Design and Implementation of a Real-Time Controller for a 3-Wheeled Collapsible Vehicle

by

Brandon Smyth

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Engineering
(Computer Engineering)
in the University of Michigan-Dearborn
2020

Master's Thesis Committee:

Associate Professor Sridhar Lakshmanan, Chair
Professor Paul Richardson
Lecturer Paul Muench

Acknowledgements

I would like to thank Dr. Sridhar Lakshmanan and Dr. Paul Muench for providing the idea and general background for this thesis, and specifically Dr. Lakshmanan for providing feedback and relief during the roadblocks encountered during the development of the project. Special thanks to Cristian Adam and Katharina Grenn for providing the mechanical design and manufacturing for the vehicle, as well as providing moral support during the development of the project. I would further like to thank Dr. Michael Putty for his assistance with the control loops and signal processing along with providing useful feedback on design ideas for the controller software and communication protocols.

Table of Contents

List of Figures

List of Tables

Abstract

This thesis focuses on designing and implementing a real-time vehicle controller. The vehicle controller is a microcontroller that is connected to a set of sensors, such as an inertial measurement unit, and actuators, such as drive motors. The controller takes advantage of Kalman filters and PI controllers to calculate, in real-time, feedback messages and control actuations. This report describes what methods were chosen and implemented in the controller and why they were chosen. It describes a communication protocol to create an interface with the controller for a computer to interact with, describes how a real-time operating system is leveraged to use multitasking to operate different controllers and feedback processes, and describes how the hardware is implemented and tested. The communication protocol is described in depth as to its format and interaction. It provides a set of tests performed on the various algorithms to show their effectiveness in creating the real-time control and accurate feedback. Finally, the paper sets out a set of future works that can be achieved by leveraging this vehicle controller to provide autonomous control and self-balancing systems as is desired for this vehicle's final product.

Chapter 1: Introduction

1.1 Problem Statement

There are times when it is beneficial for an autonomous vehicle to be narrow to be able to navigate through narrow areas, or to reduce drag and allow for higher speed travel. However, narrow vehicles are inherently unstable and prone to falling over, especially at low speeds. A potential solution to this is a 3-wheeled vehicle that could collapse its wheel track to allow for more narrow travel locations, as well as higher speeds, but maintaining its balance by expanding the track to compensate for when the vehicle begins to fall over. This allows for travel in narrow spaces or at high speed while maintaining stability. Having a third wheel will also allow significantly greater stability when travelling on unstable terrain. The vehicle will also be able to carry payloads when in its expanded form. While unable to navigate narrow passages in this mode, on a return trip it may be able to take a more optimal route in narrow areas. This thesis will explore the embedded system design and control systems vehicle dynamics and the control systems required to record and receive data to control the vehicle remotely or via a more powerful computer for autonomous control, as well as a machine learning algorithm to attempt to mimic a human controlling the balance of the vehicle. The investigation will be performed using a scale model of the proposed vehicle design. Data collected from a human driver will be used to facilitate supervised training on a neural network in an attempt to control the balance of the vehicle.

1.2 Design Overview

This thesis investigates a design for a control system for a dynamic 3-wheeled vehicle. The vehicle is designed such that it will be able to carry loads, with the rear wheels extended out, at low speeds and when unloaded will be able to collapse the rear wheels in and navigate back to a set location at higher speeds. Specifically, this paper will investigate the real-time control systems for the entire vehicle (steering control, vehicle speed, rear track control, etc.) and sensor data feedback. Self-balance will be achieved by expanding the rear wheel track to compensate for the vehicle tilting as it travels. The expanded track will self-right the vehicle preventing the vehicle from rolling over and allowing it to continue its journey in a narrow cross-sectional area of travel along the ground. The vehicle will initially be controlled remotely by hand, recording data about the state of the vehicle. This data can then be used to train a neural network via supervised learning [2] to allow a computer to map the vehicle dynamics of self-balance, without the engineer or computer needing to fully know the underlying physical behavior.

The prototype design will be a scale model in which the steering and speed of the vehicle are remotely controlled. The prototype will be used to act as a proof of concept for the vehicle control algorithms, the control interface, and the systems needed to train and interface with a neural network that can be taught to map the non-linear vehicle dynamics of vehicle balance.

1.3 Review of Existing Strategies and Related Works

The most common works explored that relate to this thesis would be the self-balancing bicycle. It is a complex problem to solve as it is deeply involved with both vehicle dynamics and nonlinear control systems, due to the inherent problem of the instability of a bicycle. There is also a lot of work that can be explored regarding unstable vehicle balancing, sensor fusion, and

control algorithms. Neural networks also provide an interesting research route as they can allow a system to derive a control algorithm without knowing the underlying dynamics, relying solely on data it has been trained on and current feedback data.

1.3.1 Control Methods and Algorithms

The self-balancing bicycle has been a problem commonly studied in vehicle dynamics for years. There have been multiple methods implemented to solve this problem, including a gyroscope, a moving mass, and steering. The method most related to this thesis is the steering method, as it deals more with controlling the vehicle with its wheels rather than balancing by adjusting a gyroscope or heavy mass.

In the paper written by S. Vatanashevanopakorn and M. Parnichkun, they investigate creating a self-balancing bicycle by creating a mathematical model for both the bicycle itself and the motor that controls the steering [1]. The models for the bicycle and motor are used in a set of two Proportional Integral Derivative (PID) control loops. They also use a Linear Quadratic Regulator (LQR) to set the optimal gain for the different states of the control loop, such as the current steering angle and current lean. Simulations performed on their model and algorithm showed that their bicycle would stabilize in about 2.5 seconds from an initial lean of 5 degrees.

1.3.2 Sensor Fusion and Pose Estimation

W. Ding, et al. created a set of algorithms that allow them to estimate the attitude (specifically, pitch and roll) of a tricycle. The issue they encountered is that standard Inertial Measurement Unit (IMU) sensors are not accurate enough on their own. Accelerometers are slow to react and prone to high-frequency noise, while gyroscopes are prone to drift over time. Using a Kalman filter they can fuse the output of these two sensors together to mitigate the flaws of the

separate sensors. Because the tricycle is a nonlinear system a standard Kalman filter cannot create accurate enough data, so the authors instead use an extended Kalman filter. The extended filter is a nonlinear variant that can linearize an estimate of the information. In their design they have two different modes for fusing the gyroscope measurements. During the dynamic movement of the tricycle they integrate the values from the gyroscope directly into the pitch and roll estimations, but when the tricycle is in a static state, they feed the gyroscope output into the Kalman filter. A standard filter would simply use the piece they use in the static state. Through their experimentation they were able to prove that this change to the filter eliminated erroneous changes in the attitude estimation when the tricycle switched from a dynamic to a static state. This paper provided a good set of algorithms on how to provide accurate vehicle state estimation while building upon commonly used algorithms to mitigate their problems.

A great investigation written by M. Nowicki, et al. was created to compare the extended Kalman Filter with the complementary filter [4]. They wanted to see if a complementary filter had comparable accuracy in orientation estimation of a device when compared against an extended Kalman filter. A complementary filter is a sensor fusion filter that works very well with IMU data, while remaining relatively simple in design. In its simplest form it applies a low-pass filter on the accelerometer data and weights it against the gyroscopic data. This provides the short-term accuracy that the gyroscope provides, while using the low-weighted accelerometer to diminish the drift of the gyroscope in the long term. The complementary filter is very useful in an embedded environment where computing resources are limited. Its simplicity means that it uses very little memory and computation time. The authors also note that being much simpler, it has only 1 parameter that would need to be tuned, making accurate orientation estimate projects quick to implement. They did notice that the complementary filter was slightly less accurate than

the extended Kalman filter, by about 2–4%, depending on the device they used to test with. They noted that the extended Kalman filter was only slightly slower in computation time in their tests but note that this is likely due to careful implementation of new mathematical libraries that were provided for the Android operating system and their particular hardware. Overall, they believe that the choice of filter used depends entirely on the project requirements, with the designer needing to consider their required precision and time needed to implement the filter.

1.3.3 Deep Learning Control

As computers are becoming more powerful it is becoming much more realistic to train a neural network in control processes and have them operate in real time. This allows for the creation of self-driving cars and other systems that control themselves based on data previously collected in relation to the task at hand. A neural network can be taught to steer a car entirely on its own by using just the raw data from cameras [5]. Collecting steering wheel angle data and synchronizing it with camera data allowed researchers to teach a neural network what angle the steering wheel needed to be at just by using the camera data of a camera pointed towards the road in front of the vehicle. This allowed the network to determine what angle the steering wheel needed to be at in a very human-like fashion. These systems, however, take in only the current information from the camera and do not take into account how the system previously set its steering angle. Systems like these could benefit from using a recurrent neural network, as shown in Chi, Lu, and Yadong Mu [6] and Eraqi, H. M., Moustafa, M. N., and Honer, J. [7]. These papers utilized Long Short-Term Memory (LSTM), which allows a neural network to remember information from previous forward passes. This allows the neural network to learn information temporally as well as spatially, as the feedback lets the network remember

previous information [8]. This can be helpful in control systems, as they are sometimes reliant on previous states, such as what is required in a PID controller.

1.4 Hardware and Signal Layout

The heart of the project is a microcontroller, the TI *TM4C123G*, that handles the real-time communication and signals processing. The entire prototype hardware consists of the microcontroller, an inertial measurement unit (IMU), 5 electronic speed controllers (ESCs), 5 brushless DC (BLDC) motors, and 2 potentiometers. The IMU communicates over the inter-integrated circuit (I^2C) protocol and provides acceleration and gyroscopic data. The ESCs are used to control the motors as they high power electronics and control methods to drive them. Each ESC has a universal asynchronous receive transmit (UART) serial connection to allow external controllers to send commands and receive feedback through a high-level protocol. The 2 potentiometers are used as absolute position sensors for feedback on control loops described in chapter 4. All of these peripherals need to be able to be read and/or written to by the microcontroller. A basic schematic of how the peripherals are connected is shown in Figure 1 below.

