ARTIFICIAL NEURAL NETWORK-BASED ROBOTICS

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 Electrical Engineering

by

Justin Ng

June 2018

© 2018
Justin Ng
ALL RIGHTS RESERVED

COMMITTEE MEMBERSHIP

TITLE:	Artificial Neural Network-Based Robotics
AUTHOR:	Justin Ng
DATE SUBMITTED:	June 2018
COMMITTEE CHAIR:	Andrew Danowitz, Ph.D. Assistant Professor of Electrical Engineering
COMMITTEE MEMBER:	Xiao-Hua Yu, Ph.D. Professor of Electrical Engineering
COMMITTEE MEMBER:	Fred W. DePiero, Ph.D. Professor of Electrical Engineering

ABSTRACT

Artificial Neural Network-Based Robotics

Justin Ng

Artificial neural networks (ANNs) are highly-capable alternatives to traditional problem solving schemes due to their ability to solve non-linear systems with a non-algorithmic approach. The applications of ANNs range from process control to pattern recognition and, with increasing importance, robotics. This paper demonstrates continuous control of a robot using an actor-critic algorithm based on deep deterministic policy gradients (DDPG) originally conceived by Google DeepMind. The robot performs tasks such as locomotion within an enclosed area and object transportation. The paper also details the robot design process and explores the challenges of implementation in a real-time system.

ACKNOWLEDGMENTS

Thanks to:

- Everyone for everything.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

Note: All project files can be found at `https://www.github.com/okayjustin/roborodentia2017`

1.1 Cal Poly Roborodentia

The robot is designed to compete in the 2018 Cal Poly Roborodentia, the university's annual intramural robotics competition, and thus conforms to its particular specifications and requirements. Briefly, competitors must produce autonomous robots to collect and fire Nerf Rival Balls into nets to win points. A drawing of the field is shown below in Figure 1.1. Two robots compete separately in each half so the effective field is a 4' wide by 8' long area enclosed by 4" walls. 1 inch PVC tubes along the 4' walls hold the balls which the robots fire into rectangular nets located along the 8' wall. The rules provide additional restrictions on robot dimensions, capabilities, and other aspects, to be covered specifically in the following chapters.

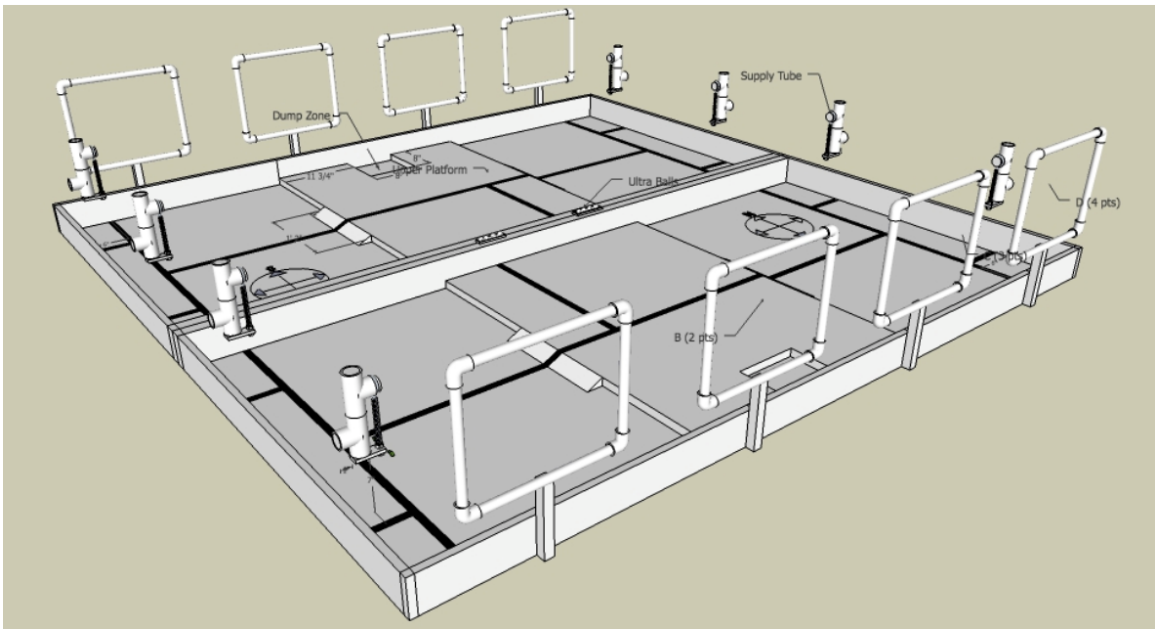


Figure 1.1: Roborodentia Field [4]

Chapter 2

MECHANICAL DESIGN

2.1 Introduction

The robot meets the design specification shown in Table 2.1. It consists of four subassemblies: the base platform, shooting mechanism, ball hopper, and control unit. Each section was first modeled in SolidWorks, an industry-standard solid modeling CAD program. The designed parts were then fabricated using a laser cutter or 3D printer and assembled with metric hardware. Figures 2.1 through 2.4 show standard view renders of the robot. Note that the robot uses mecanum wheels (a type of omni-directional wheel) which are modeled here as regular wheels for simplicity.

Table 2.1: Roborodentia 2018 Mechanical Requirements

	Requirement
1	Maximum footprint of 12" x 14" or smaller at start of match but may expand up to 14" x 17" during match.
2	Maximum height of 14" at start of match but no restriction during match.
3	Robot may not disassemble into multiple parts.
4	Robot may not be airborne.
5	Shooting mechanisms may not accelerate balls past 50 feet per second.

