Hornet Motor Controller

HORNET

Hybrid Brushed/Brushless Motor Controller

Sahil Kale
Aaditya Chaudhary

Rev 1.3
STM32F446RET6

Section 1 Overview

1.1 General Summary

The HORNET MOTOR CONTROLLER is a 5A/24V hybrid brushed and brushless motor controller designed for robotic applications based on an ARM Cortex STM32 processor

1.2 Feature Overview

- Servo-style PWM control interface
- Quadrature Encoder Support
- RGB Status LED
- PIDF feedback control
- Motion Profiling
- CAN, UART, and USB data interfaces
- Brake/Coast mode

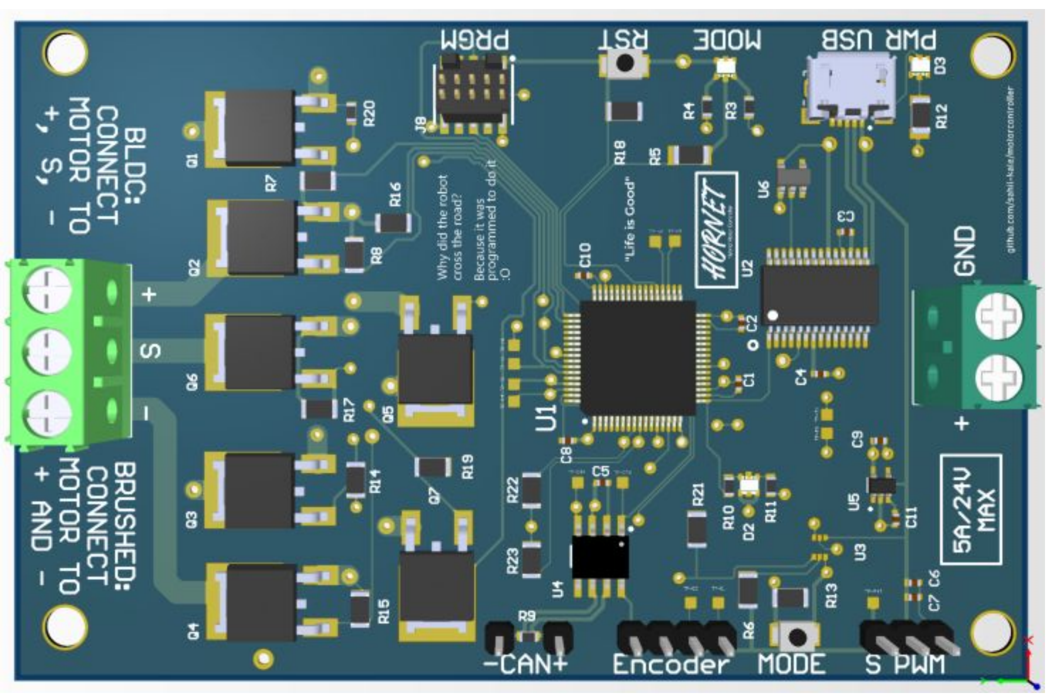


Figure 1.1 Board View

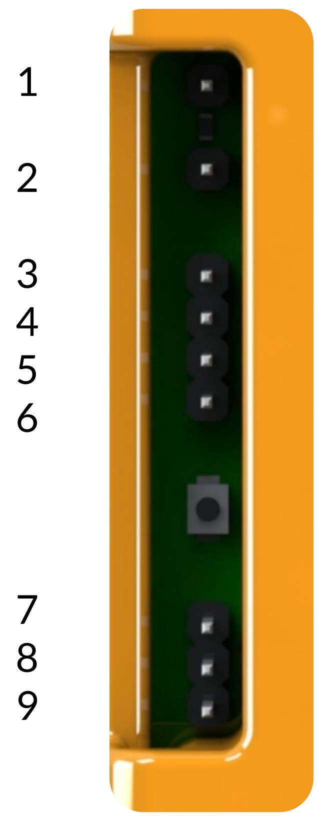
1.3 Pin Mappings

The HORNET MOTOR CONTROLLER features several communication methods . The wiring for these interfaces are located on the left side of the casing. The inputs are broken up into pin headers, in order from top to bottom.