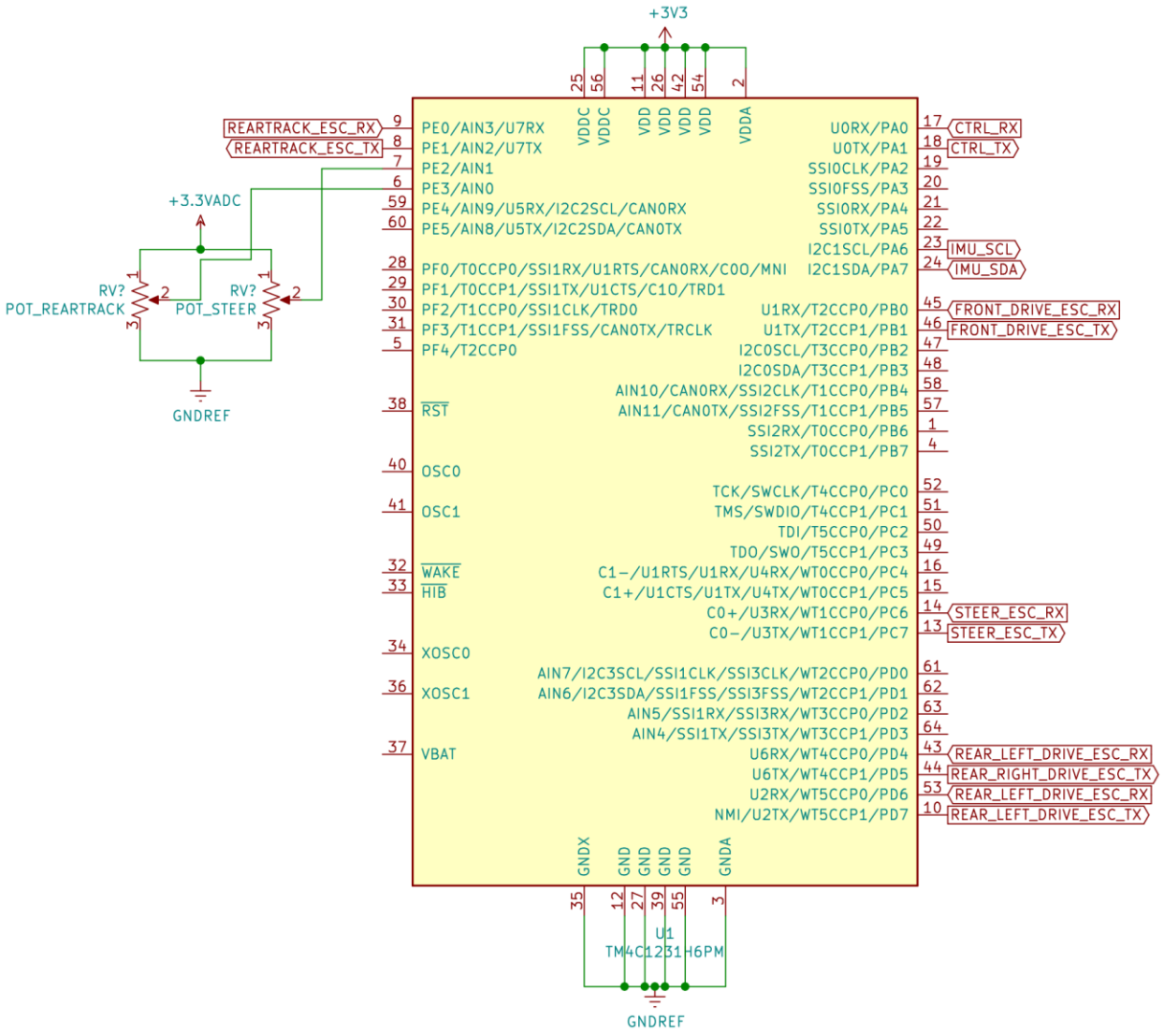

Figure 1. Microcontroller Peripheral Hardware Layout

The vehicle design, shown in Figure 2, shows how the actuators and sensors are mounted on the vehicle. Using this system, it is possible to build a machine that can fully control itself in real time and also provide sensor and vehicle status feedback to a machine. This is common in vehicle systems, with the unique opportunity in this vehicle to control a collapsible rear set of wheels.

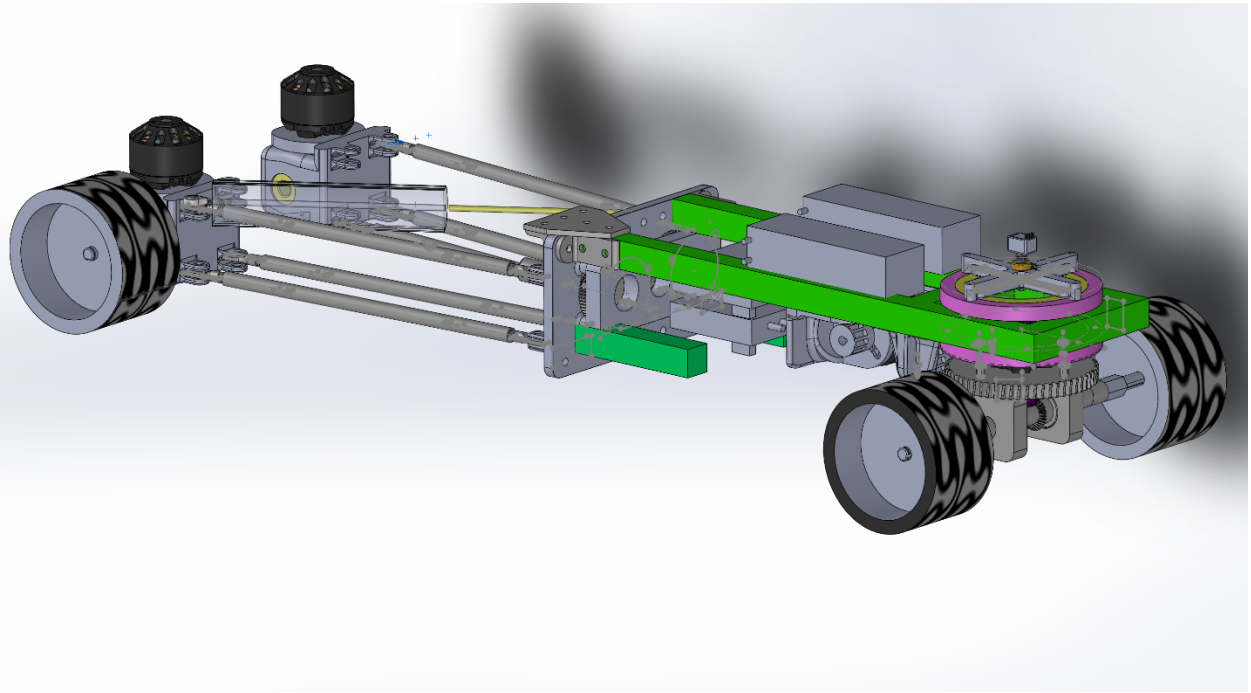

Figure 2. Model of Overall Vehicle Design

Chapter 2: Controller Software Architecture

The number of control systems, feedback loops, and communication systems is enough to warrant a more complex real-time operating system (RTOS)-based software platform. Using a pre-emptive RTOS allows the ability to run multiple tasks concurrently, with higher-priority tasks (such as sensor data acquisition) being run before the lower-priority tasks (such as setting vehicle speed). This also allows the system to handle interrupts more quickly and reduce interrupt request time by posting data received from an interrupt into a mailbox for a low-priority task to then process when allocated CPU time. Using a multitasking system also allows for finer control of each individual loop. Each control loop can be given its own individual time between executions that best suits the use-case for that loop. A control loop that needs to run every 100 milliseconds can run alongside a loop that needs to run every 20 milliseconds without having to worry about timing concerns of every other loop's execution as the task scheduler will handle when each loop needs to run. Figure 3 shows the sequence of actions performed by each task and how variables are shared between said tasks.

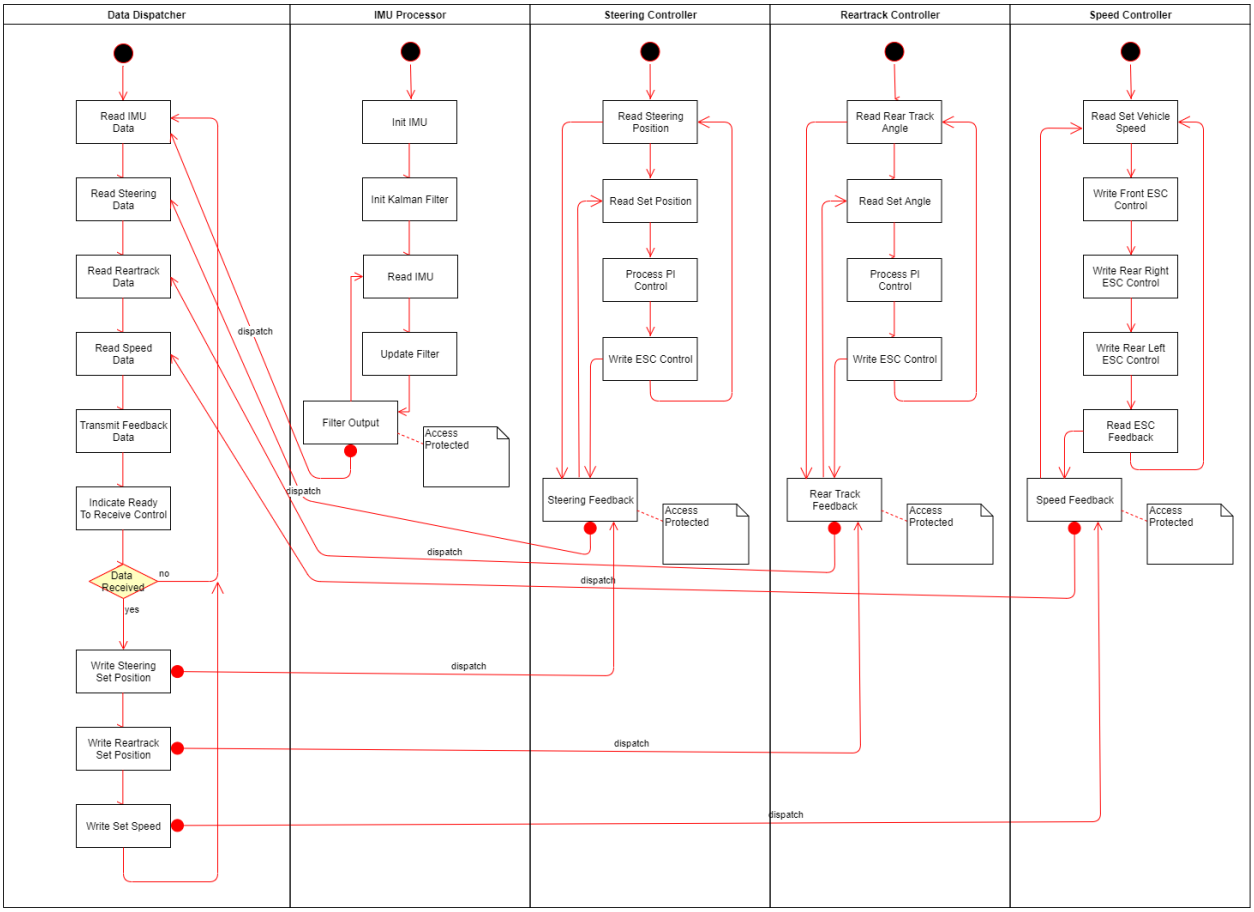

Figure 3. Controller Tasks, Internal Modules, and Inter Task Communications

2.1 Data Dispatcher and Communication Task

The data dispatcher is the most critical piece of the controller software. It is responsible for aggregating together all the data from various components and dispatching them to the various tasks and system peripherals that need the data. This task also acts as the main driver for communications to the outside world (i.e. the AI control computer). It first collects the necessary data from each task, such as current vehicle tilt, vehicle speed, steering position, rear track width, and battery status. It then packs this data into a packet to be sent to the control computer. The task will then negotiate a data transfer with the control computer using the communication protocol described in chapter 3. The control parameters from the control computer are finally received and these are then dispatched to their respective control tasks through either semaphore-

protected global variables or mailboxes, dependent upon whether the tasks require queued data or only cares about the presently requested data state. Figure 4 visualizes the functions called by the task.