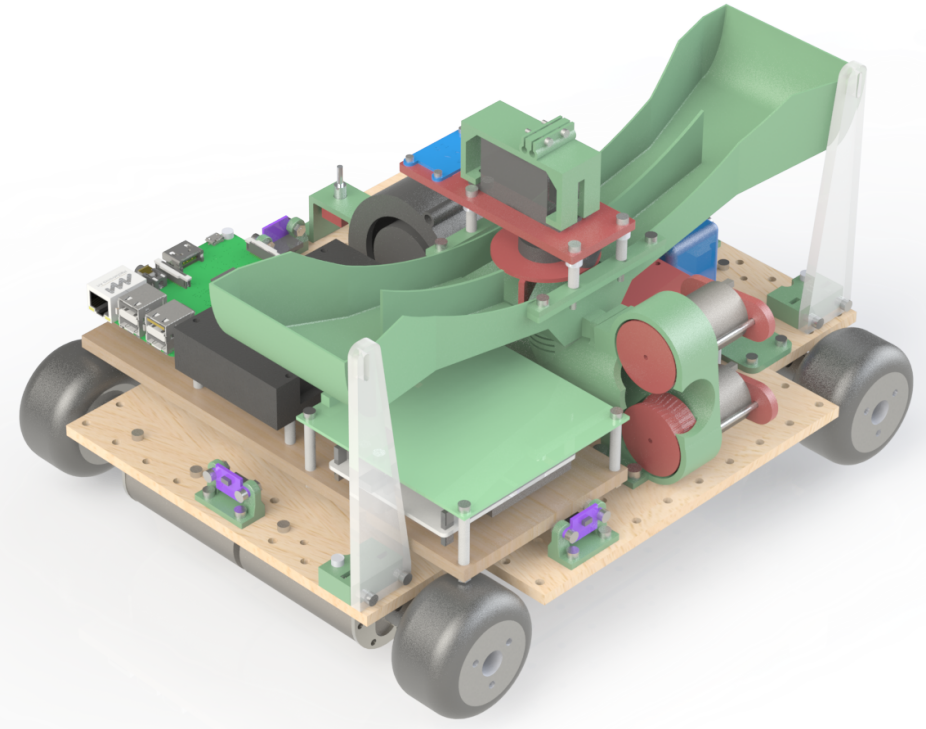


Figure 2.1: Full Robot Render – Isometric View

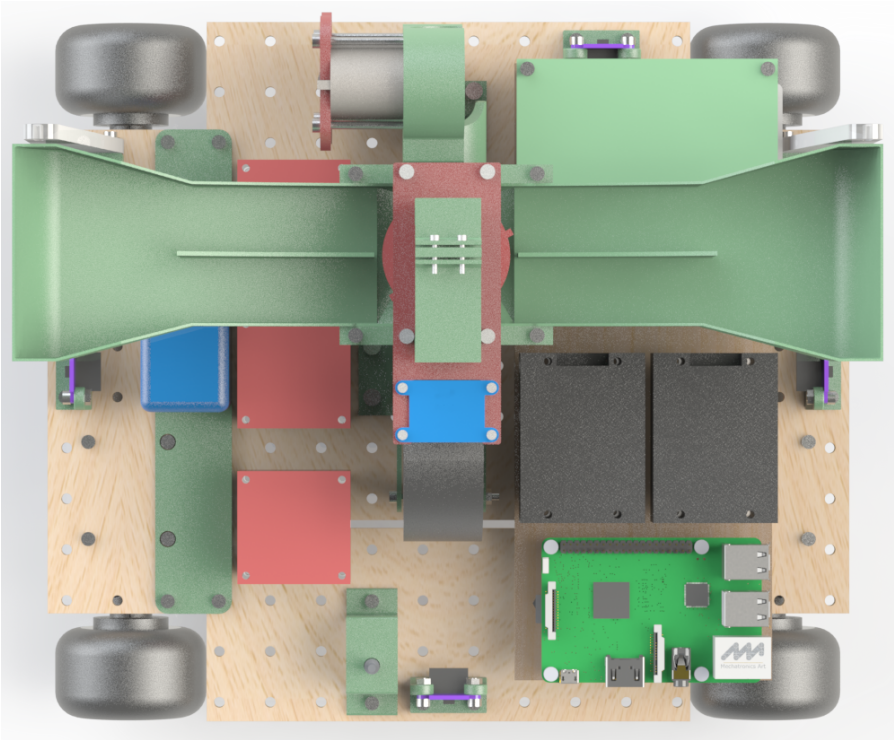


Figure 2.2: Full Robot Render – Top View

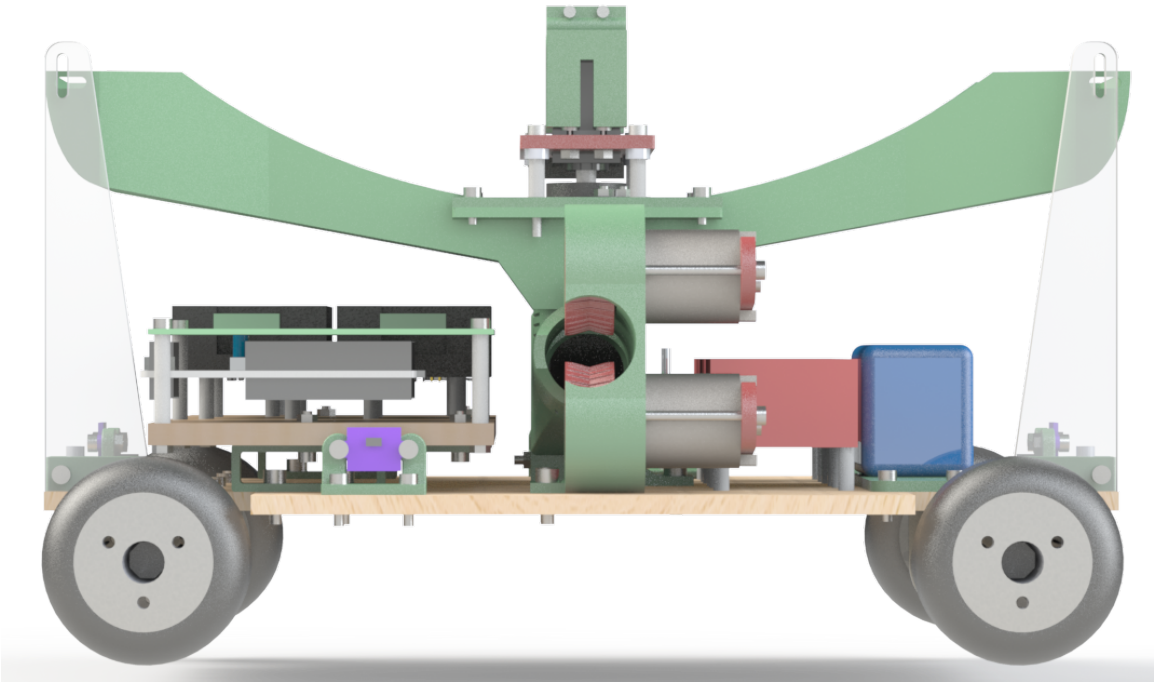


Figure 2.3: Full Robot Render – Front View

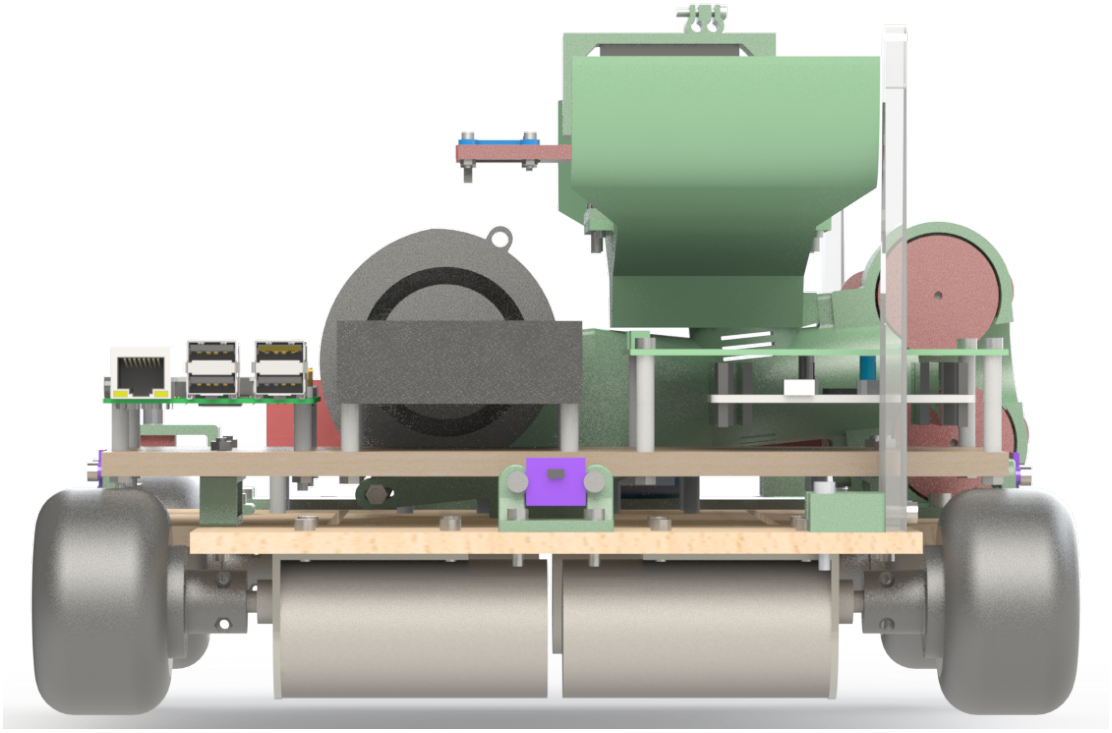


Figure 2.4: Full Robot Render – Right View

2.2 Base Platform

The base platform of the robot, made from 1/4" medium density fiberboard (MDF), serves as the primary structural component and a mounting point for the motors, electronics, shooting mechanism, and hopper. The wood is laser cut with a 20 mm grid of 4.5 mm holes to allow modular placement of components and the corners are removed to allow clearance for the wheels. Figures 2.5 shows the assembled view of the subassembly.

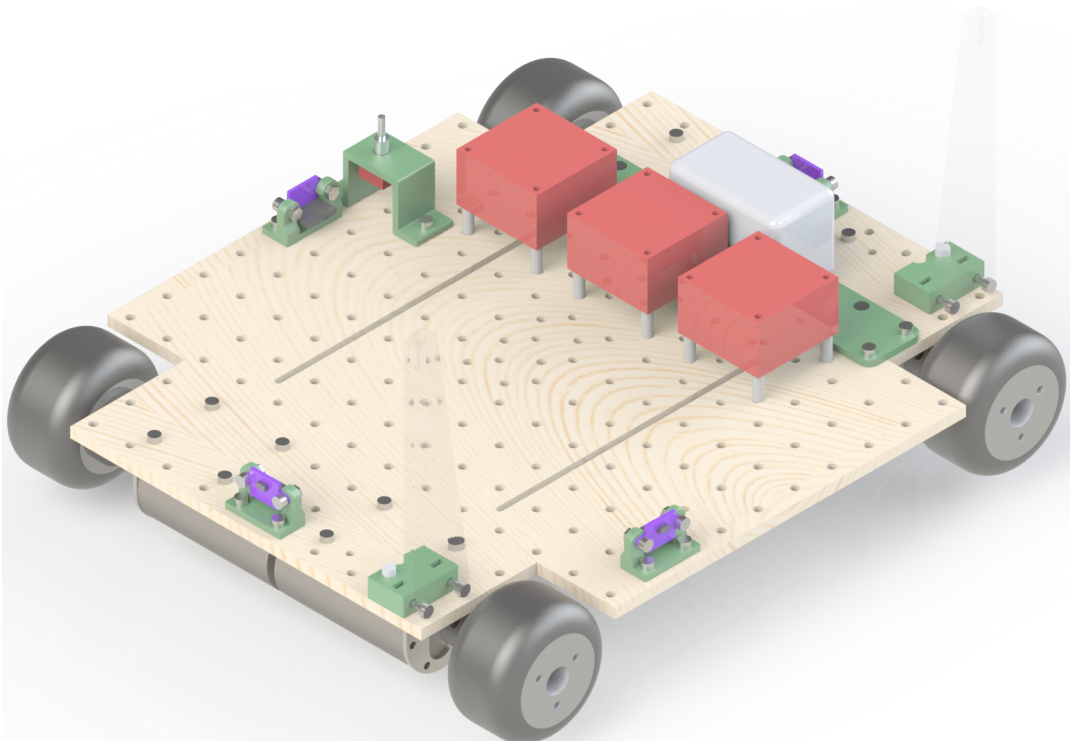


Figure 2.5: Base Platform

Four 12V Pololu 37D motors geared at a 70:1 ratio drive each of the 60 mm mecanum wheels. Each wheel contains eight angled rollers so unlike regular wheels which only produce a force vector perpendicular to the axis, mecanum wheels also produce a vector parallel to the axis. With the appropriate combination of speed and direction of each wheel, the robot can achieve simultaneous translation and rotation

in any direction.

Several electronic components are mounted directly on the base platform. Along the four edges of the platform, four ST Microelectronics VL53L0X laser rangefinders mounted on 3D printed brackets sense distance. These sensors cost between $6 to $20 mounted on a small PCB with supporting circuitry and can sense distances between 30 mm and 2000 mm at a rate of 30 Hz and less than 10% error in most test conditions [15]. A 4S, 1200 mAH LiPo battery powers the system through a on/off toggle switch.

2.3 Shooting Mechanism

The shooting mechanism naturally takes inspiration from the official Nerf Rival Blaster toys since the manufacturer specifically optimized them to fire Nerf Rival balls in a way similar to baseball pitching machines. Figure 2.6 shows an exploded view of the subassembly while Figure 2.7 displays the top view. The mechanism consists of two sections: the **barrel** (left green part in Figure 2.6) and the **wheel housing** (right green part in Figure 2.6). Both parts were fabricated using a fused deposition modeling (FDM) 3D printer as the geometries are highly complex. Therefore, the shooting mechanism consists of two separate components versus a unibody design to allow each half to be fabricated with optimal print direction, strength, and finish quality. The barrel is angled 6° above horizontal, targeting the vertical center of the nets.

