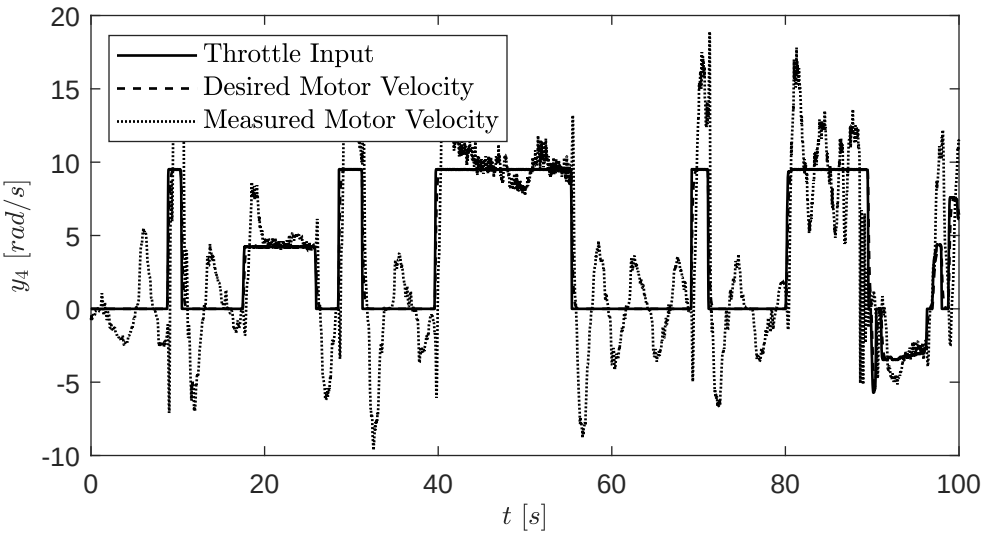


Figure 10.4: Measured system response: motor shaft velocity, $\hat{x}_4$.

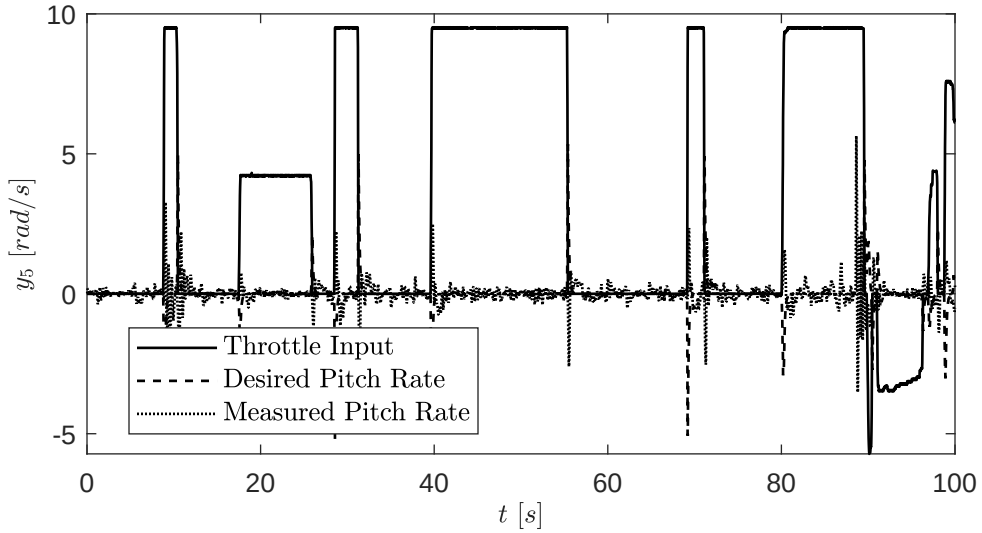


Figure 10.5: Measured system response: pitch velocity, $\hat{x}_5$.

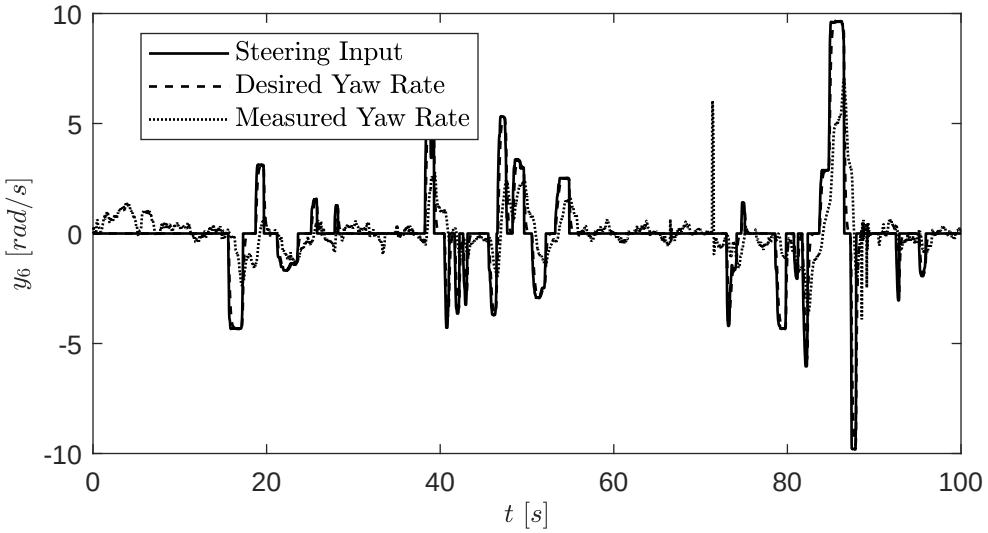


Figure 10.6: Measured system response: yaw velocity, $\hat{x}_6$.

Consider the plots shown in Figures 10.7 through 10.12. These figures show the same collection of data as the previous plots, but focus on only 30 $[s]$ of data during the middle of the test duration. The zoomed-in plots more clearly show the behavior of the balance bot during steady periods of motion.

Consider Figure 10.10 specifically: at first it may not appear that the results track effectively. However, recognizing that the bot must actively move to remain upright, the results are reasonable. The plot shows the transformed state $\hat{x}_4$ representing the motor shaft velocity. The transformed state vector $\hat{x}_4$ includes effects of the pitch velocity, $\dot{\theta}$, which are effectively overlaid on top of the horizontal velocity. From this perspective Figure 10.10 shows well performing tracking results with overlaid oscillation due to the active balancing.

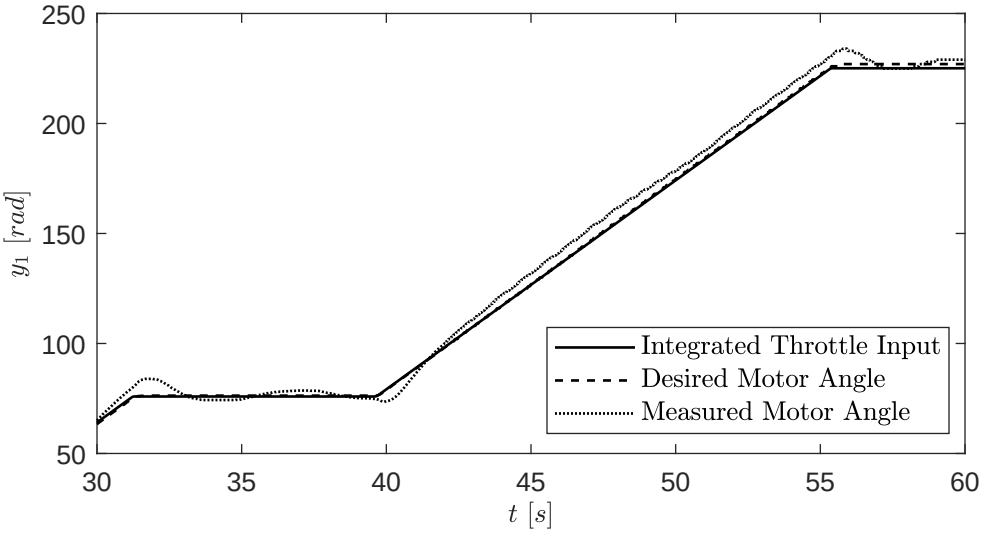


Figure 10.7: Measured system response: motor shaft angle, $\hat{x}_1$.

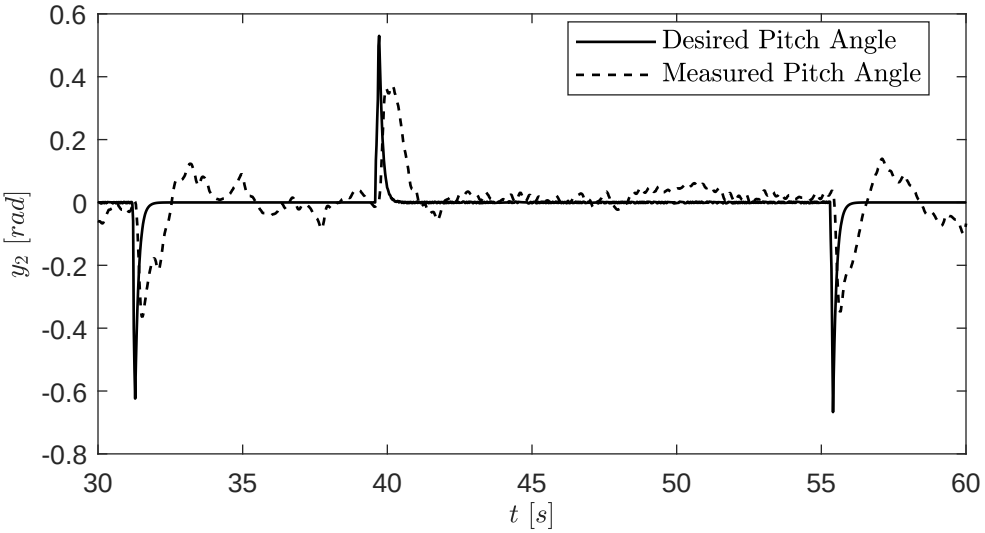


Figure 10.8: Measured system response: pitch angle, $\hat{x}_2$.

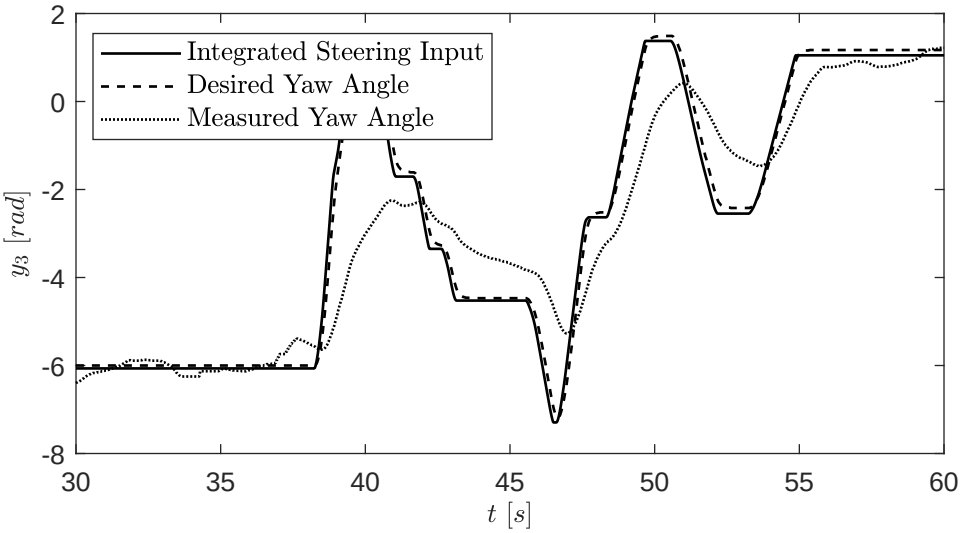


Figure 10.9: Measured system response: yaw angle, $\hat{x}_3$.

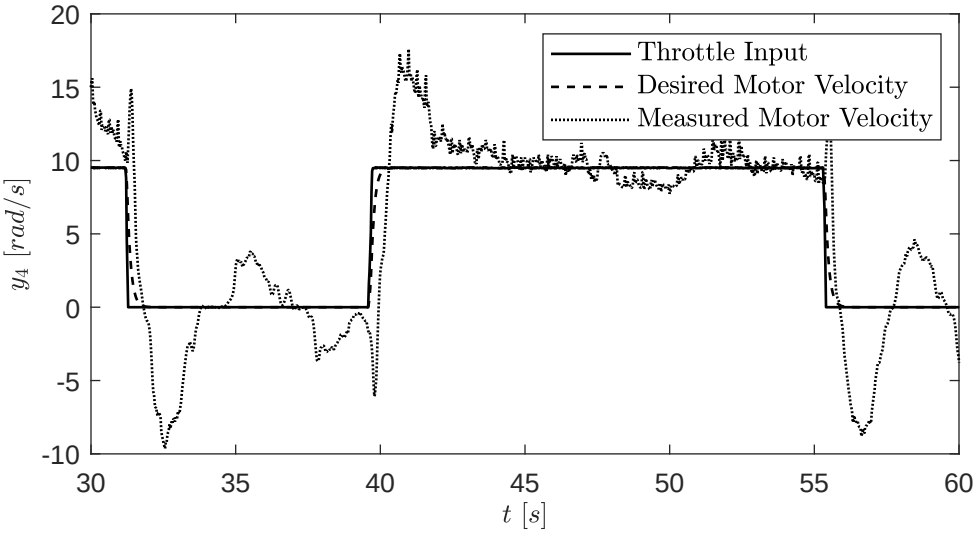


Figure 10.10: Measured system response: motor shaft velocity, $\hat{x}_4$.

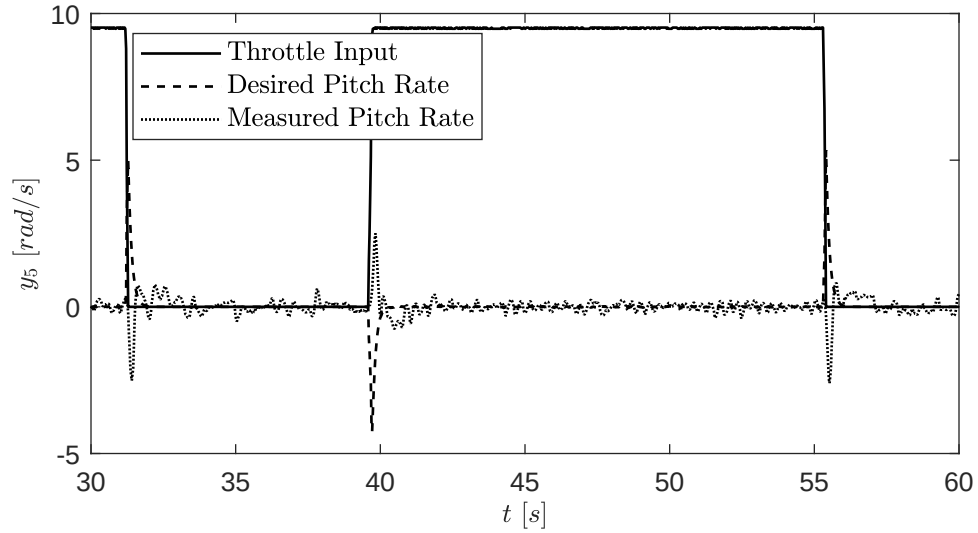


Figure 10.11: Measured system response: pitch velocity, $\hat{x}_5$.

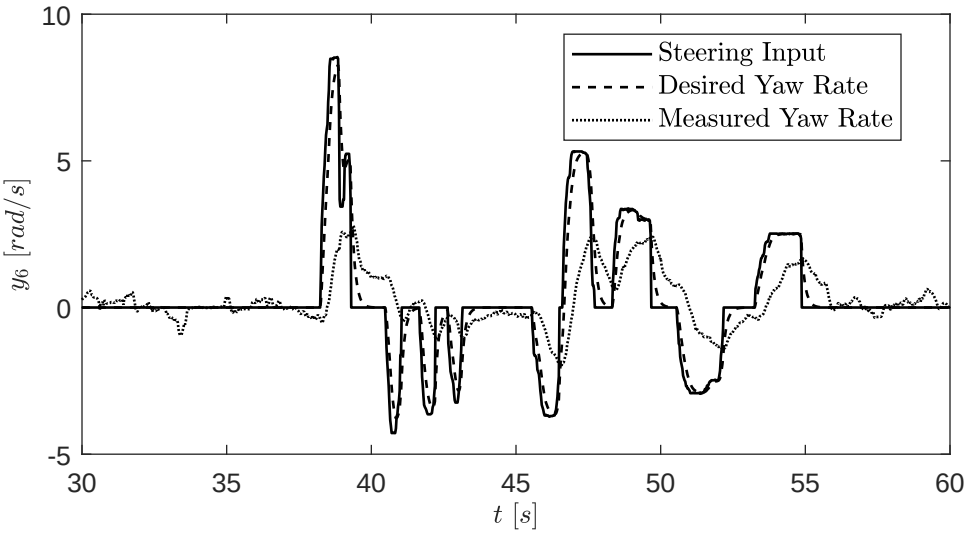


Figure 10.12: Measured system response: yaw velocity, $\hat{x}_6$.

Chapter 11

CONCLUSIONS AND RECOMMENDATIONS

Simulation and hardware testing show the effectiveness of the balance bot, but do not portray the dynamic quality that can only be observed by witnessing the bot balance remain upright while driving and steering. This chapter discusses brief qualitative analysis of the balance bot performance flaws and recommends areas for further research and testing.

In summary, the balance bot described in this document is able to stably balance and track external steering commands both in simulation and through physical testing. The hardware design supports many different possible controller architectures and can be used in the future to investigate other types of modern control theory. The prefilter is a novel introduction through this effort and effectively attenuates the high frequency components of the input command signals, improving the tracking response.

Nonetheless, listed below are several recommendations for future work that would benefit the balance bot.

11.1 Tracking versus Regulating

One of the main shortcomings of the balance bot is inconsistent operation across a wide range of input commands for a given set of controller gains. In order to track a reference profile commanded wirelessly by the operator, the feedback controller must be aggressively tuned in order to quickly adapt to fast changes in setpoint.

An aggressively tuned tracking controller does not perform well when the controller is used to regulate the state of the system back to zero. When the operator provides no throttle or steering commands, the tracking controller effectively reduces to a regulator; in these circumstances, the aggressive tuning causes small oscillations about the setpoint. For the controller to work effectively as a regulator, the gains must be reduced to such a degree that tracking performance suffers.

The prefilter, discussed in detail in Chapter 7, mitigates some performance problems by filtering out abrupt changes in user input, allowing the controller to be tuned more conservatively without losing

performance. The prefilter could be adjusted to remove even more high-frequency content from the input commands, but then the operator may experience lag in the response from the balance bot.

One recommended solution is gain scheduling. Gain scheduling is a method that allows the tuning of the controller to change depending on the current operating state of the system. Gain scheduling would let the controller behave aggressively when the input commands are highly dynamic, while allowing the controller to relax when the input commands are less dynamic.

11.2 Modeling for Control and Variable Structure Methods

The methods proposed previously in this document focus entirely on the application of linear control theory to a nonlinear system. Linear control theory is powerful, but comes with many restrictions and can be very limiting when it comes to controlling nonlinear systems.

Many other approaches exist for control of nonlinear systems, but one method proposed in [1] has significant promise for the balance-bot application. The proposed method combines many nonlinear modeling and control methods into one.

With the method proposed in [1], the controller design is considered during the modeling phase. When modeling for control purposes, the model becomes approximate and includes several uncertain parameters that are estimated in real time as the controller runs. The approximate system model is defined such that the model is in nonlinear controller canonical form; that is, the model is described such that the states form an integrator chain where the lowest-order state is the parameter to be controlled. All of the system states are derivatives of the control parameter and all of the system dynamics are associated with only the highest-order derivative of the control parameter.

As an example, consider how the 2-DOF model of the balance bot could be described in nonlinear controller canonical form. For the following example let the control parameter be the horizontal displacement of the center of mass of the balance bot body. That is, $y = x + r_b \sin\theta$. With this control parameter, the model for control would be of the following form:

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} x_2 \\ x_3 \\ x_4 \\ a(\boldsymbol{x}, \boldsymbol{p}) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ b(\boldsymbol{x}, \boldsymbol{p}) \end{bmatrix} \begin{bmatrix} u \end{bmatrix} \tag{11.1}$$

$$\begin{bmatrix} y \end{bmatrix} = \begin{bmatrix} x_1 \end{bmatrix} \tag{11.2}$$

Where the vector $\boldsymbol{p}$ contains uncertain parameters about the system determined by an estimation algorithm.