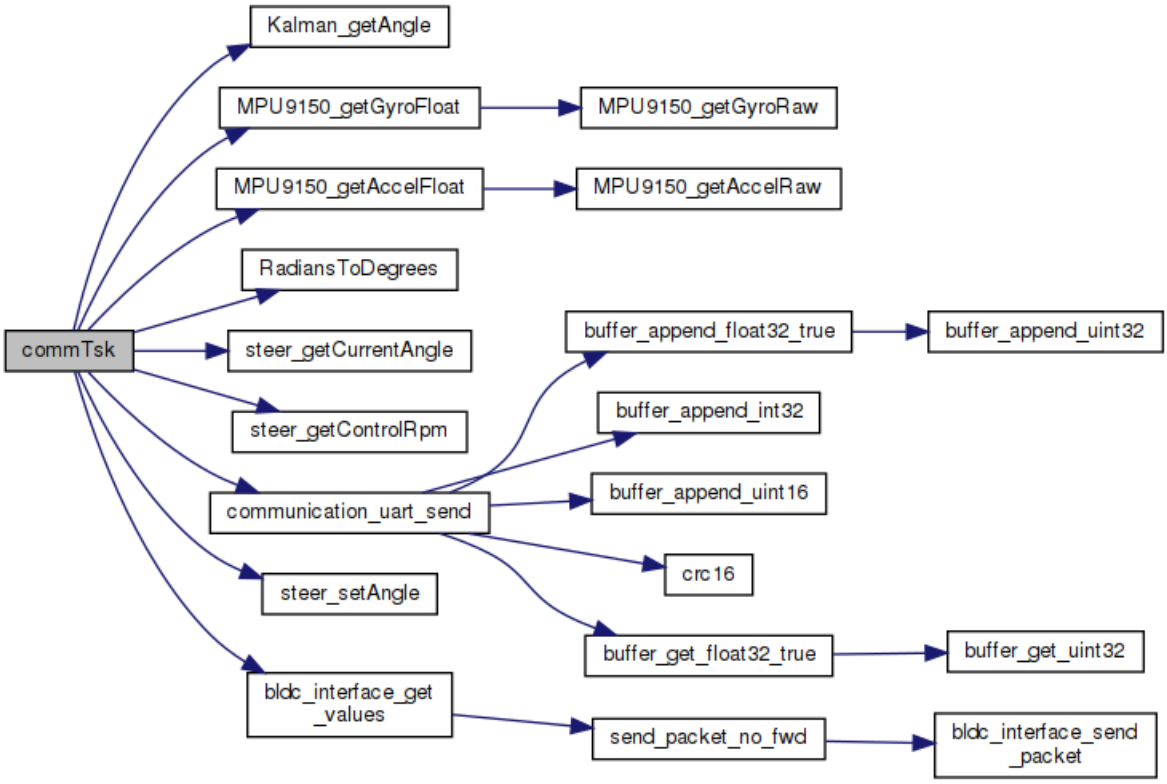

Figure 4. Data Dispatcher Call Diagram

2.2 Inertial Measurement Unit Task

The Inertial Measurement Unit (IMU) task is the task that is responsible for requesting and reading the raw data from the IMU on the I^2C bus. It starts by sending I^2C commands to the IMU to initialize and configure it. It then initializes the Kalman filter, described in chapter 4. It runs at a frequency of 200 Hz with a high task priority; this allows the Kalman filter to react quickly to changes on the IMU and report accurate roll and pitch angles for the vehicle.

At the beginning of each loop iteration, the controller requests the raw accelerometer and gyroscope data from the IMU. It then converts these binary values into values understandable for

humans, acceleration in m/s^2 and radians per second for the accelerometer and gyroscope respectively. These values are then converted into the required format by the task, as described in chapter 4, and the Kalman filter is updated. After the Kalman filter is finished updating, the new filtered values are stored in a mutex-protected global variable.

2.3 Motor Controllers Task

This task is mainly responsible for maintaining state data for the controllers. It waits for data to be received via an interrupt. This interrupt triggers when a byte of data is received over UART from a controller, and it then passes the data via a mailbox to the waiting task. The task processes the data via a state machine. Once a full packet has been received, it then checks the cyclic redundancy check (CRC) to ensure that the packet is valid. If the packet is valid, the task processes it into a local data structure and executes a callback function. This callback function sorts the feedback data from the electronic speed controller (ESC) into an appropriate mailbox for the data dispatcher task to process when needed.

2.4 Control Loops Tasks

There are 3 critical tasks that control the actuators of the vehicle. One manages the steering module, one the rear track control module, and one the vehicle speed. As they are critical, they all operate differently from standard tasks due to the tight timing constraints.

2.4.1 Steering Module Task

The steering module requires tight timing constraints for its control loop to ensure accurate control. This is achieved through using a hardware timer peripheral provided by the microcontroller. The timer interrupt is configured to trigger a hardware interrupt on an even periodic variable. This interrupt is then used to trigger a software interrupt on the RTOS.

Software interrupts preempt every task in the system and therefore run at the highest priority, while also allowing other hardware interrupts to trigger on the system. The software interrupt is used to process the PI controller, covered in chapter 4. The output of the controller is then sent with an event to the task itself, as the task to control the motor can handle being interrupted and run at a less tightly coupled time. The output value is sent via UART to the ESC responsible for controlling the motor on the steering module. Functions called to operate the PI controller are listed in Figure 5.

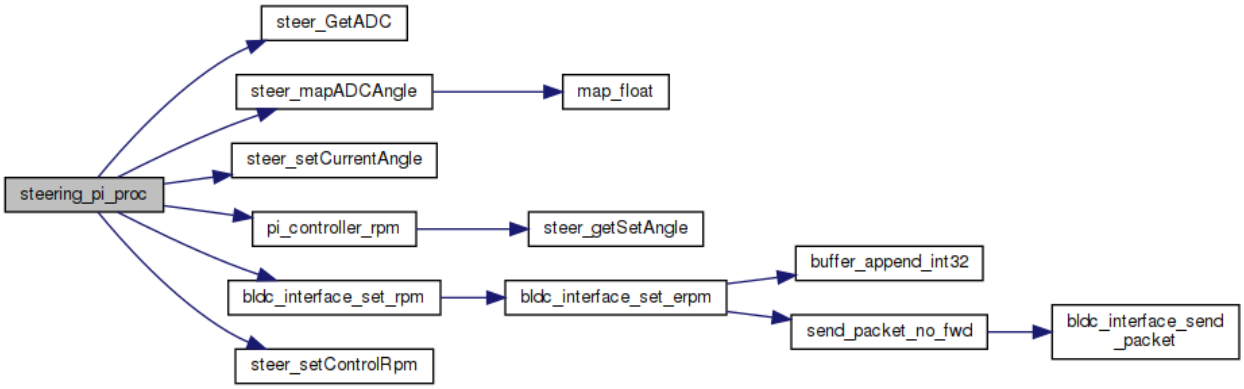

Figure 5. Steering Module Call Diagram

2.4.2 Rear Track Module Task

The rear track module task functions identically to the steering module task but uses a different set of controller peripherals to achieve its control methods. The main differences between the steering module and rear track module are covered in chapter 4.

2.4.3 Vehicle Speed Controller Task

The vehicle speed controller task is an extremely simple task that takes the speed requested by the control computer via the data dispatcher task. It takes the requested speed in m/s and converts it to motor RPM based on a configuration value that sets the diameter of the wheels

used. This task acts as a periodic transmission controller as the data dispatcher task is not guaranteed to receive new data in time to send a new RPM request to the ESC. This task solves that problem by sending a new RPM request at an even periodic interval by using the most recently requested speed.

2.5 Controller Usage and Performance

The processor usage for all these tasks is relatively minimal, occupied mostly by the data dispatcher because it handles most of the data processing. The second most active task is the IMU processing task since it is running the Kalman filter, which has a non-trivial amount of calculations. As can be observed in Figure 6, the overall CPU usage maxes out at around 15%. This leaves plenty of room for future expansion and additional sensors and actuators as required in the future. About 54k of program memory is used, while 23k of system RAM is used. There is plenty of program memory available for more functions, but system RAM usage will need to be reduced through static analysis to determine a more accurate stack size for each task, to minimize each task's RAM usage while preventing stack overflows.

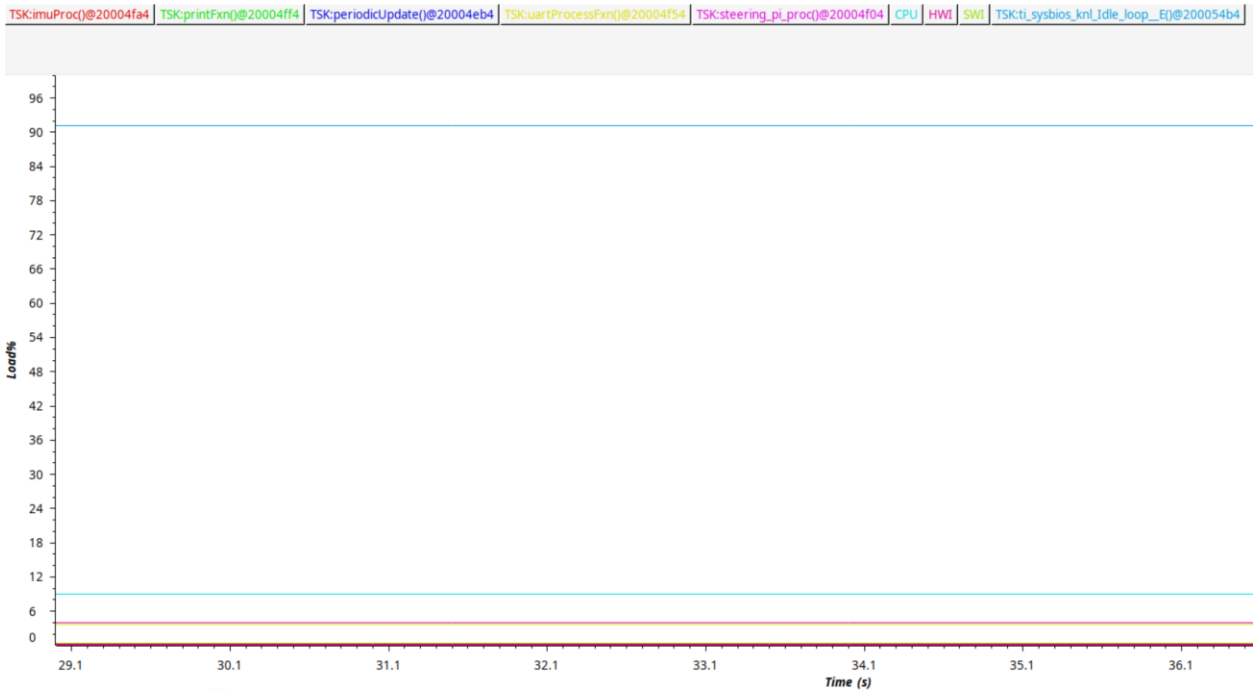

Figure 6. Per Task CPU Usage

Chapter 3: Communication Protocol

In order for the vehicle controller to act as a real-time actuator and sensor feedback computer, it must have a communication protocol to extract this information from the controller for a command computer to process. This protocol is used to synchronize communication between the computers and provide reliable communication. In order to quickly transmit the data, the controller runs the UART peripheral at a high speed (1 megabit/s). This means that even high density packets can be transmitted very quickly, but it drastically increases the need for data integrity checks as there is a higher likelihood of data corruption.