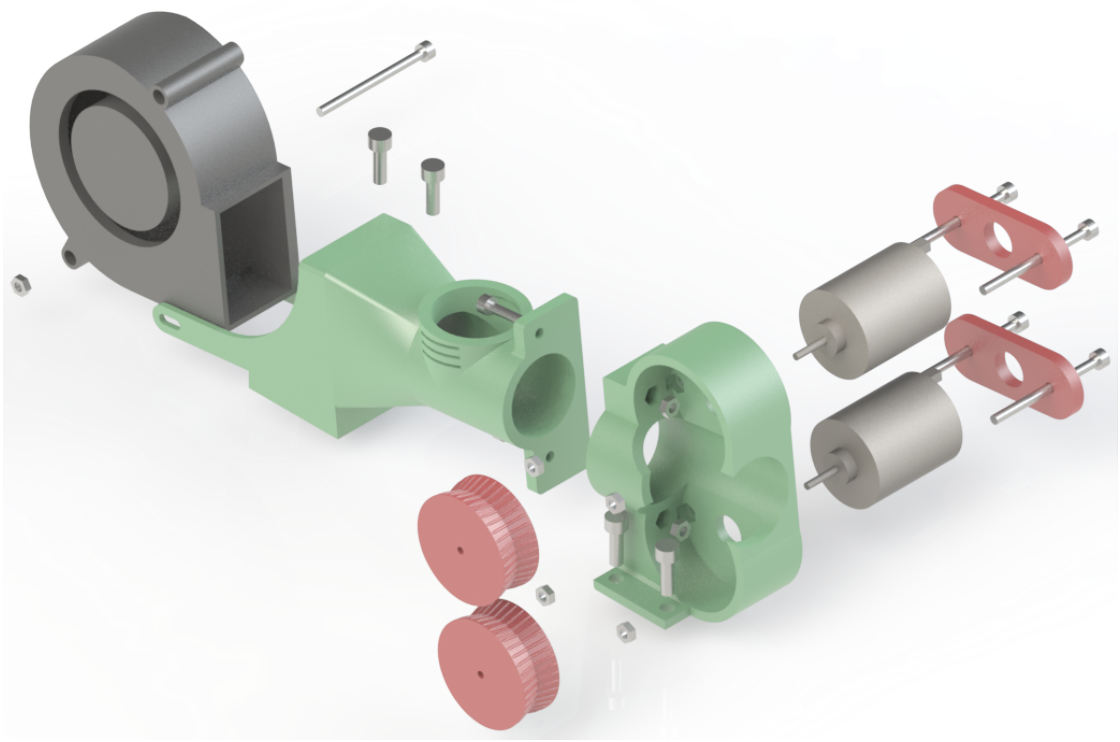


Figure 2.6: Shooting Mechanism – Exploded View

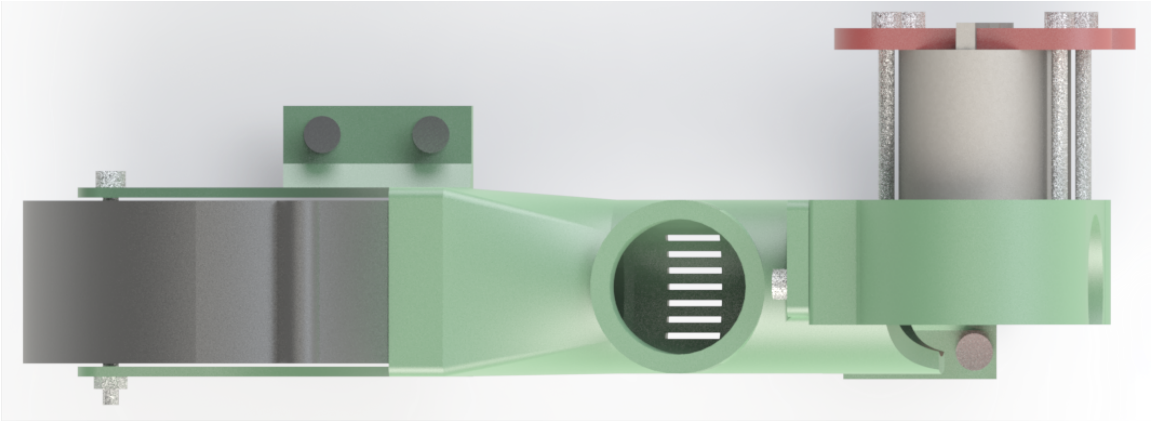


Figure 2.7: Shooting Mechanism – Top View

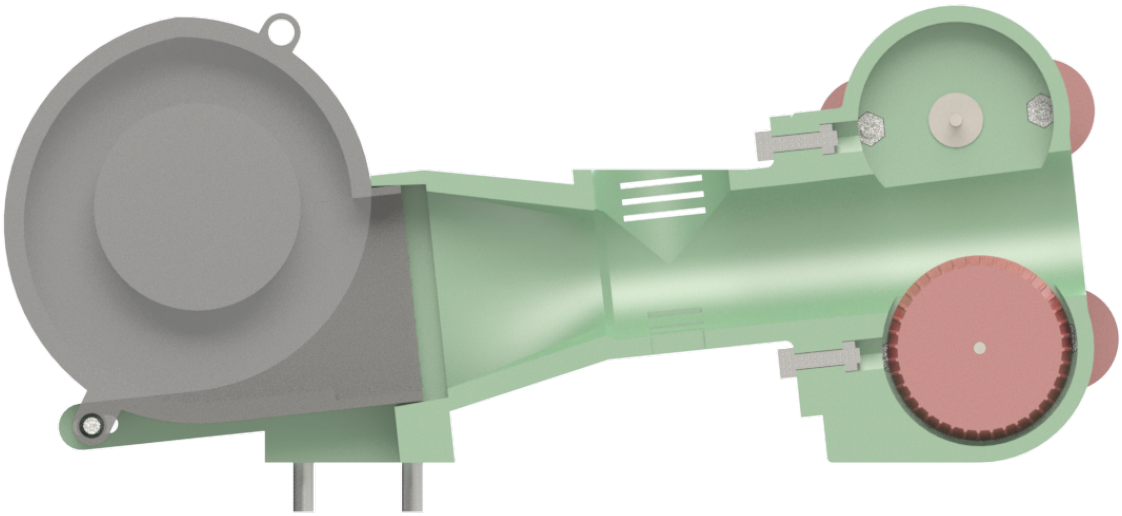


Figure 2.8: Shooting Mechanism – Cross Section View

The **barrel** directs balls from the **ball hopper** to the **wheel housing**. First, the ball enters the barrel through a vertical chute by force of gravity. As the ball falls into the barrel, a high-pressure centrifugal (or blower fan) attached at the back of the barrel pushes it into the wheel housing inlet. As seen in Figure 2.8, the barrel slightly narrows in the area behind the top chute to prevent the ball from rolling backwards towards the blower fan. A loft feature creates a smooth transition between the rectangular fan connection and the circular barrel. The foam balls, nominally 23 mm in diameter, would occasionally jam in a 24 mm barrel so all pathways are 25 mm. In the initial design, the pressure created by the blower fan was so high that it prevented the ball from falling down the vertical chute so strategically placed vents reduce the barrel pressure as the ball falls through the chute. As the ball travels down the chute into the barrel, it blocks the vents, increasing the pressure and forcing the ball into the wheel housing.

Inside the **wheel housing**, two counter-rotating 34mm wheels press fitted to two high-speed 12 V motors rapidly accelerate the foam ball up to 50 feet per second. The 14 mm gap between wheels compresses the ball to increase grip, thereby improving

energy transfer. The motors lightly press fit into the wheel housing and are secured with 3D printed braces. The perimeter of each 3D printed wheel, detailed in Figure 2.9, consists of a ribbed V-groove to increase the contact patch and grip with the compressed foam ball. Two "feet" with bolt holes at the bottom of the barrel and wheel housing secure the shooting mechanism to the base platform.

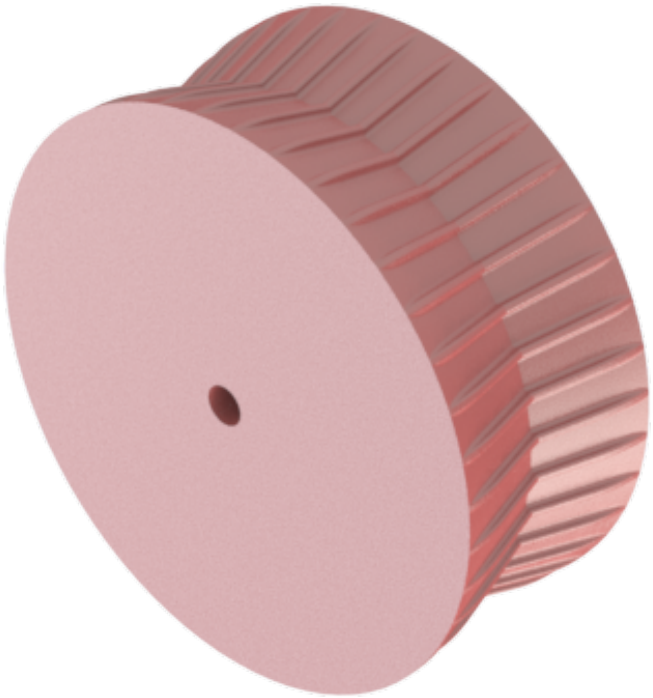


Figure 2.9: Shooting Mechanism – Shooter Wheel

2.4 Ball Hopper

The robot must obtain the foam balls from supply tubes mounted on two sides of the side. The bottoms of the supply tubes are positioned seven inches above the floor and a swinging flap holds the balls in. The ball hopper, shown in Figure 2.10 is a large 3D printed component designed to push the swinging flap away, collect the balls, store them, and dispense them into the shooting mechanism. Figure 2.11 shows an exploded view of the subassembly.

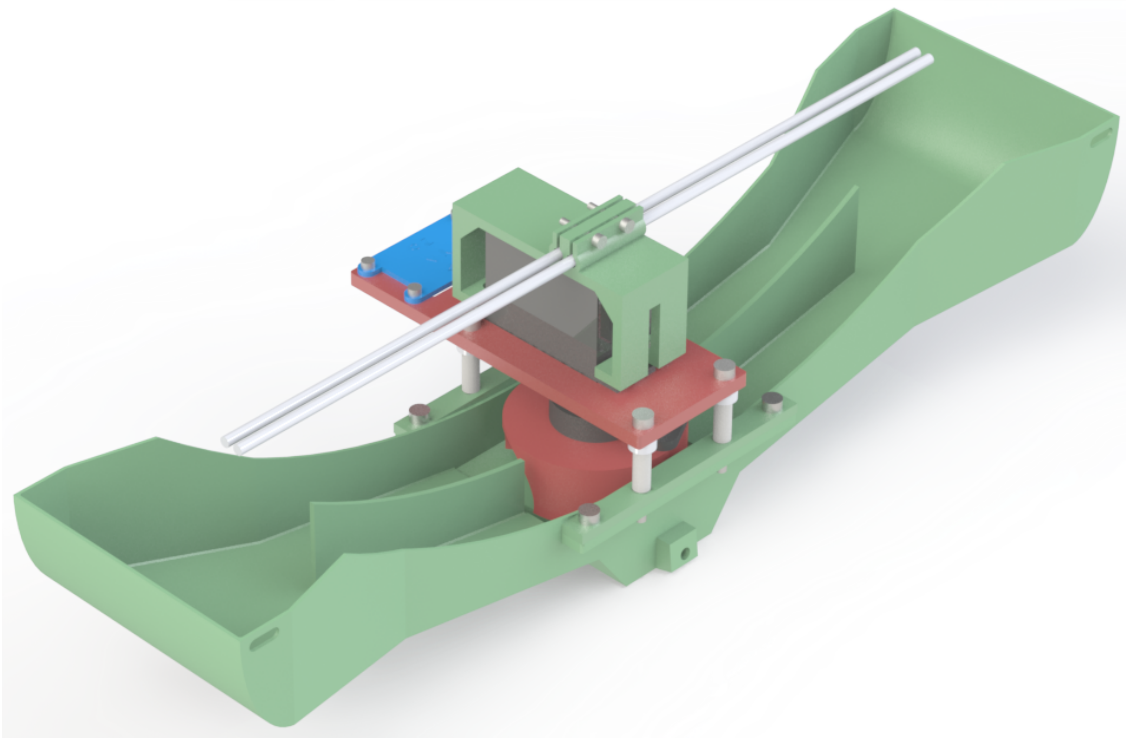


Figure 2.10: Ball Hopper

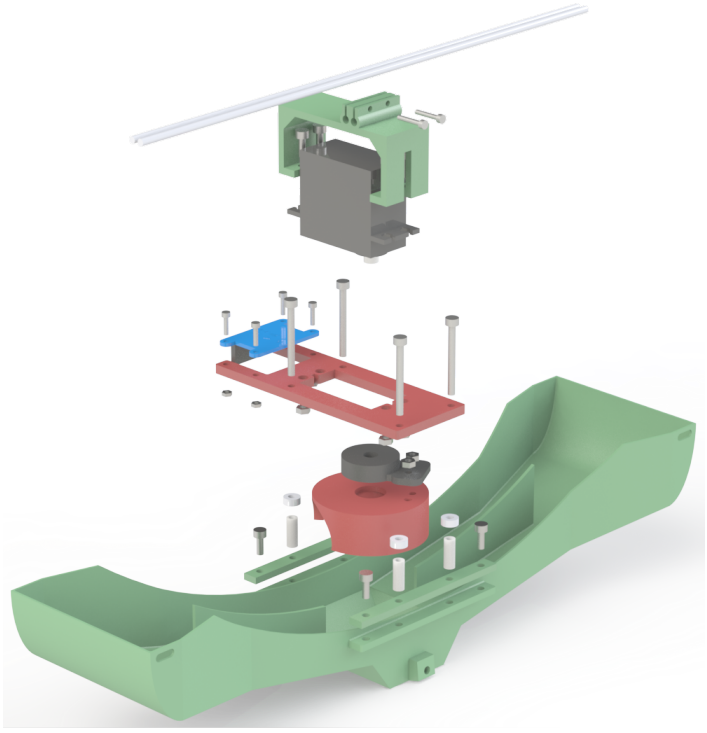


Figure 2.11: Ball Hopper – Exploded View

First, the robot moves the hopper underneath the supply tube. As the flap swings open, the balls rolls down the steep sloped portion of the hopper. Visible in the cross section view of Figure 2.12, The slope rapidly becomes less steep in order to transition the balls downward momentum into sideways momentum, keeping balls from jamming against each other. The balls then roll into one of two channels before stopping at the dispensing gate.

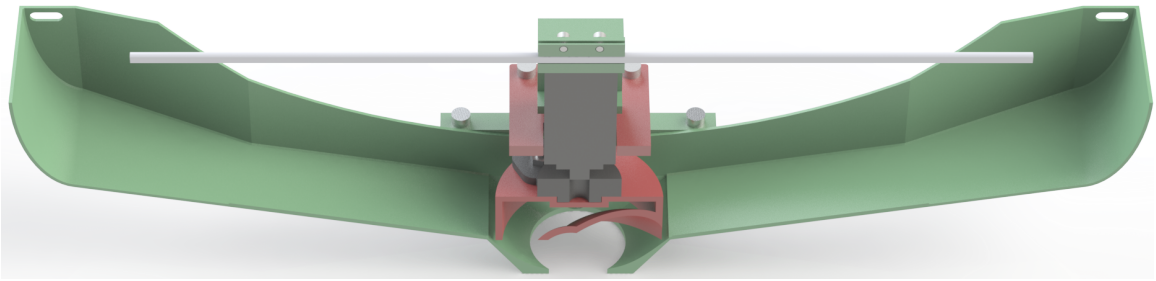


Figure 2.12: Ball Hopper – Cross Section View

The dispensing gate, shown in Figure 2.13, controls the movement of balls between hopper channel and shooting mechanism entrance. Its complex shape directs balls into the center of the ball hopper from one channel at a time to prevent jamming. A common 180° movement servo, mounted in a 3D printed bracket above the center of the hopper, controls the dispensing gate. Fastened to the same bracket, an inertial measurement unit (IMU) measures magnetic compass heading and acceleration in three dimensions.