Controller Area Network (CAN) Wiring
- 1. CAN DIFFERENTIAL LOW
- 2. CAN DIFFERENTIAL HIGH

Quadrature Encoder Pins
- 3. GND
- 4. 5V OUT*
- 5. ENCODER IN 1
- 6. ENCODER IN 2

Servo-Style PWM
- 7. SIGNAL
- 8. 5V I/O*
- 9. GND



*Note: These pins capable of powering an external device at 5V/150 mA within the specified operating limits.

2.1 Motor Control Specifications

Parameter	Min	Typ	Max	Units
Supply Voltage	5.5	-	24	V
Max Current	-	-	5	A
Output Frequency	-	2	-	MHz

Table 2.1 General Electrical Specifications

Parameter	Min	Typ	Max	Units
Default Full-Forward Pulse	-	2000	-	us
Default Neutral Pulse	1450	1500	1550	us
Default Full-Reverse Pulse	-	1000	-	us
Input Frequency	25	50	100	Hz
Input Voltage	3.8	5	5.5	V

Table 2.2 PWM input specifications

2.2 Brake/Coast Mode

The HORNET MOTOR CONTROLLER features the ability to utilize the connected motor’s counter-electromotive force in order to lock the motor in place. The Brake/Coast mode is toggled in firmware with CAN or UART signals (see Section 3.3.2).

3.1 LED Status Codes

The HORNET MOTOR CONTROLLER features an RGB LED that is used to communicate information about the current state of the controller.

Name	Interval	Interval	Interval	Interval
Proportional Forward				
Proportional Back				
No Signal, Coast				
Neutral Coast				
Neutral Brake				

Figure 3.1.1 LED Status Codes

3.2 CAN/UART Interface

3.2.1 CAN Interface Specifics

- CAN ID: 0x254
- Speed: 1Mbit/s

Note that the HORNET MOTOR CONTROLLER has an integrated 120 ohm resistor, allowing it to terminate an existing CAN network

3.2.2 UART Interface Specifics

- Baud Rate: 115200

3.3 CAN/UART Command Specification

3.3.1 Command Format

- Commands are made up of a leading value, to specify the command
- All values following the command specifier are parameters fed into the command

Note that the HORNET MOTOR CONTROLLER can accept a max of 7 parameters and that they must be unsigned 8 bit integers

3.3.2 Available Commands

Note that the parameters have been listed

- Set Speed
  - Identifier: 0x00
  - Parameters:
    - Speed, as an unsigned integer from 0-1023, where 0 represents full speed reversed, and 1023 represents full speed forwards

- Set Brushed
  - Identifier: 0x01
  - Parameters:
    - Set Brushed, as an unsigned integer where 0 is false and 1 is true

- Set CAN
  - Identifier: 0x02
  - Parameters:
    - Set whether to use CAN for speed or PWM, as an unsigned 8 bit integer. Send 0 for CAN and 1 for PWM

- Set Coast/Brake
  - Identifier: 0x03
  - Parameters:
    - Unsigned 8 bit integer. Send 0 for coast mode and 1 for brake mode

- Set Acceleration
  - Identifier: 0x04
  - Parameters:
    - Profile number for the trapezoidal profile to modify, starting at 0
    - Acceleration, as a double that must be sent as 2 unsigned 8 bit integers

- Set Velocity
  - Identifier: 0x05
  - Parameters:
    - Profile number for the trapezoidal profile to modify, starting at 0
    - Max Velocity, as a double that must be sent as 2 unsigned 8 bit integers

- Set P coefficient
  - Identifier: 0x06
  - Parameters:
    - PIDF Profile number, starting at 0
    - P coefficient as a double sent as 2 unsigned 8 bit integers

- Set I coefficient
  - Identifier: 0x07
  - Parameters:
    - PIDF Profile number, starting at 0
    - I coefficient as a double sent as 2 unsigned 8 bit integers

- Set D coefficient
  - Identifier: 0x08
  - Parameters:
    - PIDF Profile number, starting at 0
    - D coefficient as a double sent as 2 unsigned 8 bit integers

- Set F coefficient
  - Identifier: 0x09
  - Parameters:
    - PIDF Profile number, starting at 0
    - F coefficient as a double sent as 2 unsigned 8 bit integers

- Start Triangle Motion Profile
    - Identifier: 0x10
    - Parameters:
        - Distance, in encoder ticks,as a 32 bit signed integer sent as 4 unsigned 8 bit integers
        - Time, in seconds, as a double sent as 2 unsigned 8 bit integers