3.1 Communication Flow

It is important to have proper control over the flow of communication between computers. This is accomplished by following a handshake protocol. The technique implemented in this controller is similar to the transmission control protocol (TCP) three-way handshake, which allows the two computers to synchronize their communication and provide a reliable communication path [15]. However, this method acts more like a two-way handshake. In order for the controller to transmit its feedback data, it sends a ready-to-send (RTS) byte to the computer. The computer will then respond with an acknowledgement (ACK), which also acts as a clear-to-send (CTS). This means that the computers are now synchronized, and the controller will then send the feedback data to the computer. The computer can then process this data in any way it deems necessary, for example using it in a neural network to control the vehicle rear track.

For the computer to send control requests to the controller the RTS and ACK are reversed for which side sends each. This protocol is visualized in Figure 7 and Figure 8.

Figure 7. Packet Flow

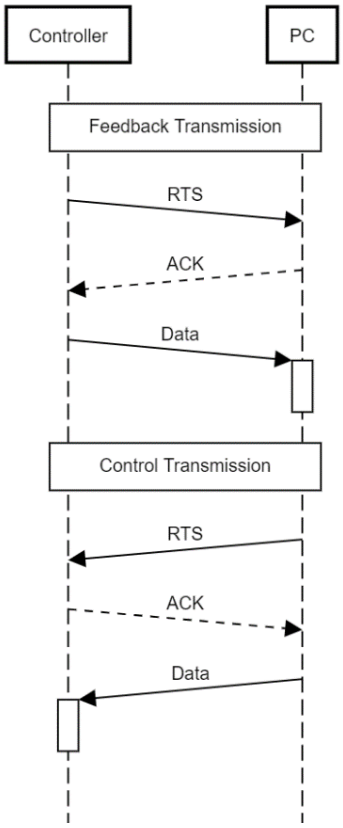

Figure 8. Communication Sequence

3.2 Cyclic Redundancy Check

Data must be reliably exchanged between computers to ensure it is interpreted correctly. Data integrity checking can be accomplished with minimal computation by using the cyclic redundancy check (CRC). This is an error-detecting code that can be used to detect corruption of data in the packet, and if detected throw it away [10]. The algorithm works such that it calculates a value from the data packet. This value is then placed at the end of the packet, and when checking the integrity, the calculation is performed again on the packet plus the CRC value. If the output of the algorithm is zero then the data was transmitted intact; if it is any other value, the data was corrupted in transit. The CRC algorithm works by using polynomial division. This division is performed on the binary data, with the data stream being the dividend and a prechosen polynomial interpreted as a binary value as the divisor. The division results in a remainder value, and this is what is appended to the end of the packet to perform the CRC post-transit. An example of this computation is shown in Figure 9.

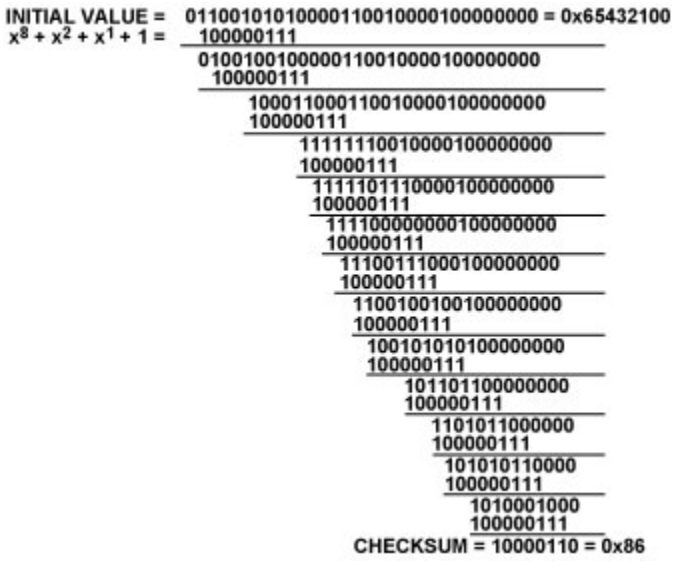

Figure 9. CRC Division Calculation [16]

3.3 Communication Packet Format

The following tables describe the format of the data packet for the controller feedback and control request messages. All data types are assumed signed unless otherwise stated

Table 1. Feedback Message Format

Name	Type	Size
Start of Frame	Byte	1
Status Flags	Byte	1
Steering Column Current Position	Float	4
Steering Motor Set RPM	Integer	2
Steering Motor Current RPM	Integer	2
Rear Track Width	Float	4
Rear Track Angle	Float	4
Rear Track Motor Set RPM	Integer	2
Rear Track Motor Current RPM	Integer	2
Vehicle Speed	Float	4
Vehicle Roll	Float	4
Accelerometer X	Float	4
Accelerometer Y	Float	4
Accelerometer Z	Float	4
Gyroscope X	Float	4
Gyroscope Y	Float	4
Gyroscope Z	Float	4
CRC	Unsigned Integer	2
	Total	56

Table 2. Control Message Format

Name	Type	Size
Start of Frame	Byte	1
Steering Set Angle	Float	4
Reartrack Set Angle	Float	4
Vehicle Set Speed	Float	4
CRC	Unsigned Integer	2
	Total	15

3.4 Communication Failsafe

The controller requires a failsafe system if communications become corrupted or stop altogether. These two key points are covered by separate checks in the failsafe system. First, if the controller has not received any new messages for 500 milliseconds it will trigger the failsafe, or if it receives 5 consecutive corrupted packets it will trigger the failsafe. Triggering on a set of consecutive corrupted packets will usually indicate there is something wrong with the communication link between the controller and computer. When the failsafe is triggered, the rear track is expanded to its widest width to ensure maximum stability and the vehicle is driven to a stop.

Chapter 4: Controls and Signals

4.1 Inertial Measurement Unit – Accelerometer and Gyroscope Fusion

In order to minimize the drawbacks of using either an accelerometer or gyroscope to measure vehicle roll the data from each sensor must be fused together such that the short-term accuracy of the gyroscope is combined with the long-term frequency of the accelerometer. These can be fused with a multitude of different types of filtering algorithms. As discussed in chapter 1, both the complementary filter and the Kalman filter are good candidates. It was determined that the microcontroller used had enough processing power to integrate a Kalman filter to obtain more accurate readings.

4.1.1 Kalman Filter Equations

In order to fuse the data, each sensor's output must first be transformed into an angle that can be processed by the filter. Degrees is used over radians for better readability.

For the accelerometer, the following transformation is used to calculate the roll of the vehicle.

$$\theta = atan2(a_y, a_z)\frac{180}{\pi} \quad (1)$$

Where a_y and a_z are the accelerations measured across the y and z axes, respectively, in $\frac{m}{s^2}$.

Using the 2-argument arctangent function gives the angle in the range of -180 to 180 degrees, where 0 is where the vehicle is perfectly upright.

For the gyroscope, the following transformation is used to calculate the roll of the vehicle.

$$\theta = \sum \dot{\theta}_x \Delta t \frac{180}{\pi} \tag{2}$$

Where $\dot{\theta}_x$ is the change in rotation about the x-axis in $\frac{degrees}{s}$ and Δt is the change in time since the last measurement.

Using the Kalman filter system derived by K. Lauszus [14], a controller can fuse the gyroscope and accelerometer with the following set of equations [12][13].

The state of the system at time *k*:

$$\boldsymbol{x}_k = \boldsymbol{F}\boldsymbol{x}_{k-1} + \boldsymbol{B}\boldsymbol{u}_k + \boldsymbol{w}_k \tag{3}$$

The state matrix:

$$\boldsymbol{x_k} = \begin{bmatrix} \theta \\ \dot{\theta}_b \end{bmatrix}_k \tag{4}$$

Where θ is the roll angle and $\dot{\theta}_b$ is the bias, which is the amount the gyroscope has drifted.

The state transition model:

$$\boldsymbol{F} = \begin{bmatrix} 1 & -\Delta t \\ 0 & 1 \end{bmatrix} \tag{5}$$

The control-input model:

$$\boldsymbol{B} = \begin{bmatrix} \Delta t \\ 0 \end{bmatrix} \tag{6}$$

The process noise covariance matrix:

$$\boldsymbol{Q}_k = \begin{bmatrix} Q_\theta & 0 \\ 0 & Q_{\dot{\theta}_b} \end{bmatrix} \tag{7}$$

Using these formulas, along with the other formulas normally used in the Kalman filter, but not needing to be repeated by the author, the filter can be fully constructed. First, we must predict a state estimate.

The *a priori* state estimate:

$$\widehat{\boldsymbol{x}}_{k|k-1} = \boldsymbol{F}\widehat{\boldsymbol{x}}_{k-1|k-1} + \boldsymbol{B}\boldsymbol{u}_k + \boldsymbol{w}_k \tag{8}$$

$$\begin{bmatrix} \theta \\ \dot{\theta}_b \end{bmatrix}_{k|k-1} = \begin{bmatrix} 1 & -\Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \theta \\ \dot{\theta}_b \end{bmatrix}_{k-1|k-1} + \begin{bmatrix} \Delta t \\ 0 \end{bmatrix} \dot{\theta}_k \tag{9}$$

$$= \begin{bmatrix} \theta - \dot{\theta}_b \Delta t + \dot{\theta} \Delta t \\ \dot{\theta}_b \end{bmatrix} \tag{10}$$

The *a priori* estimate covariance:

$$\boldsymbol{P}_{k|k-1} = \boldsymbol{F}\boldsymbol{P}_{k-1|k-1}\boldsymbol{F}^t + \boldsymbol{Q}_k \tag{11}$$

$$\begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix} = \begin{bmatrix} 1 & -\Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k-1|k-1} \begin{bmatrix} 1 & 0 \\ -\Delta t & 1 \end{bmatrix} + \begin{bmatrix} Q_\theta & 0 \\ 0 & Q_{\dot{\theta}_b} \end{bmatrix} \Delta t \tag{12}$$

$$= \begin{bmatrix} P_{00} + \Delta t(\Delta t P_{11} - P_{01} - P_{10} - Q_\theta & P_{01} - \Delta t P_{11} \\ P_{10} - \Delta t P_{11} & P_{11} + Q_{\dot{\theta}_b} \Delta t \end{bmatrix} \tag{13}$$

Where Q_θ and $Q_{\dot{\theta}_b}$ are the covariances of the process noise for the accelerometer and gyroscope drift, respectively.