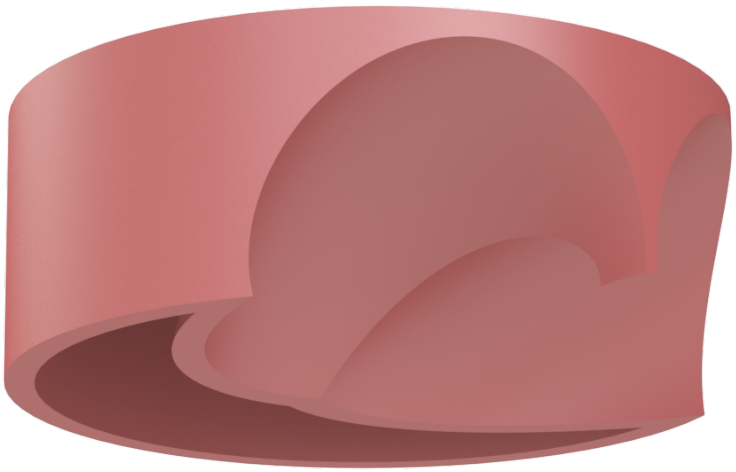


Figure 2.13: Ball Hopper – Dispensing Gate

The ball hopper is mounted at three points: the top of the shooting mechanism and the left and right edges of the robot using 3D printed and acrylic braces shown in Figure 2.14.

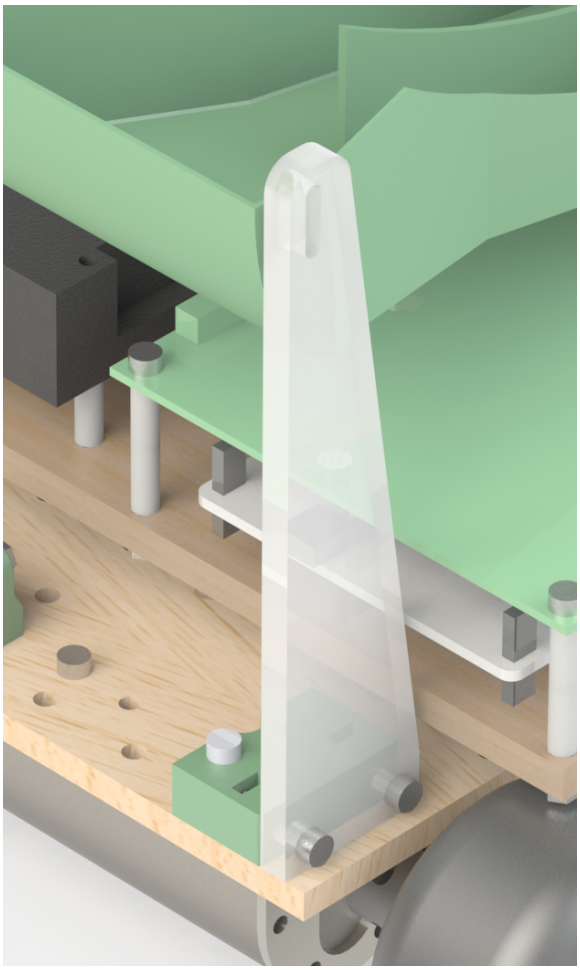


Figure 2.14: Ball Hopper – Braces

2.5 Control Unit

The control unit, shown in Figure 2.15, consists of a 1/4" MDF board with various electronic components mounted: two off-the-shelf DC-DC switching converters, a custom interconnect printed circuit board (PCB), an off-the-shelf STM32 Nucleo-64 development board, and the Raspberry Pi computer. Two 3D printed standoffs, the green parts shown in Figure 2.16, connect the control unit to the platform and raise it slightly to avoid colliding with the robot's wheels.

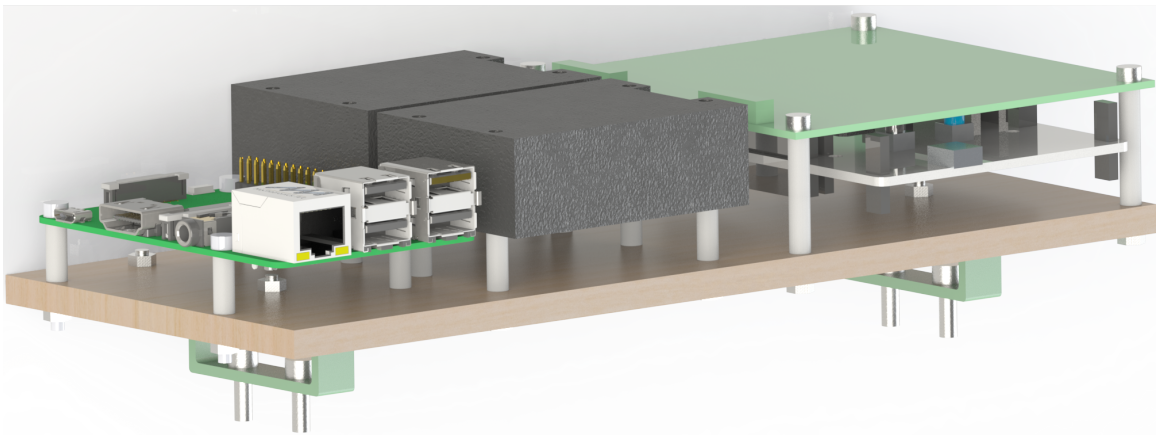


Figure 2.15: Control Unit

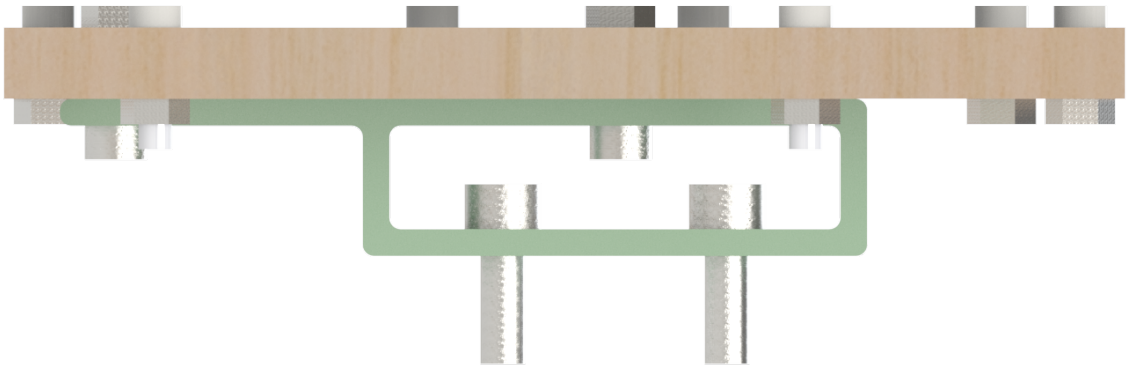


Figure 2.16: Control Unit – Standoffs

Chapter 3

ELECTRICAL DESIGN

3.1 Introduction

The electronics of the robot use a combination of off-the-shelf parts and custom designed circuits.

3.2 Power

A four cell, 1800 mAH lithium polymer (LiPo) battery powers the entire system. The battery is connected using polarized XT60 connectors to prevent reverse connection. The battery voltage varies between 16.8 V when fully charged and 14.8 V when depleted so two off-the-shelf DC-DC switching regulators, shown in Figure 3.1, buck battery voltage down to 12 V and 7 V supplies. The switching regulators accept a 7 – 40 V supply and can output 1.2 – 35 V at 8 A each. The 12 V bus powers the three motor drivers boards while the 7 V bus powers the STM32 Nucleo-64 development board and two AZ1085CD low-dropout linear regulators (LDO). One LDO produces a 5 V bus while the other provides 3.3 V, each at 3 A.

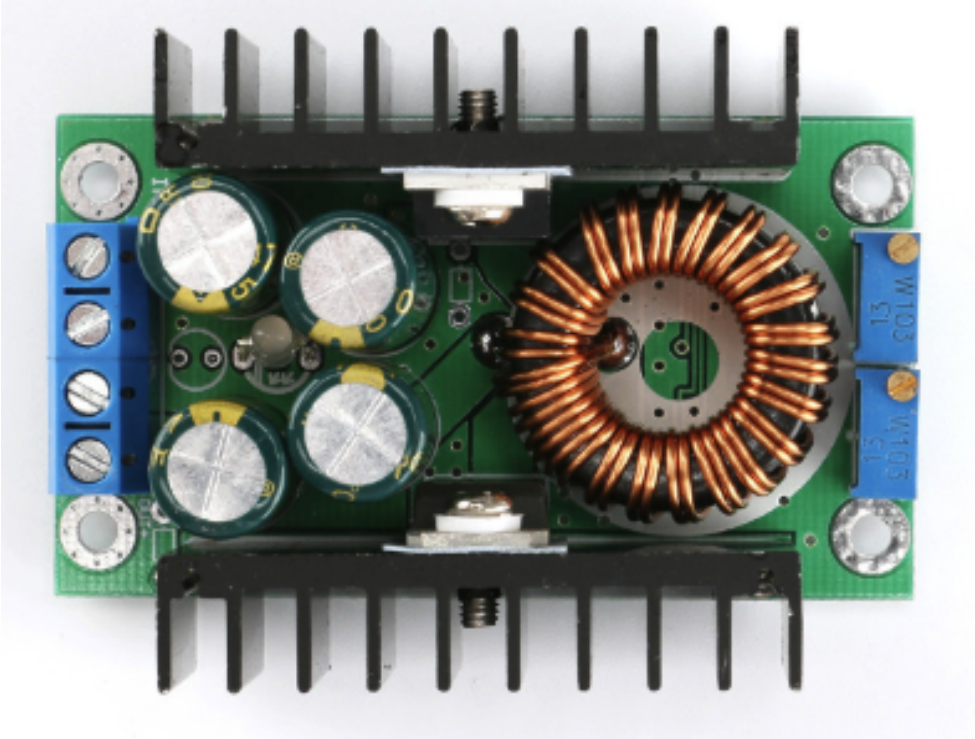


Figure 3.1: DC-DC Buck Regulator [3]

3.3 Sensors

The robot uses five off-the-shelf sensors for determining its position: four VL53L0X 1-D LIDAR rangefinders and one Adafruit 9-DOF inertial measurement unit (IMU).

3.3.1 Adafruit 9-DOF IMU

The Adafruit 9-DOF IMU incorporates the L3DG20H gyroscope and LSM303DLHC accelerometer/compass combo on a single carrier board to allow full inertial measurement in a convenient form factor [2]. Figure 3.2 shows the IMU mounted in the 3D printed bracket. The robot only utilizes the accelerometer and magnetic compass to realize a tilt-compensated compass. The LSM303DLHC can measure both acceleration and magnetic fields in three dimensions with configurable bandwidth and

full-scale ranges. It uses 400 kHz I^2C for control and data transfer and draws power from the 3.3 V supply. Raw measurements from the IMU possess significant offset and scaling error so a calibration routine is required.