With the approximate model in controller canonical form, a controller can be designed that combines approaches from the following areas of modern control theory:

- Feedback linearization
- Feedforward control
- State-estimation and data-fusion
- Sliding mode control and model order reduction
- Generalized Lyapunov functions for stability analysis and switching control
- Nonlinear controller constraints such as actuator saturation

The methods proposed in [1] may greatly improve tracking performance and allow additional features out of the scope of linear control theory such as a controlled "flip up" in which the balance bot flips itself up from a resting position to automatically begin balancing under closed-loop control. Such a maneuver would be impossible with a linear controller due to the very large deviation from the operating point used during linearization.

11.3 Hardware and Firmware Limitations

The selected hardware implementation is able to effectively actuate the BLDC motors, read from sensors, and keep the bot balancing; however, there are some lingering issues in the final hardware

implementation. As mentioned in Appendix C, a final revision is designed but unassembled and untested. The final hardware iteration aims to eliminate the remaining problems associated with the hardware revision used for testing and data collection. A brief summary of pending changes is listed below:

- Many connectors must be adjusted to allow the cables to exit vertically from the bottom of the PCB to route through the internals of the bot. This will reduce the likelihood of damaging the cabling in the event the bot falls over.
- A resistor network for battery voltage measurement is needed.
- The top silkscreen must be adjusted to include a location where the RC receiver will be mounted.

The firmware has been implemented exclusively in MicroPython. MicroPython is an excellent choice because it accelerates firmware development and allows the design to focus more heavily on the control algorithm itself and less on the implementation of the control algorithm. However, significant timing constraints are present due to the selection of MicroPython. Re-implementation in a language such as C or C++ could improve the controller timing significantly and allow more sophisticated controller implementations and better multi-tasking if additional features are to be added.

BIBLIOGRAPHY

[1] *Variable-Structure Approaches: Analysis, Simulation, Robust Control and Estimation of Uncertain Dynamic Processes*. 2016.

[2] Allegro Microsystems. *High Sensitivity, 1 MHz, GMR-Based Current Sensor IC in Space-Saving, Low Resistance QFN and SOIC-8 Packages*, May 2019. Rev. 3.

[3] Alpha & Omega Semiconductor. *AOD417 P-Channel Enhancement Mode Field Effect Transistor*, September 2008. Rev. 1.

[4] K. J. Astrom. *PID Controllers: Theory, Design, and Tuning*. 2 edition, 1995.

[5] Bosch Sensortec. *Intelligent 9-axis absolute orientation sensor*, June 2016. Rev. 1.4.

[6] W. L. Brogan. *Modern Control Theory*. 3 edition, 1991.

[7] S.-Y. J. Ching-Chih Tsai and S.-M. Hsieh. Trajectory tracking of a self-balancing two-wheeled robot using backstepping sliding-mode control and fuzzy basis function networks. 2010.

[8] Diodes Incorporated. *Low Power Hall Effect Switch*, November 2009. Rev. 8.

[9] R. Feynman. The principle of least action. 19, 2006.

[10] J. F. Gieras and M. Wing. *Permanent Magnet Motor Technology*. 1997.

[11] D. Jayakody and K. Sucharitharathna. Control unit for a two-wheel self-balancing robot. XIX, 2019.

[12] P. Lancaster and L. Rodman. *Algebraic Riccati Equations*. 01 2002.

[13] K. H. Nam. *AC Motor Control and Electric Vehicle Applications*. 2010.

[14] R. C. Ooi. Balancing a two-wheeled autonomous robot. Master's thesis, 2003.

[15] D. Rowell and D. N. Wormley. *System Dynamics*. 1 edition, October 1996.

[16] ST Microelectronics. *STM32 Nucleo-64 boards (MB1136)*, April 2019. Rev. 13.

[17] ST Microelectronics. *Ultra-low-power Arm Cortex-M4 32-bit MCU+FPU, 100DMIPS, up to 1MB Flash, 128 KB SRAM, USB OTG FS, LCD, ext. SMPS*, June 2019. Rev. 8.

[18] Texas Instruments. *DRV8313 2.5-A Triple 1/2-H Bridge Driver*, April 2016.

[19] Traxxas. *TQi Top Qualifier Owners Manual*, 2012.

APPENDICES

Appendix A

MODELING DETAILS

This appendix shows the detailed derivation of the Lagrangian and equations of motion for both the 2-DOF and 3-DOF system models.

A.1 2-DOF Model - Kinetic Energy Derivation

This section focuses on the derivation of Equation 2.4, which expresses the kinetic energy used in the derivation for the 2-DOF balance bot model. The kinetic energy must be expressed solely in terms of $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$ to use Lagrange's equation, so the kinematics must be worked out in terms of $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$ as well.

The linear velocity at the center of the wheel is only due to the velocity in the x direction.

$$\boldsymbol{v}_w = \begin{bmatrix} \dot{x} \\ 0 \\ 0 \end{bmatrix} \tag{A.1}$$

The angular velocity at the center of each wheel is due to the wheel rolling without slip on a horizontal surface.

$$\boldsymbol{v}_w = \boldsymbol{\omega}_w \times \boldsymbol{r}_w \tag{A.2}$$

$$\begin{bmatrix} \dot{x} \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} r_w \omega_{w,y} \\ 0 \\ 0 \end{bmatrix} \tag{A.3}$$

$$\boldsymbol{\omega}_w = \begin{bmatrix} 0 \\ \frac{\dot{x}}{r_w} \\ 0 \end{bmatrix} \tag{A.4}$$

The angular velocity of the body is due to the pitch velocity.

$$\boldsymbol{\omega}_b = \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} \tag{A.5}$$

The linear velocity of the body is due to the velocity of the wheel and the relative velocity of the body with respect to the wheel.

$$\boldsymbol{v}_b = \boldsymbol{v}_w + \boldsymbol{v}_{b/w} \tag{A.6}$$

$$\boldsymbol{v}_b = \boldsymbol{v}_w + \boldsymbol{\omega}_b \times \boldsymbol{r}_{b/w} \tag{A.7}$$

$$\boldsymbol{v}_b = \begin{bmatrix} \dot{x} \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} r_b \cos\theta\, \dot{\theta} \\ 0 \\ -r_b \sin\theta\, \dot{\theta} \end{bmatrix} \tag{A.8}$$

$$\boldsymbol{v}_b = \begin{bmatrix} \dot{x} + r_b \cos\theta\, \dot{\theta} \\ 0 \\ -r_b \sin\theta\, \dot{\theta} \end{bmatrix} \tag{A.9}$$

The kinetic energy depends not only on the velocity of each body, but also the mass moment of inertia of each body. The moment of inertia is trivial for the wheels because they are each rotating about an axis of symmetry and are themselves bodies of revolution.

$$\mathbb{I}_w = \begin{bmatrix} I_{w,xx} & 0 & 0 \\ 0 & I_{w,yy} & 0 \\ 0 & 0 & I_{w,zz} \end{bmatrix} \tag{A.10}$$

The mass moment of inertia of the body is more complicated. The principal axes of the body are assumed to be aligned with the body, therefore when the pitch angle θ is nonzero, the moment of inertia tensor must be rotated by θ using a coordinate transformation.

$$\mathbb{I}_b = R_\theta \left(\mathbb{I}_b\right)_{xyz} R_\theta^\top \tag{A.11}$$

$$\mathbb{I}_b = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} I_{b,xx} & 0 & 0 \\ 0 & I_{b,yy} & 0 \\ 0 & 0 & I_{b,zz} \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \tag{A.12}$$

$$\mathbb{I}_b = \begin{bmatrix} I_{b,xx}\cos^2\theta + I_{b,zz}\sin^2\theta & 0 & -I_{b,xx}\sin\theta\cos\theta + I_{b,zz}\sin\theta\cos\theta \\ 0 & I_{b,yy} & 0 \\ -I_{b,xx}\sin\theta\cos\theta + I_{b,zz}\sin\theta\cos\theta & 0 & I_{b,xx}\sin^2\theta + I_{b,zz}\cos^2\theta \end{bmatrix} \tag{A.13}$$

The kinetic energy in the system can be computed using the linear and angular velocities and moment of inertia tensors defined above.

$$T = \frac{1}{2} m_b \, \boldsymbol{v}_b^\top \boldsymbol{v}_b + \frac{1}{2} \, \boldsymbol{\omega}_b^\top \mathbb{I}_b \boldsymbol{\omega}_b + 2 \left(\frac{1}{2} m_w \, \boldsymbol{v}_w^\top \boldsymbol{v}_w + \frac{1}{2} \, \boldsymbol{\omega}_w^\top \mathbb{I}_w \boldsymbol{\omega}_w \right) \tag{A.14}$$

$$\begin{aligned} T = {} & \frac{1}{2} m_b \begin{bmatrix} \dot{x} + r_b \cos\theta \, \dot{\theta} & 0 & -r_b \sin\theta \, \dot{\theta} \end{bmatrix} \begin{bmatrix} \dot{x} + r_b \cos\theta \, \dot{\theta} \\ 0 \\ -r_b \sin\theta \, \dot{\theta} \end{bmatrix} \\ & + \frac{1}{2} \begin{bmatrix} 0 & \dot{\theta} & 0 \end{bmatrix} \begin{bmatrix} I_{b,xx} \cos^2\theta + I_{b,zz} \sin^2\theta & 0 & -I_{b,xx} \sin\theta\cos\theta + I_{b,zz} \sin\theta\cos\theta \\ 0 & I_{b,yy} & 0 \\ -I_{b,xx} \sin\theta\cos\theta + I_{b,zz} \sin\theta\cos\theta & 0 & I_{b,xx} \sin^2\theta + I_{b,zz} \cos^2\theta \end{bmatrix} \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} \\ & + m_w \begin{bmatrix} \dot{x} & 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{x} \\ 0 \\ 0 \end{bmatrix} \\ & + \begin{bmatrix} 0 & \frac{\dot{x}}{r_w} & 0 \end{bmatrix} \begin{bmatrix} I_{w,xx} & 0 & 0 \\ 0 & I_{w,yy} & 0 \\ 0 & 0 & I_{w,zz} \end{bmatrix} \begin{bmatrix} 0 \\ \frac{\dot{x}}{r_w} \\ 0 \end{bmatrix} \end{aligned} \tag{A.15}$$

$$\begin{aligned} T = {} & \frac{1}{2} m_b \left(\dot{x} + r_b \cos\theta \, \dot{\theta} \right)^2 + \frac{1}{2} m_b \left(-r_b \sin\theta \, \dot{\theta} \right)^2 + \frac{1}{2} I_{b,yy} \dot{\theta}^2 \\ & + m_w \dot{x}^2 + I_{w,yy} \left(\frac{\dot{x}}{r_w} \right)^2 \end{aligned} \tag{A.16}$$

$$\begin{aligned} T = {} & \frac{1}{2} m_b \dot{x}^2 + m_b r_b \dot{x} \dot{\theta} \cos\theta + \frac{1}{2} m_b r_b^2 \cos^2\theta \, \dot{\theta}^2 + \frac{1}{2} m_b r_b^2 \sin^2\theta \, \dot{\theta}^2 + \frac{1}{2} I_{b,yy} \dot{\theta}^2 \\ & + m_w \dot{x}^2 + \frac{I_{w,yy}}{r_w^2} \dot{x}^2 \end{aligned} \tag{A.17}$$

$$T = \frac{1}{2} \left(m_b + 2 m_w + 2 \frac{I_{w,yy}}{r_w^2} \right) \dot{x}^2 + m_b r_b \cos\theta \, \dot{x} \dot{\theta} + \frac{1}{2} \left(m_b r_b^2 + I_{b,yy} \right) \dot{\theta}^2 \tag{A.18}$$

$$T = \frac{1}{2} m_x \dot{x}^2 + m_b r_b \cos\theta \, \dot{x} \dot{\theta} + \frac{1}{2} I_\theta \dot{\theta}^2 \tag{A.19}$$

Where,

$$m_x = m_b + 2 m_w + 2 \frac{I_{w,yy}}{r_w^2},$$

$$I_\theta = m_b r_b^2 + I_{b,yy}$$

A.2 2-DOF Model - Equations of Motion Derivation

This section covers the derivation of the 2-DOF equations of motion using Lagrange's equation. Recall that the Lagrangian is defined as the difference between kinetic and potential energy; that is, $L = T - V$. The potential energy in the system is due to the elevation of the body of the balance bot and can be expressed as $V = m_b r_b \cos\theta\, g$. With the Lagrangian defined, Lagrange's equation is used to find the equations of motion.

The Lagrangian is defined as the difference between kinetic and potential energy.

$$L = T - V \tag{A.20}$$

$$L = \frac{1}{2} m_x \dot{x}^2 + m_b r_b \cos\theta\, \dot{x}\dot{\theta} + \frac{1}{2} I_\theta \dot{\theta}^2 - m_b r_b \cos\theta\, g \tag{A.21}$$

L is first differentiated with respect to x.

$$\frac{\partial L}{\partial x} = \frac{\partial}{\partial x}\left(\frac{1}{2} m_x \dot{x}^2 + m_b r_b \cos\theta\, \dot{x}\dot{\theta} + \frac{1}{2} I_\theta \dot{\theta}^2 - m_b r_b \cos\theta\, g\right) \tag{A.22}$$

$$\frac{\partial L}{\partial x} = 0 \tag{A.23}$$

Then, L is differentiated with respect to $\dot{x}$.

$$\frac{\partial L}{\partial \dot{x}} = \frac{\partial}{\partial \dot{x}}\left(\frac{1}{2} m_x \dot{x}^2 + m_b r_b \cos\theta\, \dot{x}\dot{\theta} + \frac{1}{2} I_\theta \dot{\theta}^2 - m_b r_b \cos\theta\, g\right) \tag{A.24}$$

$$\frac{\partial L}{\partial \dot{x}} = m_x \dot{x} + m_b r_b \cos\theta\, \dot{\theta} \tag{A.25}$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{x}}\right) = \frac{d}{dt}\left(m_x \dot{x} + m_b r_b \cos\theta\, \dot{\theta}\right) \tag{A.26}$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{x}}\right) = m_x \ddot{x} + m_b r_b \cos\theta\, \ddot{\theta} - m_b r_b \sin\theta\, \dot{\theta}^2 \tag{A.27}$$

And then with respect to θ.

$$\frac{\partial L}{\partial \theta} = \frac{\partial}{\partial \theta}\left(\frac{1}{2} m_x \dot{x}^2 + m_b r_b \cos\theta\, \dot{x}\dot{\theta} + \frac{1}{2} I_\theta \dot{\theta}^2 - m_b r_b \cos\theta\, g\right) \tag{A.28}$$

$$\frac{\partial L}{\partial \theta} = -m_b r_b \sin\theta\, \dot{x}\dot{\theta} + m_b r_b \sin\theta\, g \tag{A.29}$$

And finally with respect to $\dot{\theta}$.

$$\frac{\partial L}{\partial \dot{\theta}} = \frac{\partial}{\partial \dot{\theta}} \left(\frac{1}{2} m_x \dot{x}^2 + m_b r_b \cos\theta \, \dot{x}\dot{\theta} + \frac{1}{2} I_\theta \dot{\theta}^2 - m_b r_b \cos\theta \, g \right) \tag{A.30}$$

$$\frac{\partial L}{\partial \dot{\theta}} = m_b r_b \cos\theta \, \dot{x} + I_\theta \dot{\theta} \tag{A.31}$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}} \right) = \frac{d}{dt}\left(m_b r_b \cos\theta \, \dot{x} + I_\theta \dot{\theta} \right) \tag{A.32}$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}} \right) = m_b r_b \cos\theta \, \ddot{x} - m_b r_b \sin\theta \, \dot{x}\dot{\theta} + I_\theta \ddot{\theta} \tag{A.33}$$

The first equation of motion is for the x-axis and can be determined using Lagrange's equation applied to the q_1 coordinate.

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{x}} \right) - \frac{\partial L}{\partial x} = Q_x \tag{A.34}$$

$$m_x \ddot{x} + m_b r_b \cos\theta \, \ddot{\theta} - m_b r_b \sin\theta \, \dot{\theta}^2 = -2\frac{1}{r_w} T_m \tag{A.35}$$

$$m_x \ddot{x} + m_b r_b \cos\theta \, \ddot{\theta} = -2\frac{1}{r_w} T_m + m_b r_b \sin\theta \, \dot{\theta}^2 \tag{A.36}$$

The second equation of motion is for the pitch axis and can be determined using Lagrange's equation applied to the q_2 coordinate.

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}} \right) - \frac{\partial L}{\partial \theta} = Q_\theta \tag{A.37}$$

$$m_b r_b \cos\theta \, \ddot{x} - m_b r_b \sin\theta \, \dot{x}\dot{\theta} + I_\theta \ddot{\theta} + m_b r_b \sin\theta \, \dot{x}\dot{\theta} - m_b r_b \sin\theta \, g = 2T_m \tag{A.38}$$

$$m_b r_b \cos\theta \, \ddot{x} + I_\theta \ddot{\theta} = 2T_m + m_b r_b \sin\theta \, g \tag{A.39}$$

A.3 3-DOF Model - Kinetic Energy Derivation

For the 3-DOF model, the linear velocity of the wheel is also due to the velocity in the x direction, but has an additional component due to the yaw rotation about the vertical axis. For the right wheel:

$$\boldsymbol{v}_{w,r} = \boldsymbol{v}_c + \boldsymbol{v}_{w,r/c} \tag{A.40}$$

$$\boldsymbol{v}_{w,r} = \boldsymbol{v}_c + \boldsymbol{\omega}_{w,r} \times \boldsymbol{r}_{w,r/c} \tag{A.41}$$

$$\boldsymbol{v}_{w,r} = \begin{bmatrix} \dot{x} \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} \frac{w}{2}\dot{\psi} \\ 0 \\ 0 \end{bmatrix} \tag{A.42}$$

$$\boldsymbol{v}_{w,r} = \begin{bmatrix} \dot{x} + \frac{w}{2}\dot{\psi} \\ 0 \\ 0 \end{bmatrix} \tag{A.43}$$

And similarly for the left wheel:

$$\boldsymbol{v}_{w,r} = \begin{bmatrix} \dot{x} - \frac{w}{2}\dot{\psi} \\ 0 \\ 0 \end{bmatrix} \tag{A.44}$$

The angular velocities of the right wheel is due to the wheels rolling without slip and the rotation about the vertical axis.

$$\boldsymbol{v}_{w,r} = \boldsymbol{\omega}_{w,r} \times \boldsymbol{r}_{w,r} \tag{A.45}$$

$$\begin{bmatrix} \dot{x} + \frac{w}{2}\dot{\psi} \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} r_w \omega_{w,r,y} \\ 0 \\ 0 \end{bmatrix} \tag{A.46}$$

$$\boldsymbol{\omega}_{w,r} = \begin{bmatrix} 0 \\ \frac{\dot{x} + \frac{w}{2}\dot{\psi}}{r_w} \\ \dot{\psi} \end{bmatrix} \tag{A.47}$$

Similar analysis can be performed for the left wheel.

$$\boldsymbol{\omega}_{w,l} = \begin{bmatrix} 0 \\ \frac{\dot{x} - \frac{w}{2}\dot{\psi}}{r_w} \\ \dot{\psi} \end{bmatrix} \tag{A.48}$$

The angular velocity of the body is due to both the pitch angular velocity and the yaw angular velocity.

$$\boldsymbol{\omega}_b = \begin{bmatrix} 0 \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{A.49}$$

The linear velocity of the body is due to the velocity of the wheel and the relative velocity of the body with respect to the wheel.

$$\boldsymbol{v}_b = \boldsymbol{v}_c + \boldsymbol{v}_{b/\cos\theta} \tag{A.50}$$

$$\boldsymbol{v}_b = \boldsymbol{v}_c + \boldsymbol{\omega}_b \times \boldsymbol{r}_{b/\cos\theta} \tag{A.51}$$

$$\boldsymbol{v}_b = \begin{bmatrix} \dot{x} \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} r_b \cos\theta\, \dot{\theta} \\ r_b \sin\theta\, \dot{\psi} \\ -r_b \sin\theta\, \dot{\theta} \end{bmatrix} \tag{A.52}$$

$$\boldsymbol{v}_b = \begin{bmatrix} \dot{x} + r_b \cos\theta\, \dot{\theta} \\ r_b \sin\theta\, \dot{\psi} \\ -r_b \sin\theta\, \dot{\theta} \end{bmatrix} \tag{A.53}$$

The kinetic energy can be determined using the linear and angular velocities derived above.