The next step is to update the Kalman filter with predictions.

The observation:

$$\boldsymbol{z}_k = \boldsymbol{H}_k \boldsymbol{x}_k + \boldsymbol{v}_k \tag{14}$$

$$\boldsymbol{z}_k = \theta_k \tag{15}$$

In this case, $\boldsymbol{z}_k$ simply reduces to the current measured angle from the accelerometer.

The measurement pre-fit residual:

$$\tilde{\boldsymbol{y}}_k = \boldsymbol{z}_k - \boldsymbol{H}_{\hat{x}_{k|k-1}} \tag{16}$$

$$= \theta_k - [1\ 0] \begin{bmatrix} \theta \\ \dot{\theta}_b \end{bmatrix}_{k|k-1} \tag{17}$$

$$= \theta_k - \theta_{k|k-1} \tag{18}$$

The pre-fit residual covariance:

$$\boldsymbol{S}_k = \boldsymbol{H}\boldsymbol{P}_{k|k-1}\boldsymbol{H}^t + \boldsymbol{R} \tag{19}$$

$$= [1\ 0] \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} + \boldsymbol{R} \tag{20}$$

$$= P_{00\,k|k-1} + \boldsymbol{R} \tag{21}$$

Where $\boldsymbol{R}$ is a tunable constant that represents the covariance of the observation noise.

The optimal Kalman gain:

$$\boldsymbol{K}_k = \boldsymbol{P}_{k|k-1}\boldsymbol{H}^t\boldsymbol{S}_k^{-1} \tag{22}$$

$$\begin{bmatrix} K_0 \\ K_1 \end{bmatrix}_k = \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k|k-1} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \boldsymbol{S}_k^{-1} \tag{23}$$

$$= \frac{\begin{bmatrix} P_{00} \\ P_{10} \end{bmatrix}_{k|k-1}}{\boldsymbol{S}_k} \tag{24}$$

Updated *a posteriori* state covariance:

$$\hat{\boldsymbol{x}}_{k|k} = \hat{\boldsymbol{x}}_{k|k-1} + \boldsymbol{K}_k \tilde{\boldsymbol{y}}_k \tag{25}$$

$$\begin{bmatrix} \theta \\ \dot{\theta}_b \end{bmatrix}_{k|k} = \begin{bmatrix} \theta \\ \dot{\theta}_b \end{bmatrix}_{k|k-1} + \begin{bmatrix} K_0 \\ K_1 \end{bmatrix}_k \tilde{\boldsymbol{y}}_k \tag{26}$$

$$= \begin{bmatrix} \theta \\ \dot{\theta}_b \end{bmatrix}_{k|k-1} + \begin{bmatrix} K_0 & \tilde{\boldsymbol{y}} \\ K_1 & \tilde{\boldsymbol{y}} \end{bmatrix}_k \tag{27}$$

Updated *a posteriori* estimate covariance:

$$\boldsymbol{P}_{k|k} = (\boldsymbol{I} - \boldsymbol{K}_k \boldsymbol{H}) \boldsymbol{P}_{k|k-1} \tag{28}$$

$$\begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k|k} = \left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} K_0 \\ K_1 \end{bmatrix}_k \begin{bmatrix} 1 & 0 \end{bmatrix} \right) \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k|k-1} \tag{29}$$

$$= \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k|k-1} - \begin{bmatrix} K_0 P_{00} & K_0 P_{01} \\ K_1 P_{10} & K_1 P_{11} \end{bmatrix} \tag{30}$$

4.1.2 Kalman Filter Results and Performance

To test the filter's effectiveness and accuracy, the IMU is first placed flat on a level table to ensure it can accurately report the roll of the IMU. The IMU is then rotated 90 degrees to observe how quickly the filter reacts to the change in angle and how fast it converges to a stable angle in this new orientation. It is also quickly rotated back and forth to observe how it reacts to quick fluctuations in the angle of the vehicle. A final test is performed in which the IMU is again flat on the table and it is rapidly tapped to observe how vibrations, which will be experienced on a vehicle, affect the accuracy of the filter output. Figure 10 shows the output of the three

accelerometer axes, the three gyroscope axes, and the filter output angle. The filter works effectively as it is able to quickly update the angle of the IMU with very little noise. It converges very rapidly to the new angle, taking only about 100 milliseconds to converge within a small error of a few percent off the true angle. It also responds very well to vibrations that may be experienced by the IMU when mounted in a vehicle, with the filter angle diverging very minimally.

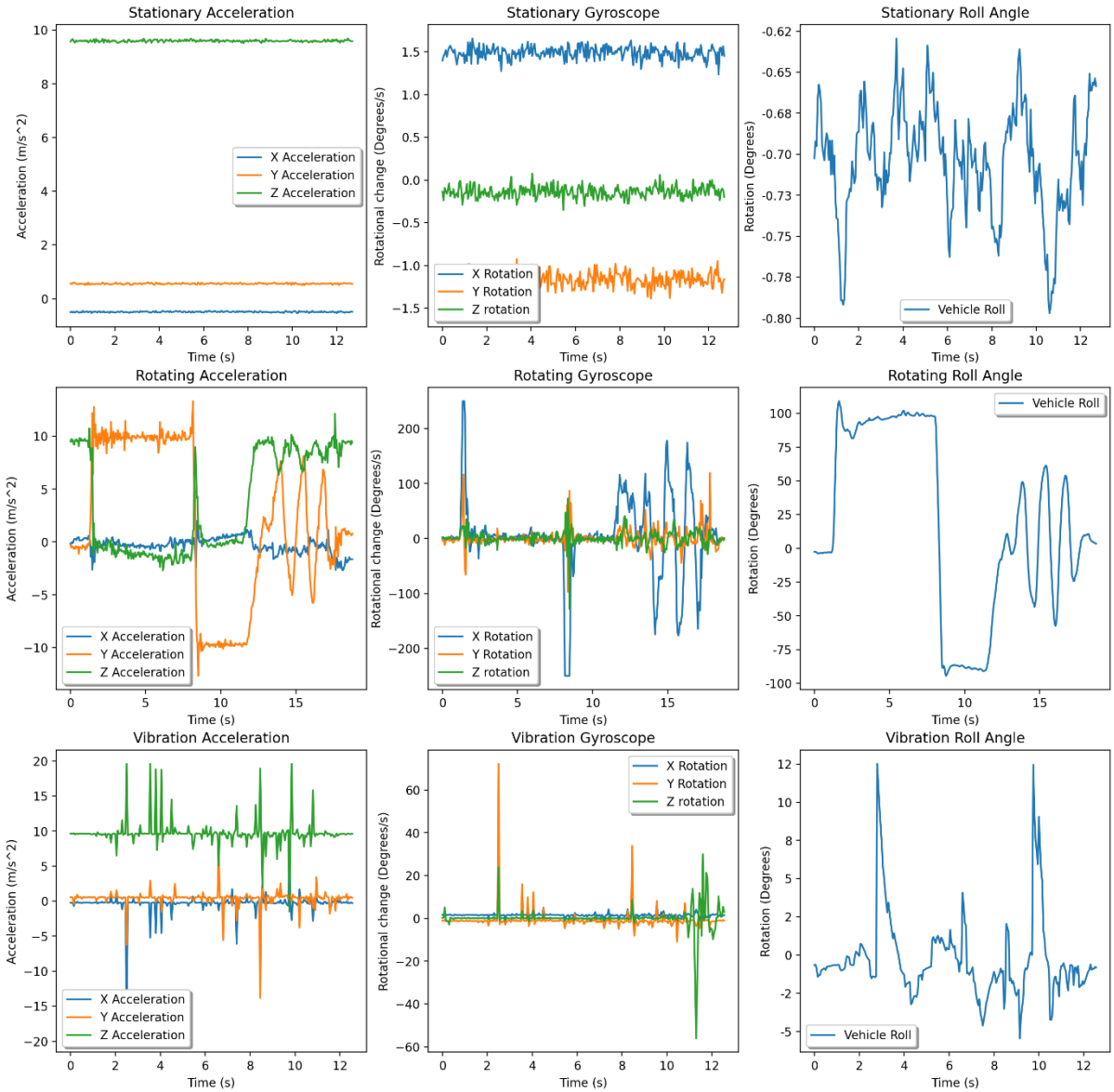

Figure 10. Accelerometer and Gyroscope Raw Measurements and Kalman Filter Output

4.2 Motor-Based Position Controllers

There are two position controllers that needed to be implemented for the system. One is used to control the steering column; one is used to control the rear track width. These controllers use a brushless DC (BLDC) motor as the rotor due to their high torque capability and minimal skipping when compared to other drive motors. These motors use three phases to drive and use a set of inverters to convert the DC power into AC signals that drive the motor poles.

In order to understand the theory and implementation of the controllers a simulation was run using Simulink. The hardware modules used to simulate the controller are shown in Figure 11. This simulation uses a three-phase cascaded controller design. The position, speed, and current controllers all use proportional integral (PI) controllers. The controller this thesis focuses on is the position controller, shown in the box in Figure 12. The speed and current controllers are managed by ESC offboard the main controller.