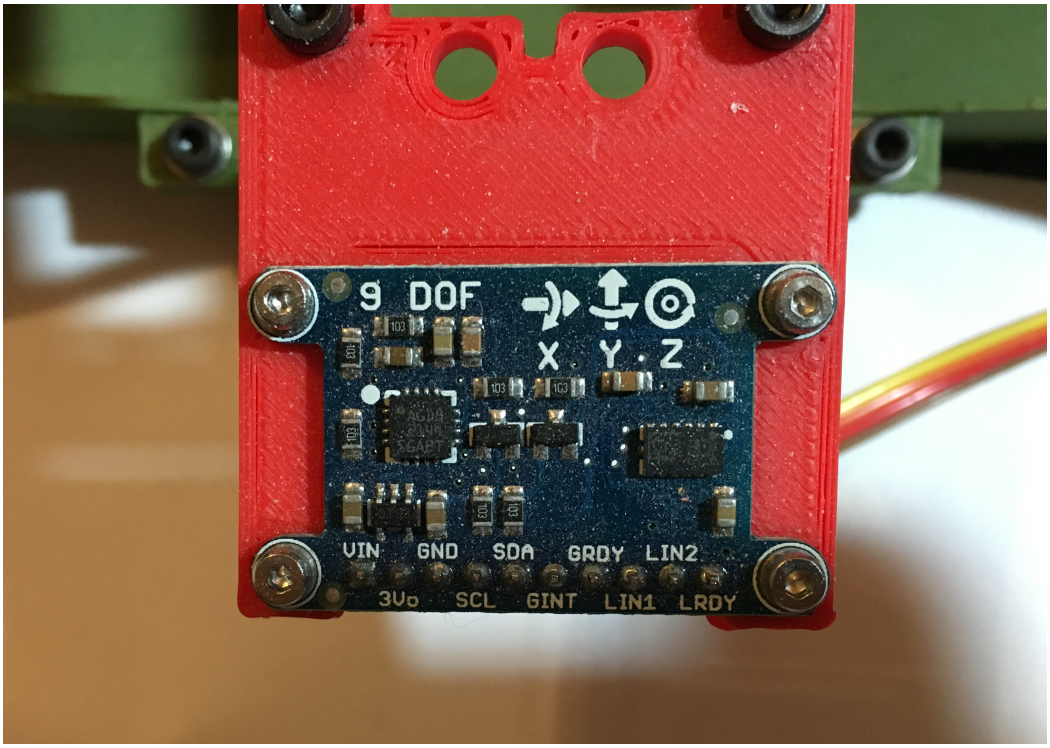


Figure 3.2: Adafruit 9-DOF IMU Mounted

The accelerometer and magnetometer calibration process utilizes the FreeIMU program written by Fabio Varesano in Python [5]. The program features a graphical user interface (GUI), shown in Figure 3.3, to display accelerometer and magnetometer measurements in real time and plots them in 3D space. The calibration program was originally designed to calibrate the open-source FreeIMU IMU when connected to an Arduino with the FreeIMU calibration firmware so the program's device communication back-end was modified to accept the LSM303DLHC IMU connected to an STM32 microcontroller with custom firmware. The calibration algorithm assumes that the sensor's measurements are linearly distorted and therefore produces a linear scaling factor and offset for each of six measurements (accelerometer X, Y, Z

and magnetometer X, Y, Z). The correction algorithm is shown below where *val* can represent any of the six measurements.

$$val_{calibrated} = m_{scale} val_{measurement} + b_{offset} \tag{3.1}$$

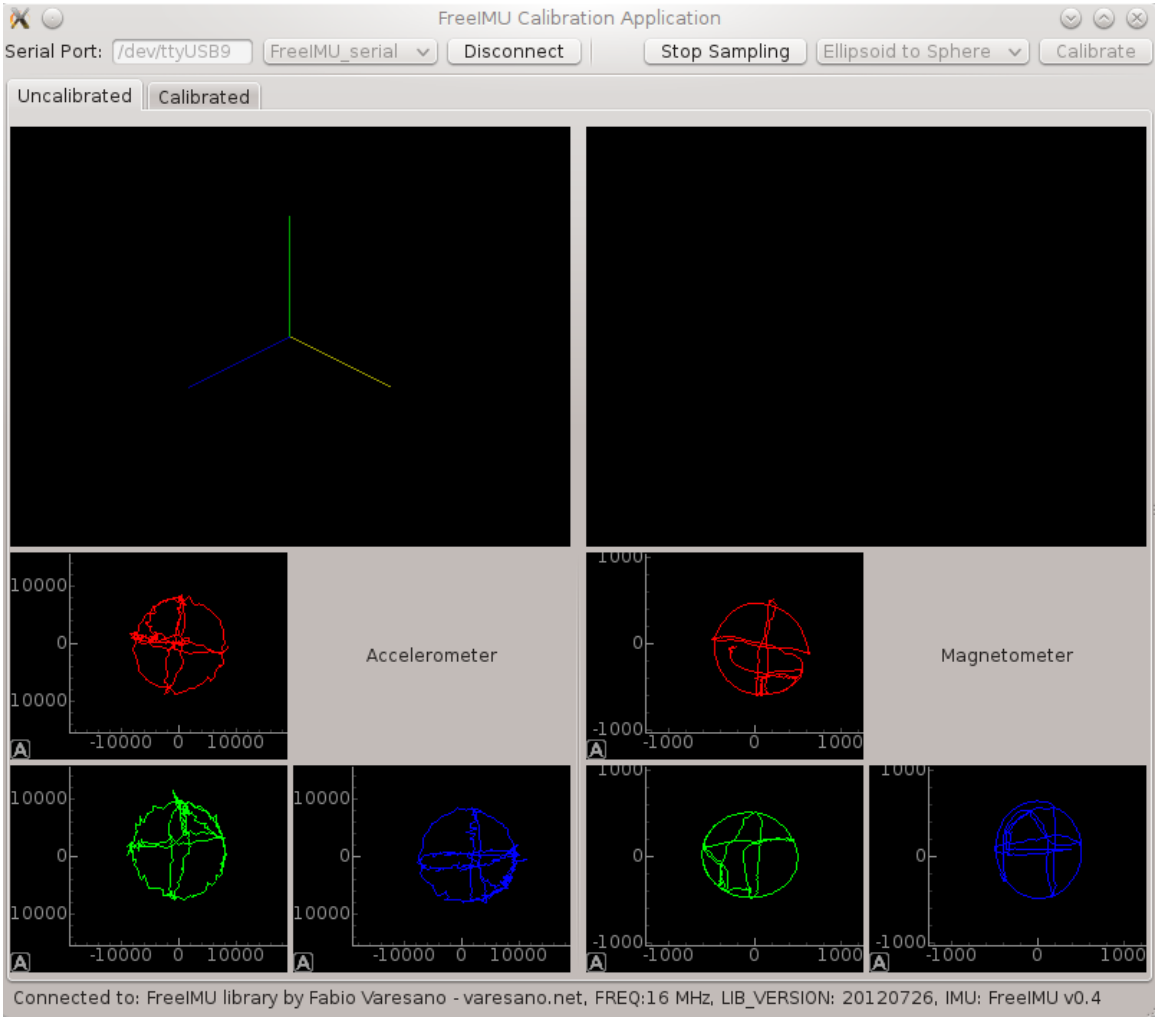


Figure 3.3: FreeIMU GUI [5]

The calibration procedure is as follows:

1. IMU should be mounted on fully assembled robot and connected to microcontroller.
2. Connect microcontroller serial port to computer through Serial-to-USB converter.

3. Click "Begin Sampling" to start recording magnetometer and accelerometer measurements.
4. Point the IMU x-axis at the ground and rotate 360° around that axis. Repeat for y-axis and z-axis.
5. Point the IMU x-axis at the sky and rotate 360° around that axis. Repeat for y-axis and z-axis.
6. Repeat Steps 3 and 4 at least twice to increase data size.
7. Click "Stop Sampling".
8. Click "Calibrate" to calculate scaling and offset constants.

The procedure ensures that ends of each axis eventually receive the maximum acceleration and magnetic field. To ensure hard-iron errors (such as from motors and permanent magnets) as well as soft iron errors (from local ferromagnetic materials like steel) are compensated for in the calibration, the process should be performed with the fully assembled robot and redone each time the robot is modified [10]. Since the expected fields are known (acceleration due to gravity, strength of Earth's magnetic field), the required linear transformation can be calculated. See [5] for details of the exact algorithm.

3.3.2 VL53L0X Rangefinders

The rangefinders, marketed by STMicroelectronics as the "world's smallest Time-of-Flight ranging sensor", are capable of measuring between 30 and 2000 mm with a 30 Hz sample rate [14]. It operates by firing pulsed light from a vertical cavity surface emitting laser (VCSEL), measuring time taken for the laser pulse to reflect back to the sensor, and calculating the distance based on the known speed of light. The robot uses cheap $8 VL53L0X breakout boards in lieu of designing and assembling custom carriers; the sensor itself comes in a 4.4 x 2.4 mm lead-less package making hand soldering prohibitively difficult. Figure 3.4 shows the sensor board mounted in

the 3D printed bracket. The sensor breakout boards include the VL53L0X module, decoupling capacitors, an LDO for the sensor's 2.8 V power supply, and level shifters for the I^2C lines. Control and data transfer both occur over 400 kHz I^2C so the breakout only requires four wires: 3.3 V, ground, and the I^2C data and clock lines. To characterize the accuracy and precision of the sensor, distance measurements were taken by targeting the sensor at a sheet of standard white printer paper. The data is compared with measurements from a tape measure; results are shown in Figure 3.5.

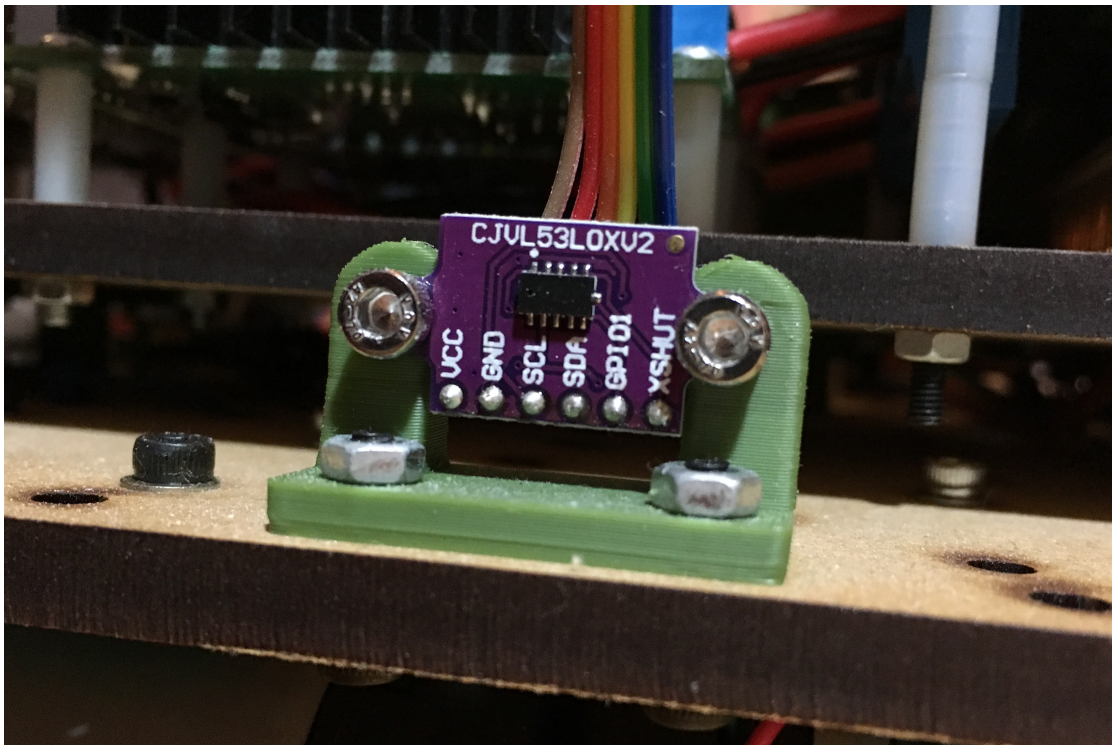


Figure 3.4: VL53L0X Rangefinder Mounted

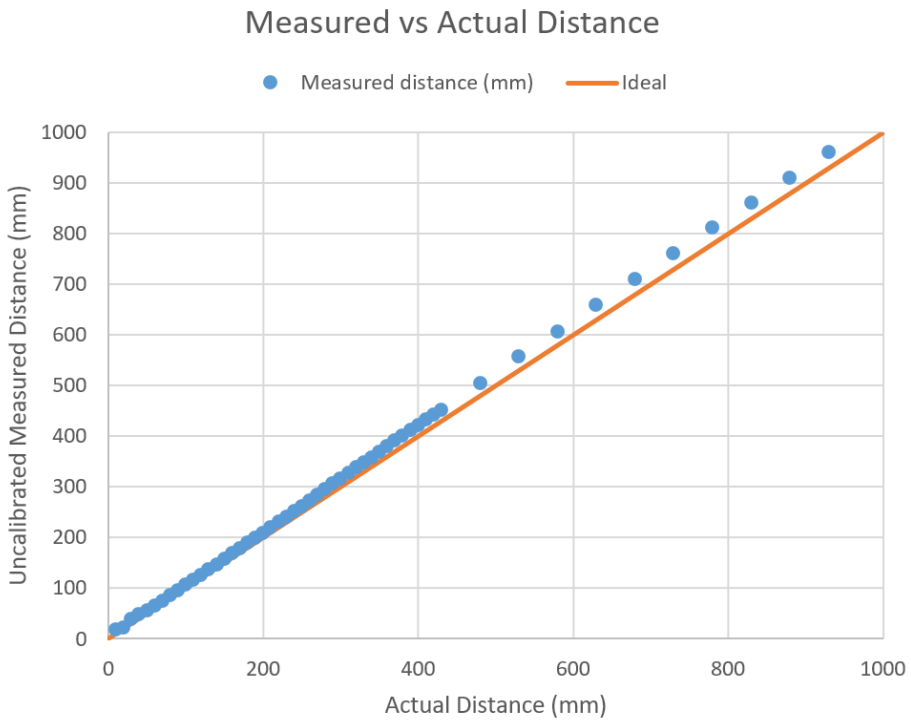


Figure 3.5: VL53L0X Measured vs. Actual Distance

Since the measurements are linear, only a linear correction is required as follows:

$$d_{calibrated} = m_{scale} d_{measurement} + b_{offset} \tag{3.2}$$

where $d_{measurement}$ is distance as returned by the sensor, $m_{scale} = 0.96502507$, $b_{offset} = -3.8534743$, and $d_{calibrated}$ is the calibrated distance. The squared error for each data point before and after calibration is shown in Figure 3.6. The mean squared error for the uncalibrated data points is 342.3 mm and 15.0 mm after calibration, indicating the use of an appropriate model and constants.