$$
\begin{aligned}
T = {} & \frac{1}{2} m_b \, \boldsymbol{v}_b^\top \boldsymbol{v}_b + \frac{1}{2} \, \boldsymbol{\omega}_b^\top I_b \boldsymbol{\omega}_b \\
& + \frac{1}{2} m_w \, \boldsymbol{v}_{w,r}^\top \boldsymbol{v}_{w,r} + \frac{1}{2} \, \boldsymbol{\omega}_{w,r}^\top I_w \boldsymbol{\omega}_{w,r} \\
& + \frac{1}{2} m_w \, \boldsymbol{v}_{w,l}^\top \boldsymbol{v}_{w,r} + \frac{1}{2} \, \boldsymbol{\omega}_{w,l}^\top I_w \boldsymbol{\omega}_{w,l}
\end{aligned} \tag{A.54}
$$

$$
\begin{aligned}
T = {} & \frac{1}{2} m_b \begin{bmatrix} \dot{x} + r_b \cos\theta \, \dot{\theta} & r_b \sin\theta \, \dot{\psi} & -r_b \sin\theta \, \dot{\theta} \end{bmatrix} \begin{bmatrix} \dot{x} + r_b \cos\theta \, \dot{\theta} \\ r_b \sin\theta \, \dot{\psi} \\ -r_b \sin\theta \, \dot{\theta} \end{bmatrix} \\
& + \frac{1}{2} \begin{bmatrix} 0 & \dot{\theta} & \dot{\psi} \end{bmatrix} \begin{bmatrix} I_{b,xx} \cos^2\theta + I_{b,zz} \sin^2\theta & 0 & -I_{b,xx} \sin\theta\cos\theta + I_{b,zz} \sin\theta\cos\theta \\ 0 & I_{b,yy} & 0 \\ -I_{b,xx} \sin\theta\cos\theta + I_{b,zz} \sin\theta\cos\theta & 0 & I_{b,xx} \sin^2\theta + I_{b,zz} \cos^2\theta \end{bmatrix} \begin{bmatrix} 0 \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \\
& + \frac{1}{2} m_w \begin{bmatrix} \dot{x} + \frac{w}{2}\dot{\psi} & 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{x} + \frac{w}{2}\dot{\psi} \\ 0 \\ 0 \end{bmatrix} \\
& + \frac{1}{2} \begin{bmatrix} 0 & \frac{\dot{x} + \frac{w}{2}\dot{\psi}}{r_w} & \dot{\psi} \end{bmatrix} \begin{bmatrix} I_{w,xx} & 0 & 0 \\ 0 & I_{w,yy} & 0 \\ 0 & 0 & I_{w,zz} \end{bmatrix} \begin{bmatrix} 0 \\ \frac{\dot{x} + \frac{w}{2}\dot{\psi}}{r_w} \\ \dot{\psi} \end{bmatrix} \\
& + \frac{1}{2} m_w \begin{bmatrix} \dot{x} - \frac{w}{2}\dot{\psi} & 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{x} - \frac{w}{2}\dot{\psi} \\ 0 \\ 0 \end{bmatrix} \\
& + \frac{1}{2} \begin{bmatrix} 0 & \frac{\dot{x} - \frac{w}{2}\dot{\psi}}{r_w} & \dot{\psi} \end{bmatrix} \begin{bmatrix} I_{w,xx} & 0 & 0 \\ 0 & I_{w,yy} & 0 \\ 0 & 0 & I_{w,zz} \end{bmatrix} \begin{bmatrix} 0 \\ \frac{\dot{x} - \frac{w}{2}\dot{\psi}}{r_w} \\ \dot{\psi} \end{bmatrix}
\end{aligned} \tag{A.55}
$$

$$
\begin{aligned}
T = & \frac{1}{2} m_b \left(\left(\dot{x} + r_b \cos\theta \, \dot{\theta} \right)^2 + \left(r_b \sin\theta \, \dot{\psi} \right)^2 + \left(-r_b \sin\theta \, \dot{\theta} \right)^2 \right) \\
& + \frac{1}{2} \left(I_{b,yy} \dot{\theta}^2 + I_{b,xx} \sin^2\theta \, \dot{\psi}^2 + I_{b,zz} \cos^2\theta \, \dot{\psi}^2 \right) \\
& + \frac{1}{2} m_w \left(\dot{x} + \frac{w}{2} \dot{\psi} \right)^2 \\
& + \frac{1}{2} \left(I_{w,yy} \left(\frac{\dot{x} + \frac{w}{2}\dot{\psi}}{r_w} \right)^2 + I_{w,zz} \dot{\psi}^2 \right) \\
& + \frac{1}{2} m_w \left(\dot{x} - \frac{w}{2} \dot{\psi} \right)^2 \\
& + \frac{1}{2} \left(I_{w,yy} \left(\frac{\dot{x} - \frac{w}{2}\dot{\psi}}{r_w} \right)^2 + I_{w,zz} \dot{\psi}^2 \right)
\end{aligned} \tag{A.56}
$$

$$
\begin{aligned}
T = & \frac{1}{2} m_b \dot{x}^2 + m_b r_b \cos\theta \, \dot{x}\dot{\theta} + \frac{1}{2} m_b r_b^2 \cos^2\theta \, \dot{\theta}^2 + \frac{1}{2} m_b r_b^2 \sin^2\theta \, \dot{\psi}^2 + \frac{1}{2} m_b r_b^2 \sin^2\theta \, \dot{\theta}^2 \\
& + \frac{1}{2} I_{b,yy} \dot{\theta}^2 + \frac{1}{2} I_{b,xx} \sin^2\theta \, \dot{\psi}^2 + \frac{1}{2} I_{b,zz} \cos^2\theta \, \dot{\psi}^2 \\
& + \frac{1}{2} m_w \dot{x}^2 + m_w \frac{w}{2} \dot{x}\dot{\psi} + \frac{1}{2} m_w \frac{w^2}{4} \dot{\psi}^2 \\
& + \frac{1}{2} \frac{I_{w,yy}}{r_w^2} \dot{x}^2 + \frac{I_{w,yy}}{r_w^2} \frac{w}{2} \dot{x}\dot{\psi} + \frac{1}{2} \frac{I_{w,yy}}{r_w^2} \frac{w^2}{4} \dot{\psi}^2 + \frac{1}{2} I_{w,zz} \dot{\psi}^2 \\
& + \frac{1}{2} m_w \dot{x}^2 - m_w \frac{w}{2} \dot{x}\dot{\psi} + \frac{1}{2} m_w \frac{w^2}{4} \dot{\psi}^2 \\
& + \frac{1}{2} \frac{I_{w,yy}}{r_w^2} \dot{x}^2 - \frac{I_{w,yy}}{r_w^2} \frac{w}{2} \dot{x}\dot{\psi} + \frac{1}{2} \frac{I_{w,yy}}{r_w^2} \frac{w^2}{4} \dot{\psi}^2 + \frac{1}{2} I_{w,zz} \dot{\psi}^2
\end{aligned} \tag{A.57}
$$

$$
\begin{aligned}
T = & \frac{1}{2} m_b \dot{x}^2 + m_b r_b \cos\theta \, \dot{x}\dot{\theta} + \frac{1}{2} m_b r_b^2 \dot{\theta}^2 + \frac{1}{2} m_b r_b^2 \sin^2\theta \, \dot{\psi}^2 \\
& + \frac{1}{2} I_{b,yy} \dot{\theta}^2 + \frac{1}{2} I_{b,xx} \sin^2\theta \, \dot{\psi}^2 + \frac{1}{2} I_{b,zz} \cos^2\theta \, \dot{\psi}^2 \\
& + m_w \dot{x}^2 + m_w \frac{w^2}{4} \dot{\psi}^2 \\
& + \frac{I_{w,yy}}{r_w^2} \dot{x}^2 + \frac{I_{w,yy}}{r_w^2} \frac{w^2}{4} \dot{\psi}^2 + I_{w,zz} \dot{\psi}^2
\end{aligned} \tag{A.58}
$$

$$
\begin{aligned}
T = & \frac{1}{2} m_b \dot{x}^2 + m_w \dot{x}^2 + \frac{I_{w,yy}}{r_w^2} \dot{x}^2 \\
& + m_b r_b \cos\theta \, \dot{x}\dot{\theta} \\
& + \frac{1}{2} m_b r_b^2 \dot{\theta}^2 + \frac{1}{2} I_{b,yy} \dot{\theta}^2 \\
& + \frac{1}{2} m_b r_b^2 \sin^2\theta \, \dot{\psi}^2 + \frac{1}{2} I_{b,xx} \sin^2\theta \, \dot{\psi}^2 + \frac{1}{2} I_{b,zz} \cos^2\theta \, \dot{\psi}^2 + m_w \frac{w^2}{4} \dot{\psi}^2 + \frac{I_{w,yy}}{r_w^2} \frac{w^2}{4} \dot{\psi}^2 + I_{w,zz} \dot{\psi}^2
\end{aligned} \tag{A.59}
$$

$$
\begin{aligned}
T = & \frac{1}{2}\left(m_b + 2m_w + 2\frac{I_{w,yy}}{r_w^2}\right)\dot{x}^2 \\
& + m_b r_b \cos\theta\, \dot{x}\dot{\theta} \\
& + \frac{1}{2}\left(m_b r_b^2 + I_{b,yy}\right)\dot{\theta}^2 \\
& + \frac{1}{2}\left(m_b r_b^2 \sin^2\theta + I_{b,xx}\sin^2\theta + I_{b,zz}\cos^2\theta + m_w\frac{w^2}{2} + \frac{I_{w,yy}}{r_w^2}\frac{w^2}{2} + 2I_{w,zz}\right)\dot{\psi}^2 \qquad \text{(A.60)}
\end{aligned}
$$

$$
T = \frac{1}{2}m_x\dot{x}^2 + m_b r_b\cos\theta\,\dot{x}\dot{\theta} + \frac{1}{2}I_\theta\dot{\theta}^2 + \frac{1}{2}\left(m_b r_b^2\sin^2\theta + I_{b,xx}\sin^2\theta + I_{b,zz}\cos^2\theta + I_\psi\right)\dot{\psi}^2 \qquad \text{(A.61)}
$$

Where,

$$
\begin{aligned}
m_x &= m_b + 2m_w + 2\frac{I_{w,yy}}{r_w^2}, \\
I_\theta &= m_b r_b^2 + I_{b,yy}, \\
I_\psi &= m_w\frac{w^2}{2} + \frac{I_{w,yy}}{r_w^2}\frac{w^2}{2} + 2I_{w,zz}.
\end{aligned}
$$

A.4 3-DOF Model - Equations of Motion Derivation

This section covers the derivation of the 3-DOF equations of motion using Lagrange's equation. Recall that the Lagrangian is defined as the difference between kinetic and potential energy; that is, $L = T - V$. The potential energy in the system is due to the elevation of the body of the balance bot and can be expressed as $V = m_b r_b \cos\theta\, g$. With the Lagrangian defined, Lagrange's equation is used to find the equations of motion.

The Lagrangian is defined as the difference between kinetic and potential energy.

$$
L = T - V \qquad \text{(A.62)}
$$

$$
L = \frac{1}{2}m_x\dot{x}^2 + m_b r_b\cos\theta\,\dot{x}\dot{\theta} + \frac{1}{2}I_\theta\dot{\theta}^2 + \frac{1}{2}\left(m_b r_b^2\sin^2\theta + I_{b,xx}\sin^2\theta + I_{b,zz}\cos^2\theta + I_\psi\right)\dot{\psi}^2 - m_b r_b\cos\theta\, g \qquad \text{(A.63)}
$$

L is first differentiated with respect to x.

$$
\frac{\partial L}{\partial x} = \frac{\partial}{\partial x}\left(\frac{1}{2}m_x\dot{x}^2 + m_b r_b\cos\theta\,\dot{x}\dot{\theta} + \frac{1}{2}I_\theta\dot{\theta}^2 + \frac{1}{2}\left(m_b r_b^2\sin^2\theta + I_{b,xx}\sin^2\theta + I_{b,zz}\cos^2\theta + I_\psi\right)\dot{\psi}^2 - m_b r_b\cos\theta\, g\right) \qquad \text{(A.64)}
$$

$$
\frac{\partial L}{\partial x} = 0 \qquad \text{(A.65)}
$$

Then, L is differentiated with respect to $\dot{x}$.

$$\frac{\partial L}{\partial \dot{x}} = \frac{\partial}{\partial \dot{x}} \left(\frac{1}{2} m_x \dot{x}^2 + m_b r_b \cos\theta \, \dot{x}\dot{\theta} + \frac{1}{2} I_\theta \dot{\theta}^2 + \frac{1}{2} \left(m_b r_b^2 \sin^2\theta + I_{b,xx} \sin^2\theta + I_{b,zz} \cos^2\theta + I_\psi \right) \dot{\psi}^2 - m_b r_b \cos\theta \, g \right) \tag{A.66}$$

$$\frac{\partial L}{\partial \dot{x}} = m_x \dot{x} + m_b r_b \cos\theta \, \dot{\theta} \tag{A.67}$$

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{x}} = \frac{d}{dt} \left(m_x \dot{x} + m_b r_b \cos\theta \, \dot{\theta} \right) \tag{A.68}$$

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{x}} = m_x \ddot{x} + m_b r_b \cos\theta \, \ddot{\theta} - m_b r_b \sin\theta \, \dot{\theta}^2 \tag{A.69}$$

And with respect to θ.

$$\frac{\partial L}{\partial \theta} = \frac{\partial}{\partial \theta} \left(\frac{1}{2} m_x \dot{x}^2 + m_b r_b \cos\theta \, \dot{x}\dot{\theta} + \frac{1}{2} I_\theta \dot{\theta}^2 + \frac{1}{2} \left(m_b r_b^2 \sin^2\theta + I_{b,xx} \sin^2\theta + I_{b,zz} \cos^2\theta + I_\psi \right) \dot{\psi}^2 - m_b r_b \cos\theta \, g \right) \tag{A.70}$$

$$\frac{\partial L}{\partial \theta} = -m_b r_b \sin\theta \, \dot{x}\dot{\theta} + m_b r_b^2 \sin\theta \cos\theta \, \dot{\psi}^2 + I_{b,xx} \sin\theta \cos\theta \, \dot{\psi}^2 - I_{b,zz} \sin\theta \cos\theta \, \dot{\psi}^2 + m_b r_b \sin\theta \, g \tag{A.71}$$

And with respect to $\dot{\theta}$.

$$\frac{\partial L}{\partial \dot{\theta}} = \frac{\partial}{\partial \dot{\theta}} \left(\frac{1}{2} m_x \dot{x}^2 + m_b r_b \cos\theta \, \dot{x}\dot{\theta} + \frac{1}{2} I_\theta \dot{\theta}^2 + \frac{1}{2} \left(m_b r_b^2 \sin^2\theta + I_{b,xx} \sin^2\theta + I_{b,zz} \cos^2\theta + I_\psi \right) \dot{\psi}^2 - m_b r_b \cos\theta \, g \right) \tag{A.72}$$

$$\frac{\partial L}{\partial \dot{\theta}} = m_b r_b \cos\theta \, \dot{x} + I_\theta \dot{\theta} \tag{A.73}$$

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{\theta}} = -m_b r_b \sin\theta \, \dot{x}\dot{\theta} + m_b r_b \cos\theta \, \ddot{x} + I_\theta \ddot{\theta} \tag{A.74}$$

And with respect to ψ.

$$\frac{\partial L}{\partial \psi} = \frac{\partial}{\partial \psi} \left(\frac{1}{2} m_x \dot{x}^2 + m_b r_b \cos\theta \, \dot{x}\dot{\theta} + \frac{1}{2} I_\theta \dot{\theta}^2 + \frac{1}{2} \left(m_b r_b^2 \sin^2\theta + I_{b,xx} \sin^2\theta + I_{b,zz} \cos^2\theta + I_\psi \right) \dot{\psi}^2 - m_b r_b \cos\theta \, g \right) \tag{A.75}$$

$$\frac{\partial L}{\partial \psi} = 0 \tag{A.76}$$

And finally with respect to $\dot{\psi}$.

$$\frac{\partial L}{\partial \dot{\psi}} = \frac{\partial}{\partial \dot{\psi}} \left(\frac{1}{2} m_x \dot{x}^2 + m_b r_b \cos\theta \, \dot{x}\dot{\theta} + \frac{1}{2} I_\theta \dot{\theta}^2 + \frac{1}{2} \left(m_b r_b^2 \sin^2\theta + I_{b,xx} \sin^2\theta + I_{b,zz} \cos^2\theta + I_\psi \right) \dot{\psi}^2 - m_b r_b \cos\theta \, g \right) \tag{A.77}$$

$$\frac{\partial L}{\partial \dot{\psi}} = \left(m_b r_b^2 \sin^2\theta + I_{b,xx} \sin^2\theta + I_{b,zz} \cos^2\theta + I_\psi \right) \dot{\psi} \tag{A.78}$$

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{\psi}} = \frac{d}{dt} \left(\left(m_b r_b^2 \sin^2\theta + I_{b,xx} \sin^2\theta + I_{b,zz} \cos^2\theta + I_\psi \right) \dot{\psi} \right) \tag{A.79}$$

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{\psi}} = \left(m_b r_b^2 \sin^2\theta + I_{b,xx} \sin^2\theta + I_{b,zz} \cos^2\theta + I_\psi \right) \ddot{\psi} \tag{A.80}$$

$$+ \left(2 m_b r_b^2 \sin\theta \cos\theta \, \dot{\theta} + 2 I_{b,xx} \sin\theta \cos\theta \, \dot{\theta} - 2 I_{b,zz} \sin\theta \cos\theta \, \dot{\theta} \right) \dot{\psi} \tag{A.81}$$

The first equation of motion is for the horizontal axis and can be determined using Lagrange's equation applied to the q_1 coordinate.

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{x}}\right) - \frac{\partial L}{\partial x} = Q_x \tag{A.82}$$

$$m_x\ddot{x} + m_b r_b \cos\theta\,\ddot{\theta} - m_b r_b \sin\theta\,\dot{\theta}^2 = \frac{1}{r_w}\left(-T_{m,r} + T_{m,l}\right) \tag{A.83}$$

$$m_x\ddot{x} + m_b r_b \cos\theta\,\ddot{\theta} = -\frac{1}{r_w}T_{m,r} + \frac{1}{r_w}T_{m,l} + m_b r_b \sin\theta\,\dot{\theta}^2 \tag{A.84}$$

The second equation of motion is for the pitch axis and can be determined using Lagrange's equation applied to the q_2 coordinate.

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}}\right) - \frac{\partial L}{\partial \theta} = Q_\theta \tag{A.85}$$

$$\begin{aligned}
&\Big(m_b r_b \cos\theta\, \ddot{x} - m_b r_b \sin\theta\, \dot{x}\dot{\theta} \\
&\quad + I_\theta \ddot{\theta} + m_b r_b \sin\theta\, \dot{x}\dot{\theta} - m_b r_b^2 \sin\theta\cos\theta\, \dot{\psi}^2 \\
&\quad - I_{b,xx} \sin\theta\cos\theta\, \dot{\psi}^2 + I_{b,zz} \sin\theta\cos\theta\, \dot{\psi}^2 - m_b r_b \sin\theta\, g\Big) \quad = T_{m,r} - T_{m,l}
\end{aligned} \tag{A.86}$$

$$\begin{aligned}
m_b r_b \cos\theta\, \ddot{x} + I_\theta \ddot{\theta} \quad &= T_{m,r} - T_{m,l} \\
&\quad + m_b r_b \sin\theta\, g \\
&\quad + \left(m_b r_b^2 + I_{b,xx} - I_{b,zz}\right) \sin\theta\cos\theta\, \dot{\psi}^2
\end{aligned} \tag{A.87}$$

The third and final equation of motion is for the yaw axis and can be determined using Lagrange's equation applied to the q_3 coordinate.

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\psi}}\right) - \frac{\partial L}{\partial \psi} = Q_\psi \tag{A.88}$$

$$\begin{aligned}
&\Big(\left(m_b r_b^2 \sin^2\theta + I_{b,xx} \sin^2\theta + I_{b,zz} \cos^2\theta + I_\psi\right) \ddot{\psi} \\
&\quad + \left(2 m_b r_b^2 \sin\theta\cos\theta\, \dot{\theta} + 2 I_{b,xx} \sin\theta\cos\theta\, \dot{\theta} - 2 I_{b,zz} \sin\theta\cos\theta\, \dot{\theta}\right) \dot{\psi}\Big) \quad = \frac{w}{2r_w}\left(-T_{m,r} - T_{m,l}\right)
\end{aligned} \tag{A.89}$$

$$\begin{aligned}
\left(m_b r_b^2 \sin^2\theta + I_{b,xx} \sin^2\theta + I_{b,zz} \cos^2\theta + I_\psi\right) \ddot{\psi} \quad &= -\frac{w}{2r_w} T_{m,r} - \frac{w}{2r_w} T_{m,l} \\
&\quad - 2\left(m_b r_b^2 + I_{b,xx} - I_{b,zz}\right) \sin\theta\cos\theta\, \dot{\theta}\dot{\psi}
\end{aligned} \tag{A.90}$$

Appendix B

OBSERVABILITY AND CONTROLLABILITY CHECKS

This brief appendix covers confirmation of the system controllability and observability for the 2-DOF and 3-DOF system models.