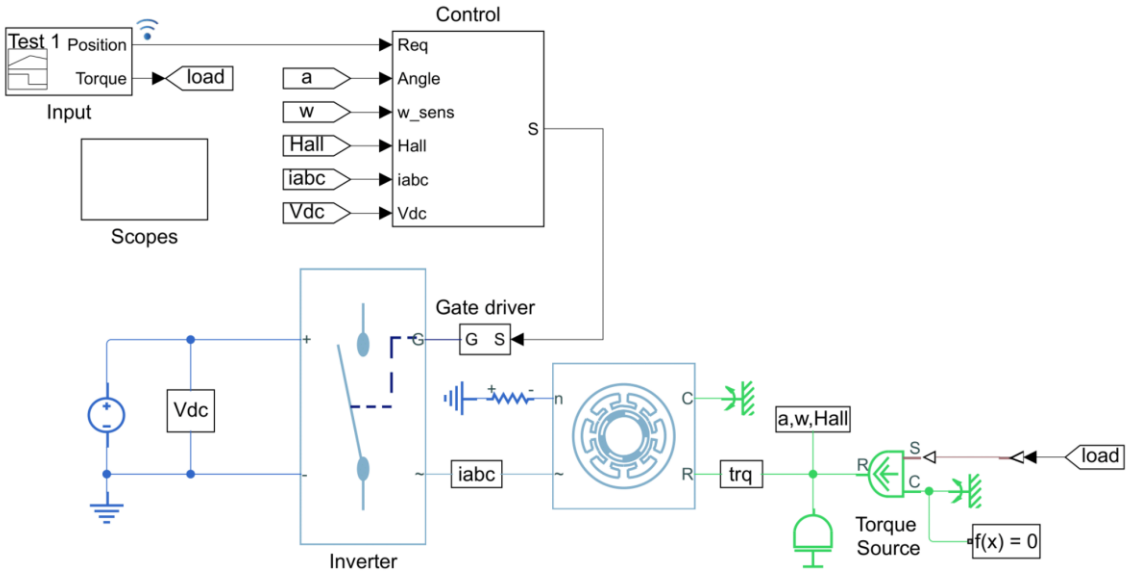

Figure 11. BLDC Motor Model

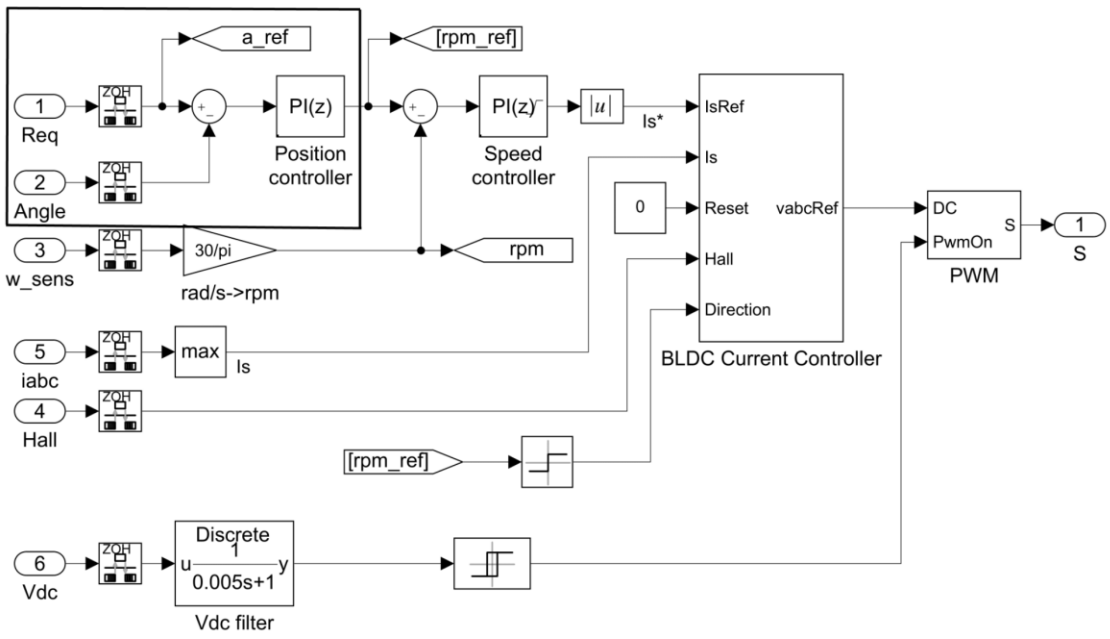

Figure 12. BLDC Motor Controller

As shown in the simulation results in Figure 13, the motor can provide accurate and reactive position setting by using these cascaded control loops to manage each portion of the overall control loop.

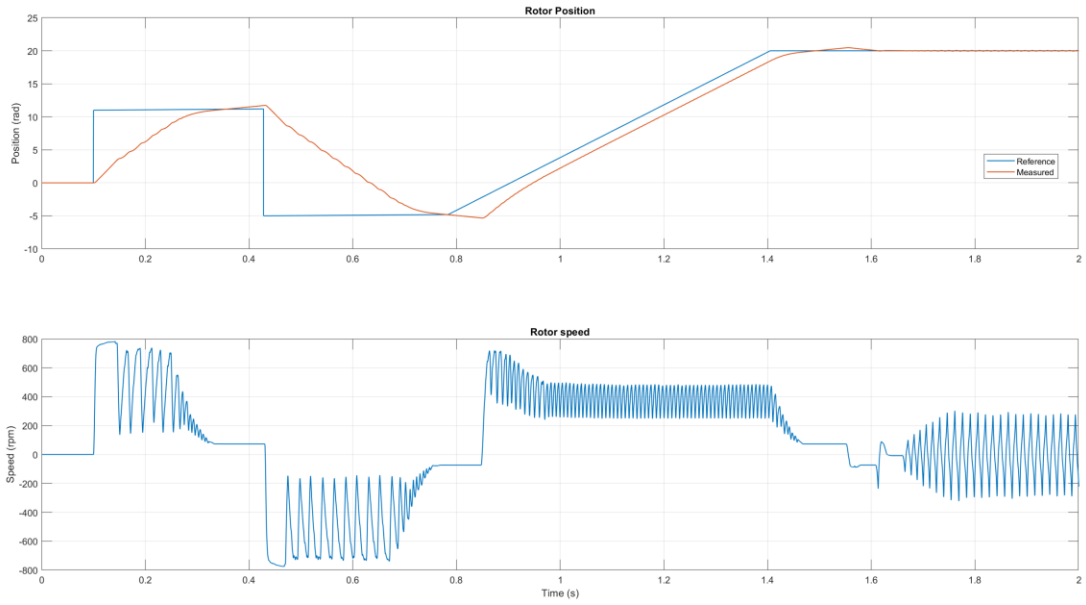

Figure 13. BLDC Motor Controller Simulation Results

4.2.1 Steering Column Controller

The steering column uses a BLDC motor that drives a worm gear through a belt, which drives a geared turntable that acts as the steering column. The worm gear to the turntable has a 50-to-1 gear reduction. The feedback for the controller is a potentiometer that is attached to the turntable. The potentiometer is configured such that its center point is set to the steering column facing straight for the vehicle. The potentiometer is connected to the microcontroller through a 12-bit analogue-to-digital converter (ADC) which turns the analog voltage into a digital value that ranges from 0 to 4096. This value can then be mapped to a range based on the maximum sweep angle of the potentiometer, in this case about 250 degrees. Using a simple mapping function, the ADC value from 0 to 4096 can be mapped to -125 to 125 degrees to give an accurate angle of the steering column to the PI controller. This value can then be easily used to calculate the error for the controller. As the motor spins, the belt will spin the worm gear which will rotate the turntable. Through a ball bearing, the potentiometer is rotated along with the turntable, which provides the feedback mechanism for the control loop. The model for the steering column can be seen in Figure 14.

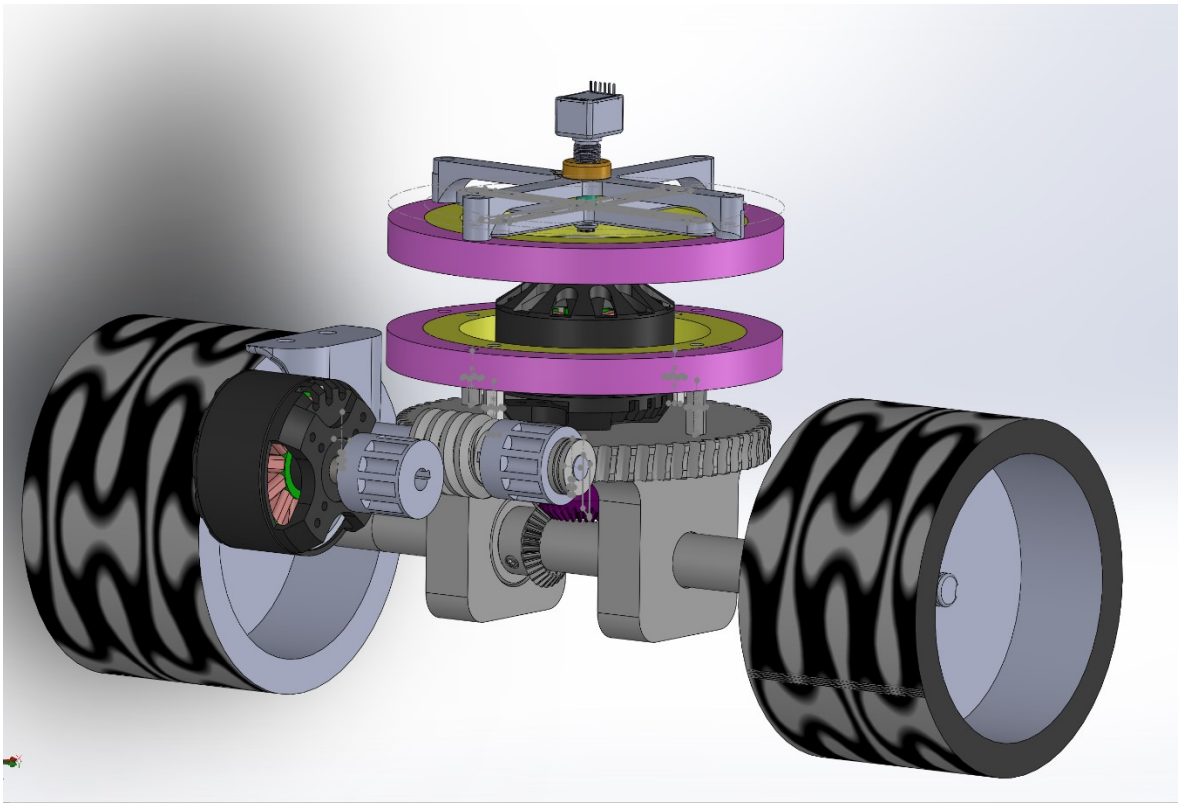

Figure 14. Steering Column CAD Model

The PI controller was empirically tuned until a fast and stable control loop was achieved that quickly arrives to the set point and maintains an accurate position relative to the requested set point. After tuning the controller, the constants that were arrived at are $[K_p \quad K_i] = [55.0 \quad 20.4]$. The controller shown in Figure 12 runs in the continuous time domain. In order to use it in the microcontroller, it must be converted into the sampled data domain. This is achieved through the following formula.

$$U_t = U_{t-1} + \left(K_p + K_i \Delta t\right) E_t - K_p E_{t-1} \tag{31}$$

Where U_t is the controller output as motor RPM and E_t is the error in degrees, which is calculated by:

$$E_t = \theta_{set} - \theta_{pot} \tag{32}$$

The controller is tested by quickly setting the set point to a steep angle (0 to 90 degrees) and observing how fast it achieves this set point. It is also tested by setting the set point along a linear slope to see how accurately it follows a constantly changing set point. The graphs in Figure 15 show both the set position and the current column position as well as the set RPM from the controller and the ESC's current RPM. These graphs show relatively accurate control with minor overshoot that is quickly corrected by the integral portion of the PI controller.