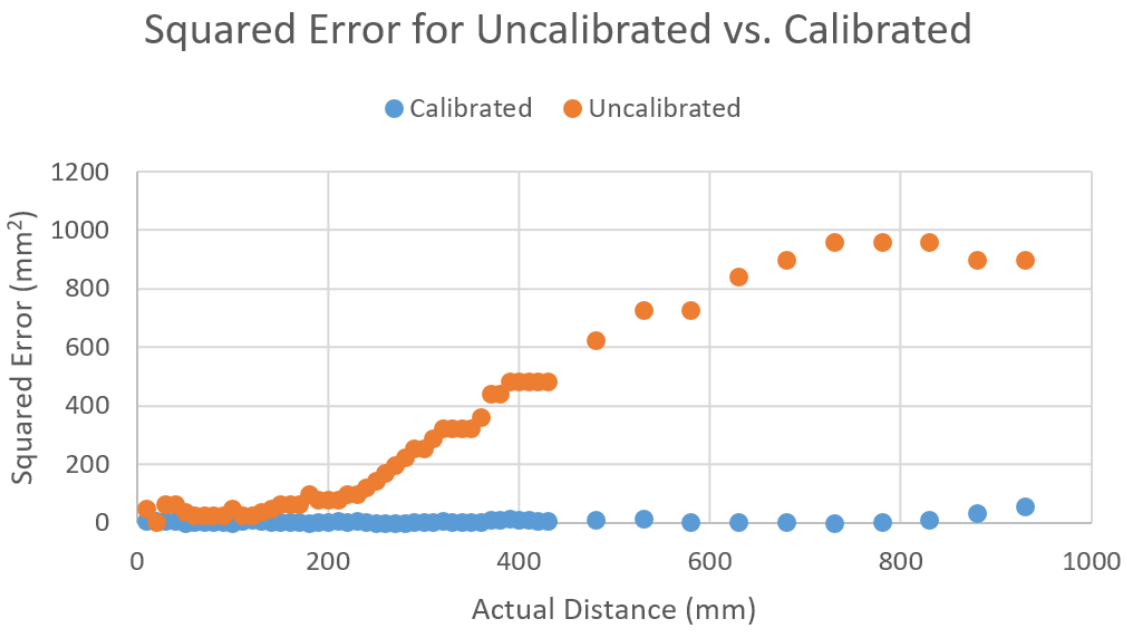


Figure 3.6: VL53L0X Squared Error for Uncalibrated vs. Calibrated

Additionally, 100 measurements were taken at each distance and the variance computed to characterize the spread. The results are shown in 3.7. The rangefinders are very accurate; at 900 mm with calibration, the error is only 7 mm or 0.8% error. Of course, this error is with respect to the average measurement at 900 mm. At 900 mm, the standard deviation is 14 mm so individual measurements can vary, especially with non-ideal surfaces. The closer the target, the more accurate and less varied the measurement. No experiments were carried out to determine the measurement characteristics on various colored, textured, transparent, or angled surfaces as the expected target surface is white painted wood.

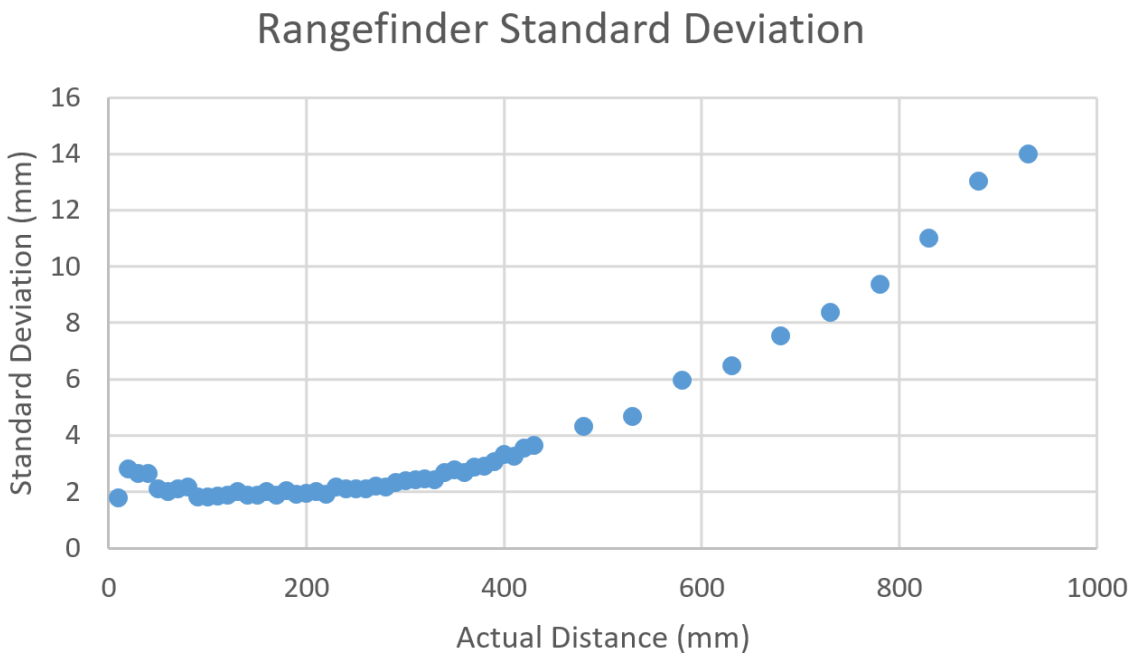


Figure 3.7: VL53L0X Standard Deviation

3.4 Motor Drivers

The system uses three off-the-shelf L298N motor driver boards since they are easily obtainable for less than $6 each and incorporate features such as heat-sinking, flyback voltage protection, supply filtering, and screw terminal connections. Implementing comparable motor drivers with a similar feature set would undoubtedly cost more. Each L298N is a dual H-bridge driver with 2 A maximum output per bridge using a 5 – 35 V supply. Two motor drivers handle the four robot drive motors while the third powers the blower fan and shooting mechanism motors.

Figure 3.8 shows a wiring diagram for each motor driver. The board uses four digital control inputs, each controlling the state of one half-bridge. Each motor uses a pair of inputs: IN1 and IN2 control one motor while IN3 and IN4 control the other. To achieve direction and speed control, IN1 and IN3 are pulse width modulated (PWM) while IN2 and IN4 are digitally set. Table 3.1 is a truth table of the motor state

versus inputs.

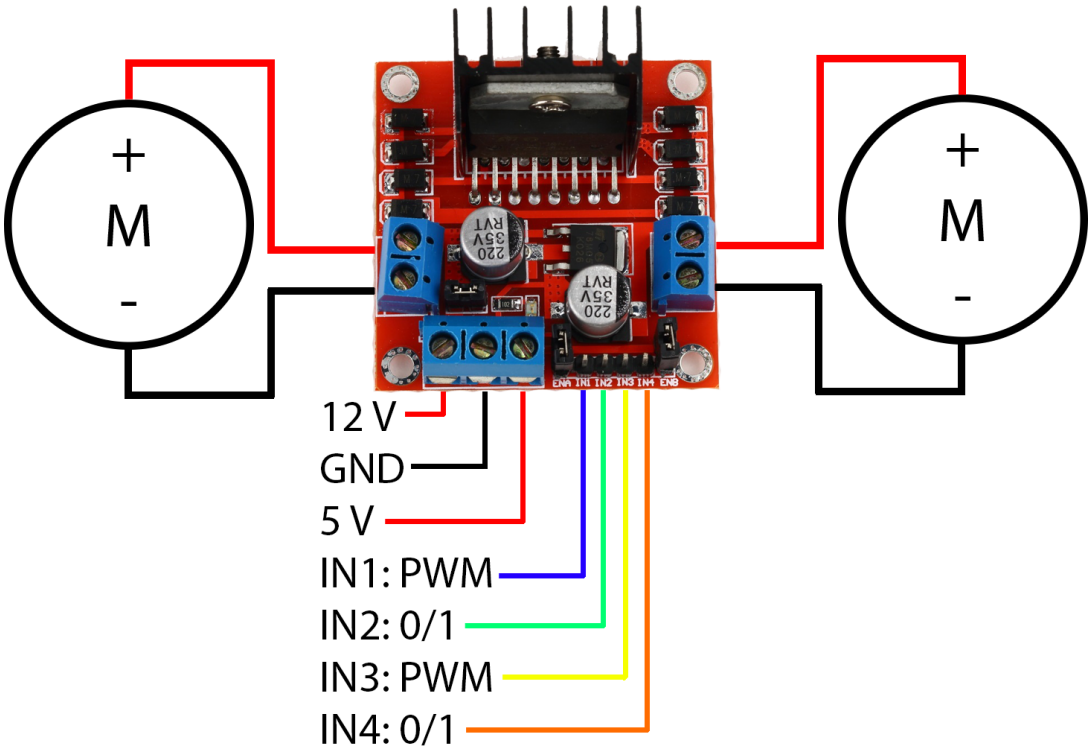


Figure 3.8: L298N Motor Driver Wiring Diagram [8]

Table 3.1: Motor Control Truth Table

IN1/IN3 Duty Cycle	IN2/IN4 State	Motor State
0%	0	Stopped
>0%	0	Forward, speed increases with duty cycle
<100%	1	Reverse, speed decreases with duty cycle
100%	1	Stopped

3.5 Servo

A GWS S03N standard servo powered from the 5 V bus actuates the gating mechanism in the ball hopper. Most servos are controlled by driving the control wire with a pulse

width modulated (PWM) signal with a period of 15 – 25 ms and a pulse width between 0.5 ms and 2.5 ms where the pulse width determines the position of the servo as shown in Figure 3.9.