B.1 2-DOF Model Controllability and Observability Confirmation

The controllability and observability of the 2-DOF system model are confirmed using features of the symbolic toolbox in MATLAB®. Consider the code shown in Listings B.1 and B.3 which symbolically calculate the controllability and observability Gramians from the matrices A, B, and C.

Listing B.1: Controllability check for 2-DOF system model: `controllability_2DOF.m`.

```
% Define symbolic variables for matrix A
syms A_32 A_42

% Define the matrix A
A = [   0   0       1   0
        0   0       0   1
        0   A_32    0   0
        0   A_42    0   0 ];

% Define symbolic variables for matrix B
syms B_3 B_4

% Define the matrix B
B = [   0
        0
        B_3
        B_4 ];

% Compute the controllability Gramian
ctrl = [B A*B A^2*B A^3*B];

% Check the rank of the Gramian
rank(ctrl)
```

Listing B.2: Output of controllability check script for 2-DOF system model.

```
>> controllability_2DOF

ans =

     4
```

Listing B.3: Observability check for 2-DOF system model: `observability_2DOF.m`.

```
% Define symbolic variables for matrix A
syms A_32 A_42

% Define the matrix A
A = [   0   0       1   0
        0   0       0   1
        0   A_32    0   0
        0   A_42    0   0 ];

% Define symbolic variables for matrix C
syms C_11 C_33

% Define the matrix C
C = [   C_11   -1   0       0
        0       1   0       0
        0       0   C_33   -1
        0       0   0       1 ];

% Compute the controllability Gramian
obsv = [C
        C*A
        C*A^2
        C*A^3];

% Check the rank of the Gramian
rank(obsv)
```

Listing B.4: Output of observability check script for 2-DOF system model.

```
>> observability_2DOF

ans =

     4
```

Both the controllability Gramian, $\mathcal{C}$, and the observability Gramian, $\mathcal{O}$, are of full rank indicating that the 2-DOF system model is fully controllable and observable. The controllability Gramian is shown below in Equation B.1 and the observability Gramian is shown below in Equation B.2.

$$\mathcal{C} = \begin{bmatrix} 0 & B_3 & 0 & A_{3,2}B_4 \\ 0 & B_4 & 0 & A_{3,2}B_4 \\ B_3 & 0 & A_{3,2}B_4 & 0 \\ B_4 & 0 & A_{4,2}B_4 & 0 \end{bmatrix} \tag{B.1}$$

$$\mathcal{O} = \begin{bmatrix} C_{1,1} & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & C_{3,3} & -1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & C_{1,1} & -1 \\ 0 & 0 & 0 & 1 \\ 0 & A_{3,2}C_{3,3} - A_{4,2} & 0 & 0 \\ 0 & A_{4,2} & 0 & 0 \\ 0 & A_{3,2}C_{1,1} - A_{4,2} & 0 & 0 \\ 0 & A_{4,2} & 0 & 0 \\ 0 & 0 & 0 & A_{3,2}C_{3,3} - A_{4,2} \\ 0 & 0 & 0 & A_{4,2} \\ 0 & 0 & 0 & A_{3,2}C_{1,1} - A_{4,2} \\ 0 & 0 & 0 & A_{4,2} \\ 0 & A_{3,2}A_{4,2}C_{3,3} - {A_{4,2}}^2 & 0 & 0 \\ 0 & {A_{4,2}}^2 & 0 & 0 \end{bmatrix} \tag{B.2}$$

B.2 3-DOF Model Controllability and Observability Confirmation

The controllability and observability of the 3-DOF system model are confirmed using features of the symbolic toolbox in MATLAB®. Consider the code shown in Listings B.5 and B.7 which symbolically calculate the controllability and observability Gramians from the matrices A, B, and C.

Listing B.5: Controllability check script for 3-DOF system model: `controllability_3DOF.m`.

```
% Define symbolic variables for matrix A
syms A_42 A_52

% Define the matrix A
A = [   0   0      0   1   0   0
        0   0      0   0   1   0
        0   0      0   0   0   1
        0   A_42   0   0   0   0
        0   A_52   0   0   0   0
        0   0      0   0   0   0 ];

% Define symbolic variables for matrix B
syms B_41 B_42 B_51 B_52 B_61 B_62

% Define the matrix B
B = [   0       0
        0       0
        0       0
        B_41    B_42
        B_51    B_52
        B_61    B_62 ];

% Compute the controllability Gramian
ctrl = [B A*B A^2*B A^3*B A^4*B A^5*B];

% Check the rank of the Gramian
rank(ctrl)
```

Listing B.6: Output of controllability check script for 3-DOF system model.

```
>> controllability_3DOF

ans =

     6
```

Listing B.7: Observability check script for 3-DOF system model: `observability_3DOF.m`.

```
% Define symbolic variables for matrix A
syms A_42 A_52 B_41

% Define the matrix A
A = [   0   0       0   1   0   0
        0   0       0   0   1   0
        0   0       0   0   0   1
        0   A_42    0   0   0   0
        0   A_52    0   0   0   0
        0   0       0   0   0   0 ];

% Define symbolic variables for matrix C
syms C_11 C_44

% Define the matrix C
C = [   C_11   -1   0   0        0   0
        0       1   0   0        0   0
        0       0   1   0        0   0
        0       0   0   C_44    -1   0
        0       0   0   0        1   0
        0       0   0   0        0   1];

% Compute the controllability Gramian
obsv = [C
        C*A
        C*A^2
        C*A^3
        C*A^4
        C*A^5];

% Check the rank of the Gramian
rank(obsv)
```

Listing B.8: Output of observability check script for 3-DOF system model.

```
>> observability_3DOF

ans =

     6
```

Both the controllability Gramian, $\mathcal{C}$, and the observability Gramian, $\mathcal{O}$, are of full rank indicating that the 3-DOF system model is fully controllable and observable. The controllability Gramian is shown below in Equation B.3 and the observability Gramian is shown below in Equation B.4.

$$\mathcal{C} = \begin{bmatrix} 0 & 0 & B_{4,1} & B_{4,2} & 0 & 0 & A_{4,2}B_{5,1} & A_{4,2}B_{5,2} & 0 & 0 & A_{4,2}A_{5,2}B_{5,1} & A_{4,2}A_{5,2}B_{5,2} \\ 0 & 0 & B_{5,1} & B_{5,2} & 0 & 0 & A_{5,2}B_{5,1} & A_{5,2}B_{5,2} & 0 & 0 & {A_{5,2}}^2 B_{5,1} & {A_{5,2}}^2 B_{5,2} \\ 0 & 0 & B_{6,1} & B_{6,2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ B_{4,1} & B_{4,2} & 0 & 0 & A_{4,2}B_{5,1} & A_{4,2}B_{5,2} & 0 & 0 & A_{4,2}A_{5,2}B_{5,1} & A_{4,2}A_{5,2}B_{5,2} & 0 & 0 \\ B_{5,1} & B_{5,2} & 0 & 0 & A_{5,2}B_{5,1} & A_{5,2}B_{5,2} & 0 & 0 & {A_{5,2}}^2 B_{5,1} & {A_{5,2}}^2 B_{5,2} & 0 & 0 \\ B_{6,1} & B_{6,2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{B.3}$$

$$\mathcal{O} = \begin{bmatrix}
C_{1,1} & -1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & C_{4,4} & -1 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & C_{1,1} & -1 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 \\
0 & A_{4,2}\,C_{4,4} - A_{5,2} & 0 & 0 & 0 & 0 \\
0 & A_{5,2} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & A_{4,2}\,C_{1,1} - A_{5,2} & 0 & 0 & 0 & 0 \\
0 & A_{5,2} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & A_{4,2}\,C_{4,4} - A_{5,2} & 0 \\
0 & 0 & 0 & 0 & A_{5,2} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & A_{4,2}\,C_{1,1} - A_{5,2} & 0 \\
0 & 0 & 0 & 0 & A_{5,2} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & A_{4,2}\,A_{5,2}\,C_{4,4} - {A_{5,2}}^2 & 0 & 0 & 0 & 0 \\
0 & {A_{5,2}}^2 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & A_{4,2}\,A_{5,2}\,C_{1,1} - {A_{5,2}}^2 & 0 & 0 & 0 & 0 \\
0 & {A_{5,2}}^2 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & A_{4,2}\,A_{5,2}\,C_{4,4} - {A_{5,2}}^2 & 0 \\
0 & 0 & 0 & 0 & {A_{5,2}}^2 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & A_{4,2}\,A_{5,2}\,C_{1,1} - {A_{5,2}}^2 & 0 \\
0 & 0 & 0 & 0 & {A_{5,2}}^2 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & A_{4,2}\,{A_{5,2}}^2\,C_{4,4} - {A_{5,2}}^3 & 0 & 0 & 0 & 0 \\
0 & {A_{5,2}}^3 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix} \tag{B.4}$$

Appendix C

PRINTED CIRCUIT BOARD REVISION HISTORY

Throughout the development of this project, several revisions have been made to the printed circuit board design. This section will summarize the design development through the revision history.

C.1 Revision 0.0

Initial development began without a custom PCB design. This "zeroth" revision was used to validate the feasibility of operating a BLDC motor using the Nucleo hardware running Micropython as originally there were concerns regarding the precise timing necessary to commutate the BLDC motor.

This revision consisted of the Nucleo development board and an off-the-shelf motor driver development board designed for the Nucleo. The X-NUCLEO-IHM04A1 is a development board that breaks out a L6206 dual DC motor driver from ST-Microelectronics. The dual motor driver includes two full H-bridges, which is enough to drive two DC motors. One full bridge, and half of the other full bridge were used together to test commutation of a BLDC motor.

Revision 0.0 ended up showing that the timing was possible, but provided very poor performance due to limitations in the L6206. The full-bridges within the L6206 may be disabled, but not on an individual half-bridge basis. This meant that it was impossible to let just one of the motor phases float at a time preventing proper block commutation. It was possible to commutate effectively enough to produce torque and get the motor to spin, but the commutation was very inefficient and produced very large torque ripple.

This revision also proved that the microcontroller could run input capture interrupts to measure the time duration between edges on the hall sensor signals.

C.2 Revision 0.1

The next revision, R0.1, was the first revision with a custom PCB design, as shown in Figure C.1. This first revision includes the majority of the components listed above in the section on Component Selection. This board had several problems that were addressed in revisions 0.2 and 1.0. The board measured 98×69 $[mm]$.

- The pads in the footprint for the DRV8313 were too large to solder effectively by hand making the board prone to short circuits underneath the motor driver ICs.
- The screw terminals allowing connection to the motors and sensors were unreachable when the Nucleo was stacked on the PCB.
- The jumper JP1 was intended to allow the additional USB pins meant for auto-detection and identification to be selected instead of the PWM signal used for motor commutation. This ended up being unnecessary because the USB pins are only utilized when the USB-OTG is in host mode.
- The routing on this board was done with the Autorouter tool available in Autodesk Eagle in order to rapidly produce and test the initial PCB design. Therefore, the trace routing was nonoptimal.
- The screw terminals labelled Li3s were intended to allow two batteries to be used in series, providing a 24 $[V]$ rail to supply the motors. The motors ended up being sufficiently powerful running at 12 $[V]$ so the second battery port was removed in R1.0.
- This revision did not include the ACS70331 current sensors selected to measure the BLDC motor phase currents.
- The E5V pin providing power to the Nucleo from the USB port was connected to the wrong pin and needed to be adjusted with a wire modification.
- The battery fuse was not included in this revision.

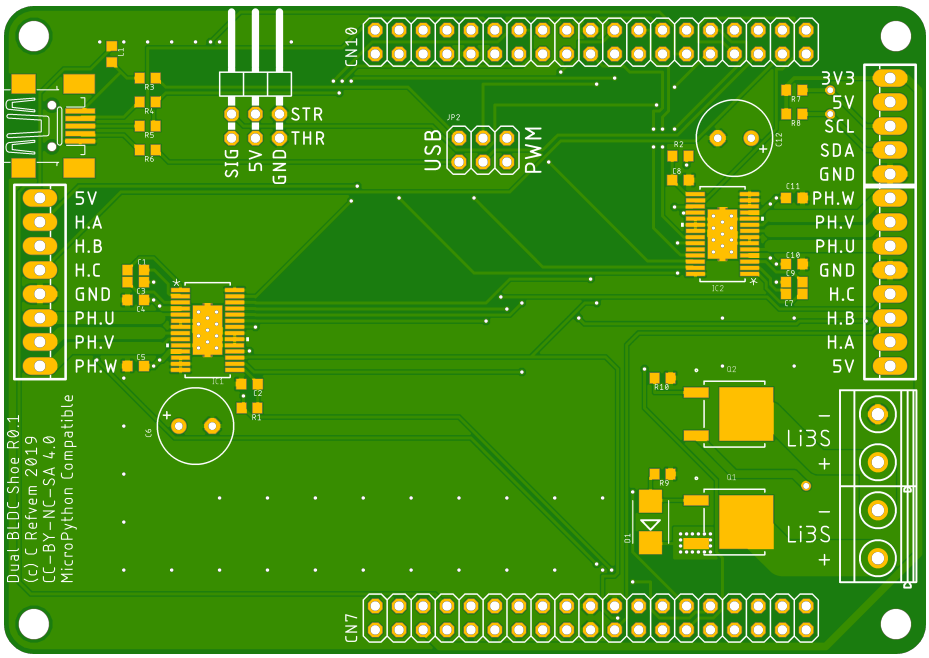


Figure C.1: Render of PCB revision 0.1, created with Autodesk Eagle.

C.3 Revision 0.2

The next revision, R0.2, was intended as a quick fix for the issues in R0.1 that prevented proper testing of the balance bot. The board dimensions increased slightly to 100×71 $[mm]$. There were several small changes in this circuit board design compared to R0.1 that allowed the device to be tested, but there were no major changes in circuit design.

- The routing of every trace was redone manually. The main goal of this effort was to prevent fragmentation of the ground plane on the top and bottom of the PCB. Additionally, the high speed USB traces were impedance matched, although the necessity of this change has not been confirmed.
- The pads on the DRV8313 footprint were made smaller to prevent short circuits underneath the IC.
- The dimensions of the board were extended and the Nucleo headers were relocated slightly to allow access to the screw terminals while the Nucleo was stacked.
- The E5V pin was connected properly to the USB power eliminating the need for a wire modification.

- The jumper allowing USB-OTG functionality was removed as that feature was deemed unnecessary for this project.

- Several components were spread out to make assembly and hand soldering easier.

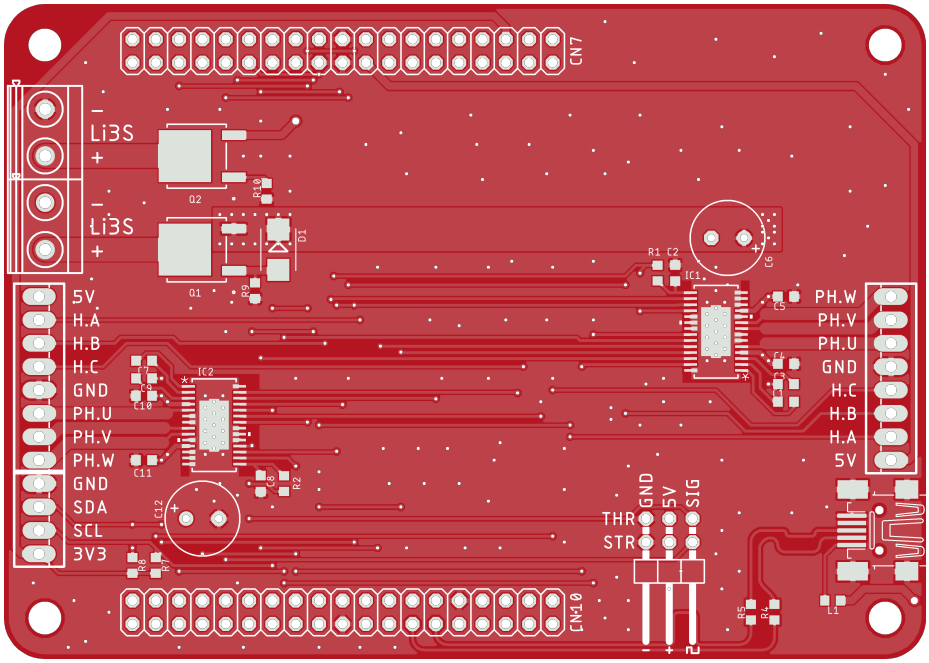


Figure C.2: Render of PCB revision 0.2, created with Autodesk Eagle.

Even with all of these changes, there were still many issues that motivated a significant amount of redesign in R1.0.

- Assembly and testing required frequent connection and disconnection from the external components. The screw terminals ended up being a nuisance in this regard because they required a lot of time and attention to connect properly. The screenprint labeling each pin was also difficult to see when the Nucleo was stacked.

- During testing it was verified that the motors would provide sufficient torque operating at half their nominal voltage making the dual battery configuration unneeded.

- Even though there is built in reverse polarity protection in R0.1 and R0.2, the unpolarized screw terminals were not ideal. The '+' and '-' symbols were difficult to see when the Nucleo was stacked.

- The RC radio header was awkwardly placed and hard to reach when the Nucleo was stacked.

- Neither R0.1 nor R0.2 supported interfacing with quadrature encoders. At the time, testing was indicating that the speed and position measurements coming from the Hall sensors were insufficiently precise to effectively control the balance bot.
- After rerouting the traces manually the ground plane fragmentation was reduced, but not eliminated satisfactorily. The nature of the Nucleo pinout unavoidably forces traces to cross all over the board which inevitably slices up the ground planes.
- The battery fuse was still not included in this revision and it was getting to the point where battery testing was necessary and the lab supply had to be put away.

C.4 Revision 1.0

The remaining problems from R0.2 were addressed in R1.0 resulting in a board that was able to make the bot balance successfully for the first time. Unfortunately this revision was significantly different from R0.2 and required a considerable amount of time to prepare compared to the changes from R0.1 to R0.2. The board size was increased slightly to 109×71 $[mm]$ to accommodate some changes in the mechanical design.

- One of the bigger changes was the switch from a 2-layer PCB to a 4-layer PCB. This greatly improved the routing on the board and entirely prevented the ground plane fragmentation issues in R0.1 and R0.2.
- All of the screw terminals and headers were replaced by polarized connectors. This proved to be convenient in some ways but a big problem in other ways that will be addressed shortly. This allowed the polarized battery connector to be plugged directly into the PCB, avoiding the risk of incorrect orientation entirely.
- The dual battery configuration was abandoned so that the whole system may run off of the single three cell LiPo battery.
- Current sensors were added to each phase of each motor.
- The battery protection fuse was integrated now that battery testing was absolutely necessary.
- Additional connectors were added allowing interface with two quadrature encoders. Unfortunately this change required significant shuffling of pins compared to R0.1 and R0.2.

- The remaining unused GPIO pins, of which there were now only 6, were exposed through test-points in case further functionality was to be investigated once the bot was balancing.

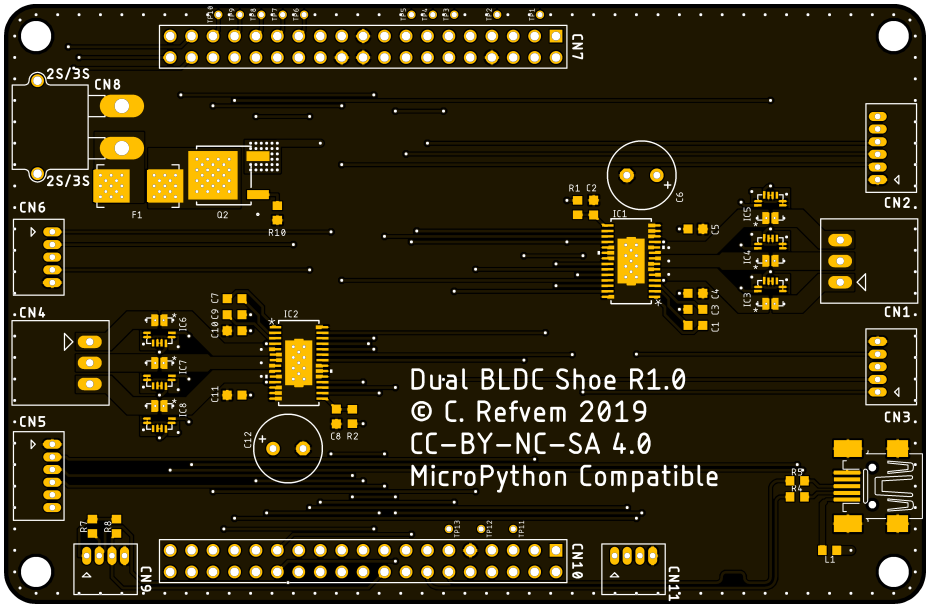


Figure C.3: Render of PCB revision 1.0, created with Autodesk Eagle; main board shown.