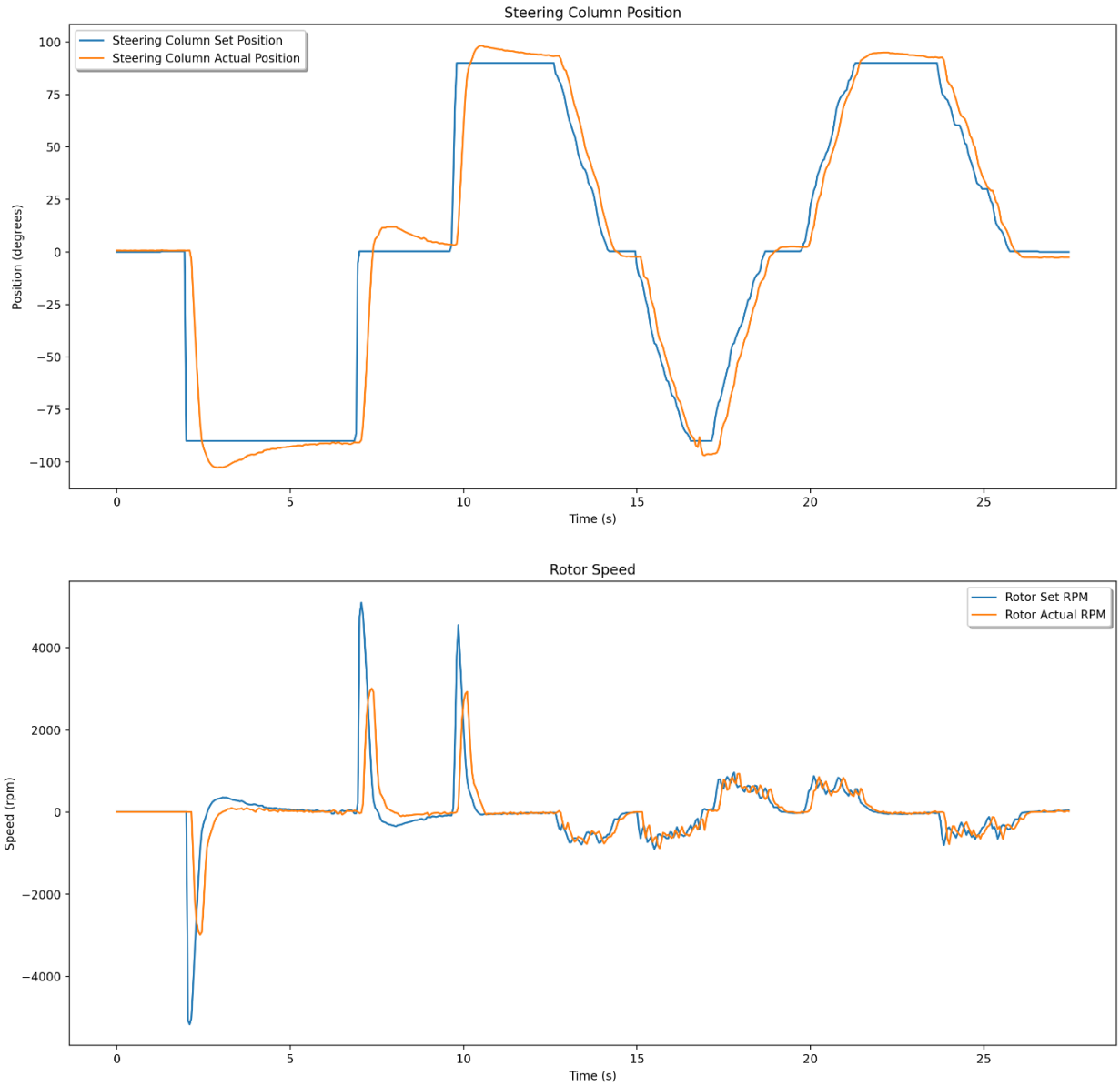

Figure 15. Steering Column PI Controller Experimental Data

4.2.2 Rear Track Controller

The rear track controller uses a nearly identical control scheme to that of the steering column. The key difference is that it uses a linear potentiometer to measure the rear track width (or angle). As the rear track expands, the linear potentiometer expands linearly with the angle of the rear track. This allows the resistance value of the linear potentiometer to drive an electrical signal to the microcontroller that can be read by the ADC similarly to the steering column potentiometer. The key difference is that the linear potentiometer may not fully expand or contract with the rear track but the full range of the ADC is still desired. This can be solved by measuring the resistance when the track is fully expanded and closed and using those values to create a Wheatstone bridge amplifier. An example Wheatstone bridge is shown in Figure 16, where R2 would be the linear potentiometer, and R1, R3, R4, Rx, and Rg are standard resistors whose value would be calculated based on the resistance range of the linear regulator when in the rear track controller. *V* of the Wheatstone bridge in the figure would be 3.3V, while *OUT* would range from 0V to 3.3V linearly as the potentiometer extends as the rear track expands. This amplifier can take a set of different resistances and amplify it to the full range of the ADC, allowing the maximum resolution to be used and gaining more precise control in the PI loop.

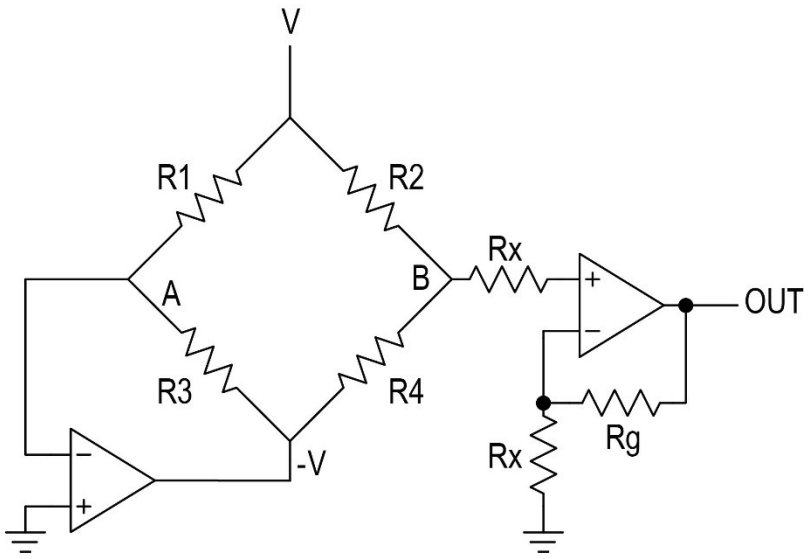

Figure 16. Wheatstone Bridge Amplifier

The BLDC motor drives a linear actuator that is responsible for expanding the rear track. This is visualized in the model shown in Figure 17. As these actuators extend, the rear track width and angle of the rear body mechanism would expand proportionally. This motion will likewise extend the linear potentiometer, increasing the output resistance, providing an analog feedback signal that can interpreted by the microcontroller ADC and be used to drive a control loop.

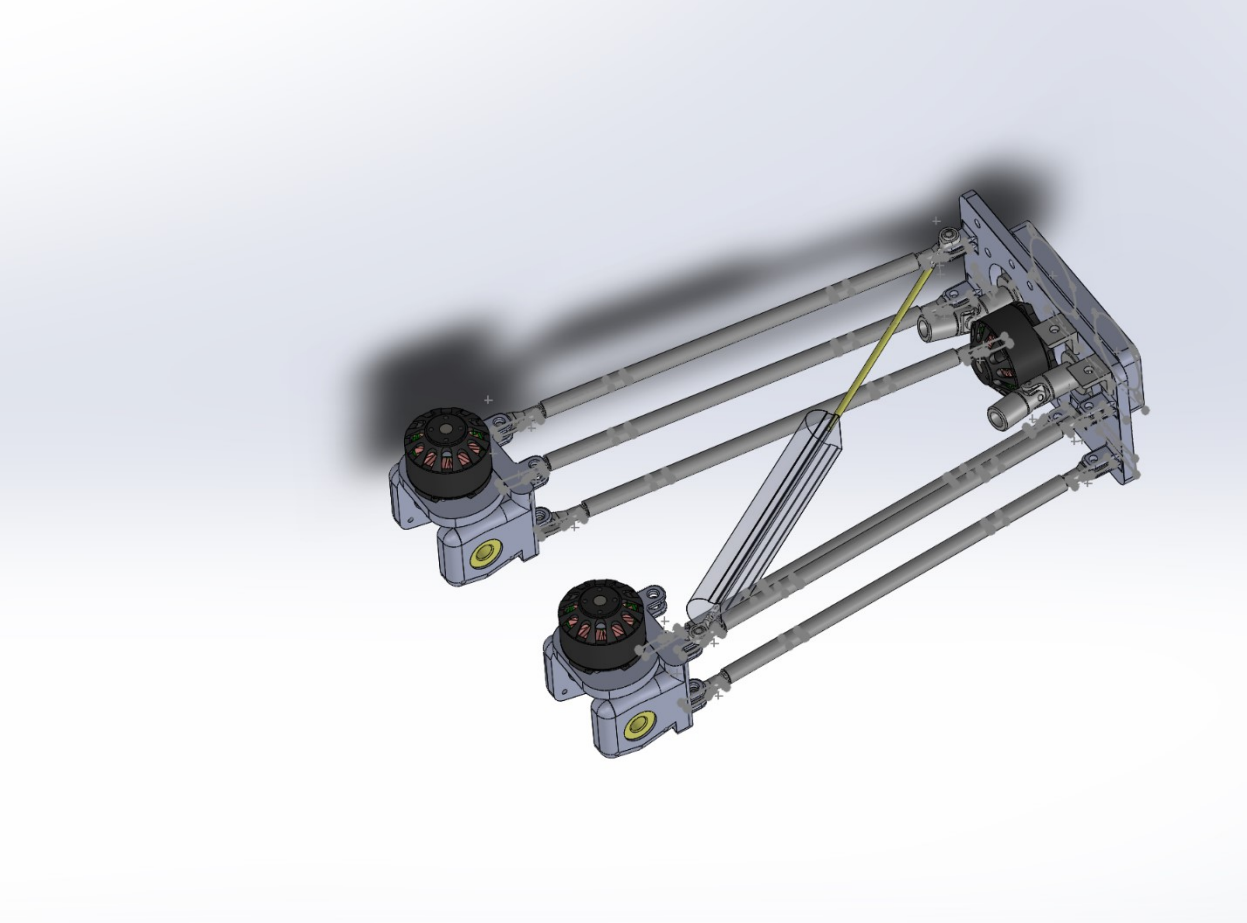

Figure 17. Rear Track Control and Feedback Mechanisms

Using a similar control loop to that of the steering column, the rear track PI controller is also empirically tuned to achieve fast and accurate set points. The controller is similarly tested, and the actuators checked for backlash and strain that may affect the control loop from behaving properly [11].

4.3 Real-Time Controls System Balancing Simulation Test

In order to test the controller's ability to respond to quick changes to the vehicle balance and ensure the communication protocol does not hinder the controller in any way, a quick self-balance mockup test was created. It uses the control computer to listen to the feedback messages from the controller, specifically the vehicle's roll angle, and then calculates a PID loop value to

send back a steering angle to the controller. The body of the vehicle was gently rocked back and forth to simulate an unstable vehicle and the steering mechanism was observed to see how quickly it would respond to the change in angle. Data was also collected and analyzed to see if it would react quickly to the sudden changes and if the controller could respond fast enough. The roll angle and steering angle values collected are plotted in Figure 18, where the difference between the control computer receiving the current vehicle roll and the controller starting to actuate the steering motor was about 15 milliseconds.