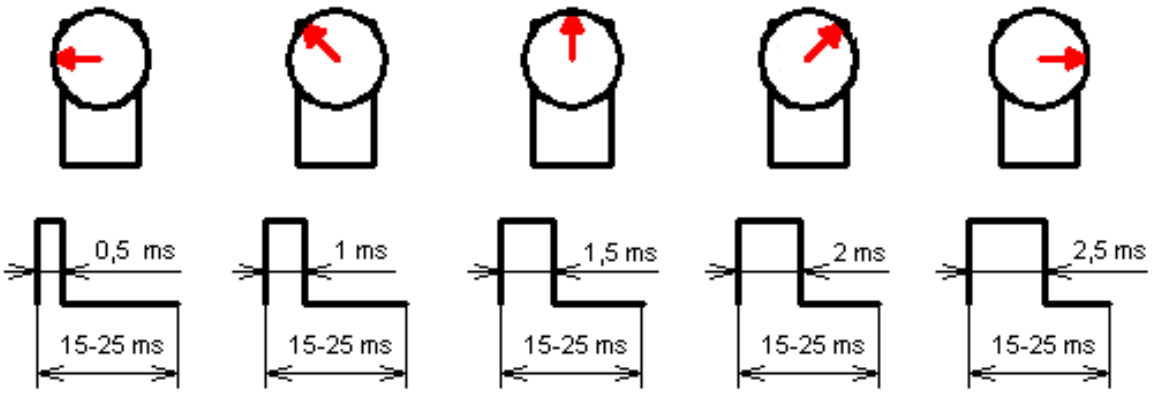


Figure 3.9: Servo PWM Control Scheme [7]

Due to cheap manufacturing and loose tolerances, the actual required pulse widths can vary from servo to servo, requiring calibration to obtain accurate positional control. The process for calibration is simple: apply various pulse widths and record the resulting servo positions. The calibrated pulse widths and servo angles are recorded in Table 3.2.

Table 3.2: Servo Required Pulse Widths

Servo Position	Pulse Width (ms)
0°	0.674
45°	1.082
90°	1.490
135°	1.898
180°	2.306

3.6 Microcontroller

An STMicroelectronics STM32F446RE microcontroller (MCU) serves as the bridge between the robot's low-level electronics and the high-level control system running on a desktop computer. Specifically, the MCU collects data from sensors over I^2C and general purpose input/output (GPIO), generates control signals for the motor drivers and servo, and services commands from UART (universal asynchronous receiver-transmitter). The MCU resides on an STMicroelectronics Nucleo-64 development board, shown in Figure 3.10, which conveniently integrates an ST-LINK V2 debugger, programmer, and USB-to-UART interface.

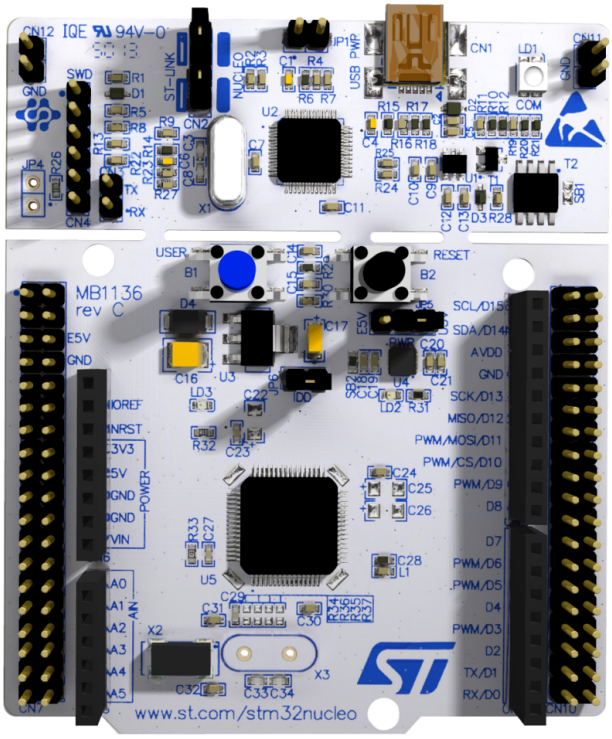


Figure 3.10: STM32 Nucleo-64 Development Board [13]

3.7 Interconnect PCB

The interconnect PCB is a custom designed board with three goals: regulate voltage, route power, and connect peripherals to the development board. The PCB routes input battery power to two external buck regulators, producing 12 V and 7 V. The 7 V bus is further regulated to 5 V and 3.3 V logic rails using two AZ1085CD LDOs and then routed to the development board, sensors, etc. Most importantly, the interconnect board provides dedicated 2.0 mm JST connectors for all sensors, motor drivers, and the servo to simplify and robustify wiring across the robot. In addition, hand-crimped cables made with JST connectors and 28 gauge ribbon cable prevent rats nest wiring. A block diagram is shown in Figure **??**. Due to uncertainty in the final robot design, extra hardware and circuits were added. Table 3.3 lists the supported features.

Table 3.3: Interconnect PCB – Supported Features

Quantity	Feature
6	VL53L0X laser rangefinder
1	Adafruit 9-DOF IMU
3	Dual H-bridge motor driver
2	5 V Standard servo
2	8 sensor IR proximity array
4	Debug LEDs
1	Debug button
1	Reset button

The system utilizes two 400 kHz I^2C buses to improve communication bandwidth with the numerous I^2C devices. Table 3.4 lists I^2C device addresses and bus allocation. The two buses are named I2C2 and I2C3 to maintain parity with the microcontroller's

naming convention.

Table 3.4: Interconnect PCB – I²C Devices

I²C Bus	Device	Address (7-bit)
I2C2	Rangefinder 4	0x52
I2C2	Rangefinder 5	0x53
I2C2	Rangefinder 6	0x54
I2C2	Adafruit 9-DOF IMU Gyroscope	0x69
I2C2	Adafruit 9-DOF IMU Accelerometer	0x19
I2C2	Adafruit 9-DOF IMU Magnetometer	0x1E
I2C2	GPIO Expander 3	0x3A
I2C3	Rangefinder 4	0x52
I2C3	Rangefinder 5	0x53
I2C3	Rangefinder 6	0x54
I2C3	GPIO Expander 1	0x38
I2C3	GPIO Expander 2	0x39

The schematic capture and board layout were created in CadSoft EAGLE 7.4 due to its simplicity of use, popularity, and community support. The schematic can be found in Appendix A. The board uses two 1 ounce copper layers for cost effectiveness. Two large headers underneath the board connect to the headers on the top of the microcontroller development board while the JST connectors are placed across the top of the PCB. 18-gauge wires soldered to large plated through-holes at the right edge of the board connect the external buck regulators. All components are placed on the top side of the board (except for the development board connectors) to make hand-soldering easier. The completed electronics assembly is shown in Figure 3.11.

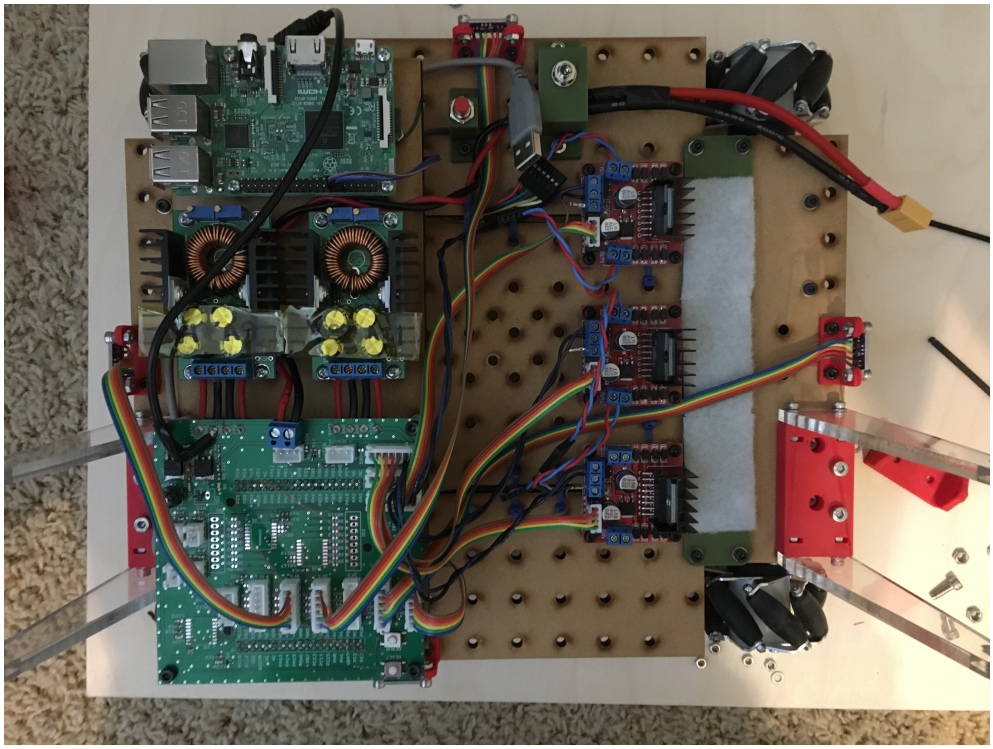


Figure 3.11: Fully Assembled Electronics

Chapter 4

FIRMWARE DESIGN

4.1 STM32CubeMX

The robot uses an STMicroelectronics STM32F446RE microcontroller (MCU) for low level sensor interfacing and motor control. Before starting the electrical design, the various hardware signals must be assigned to the MCU's GPIO with consideration for the device's peripherals such as timers and I^2C buses. STMIcroelectronics provides a program to this end called STM32CubeMX, shown in Figure 4.1. Within the program, the user selects which MCU to configure and a graphical representation of the chip is shown on the GUI. The MCU's GPIO pins are arranged into banks of up to 16 pins each called ports. For example, PC2 is the second pin in Port C while PB11 is Pin 11 in Port B. The left column of the GUI lists the device peripherals including analog-to-digital converters (ADCs); various buses such as SPI, USART, and USB; and timers. Peripheral modes can be selected here such as USART mode (asynchronous, single-wire, LIN, etc) and flow control as well as timer clock sourcing and channel modes.

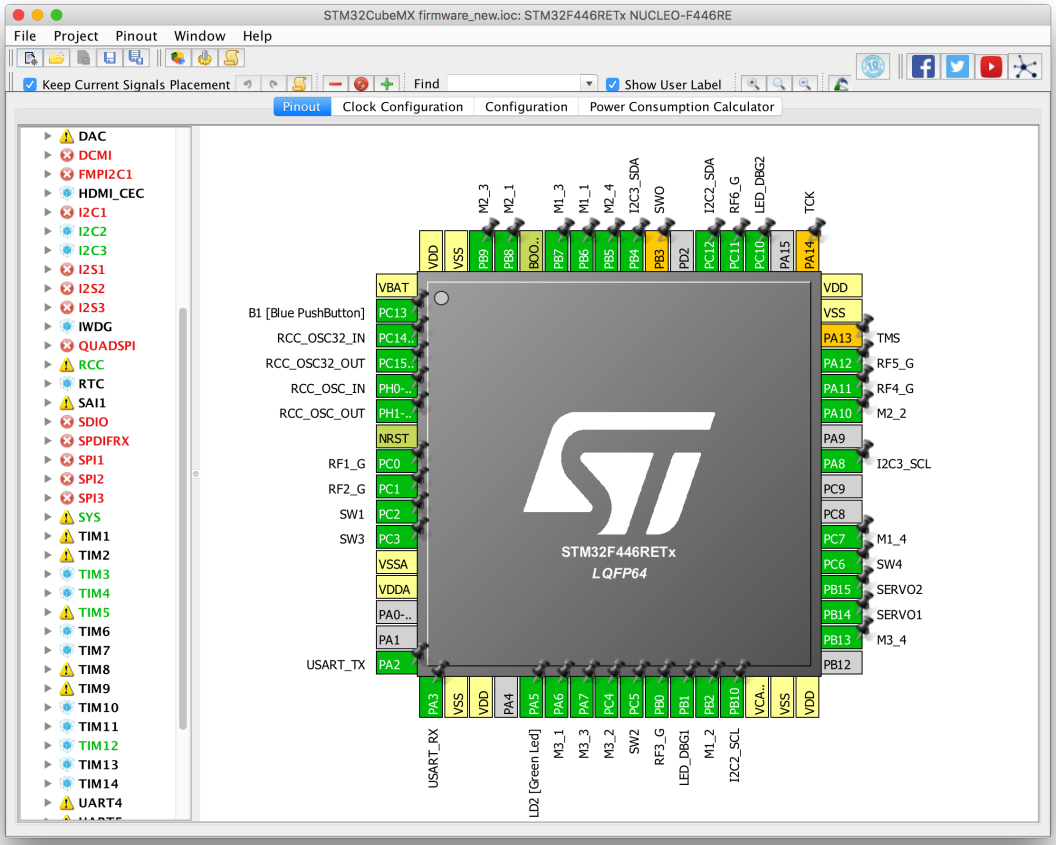


Figure 4.1: STM32CubeMX

The program greatly simplifies the puzzle-like process of pin assignment. Clicking a pin on the GUI brings up a menu of possible assignments. For example, clicking PB15, seen in Figure 4.2, shows that the pin can be used for the ADC, I2S2, as SPI2's MOSI, TIMER12's CH2 output, part of the USB differential data pair, as standard GPIO input or output, and more. As pins are assigned to various purposes, the true benefit of STM32CubeMX becomes clear. The program continually checks for compatibility issues and pin conflicts so pins can be iteratively assigned until errors are eliminated. For example, using a particular set of timers may actually prevent the use of I2C1 so either I2C2 or I2C3 must be used instead. This process is much faster and less error-prone than searching through the MCU's immense datasheet to

manually check for assignment conflicts within every peripheral.