As with any design, there are as many lingering problems as their are revisions. While R1.0 fixed many problems from R0.2, it unfortunately introduced one very significant problem: micro-pitch connector assembly. Micro-pitch connectors will be considered below in their own subsection.

Aside from the cabling trouble, there were still some remaining issues in R1.0 of the PCB design.

- The new connectors all exit the top of the PCB horizontally, creating strain relief issues and locating the cables in such a way that they are easily damaged when the balance bot falls over. If the connectors were replaced with vertical connectors on the bottom of the board, all wiring could be internal and provide significantly less risk of damage in the event of the bot falling over.

- The footprints for the current sensors have a very fine pitch of 0.5 $[mm]$ making them extremely challenging to solder by hand. The fine pitch issue is compounded by the fact that the 4-layer PCB sinks heat more rapidly than standard 2-layer PCBs. While testing, the current sensors were abandoned and were replaced by jumpers for the time-being.

C.4.1 Additional PCB Tiers

As covered in the mechanical design chapter, the balance bot was designed to be composed of multiple tiers separated by standoffs. Originally the PCB was to be one of the many tiers, and all others were to be fabricated from polycarbonate.

As part of the changes in R1.0, it was decided that all of the tiers in the balance bot would be constructed from PCBs. Modern PCB manufacturing is rapid, extremely precise, and very low cost; together these benefits make PCBs ideal choices for the tiered balance bot design. As an added bonus, the material PCBs are constructed from is FR4, which is a type of fiberglass that is extremely rigid and long lasting compared to other materials considered such as polycarbonate and aluminum.

The R1.0 design of the balance bot is composed of three unique PCBs stacked conveniently to hold all of the required components. Below, Figures C.4 and C.5 show the additional PCBs other than the main board shown in figure C.3.

The IMU board, shown in Figure C.4, allows interface with the BNO055 module.

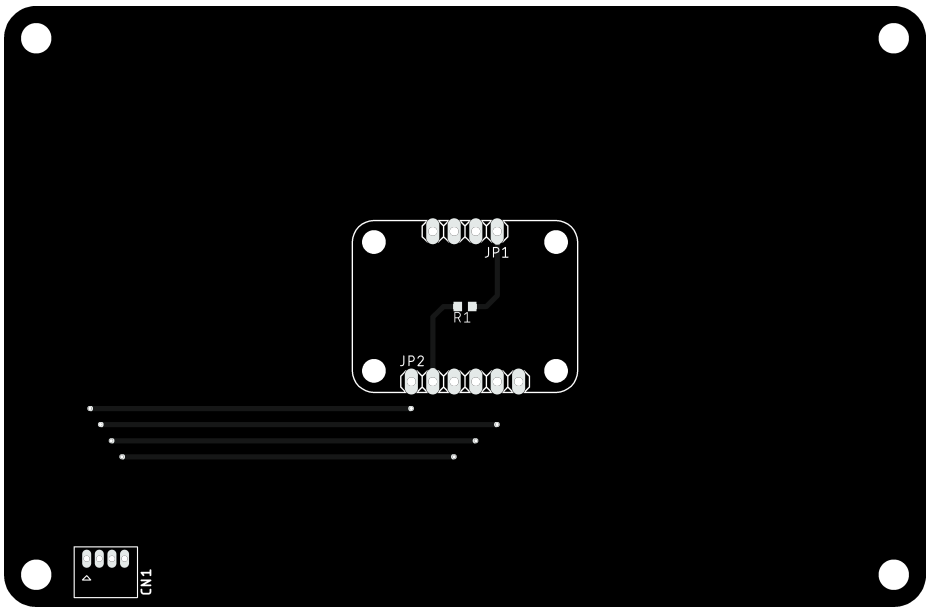


Figure C.4: Render of PCB revision 1.0, created with Autodesk Eagle; IMU board shown.

The motor board does not actually contain any electronic components, and is used only for mechanical purposes. The primary role of this board is to connect with the motor brackets.

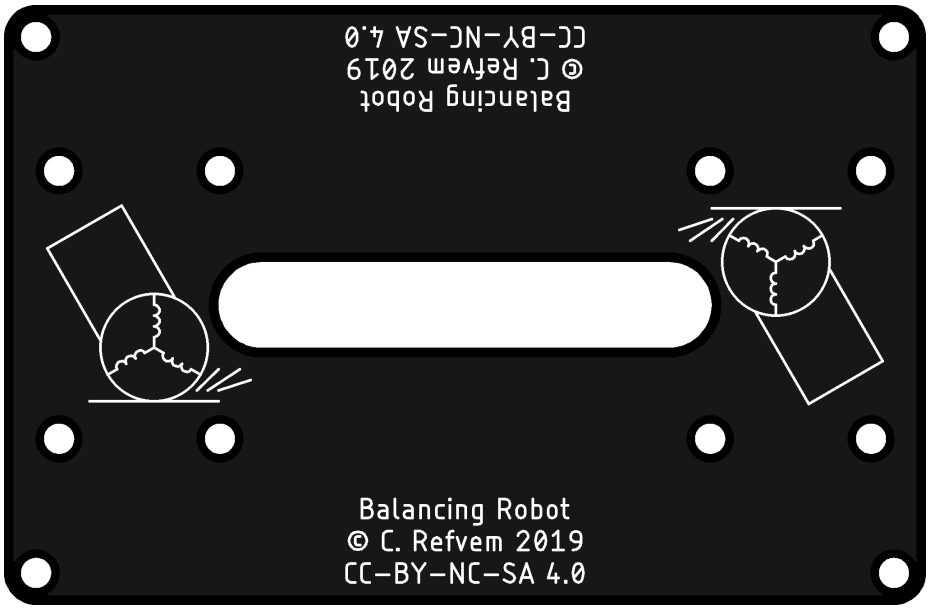


Figure C.5: Render of PCB revision 1.0, created with Autodesk Eagle; motor board shown.

C.4.2 Micro-pitch Connectors

The decision to move away from screw terminals and toward connectors required selection of connectors with sufficiently small pitch to allow enough edge space on the board to squeeze in all the connectors. To accomplish this, several connector families were selected, mainly from JST.

The battery connection is through a XT30PW male connector which allows the XT30 female plug on the battery to connect directly to the PCB. XT30 connectors are standard among hobby-grade batteries of the scale used in this project.

The JST-XH connector line was selected for the motor phase connections, as they offer a continuous current rating of 3 $[A]$, which is sufficient for the selected motors. The JST-XH line has a pitch of 2.5 $[mm]$ which is also wide enough to support the 22AWG wire used for the motor phases connections.

For all other connections, the JST-ZH line was selected. All remaining connections were of low current requirements so a fine pitch connector like the 1.5 $[mm]$ pitch JST-ZH seemed like a good choice. In terms of the PCB design and assembly, these connectors were a great choice. However, the switch to these connectors forced the mating cables to all be assembled manually. The hand crimp tools produced by JST for the ZH line range from $500 to $1100, considerably outside the budget

for this project. Hand crimp tools from other manufacturers are of much lower cost in the $20 to $100 range but are not designed with the proper geometry to crimp the ZH connectors reliably.

Many attempts were made to produce reliable cabling for the many wiring assemblies needed for this project, but the incorrect crimp tools ended up producing cables that were prone to failure. To mitigate this issue, wherever possible pre-crimped cables were ordered and re-pinned to avoid the need to crimp manually. In many cases this was impossible though; the motors came with flying leads that had to be crimped manually.

At the time of writing, this issue with micro-pitch crimping is the biggest cause of unreliability in the balance bot design.

C.5 Revision 1.1

The most recent revision, R1.1, has not been ordered or assembled yet but includes some pending changes to overcome the issues with R1.0. The main board only has a few minor changes.

- Many connectors have been adjusted to allow the cables to exit vertically from the bottom of the PCB to rout through the internals of the bot. This will reduce the likelihood of damaging the cabling in the event the bot falls over.
- The resistor network for battery voltage measurement has been added.
- The top silkscreen has been adjusted to include a location where the RC receiver will be mounted.

The IMU board has one main change. A slot has been added to allow the cabling from the main board to more easily route to the motors at the bottom of the balance bot.

The motor board has also been adjusted slightly. The slot accommodating the motor cabling has been enlarged, and additional slots have been added that will be used with Velcro straps to secure the battery.

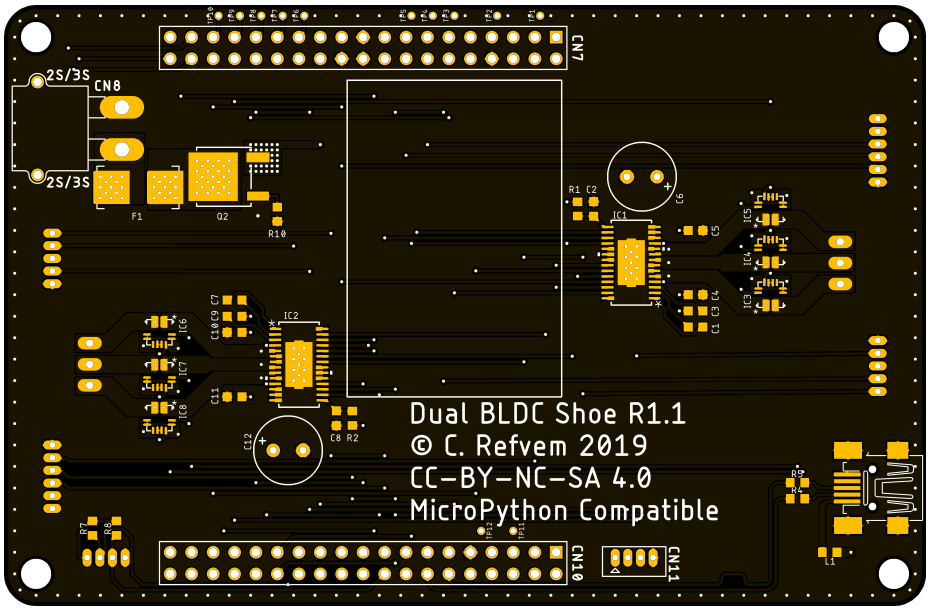


Figure C.6: Render of PCB revision 1.1, created with Autodesk Eagle; main board shown.

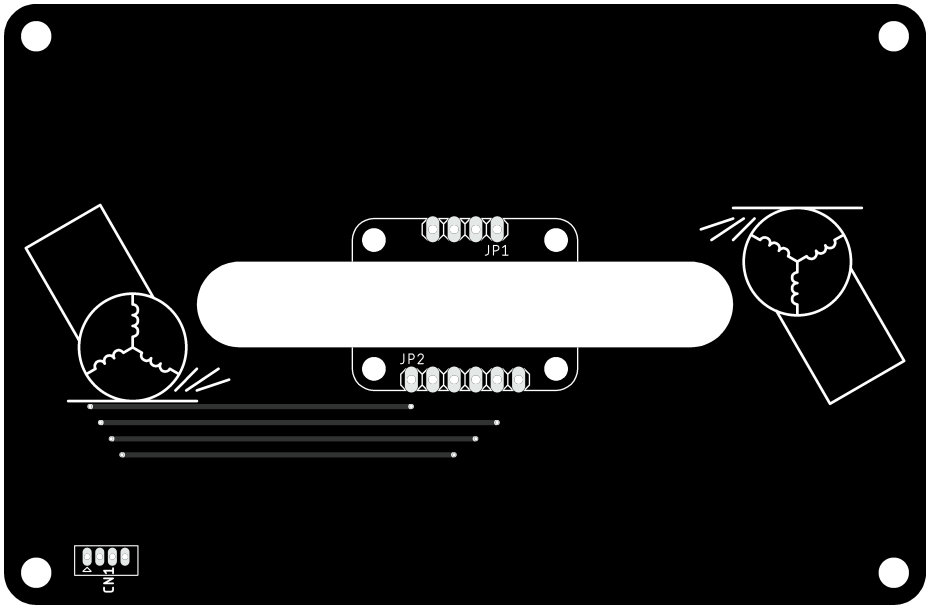


Figure C.7: Render of PCB revision 1.1, created with Autodesk Eagle; IMU board shown.

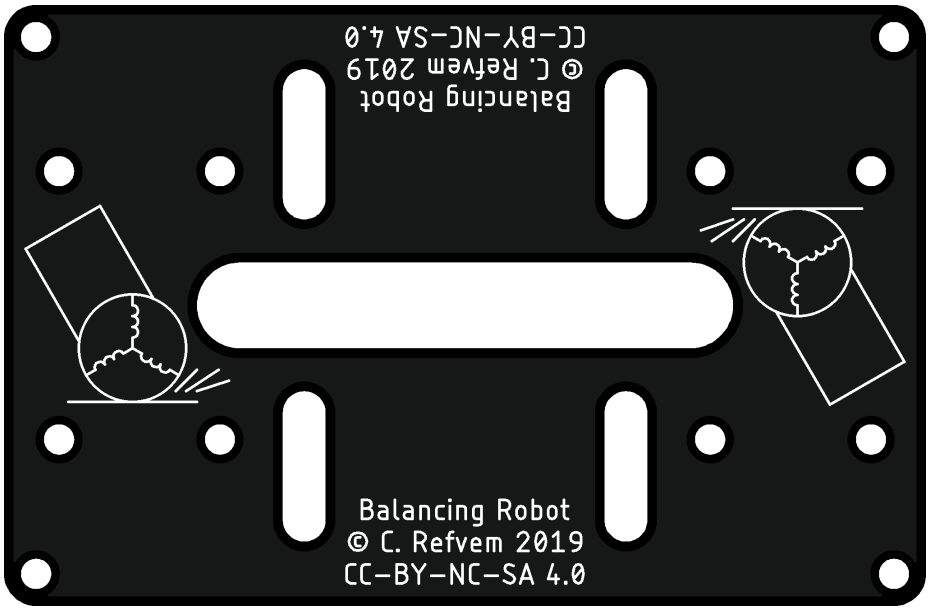


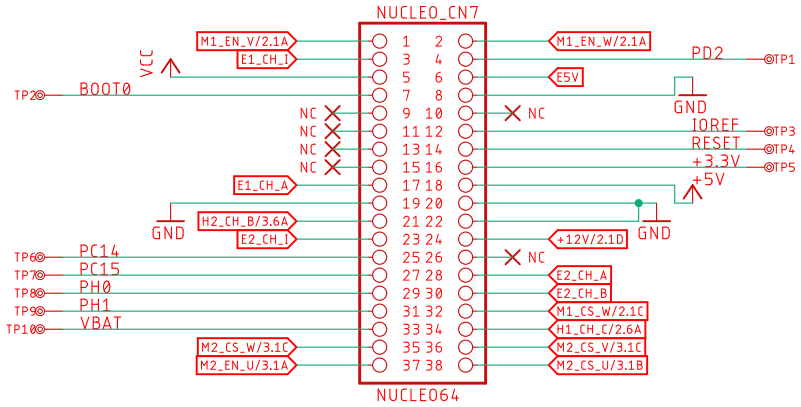
Figure C.8: Render of PCB revision 1.1, created with Autodesk Eagle; motor board shown.

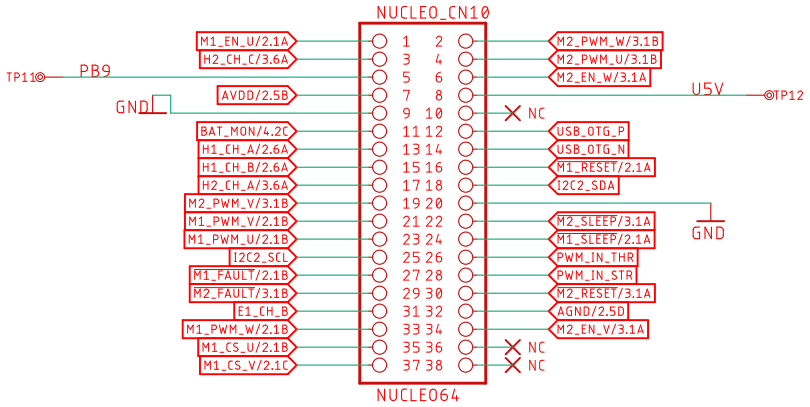
Appendix D

CIRCUIT DESIGN

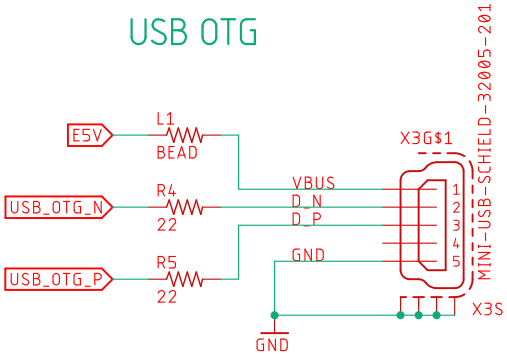
The following schematic is for the final design implementation of the balance bot PCB as described in C.

Nucleo Morpho Headers

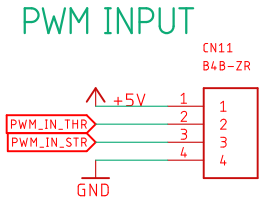




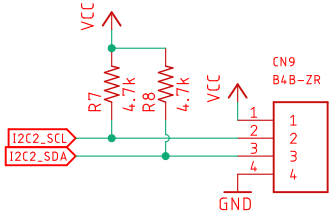
USB OTG



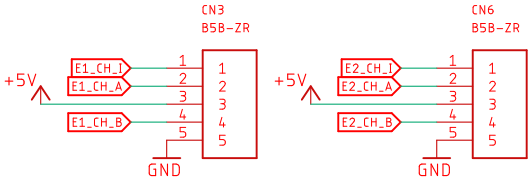
PWM INPUT



I2C



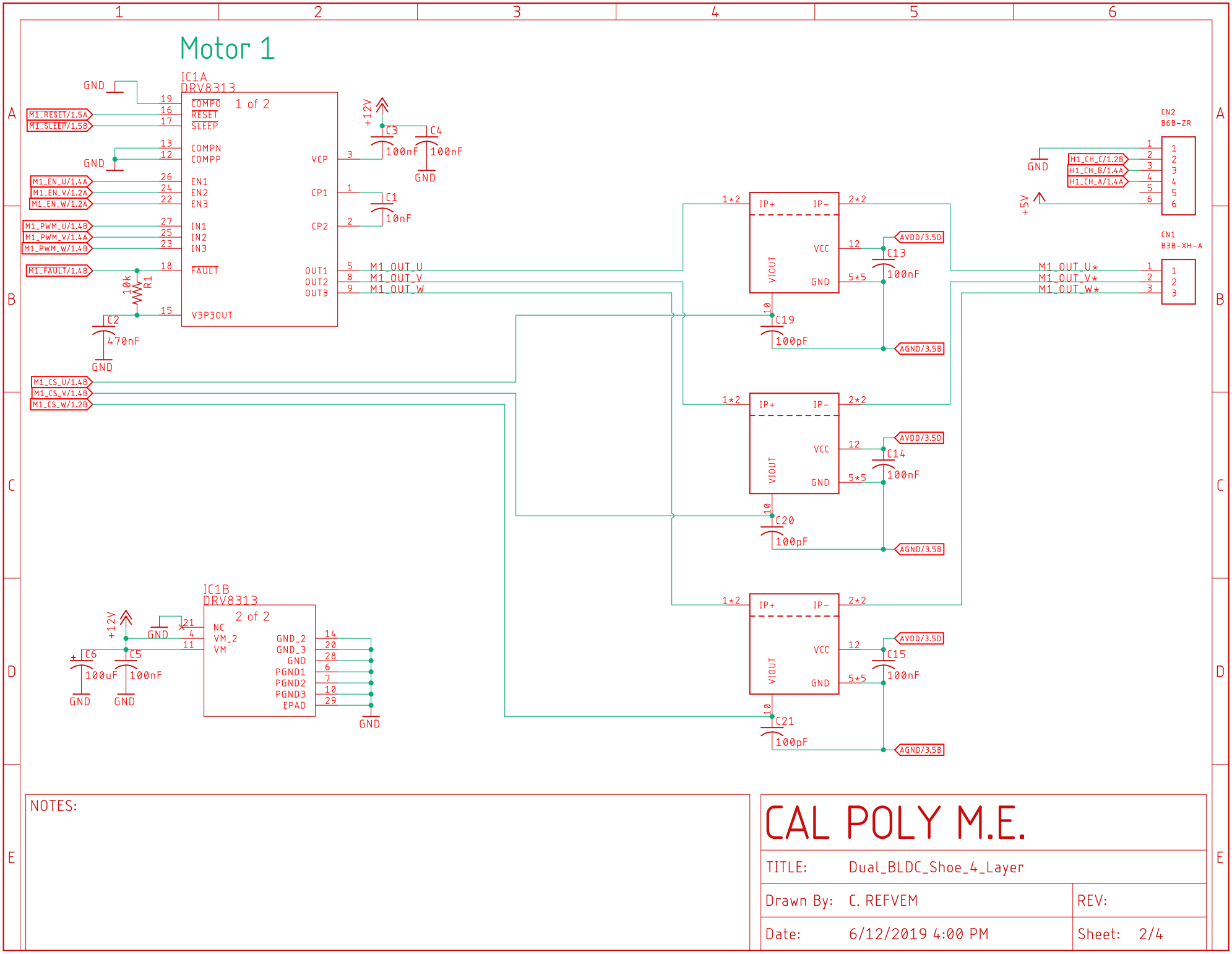
Encoders



NOTES:
- Connectors have been changed to top mount connectors mounted on the bottom side of the PCB to allow cables to pass through PCBs

CAL POLY M.E.