- Start Trapezoidal Motion Profile
    - Identifier: 0x11
    - Parameters:
        - Trapezoidal Profile number, starting at 0 [single unsigned 8 bit integer]
        - Velocity PIDF Profile to use, starting at 0 [single unsigned 8 bit integer]
        - Distance in encoder ticks to travel, as a 32 bit signed integer [4 unsigned 8 bit integers]

- Start PIDF Feedback Control
    - Identifier: 0x12
    - Parameters:
        - Input Type, 0 for encoder position and 1 for encoder velocity [unsigned 8 bit integer]
        - Target of the PIDF loop, as a double [2 unsigned 8 bit integers]

4.1 Triangle Motion Profile Generator

The HORNET MOTOR CONTROLLER can generate triangular motion profiles on the fly, when given both a distance in encoder ticks to travel and the time needed to reach the target. The resulting velocity-time profile does not require a given acceleration because it calculates the acceleration based on the given time and distance. Once the profile is generated, a velocity PIDF config will need be selected to actually follow the profile.

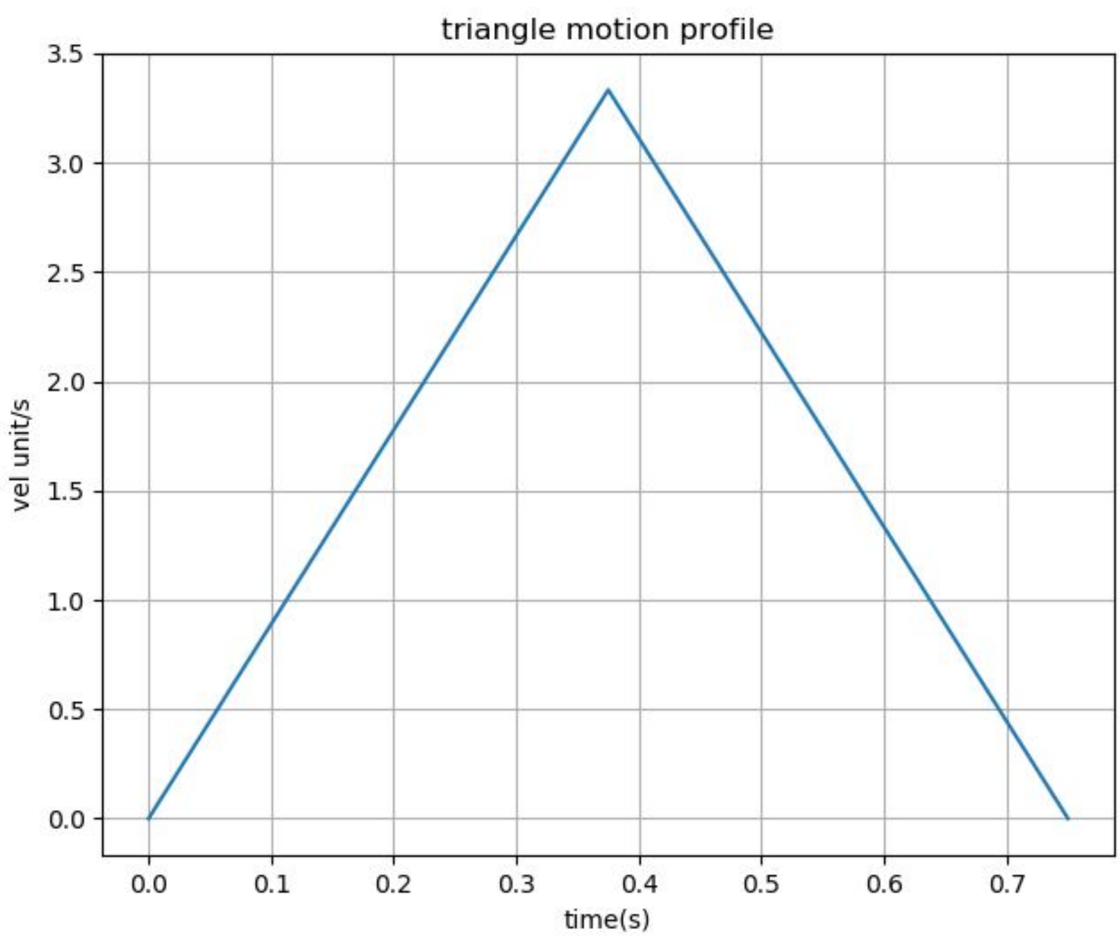


Figure 4.1.1 Triangle Motion Graph

4.2 Trapezoidal Motion Profile Generator

The HORNET MOTOR CONTROLLER can generate trapezoidal motion profiles on the fly, when given a distance in encoder ticks to travel. The controller has 8 available trapezoidal profiles. To use a profile, a max velocity and acceleration must be set for the profile. These values will be used to generate the profile. Once the profile is generated, a PIDF slot will need be selected to follow the profile. If the max velocity is not reached, but half the distance has been traveled, the profile will be a triangular profile rather than a trapezoidal one.