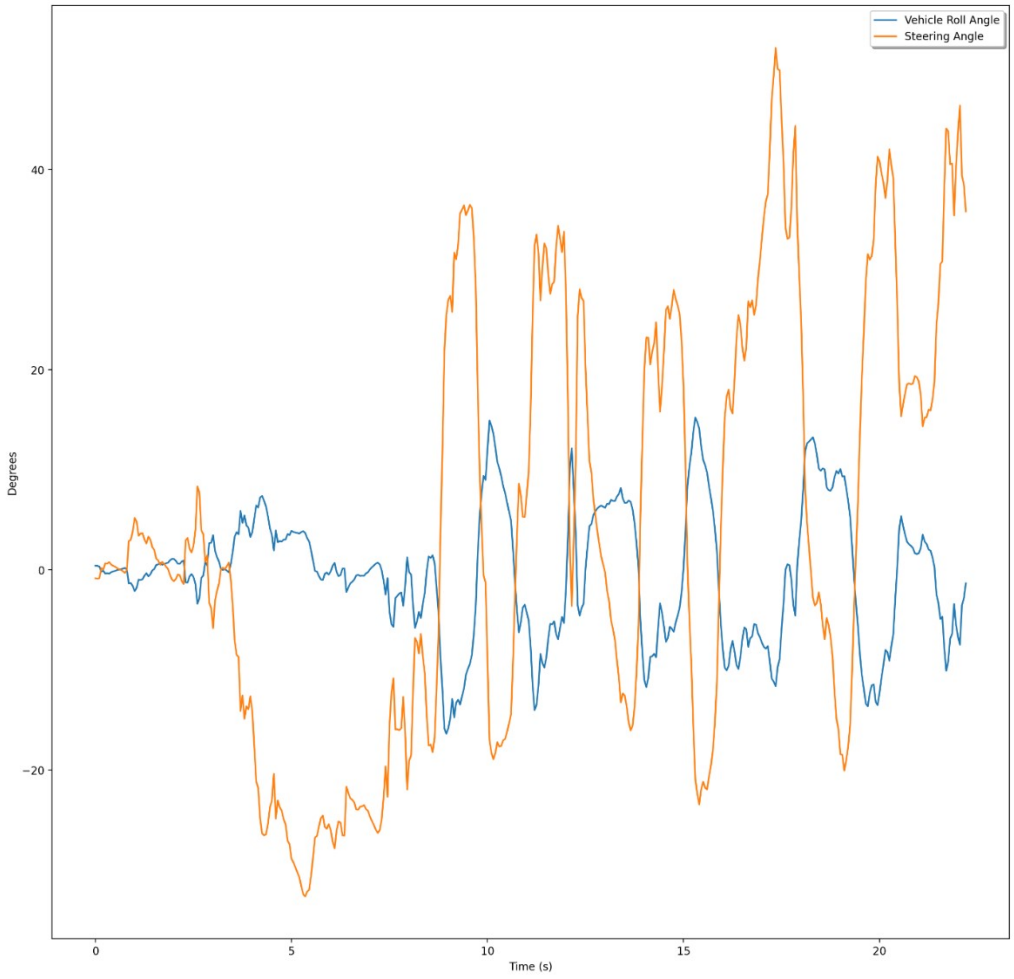

Figure 18. Steering Based Balance Output

Chapter 5: Future Work and Conclusion

5.1 Future Work

This project sets out and implements a controller for a unique vehicle with many sensors and actuators. What it does not do is provide fully autonomous operation of the vehicle, but rather gives real time sensor information and control operation that may not be possible without a deterministic controller. The next steps would be to implement a fully autonomous controller with a more powerful computer that interfaces with the vehicle controller to actuate the system itself. The narrow balancing mechanism that this vehicle is designed to achieve can be implemented using a neural network that will perform the balancing by reading the IMU data from the controller and then sending a rear track actuation command to expand the track if the vehicle begins to tip too far. This would be best achieved using a reinforcement learning algorithm to teach a neural network how to achieve balance by rewarding the algorithm when it gains balance by actuating the rear track. This would require a simulation and would not have been achievable in the time available to complete this thesis. Another option is to use supervised learning, where the vehicle is driven by a human and when the vehicle starts to tip the human will manually expand the rear track. This data is logged and can be used in a training algorithm to teach a neural network how to map this non-linear function. The best type of network to achieve this would be a recurrent neural network. This is because the recurrent neural network uses long short-term memory (LSTM). This is where the hidden layers of the neural network

feed back on themselves and allow the network to maintain a sort of memory between activations. This can allow the neural network to map the temporal effects that are involved in a tipping vehicle as it will not linearly fall over, but due to gravity will accelerate as it tips more, and as such the network can act as a sort of gain scheduler for a control loop and achieve good balance. While the network shown in Figure 19 is not a true representation of what the neural network for a self-balancing vehicle would look like, it acts as a good reference for how to design it. The inputs for such a network would likely be the current roll of the vehicle, potentially the raw data of the accelerometer as deemed necessary, and the current steering angle of the vehicle. The hidden layers would be configured such as to map these values to a hidden function, along with the LSTM being used to allow the network to maintain a temporal memory of what the network had previously done. The output of the network would be the set position for the rear track width, or angle, and an acceleration for how quickly the width needs to be achieved.

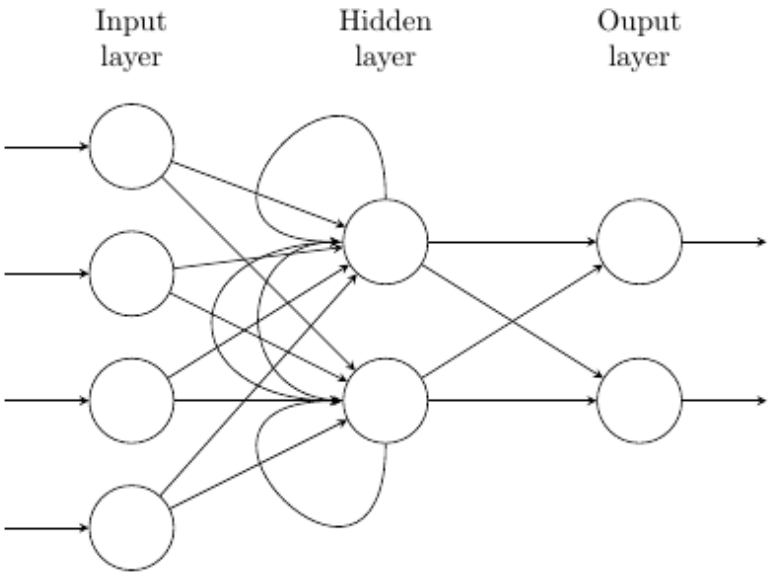

Figure 19. Recurrent Neural Network Model

5.2 Conclusion

This thesis set out to design and implement a real-time vehicle controller using a simple ARM Cortex-M4-based microcontroller. It provides a set of sensor feedback data recorded in real time as well as a set of actuators controlled in real time. Due to the deterministic nature of these control schemes, they must be performed on a computer that can manage tight timings, whereas many personal computers are not able to achieve this due to the lack of determinism in standard operating systems. Using an RTOS allowed the controller to manage many different tasks without relying on different triggers or interrupts to manage when to process the sensors and actuators. The RTOS kernel is used to manage these timings for the controller itself. Overall, this system laid out and achieved a highly performant and reliable real-time controller with rapid feedback and control mechanisms, which can be augmented with autonomous systems that are unable to achieve the deterministic control sequences required to operate vehicle actuators and sensors.

References

[1] S. Vatanashevanopakorn and M. Parnichkun, “Steering control based balancing of a bicycle robot,” *2011 IEEE International Conference on Robotics and Biomimetics*, Karon Beach, Phuket, 2011, pp. 2169-2174.

[2] R. Choudhary and H. K. Gianey, “Comprehensive Review On Supervised Machine Learning Algorithms,” *2017 International Conference on Machine Learning and Data Science (MLDS)*, Noida, 2017, pp. 37-43.

[3] W. Ding, M. Xu, Y. Ma and G. Shi, “Tricycle Attitude Estimation and Turn Control Based on MEMS Sensing Technology,” *2018 IEEE 1st International Conference on Micro/Nano Sensors for AI, Healthcare, and Robotics (NSENS)*, Shenzhen, China, 2018, pp. 30-34.

[4] P. Gui, L. Tang and S. Mukhopadhyay, “MEMS based IMU for tilting measurement: Comparison of complementary and kalman filter based data fusion,” *2015 IEEE 10th Conference on Industrial Electronics and Applications (ICIEA)*, Auckland, 2015, pp. 2004-2009.

[5] M. Nowicki, J. Wietrzykowski and P. Skrzypczyński, “Simplicity or flexibility? Complementary Filter vs. EKF for orientation estimation on mobile devices,” *2015 IEEE 2nd International Conference on Cybernetics (CYBCONF)*, Gdynia, 2015, pp. 166-171.

[6] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Xhang, J. Zhao, “End to End Learning for Self-Driving Cars”, *arXiv: 1604.07316*, 2016.

[7] Chi, Lu and Yadong Mu. “Deep Steering: Learning End-to-End Driving Model from Spatial and Temporal Visual Cues,” *arXiv*: 1708.03798, 2017.

[8] Eraqi, H.M., Moustafa, M.N., Honer, J. “End-to-End Deep Learning for Steering Autonomous Vehicles Considering Temporal Dependencies,” *arXiv*: 1710.03804, 2017.

[9] Kuo, C.-C. Jay. “Understanding convolutional neural networks with a mathematical model.” *J. Vis. Commun. Image Represent.* 41, 2016, pp. 406-413.

[10] S. Sheng-Ju, “Implementation of Cyclic Redundancy Check in Data Communication,” *2015 International Conference on Computational Intelligence and Communication Networks (CICN)*, Jabalpur, 2015, pp. 529-531.

[11] T. Wescott, “Controlling Motors in the Presence of Friction and Backlash.” (2014).

[12] H. Ferdinando, H. Khoswanto and D. Purwanto, “Embedded Kalman Filter for Inertial Measurement Unit (IMU) on the ATMega8535,” *2012 International Symposium on Innovations in Intelligent Systems and Applications*, Trabzon, 2012, pp. 1-5.

[13] G. Welch and G. Bishop, “An Introduction to The Kalman Filter,” technical report, Dept. of Computer Science, Univ. of North Carolina at Chapel Hill, 2002.

[14] K. Lauszus, “A Practical Approach to Kalman Filter and How to Implement It,” Web, TKJ Electronics, 2012.

[15] S. Papanastasiou, M. Ould-Khaoua and V. Charissis, “The Effect of the RTS/CTS Handshake on TCP,” *21st International Conference on Advanced Information Networking and Applications Workshops (AINAW'07)*, Niagara Falls, Ont., 2007, pp. 940-946.

[16] K. Kavanagh, “Cyclic Redundancy Checking Ensures Correct Data Communications,” technical report, Analog Dialogue, 2011.