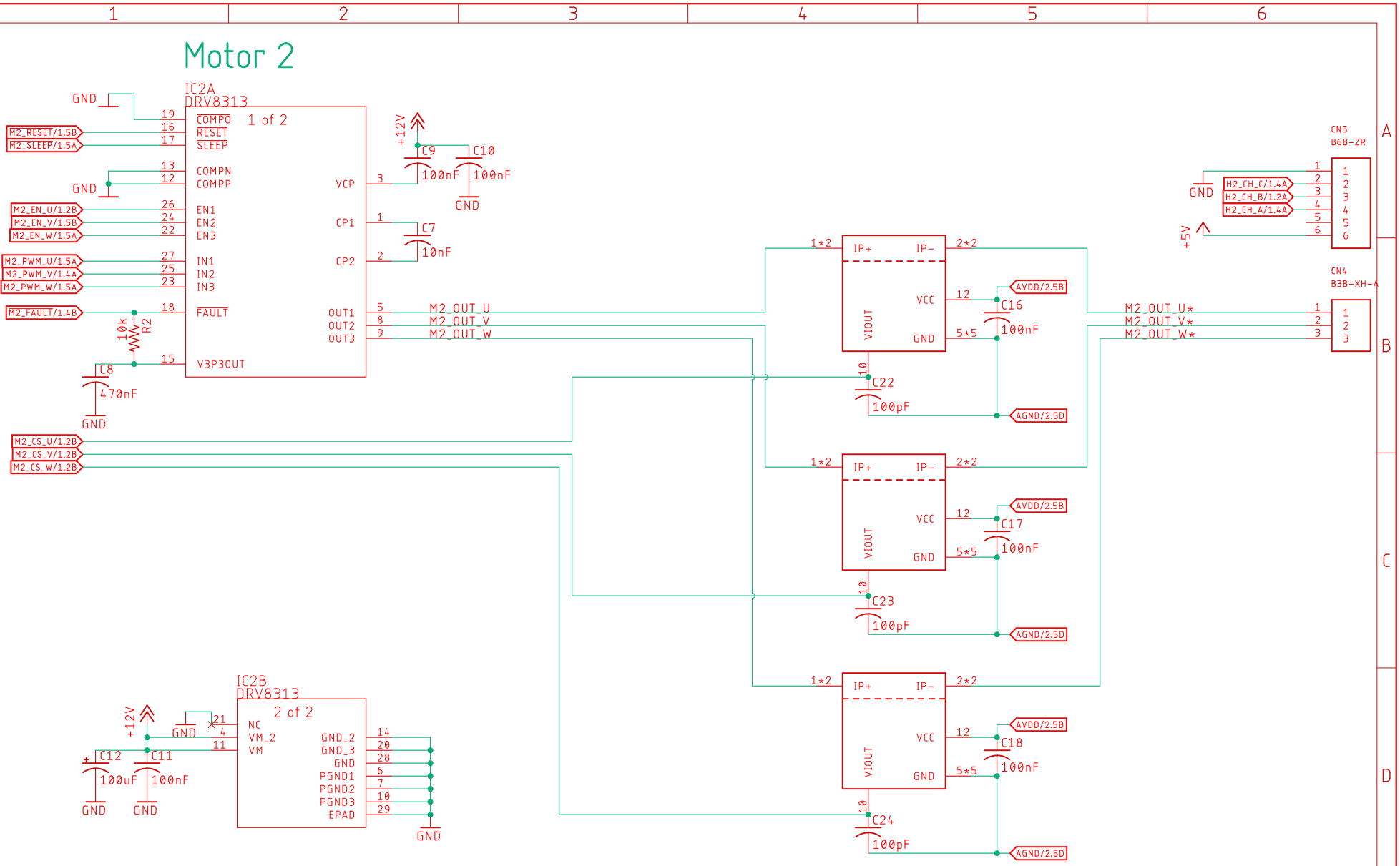
TITLE:	Dual_BLDC_Shoe_4_Layer		
Drawn By:	C. REFVEM	REV:	
Date:	6/12/2019 4:00 PM	Sheet:	1/4



NOTES:

CAL POLY M.E.

TITLE:	Dual_BLDC_Shoe_4_Layer		
Drawn By:	C. REFVEM	REV:	
Date:	6/12/2019 4:00 PM	Sheet:	2/4

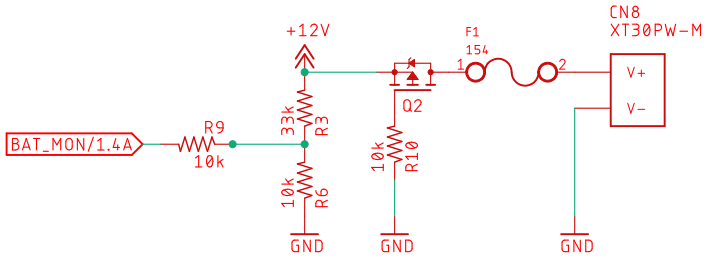


NOTES:

CAL POLY M.E.

TITLE:	Dual_BLDC_Shoe_4_Layer		
Drawn By:	C. REFVEM	REV:	
Date:	6/12/2019 4:00 PM	Sheet:	3/4

Fused power input



NOTES:
- Use ≤10A fast acting fuse
- Q2 provides reverse voltage protection
- R3, R6, R9 form a resistor divider to measure the battery voltage

CAL POLY M.E.		
TITLE:	Dual_BLDC_Shoe_4_Layer	
Drawn By:	C. REFVEM	REV:
Date:	6/12/2019 4:00 PM	Sheet: 4/4

Appendix E

SIMULATION RESULTS

E.1 Simulation #1: 2-DOF Open-loop Testing

An open-loop free-response is a logical first simulation as it allows the underlying dynamic model to be considered by itself. Consider the plots shown in Figures E.2 through E.5 which depict the 2-DOF system model's free response due to initial conditions of $x = 0$, $\theta = \pi/24$, $\dot{x} = 0$, and $\dot{\theta} = 0$. These initial conditions represent a configuration where the balance bot is motionless and nearly upright, with a deviation of 15° from vertical.

Given these initial conditions, one would expect the balance bot to fall downward under gravity and begin oscillating, similar to a pendulum, about a downward hanging position[1]. As the balance bot swings back and forth, the wheels also roll forward and backward such that the composite center of mass for the entire system moves very little in the horizontal direction.

All of the 2-DOF open-loop plots were produced using data from Simulink®; a screen capture of the block diagram used for the 2-DOF open-loop simulation is shown in Figure E.1.

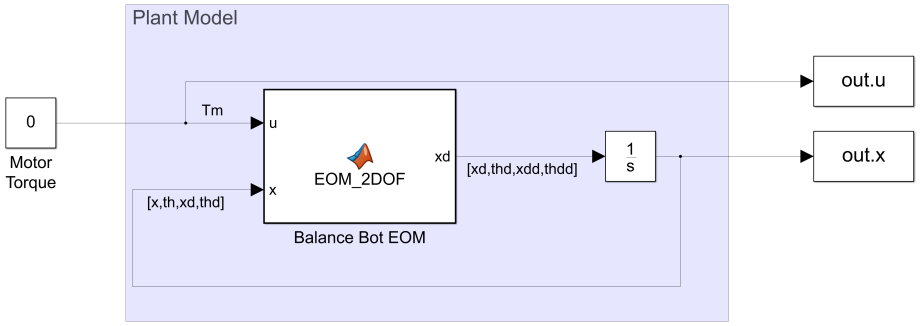


Figure E.1: Block diagram for open-loop 2-DOF model.

The 'Matlab Function' block titled `EOM_2DOF` is used to implement the equations of motion for the balance bot. The contents of this function are shown below in the following code listing:

[1] If this test were performed with the physical hardware the bot would fall until it hits the surface it is driving on. The system model does not consider the driving surface, so in simulation the balance bot can hang upside down.

Listing E.1: 2-DOF equations of motion function file: `EOM_2DOF.m`.

```
function xd = EOM_2DOF(u,x,p)
% u = [ Tm ]
%
% x = [ x
%       th
%       xd
%       thd ]
%
% p = data structure containing model constants

c = cos(x(2));
s = sin(x(2));

M = [   1   0   0           0
        0   1   0           0
        0   0   p.mx        p.mb*p.rb*c
        0   0   p.mb*p.rb*c p.Ith ];

b = [   x(3)
        x(4)
        -2/p.rw*u + p.mb*p.rb*s*x(4)^2
        2*u + p.mb*p.rb*s*p.g ];

xd = M\b;
```

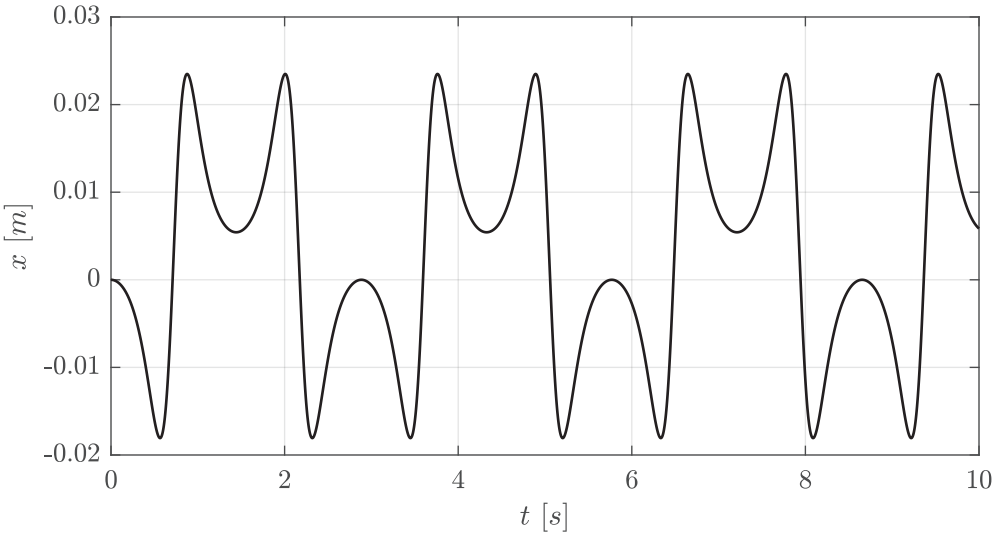


Figure E.2: Open-loop response for 2-DOF model: horizontal displacement, x.

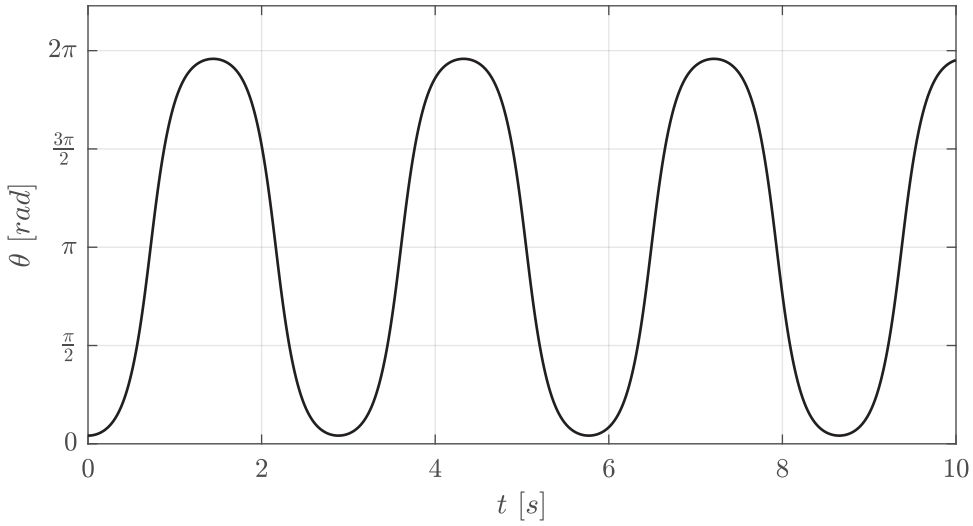


Figure E.3: Open-loop response for 2-DOF model: pitch angle, θ.

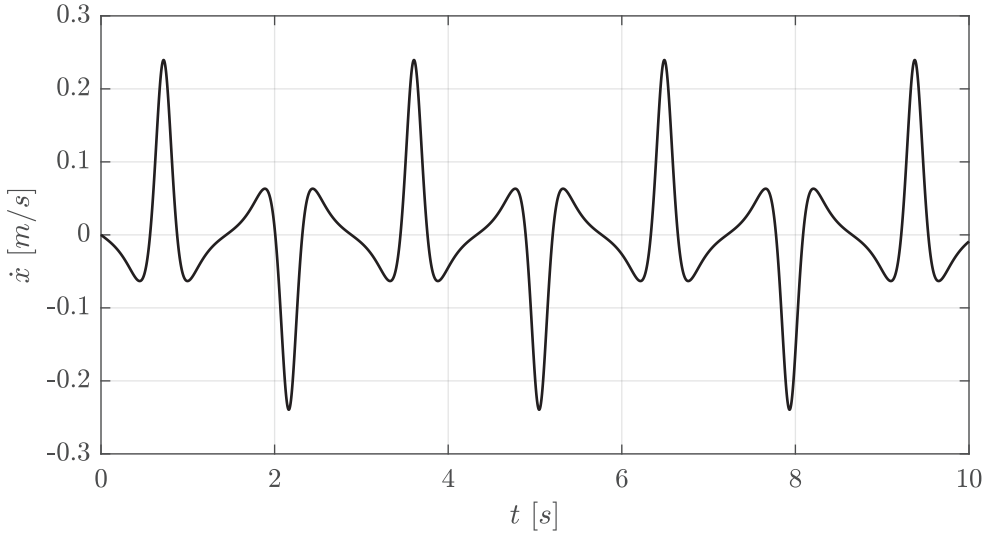


Figure E.4: Open-loop response for 2-DOF model: horizontal velocity, $\dot{x}$.

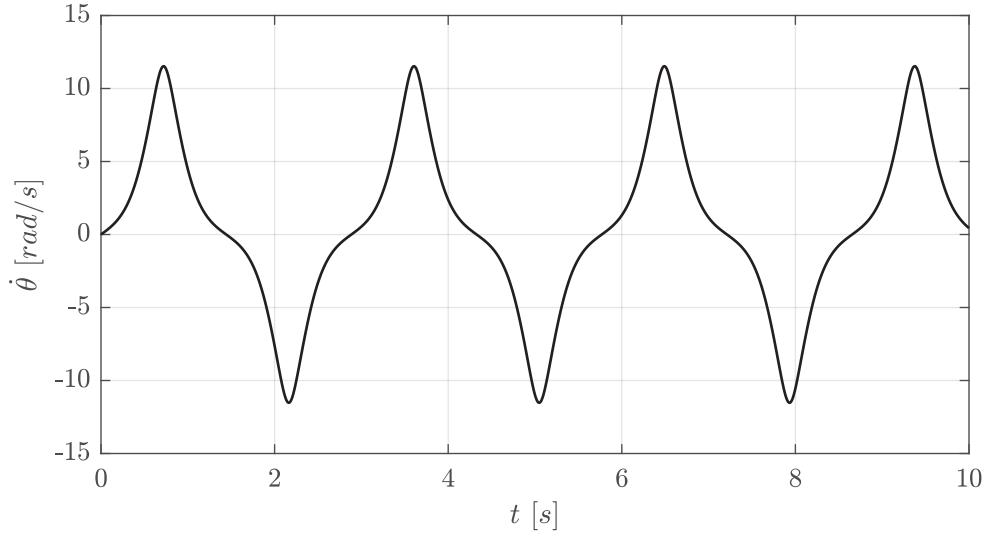


Figure E.5: Open-loop response for 2-DOF model: pitch velocity, $\dot{\theta}$.

E.2 Simulation #2: 2-DOF Closed-loop Testing

Once the open-loop testing was finished, testing proceeded toward closed-loop simulation. Closed-loop simulation validates the proposed control laws. The open-loop Simulink® model was modified for state-feedback and is shown in Figure E.6. The 'Matlab Function' block titled `EOM_2DOF` is unchanged for this simulation.

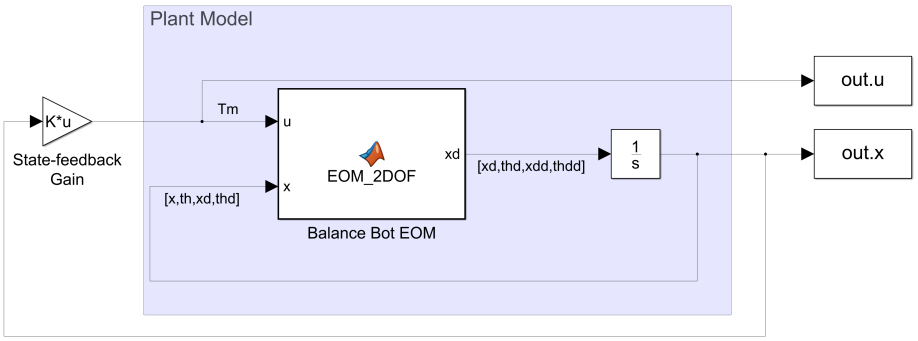


Figure E.6: Block diagram for closed-loop 2-DOF model.

Consider Figures E.7 through E.11 which depict results from a closed-loop simulation with the same initial conditions as those for the open-loop test shown above.

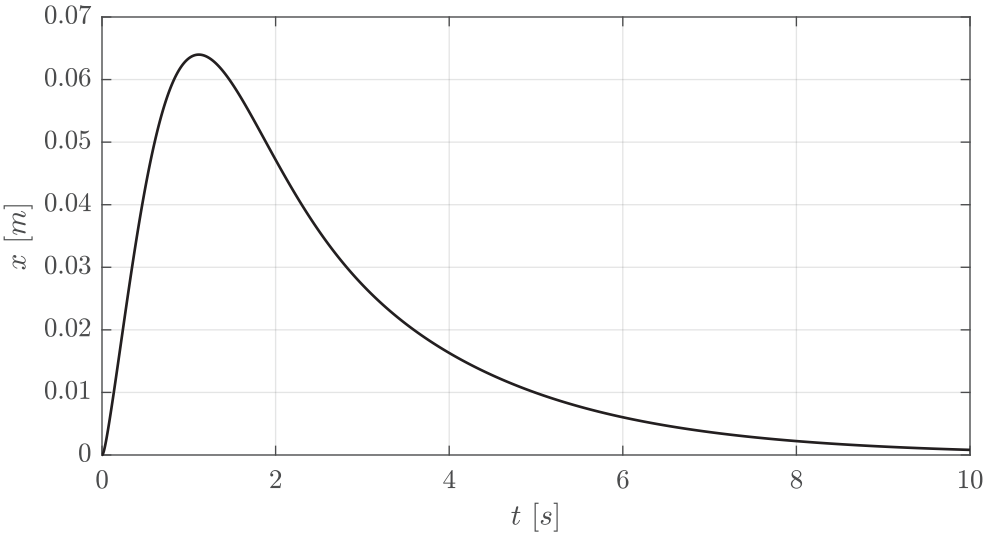


Figure E.7: Closed-loop response for 2-DOF model: horizontal displacement, x.