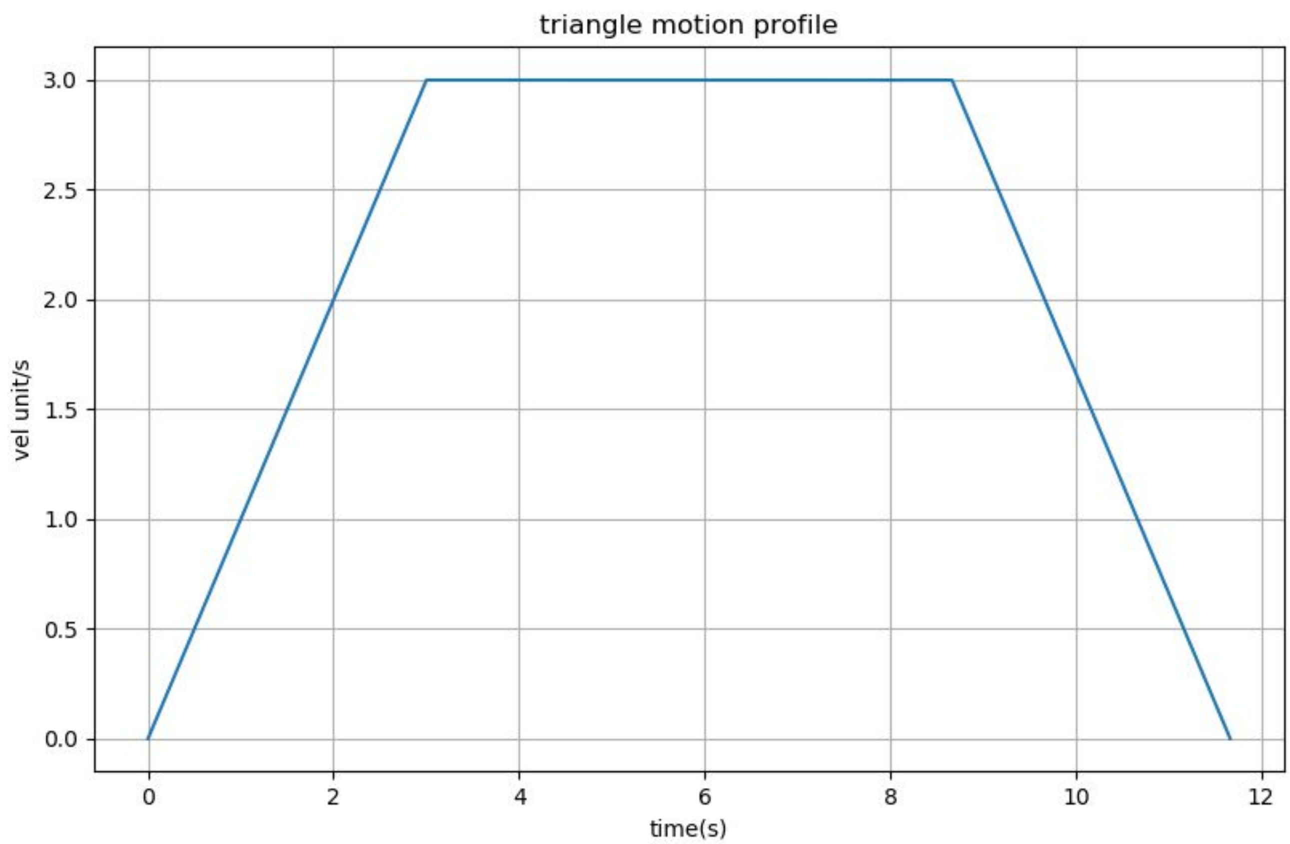


Figure 4.2.1 Trapezoidal Motion profile graph

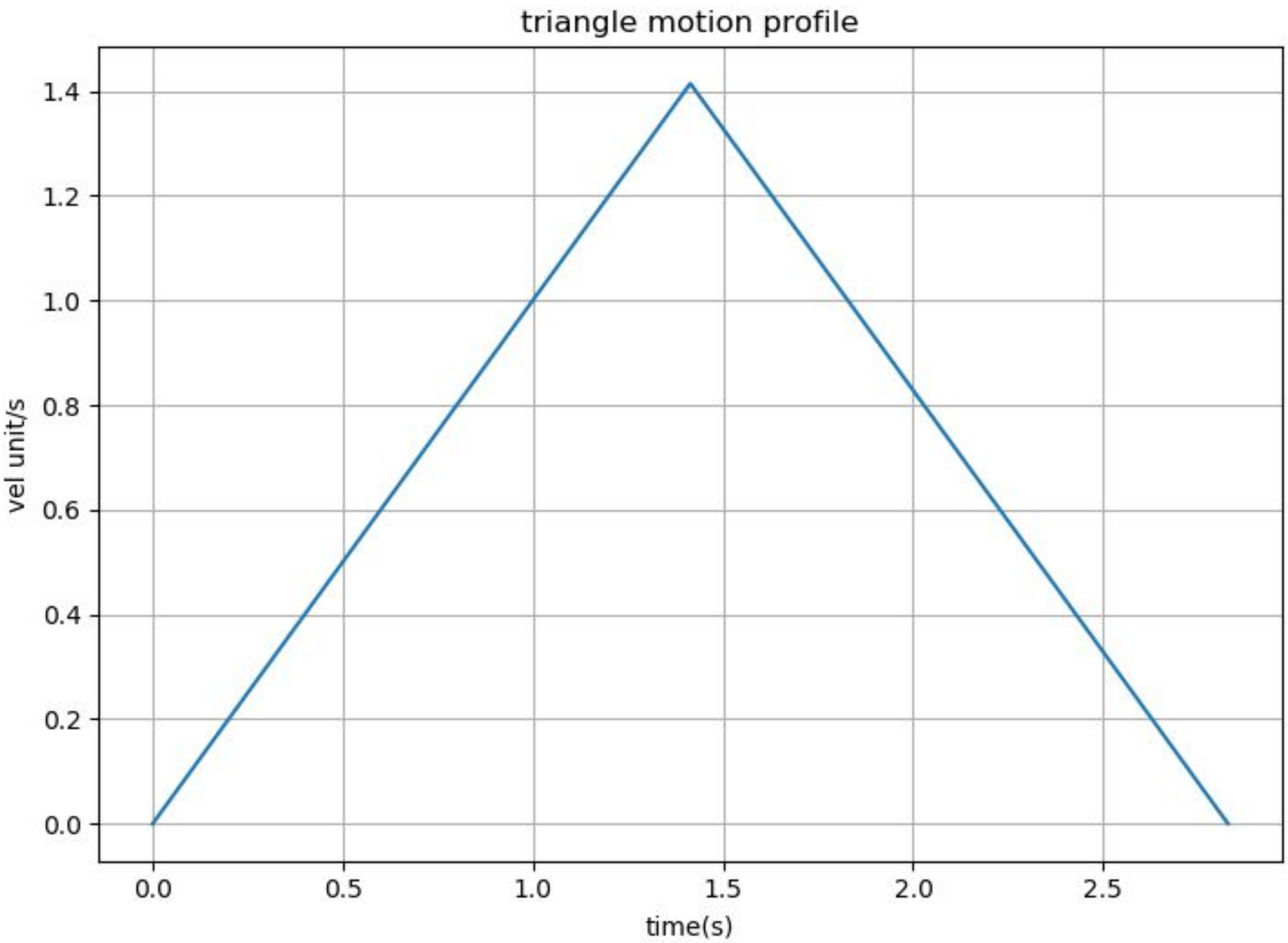


Figure 4.2.2 Trapezoidal motion profile graph in fallback mode

4.3 User Notes

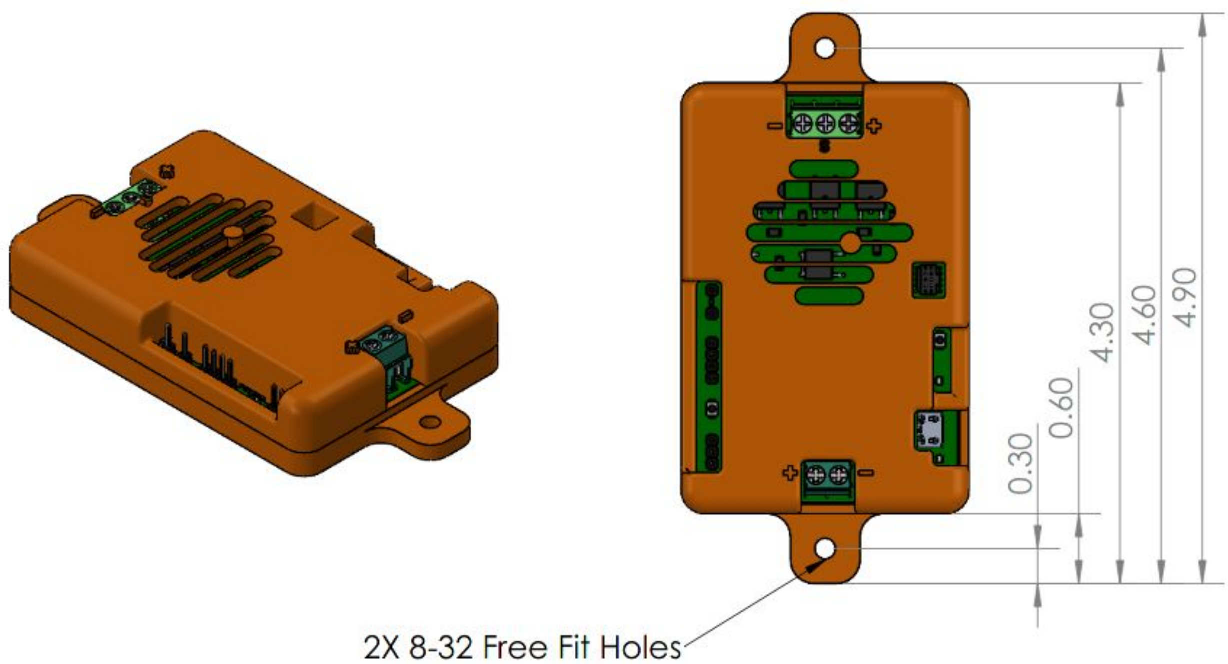
- It is a good idea to test out possible displacement, velocity, and acceleration values with the graphs before using them in practice.
- For most applications, trapezoidal profiles work better and are safer than triangular profiles.
- Trapezoidal profile generator will fallback to a triangle profile if a trapezoidal one can't be made with the given displacement.

A.1 - Schematic

The schematics for the HORNET MOTOR CONTROLLER are available at the following link:

https://github.com/sahil-kale/MotorController/blob/main/HORNET_MOTOR_CONTROLLER/HORNET_MOTOR_CONTROLLER.pdf

A.2 - Mechanical Drawing



The full mechanical drawing for the HORNET MOTOR CONTROLLER is available at the following link:

https://github.com/sahil-kale/MotorController/blob/3d-model/3d-model/Hornet_Controller.pdf

A.3 Design Decisions

A.3.1 Electrical Design Decisions

- A 4 layer PCB was chosen in order to aid with the relatively high amount of current that the board would be subjected to. Having dedicated ground and power planes ensured that power integrity was not an issue that would be faced by the HORNET MOTOR CONTROLLER.

A.3.2 Processor Decision

- An STM32F446RET6 was chosen as it was a proven microcontroller with 7 unique timer modules (3 are currently used), as well as UART, CAN, and SPI capability. Although the requirement for SPI was removed partway through development in favour of quadrature encoder support, the capability exists for the HORNET MOTOR CONTROLLER to support SPI encoders.

A.3.3 Firmware Decisions

- As a result of the variety of managers and classes being called by interrupts, singletons were employed in order to ensure that timer modules are utilized in the intended method and that copies of the managers are not inadvertently made. Additionally, compared to static functions, singletons offered the ability to encapsulate our code, increasing its reliability and maintainability.