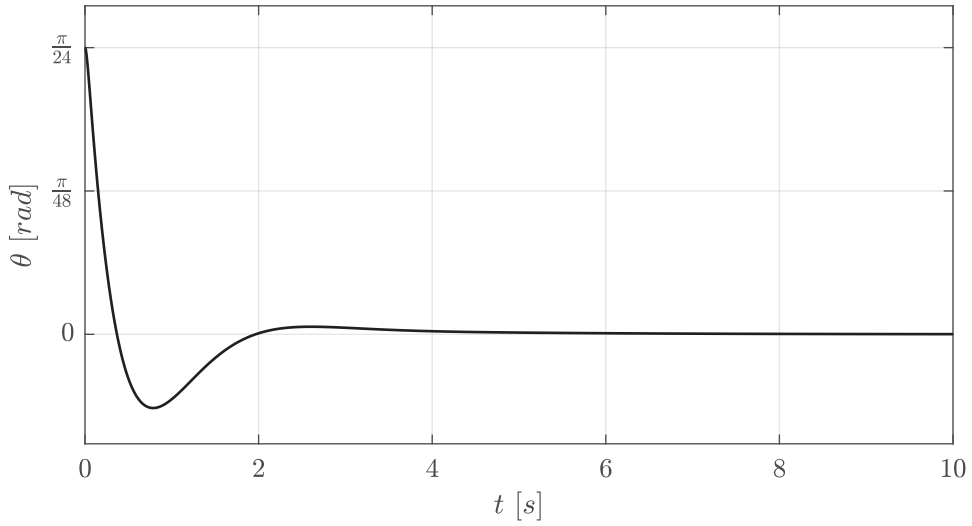


Figure E.8: Closed-loop response for 2-DOF model: pitch angle, θ.

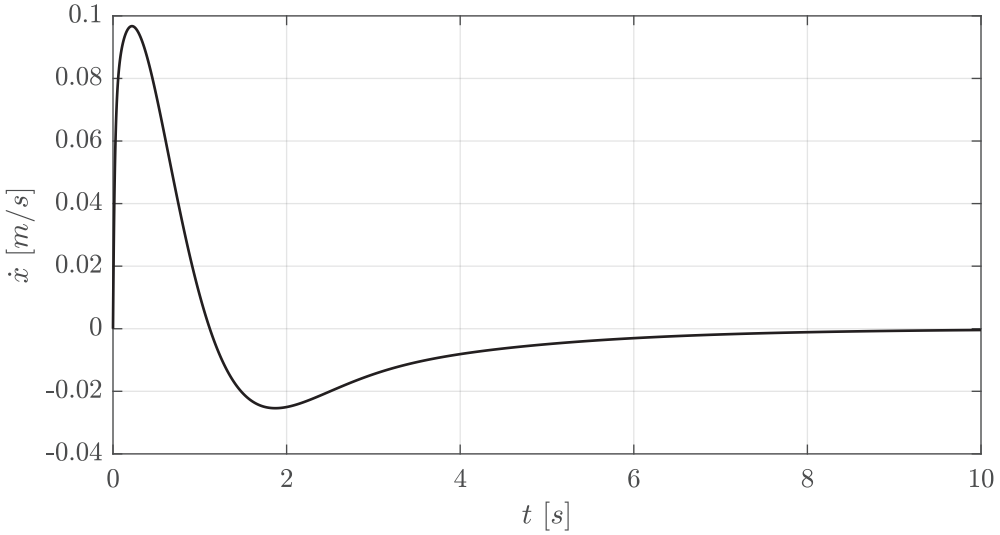


Figure E.9: Closed-loop response for 2-DOF model: horizontal velocity, $\dot{x}$.

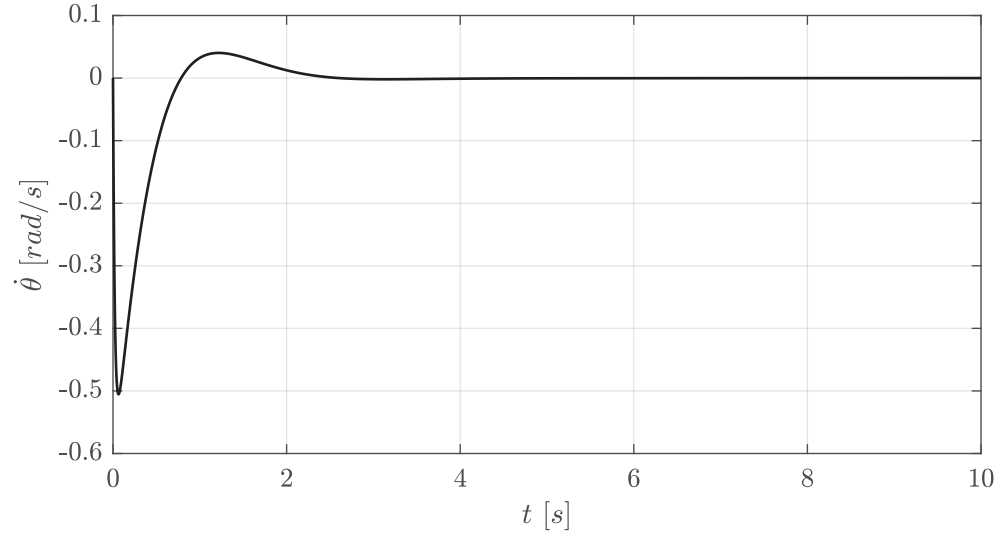


Figure E.10: Closed-loop response for 2-DOF model: pitch velocity, $\dot{\theta}$.

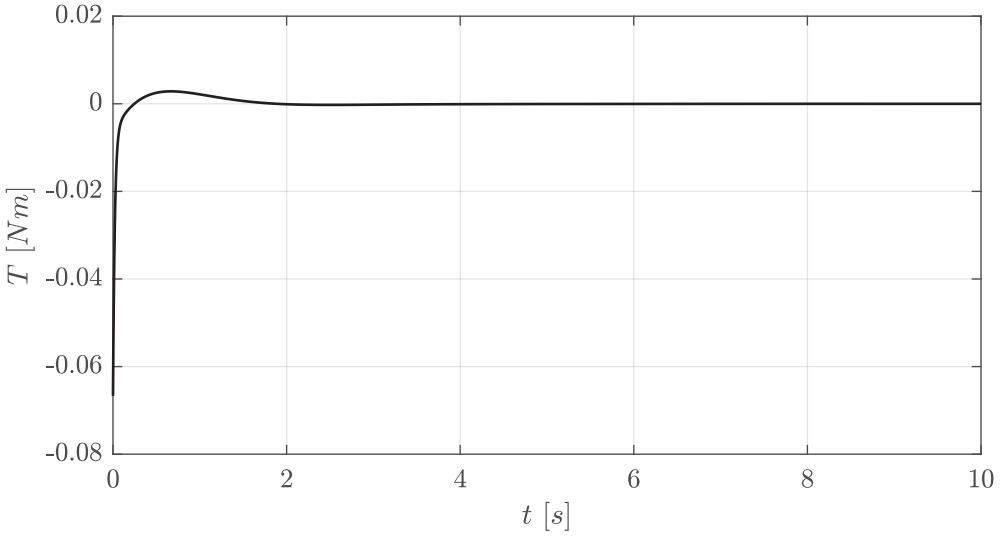


Figure E.11: Closed-loop response for 2-DOF model: input torque, T_m.

E.3 Simulation #3: 3-DOF Closed-loop Testing

Now that the 2-DOF model is validated, the 3-DOF model is run in closed-loop with the same initial conditions as the previous 2-DOF tests. The plots shown in Figures E.13 through E.19 compare the 2-DOF results with the new 3-DOF results to confirm that the two models are consistent. One key takeaway from these plots is the negligible affect that balancing has on the yaw angle ψ and yaw velocity $\dot{\psi}$; close consideration of Figure E.15 and Figure E.18 shows that the yaw angle and yaw velocity are only affected on the order of 10^{-15} $[rad]$, which clearly shows that the yaw axis is nearly uncoupled from the pitch and horizontal axes. In fact, after linearization, the motion in the yaw axis is entirely uncoupled from the motion in other axes, suggesting that near the operating point the non-linear model will be nearly uncoupled as well.

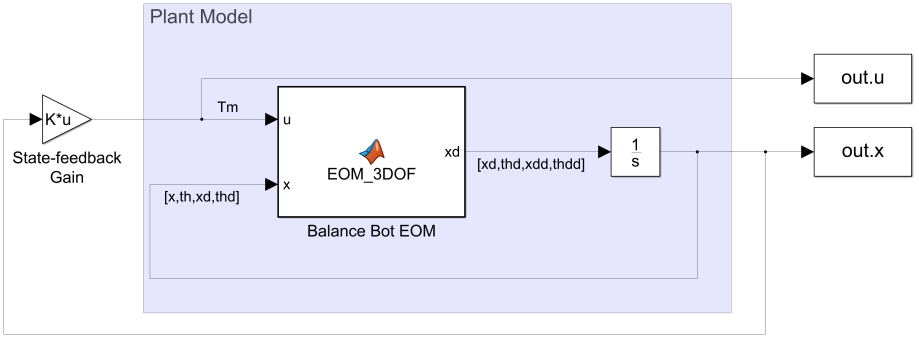


Figure E.12: Block diagram for closed-loop 3-DOF model.

The 2-DOF closed-loop Simulink® model was modified for the 3-DOF model and is shown in Figure E.12. The 'Matlab Function' block titled `EOM_3DOF` is updated for the 3-DOF model as well. The contents of this function are shown below in the following code listing:

Listing E.2: 3-DOF equations of motion function file: `EOM_3DOF.m`.

```
function xd = EOM_3DOF(u,x,p)
% u = [ Tmr
%       Tml ]
%
% x = [ x
%       th
%       psi
%       xd
%       thd
%       psid ]
%
% p = data structure containing model constants

c = cos(x(2));
s = sin(x(2));

M = [   1   0   0   0           0           0
        0   1   0   0           0           0
        0   0   1   0           0           0
        0   0   0   p.mx        p.mb*p.rb*c 0
        0   0   0   p.mb*p.rb*c p.Ith       0
        0   0   0   0           0           p.mb*p.rb^2*s^2+p.Ibxx*s^2+p.Ibzz*c^2+p.Ips];

b = [   x(4)
        x(5)
        x(6)
        -1/p.rw*u(1) + 1/p.rw*u(2) + p.mb*p.rb*s*x(5)^2
        u(1) - u(2) + p.mb*p.rb*s*p.g + (p.mb*p.rb^2 + p.Ibxx - p.Ibzz)*s*c*x(6)^2
        -p.w/(2*p.rw)*u(1)-p.w/(2*p.rw)*u(2)-2*(p.mb*p.rb^2+p.Ibxx-p.Ibzz)*s*c*x(5)*x(6)];

xd = M\b;
```

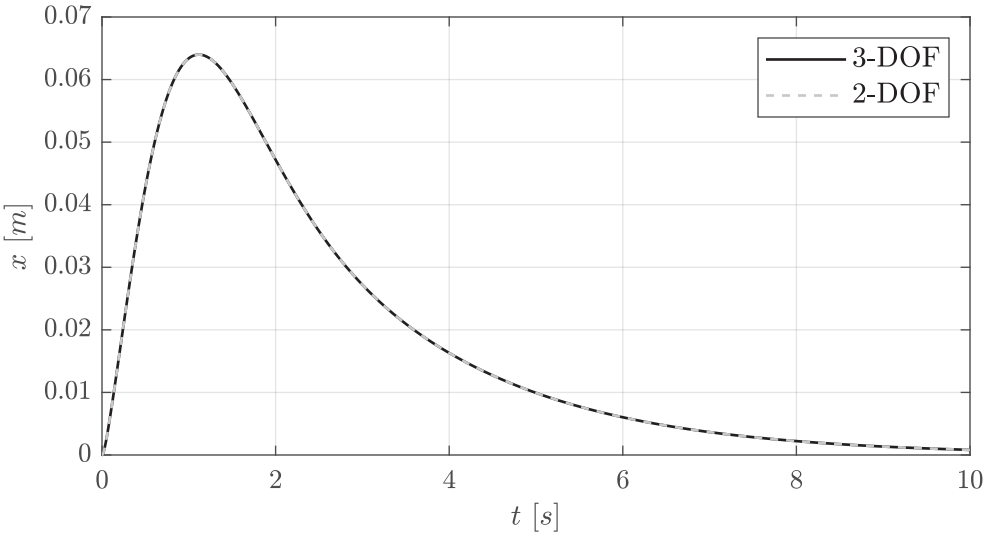


Figure E.13: Closed-loop response for 3-DOF and 2-DOF models: horizontal displacement, x.

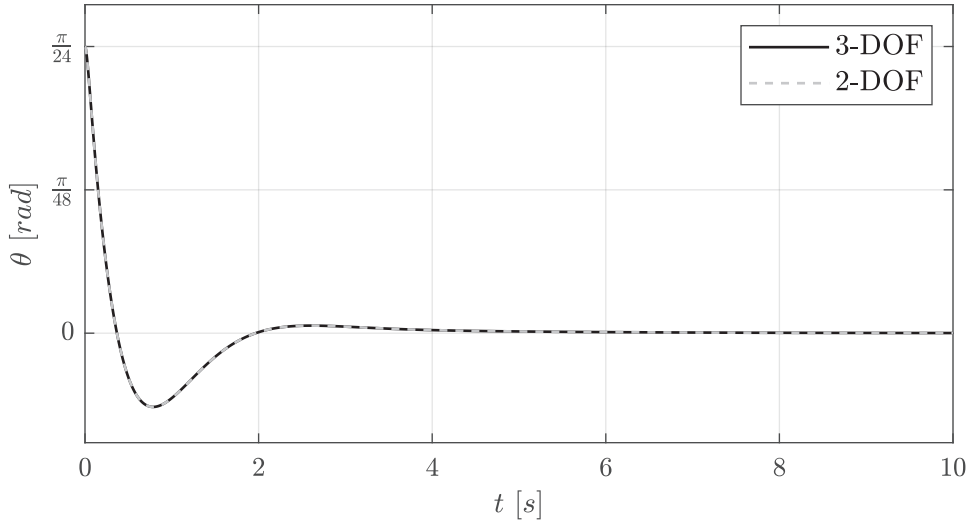


Figure E.14: Closed-loop response for 3-DOF and 2-DOF models: pitch angle, θ.

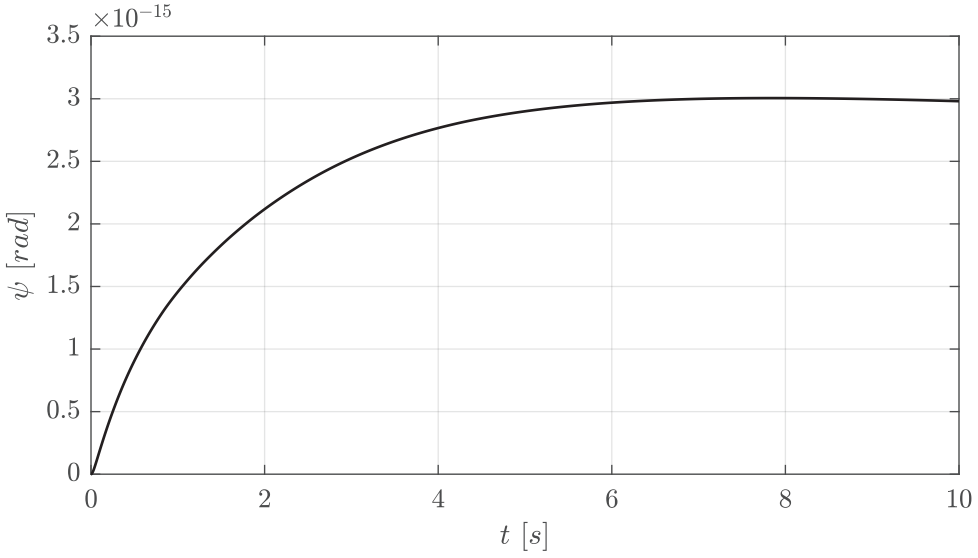


Figure E.15: Closed-loop response for 3-DOF model: yaw angle, ψ.

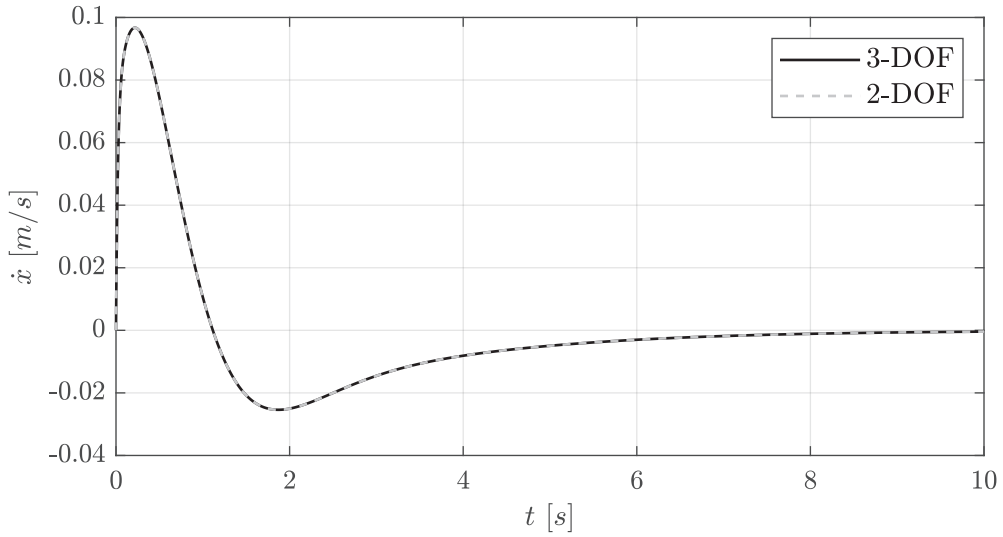


Figure E.16: Closed-loop response for 3-DOF and 2-DOF models: horizontal velocity, $\dot{x}$.

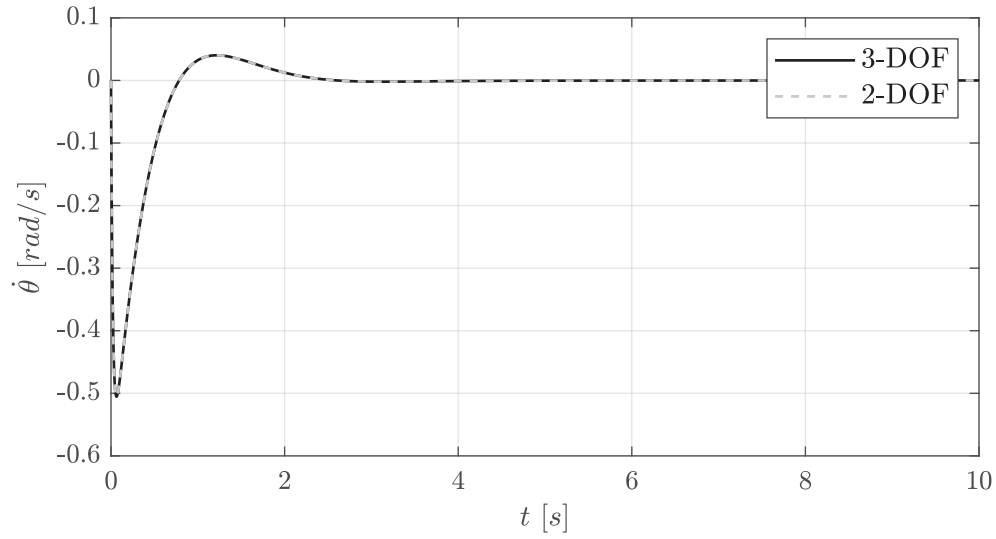


Figure E.17: Closed-loop response for 3-DOF and 2-DOF models: pitch velocity, $\dot{\theta}$.

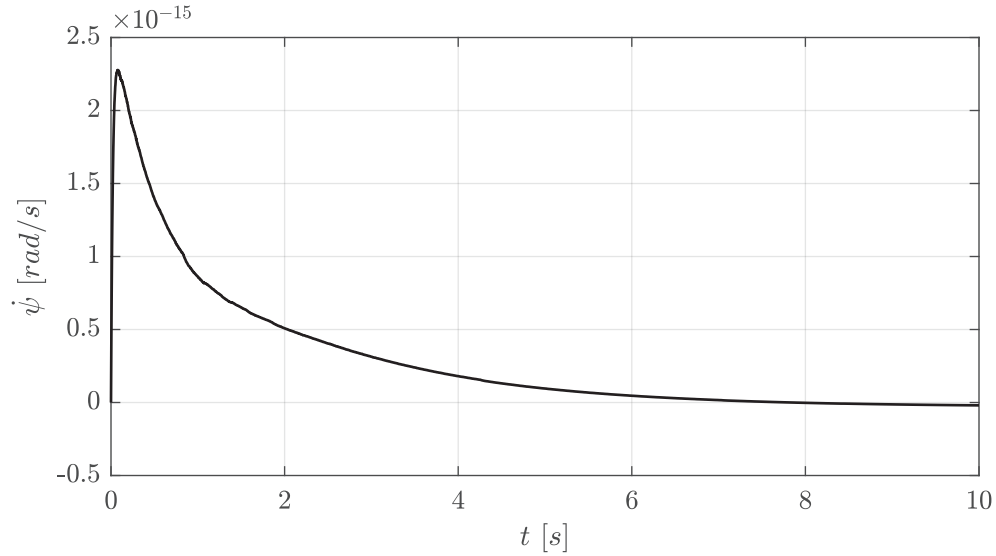


Figure E.18: Closed-loop response for 3-DOF model: yaw velocity, $\dot{\psi}$.

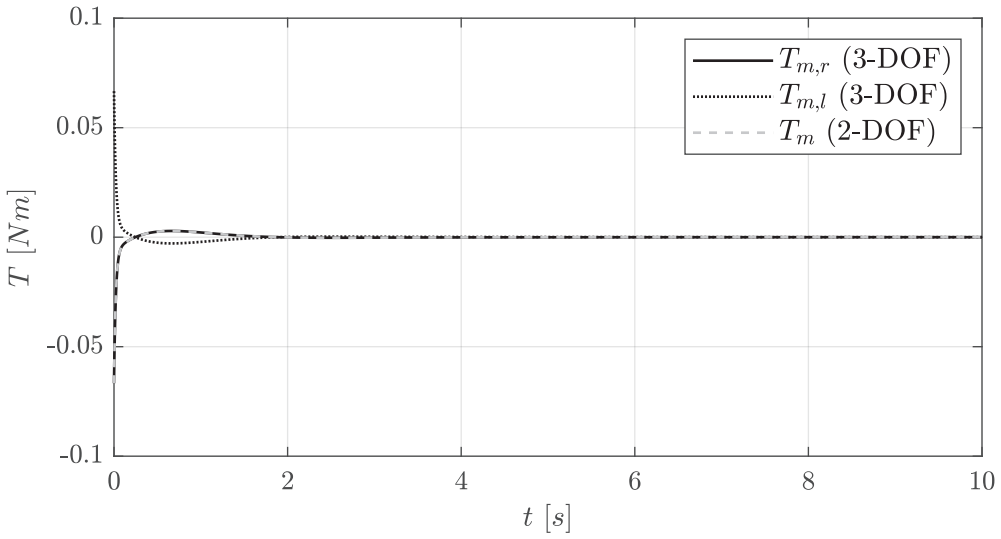


Figure E.19: Closed-loop response for 3-DOF and 2-DOF models: input torque, T_m.

E.4 Simulation #4: 3-DOF Closed-loop Testing with Prefilter

Both the 2-DOF and 3-DOF system models have been validated in closed-loop, but both models only regulate the states back to zero. The final simulation shows that the system can not only regulate the system states, but also track a desired trajectory. The plots in Figures E.23 through E.29 show the reference states coming from the prefilter in addition to the tracking results from the 3-DOF model. For this simulation the initial conditions are all set to zero, but time-dependent throttle and steering commands are present. The throttle and steering commands are both driven by rectangular pulses.

The 3-DOF closed-loop Simulink® model was modified for the 3-DOF tracking model and is shown in Figure E.20. The 'Matlab Function' block titled `EOM_3DOF` remains unchanged for this simulation.

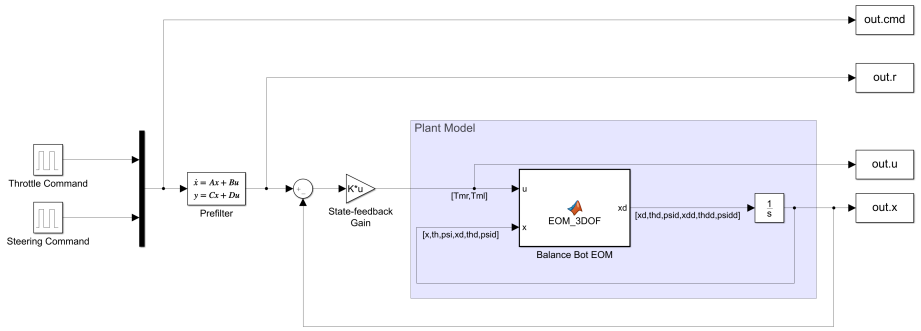


Figure E.20: Block diagram for closed-loop 3-DOF model.

The throttle command, shown in Figure E.21, is a rectangular pulse with a width of 2 $[s]$ and a height of 1 $[m/s]$ occurring 1 $[s]$ after the simulation begins. This command will cause the bot to drive forward 2 $[m]$ over a duration of 2 $[s]$.

The steering command, shown in Figure E.22, is a rectangular pulse with a width of 2 $[s]$ and a height of π $[rad/s]$ occurring 11 $[s]$ after the simulation begins. This command will cause the bot to pivot counter-clockwise one full revolution over a duration of 2 $[s]$.

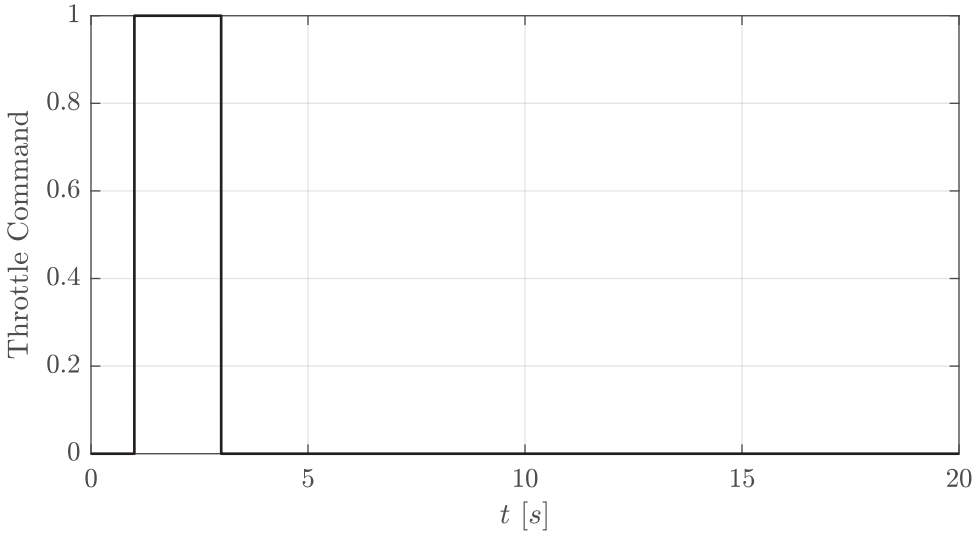


Figure E.21: Throttle command for closed-loop 3-DOF model.

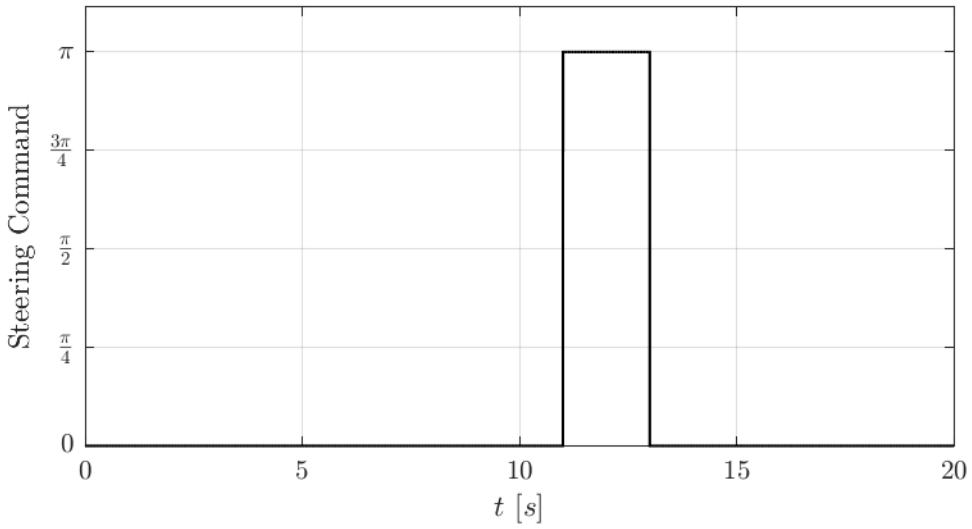


Figure E.22: Steering command for closed-loop 3-DOF model.

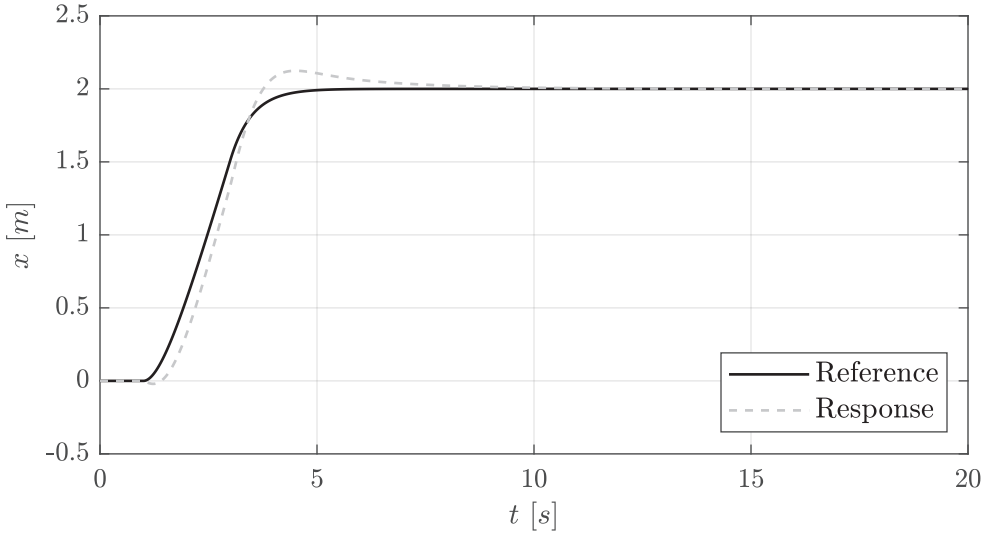


Figure E.23: Tracking response for 3-DOF model: horizontal displacement, x.

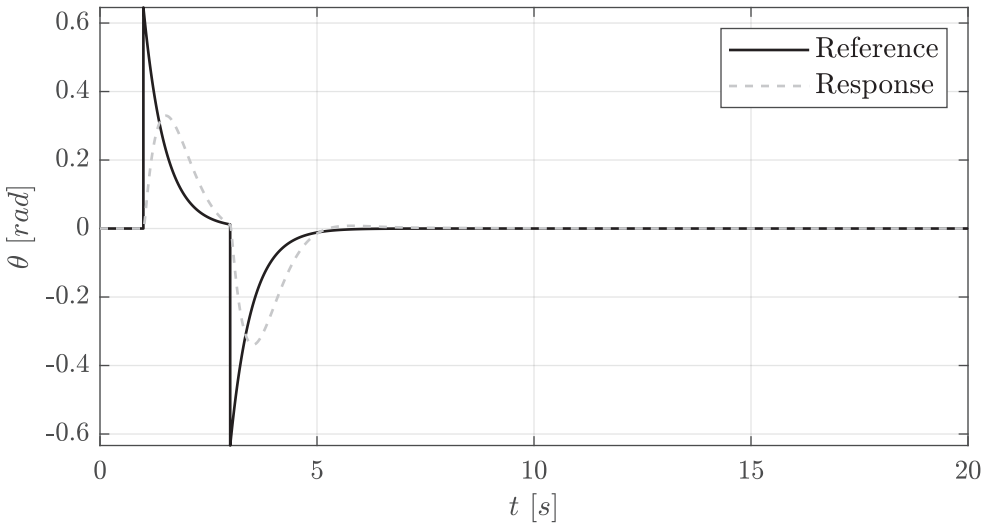


Figure E.24: Tracking response for 3-DOF model: pitch angle, θ.

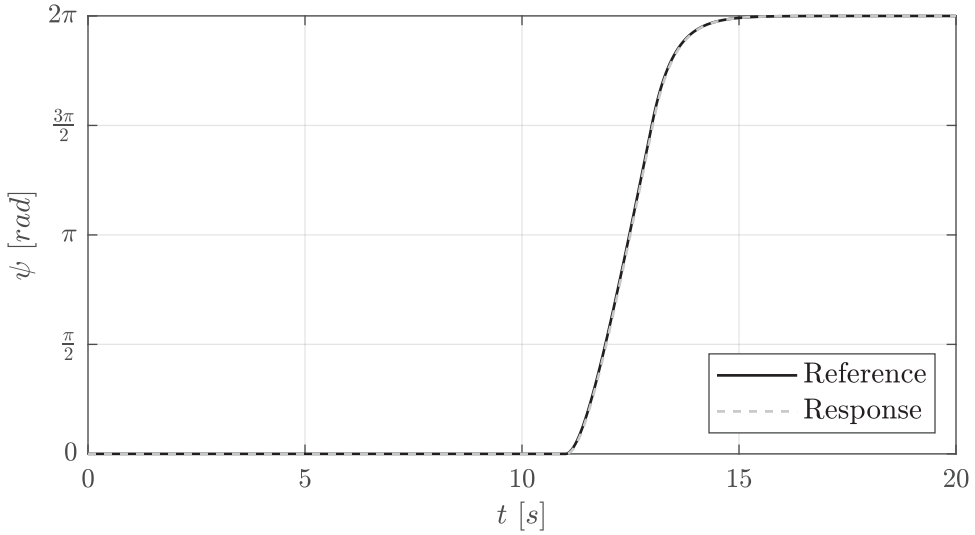


Figure E.25: Tracking response for 3-DOF model: yaw angle, ψ.

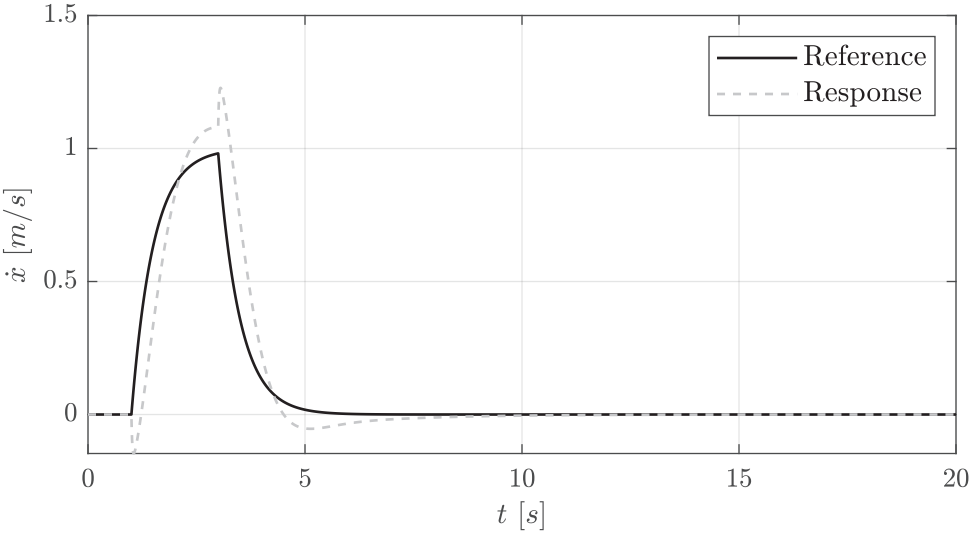


Figure E.26: Tracking response for 3-DOF model: horizontal velocity, $\dot{x}$.

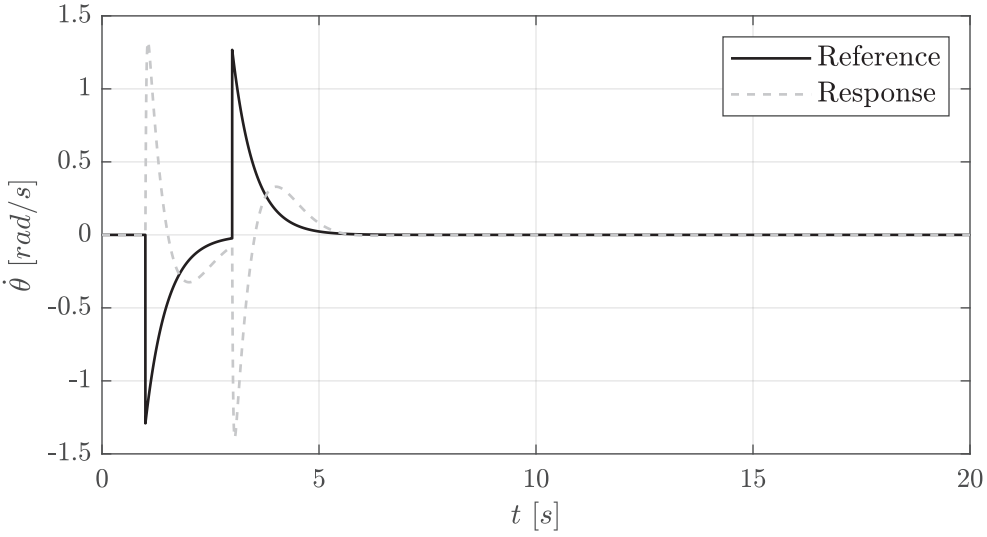


Figure E.27: Tracking response for 3-DOF model: pitch velocity, $\dot{\theta}$.

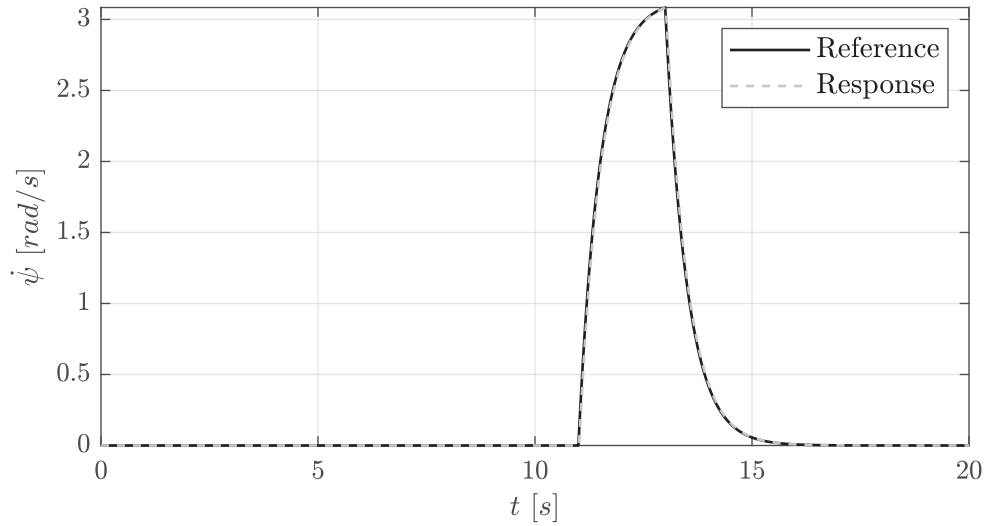


Figure E.28: Tracking response for 3-DOF model: yaw velocity, $\dot{\psi}$.

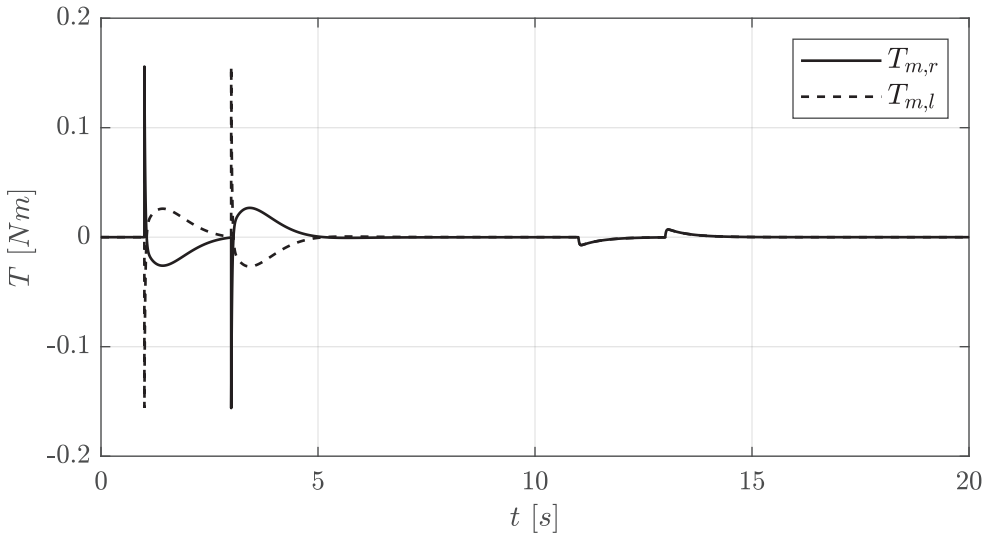


Figure E.29: Tracking response for 3-DOF model: input torque, T_m.

Appendix F

SOFTWARE DOCUMENTATION

The source code and documentation for the entire project can be found online at `https://bitbucket.org/charlierefvem/balancebot`.