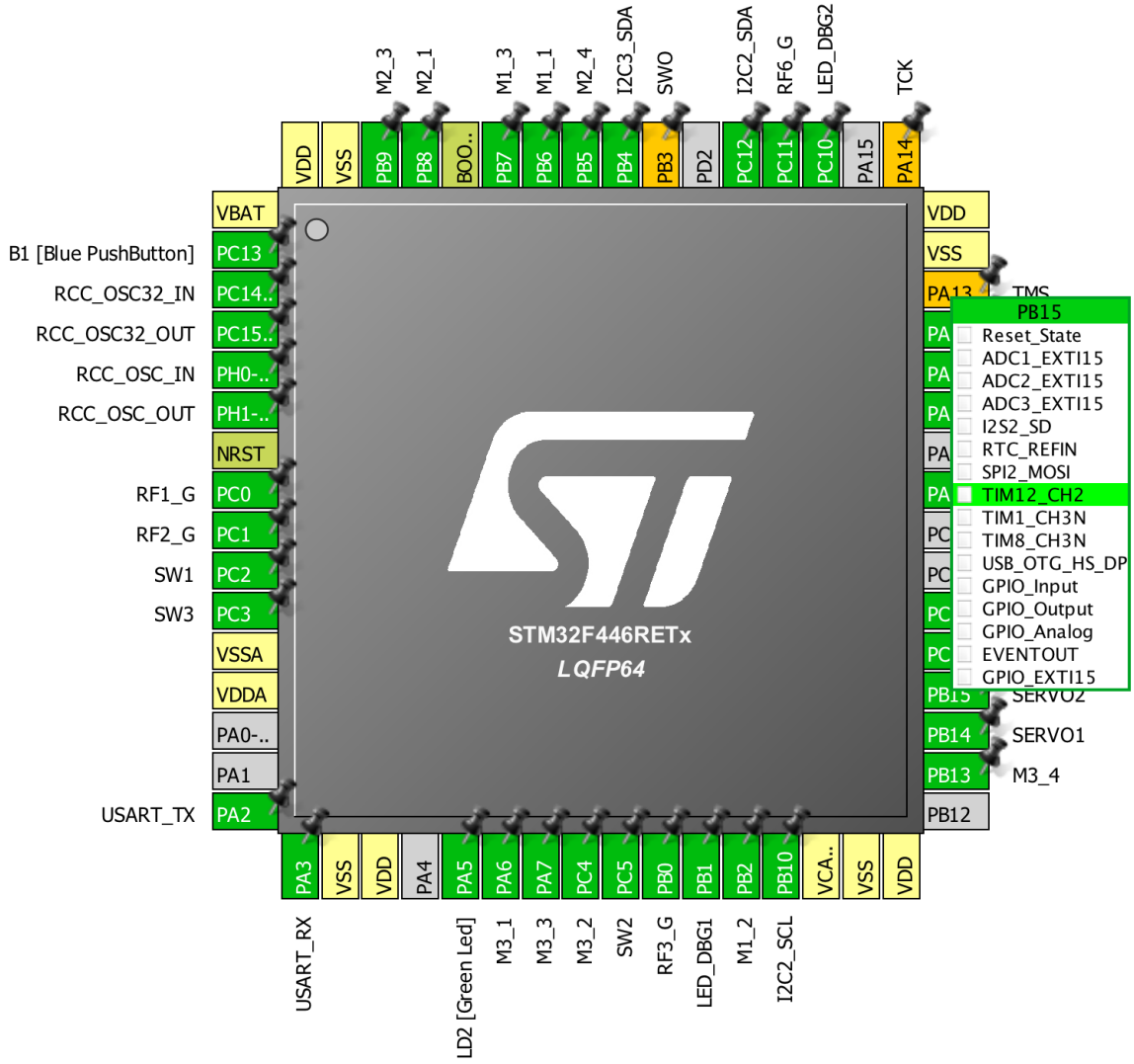


Figure 4.2: STM32CubeMX – Pin Menu

The software also provides device-wide clock and PLL configuration as shown in Figure 4.3. The MCU uses an external 8 MHz crystal to drive the internal PLL which then generates a 180 MHz system clock. Since the system is optimized for performance instead of power-saving, all advanced peripheral bus (APB) clocks are run at max frequency of 45 MHz for APB1 and 90 MHz for APB2.

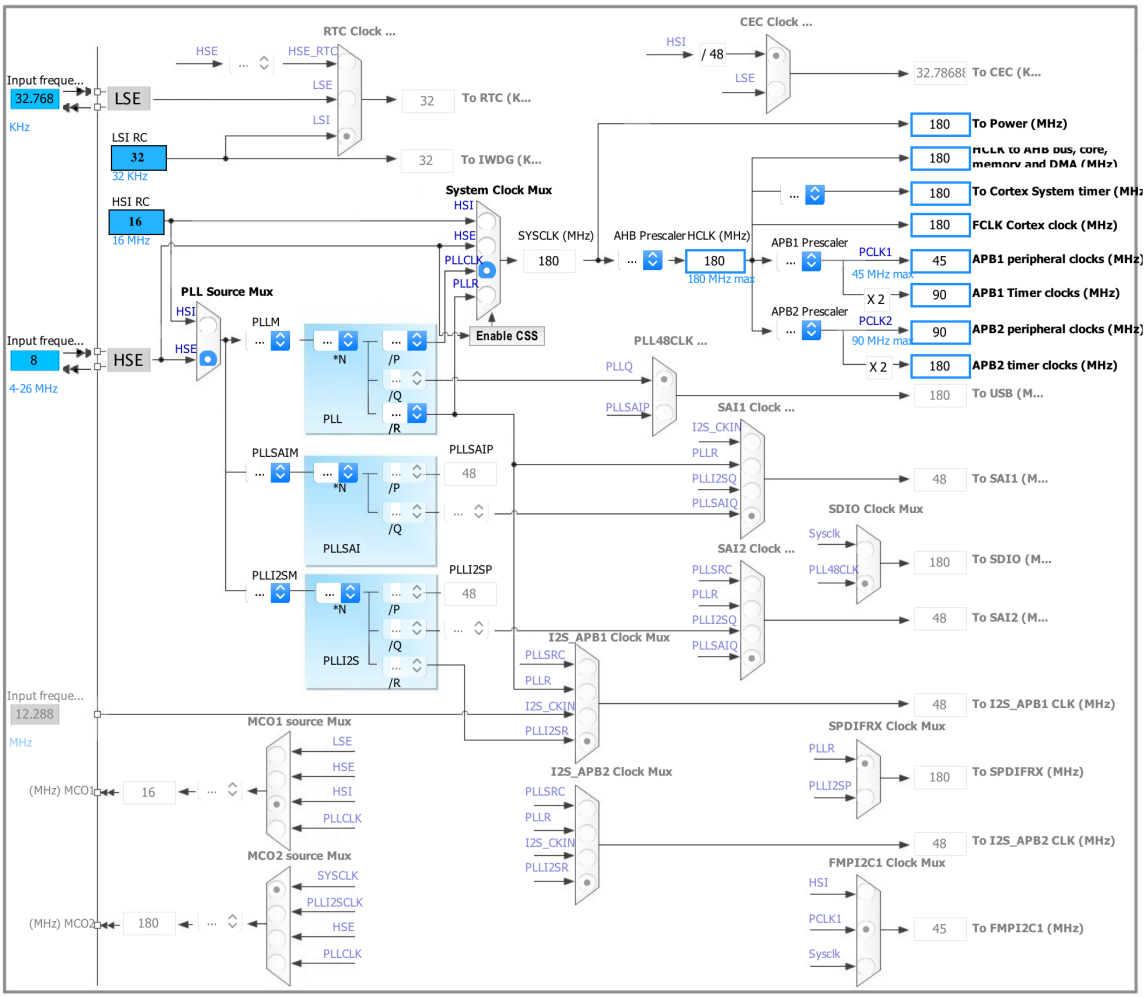


Figure 4.3: STM32CubeMX – Clock Configurator

Another tab within STM32CubeMX provides detailed peripheral configuration, shown in Figure 4.4. Both I^{2}C buses are set to 400 kHz with a 2:1 T_{low}:T_{high} duty cycle to meet the minimum pulse width requirement of slave devices. The USART peripheral is configured to 921,600 bits/s, 8-bit words, no parity, and one stop bit. The direct memory access (DMA) controller is set to process requests from the I^{2}C and UART peripherals to reduce processor load.

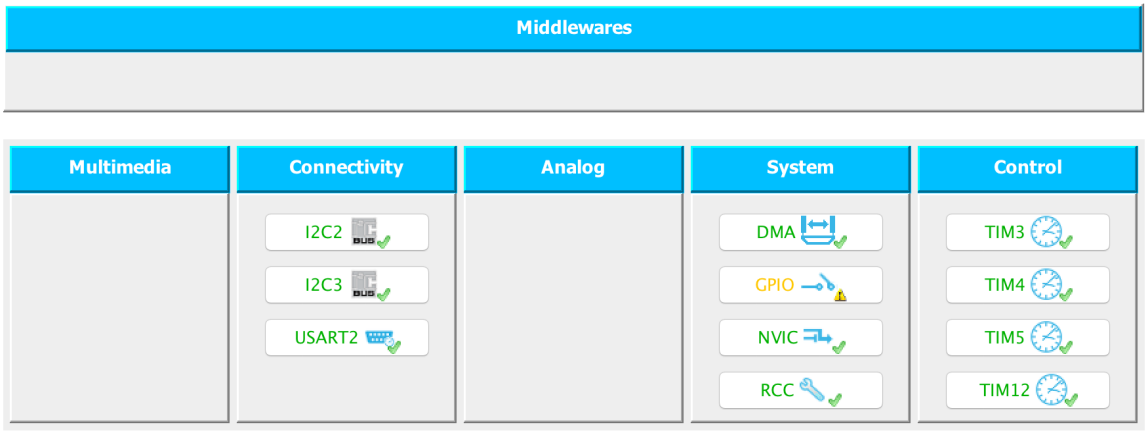


Figure 4.4: STM32CubeMX – Peripheral Configurator

The robot uses four of the MCU's timers: TIMER3, TIMER4, TIMER5, and TIMER12. Each of these timers sources its base clock from the APB1 clock which is 90 MHz. TIMER3 and TIMER4 output 6 PWM control signals for the motor drivers. They are set to count upward and reset after the counter reaches 2047 to produce a 43.9 kHz PWM frequency in the ultrasonic range. TIMER5, used for delay and timing functions, employs a clock prescaler of 9000 so the counter increases every 0.1 ms and never resets; the 32-bit counter has a maximum value of 4,294,967,295 corresponding to a counter rollover period of about 5 days. Finally, TIMER12 drives the PWM signals for servo control. It uses a 45 prescaler and a 50,000 counter period to produce a 40 Hz PWM frequency.

After configuring the various items above, STM32CubeMX can generate common boilerplate plate code to initialize and configure all the devices. Include and source files are generated on a by-peripheral basis to compartmentalize code. The program also generates a report of the device configuration, attached in Appendix D.

4.2 Microcontroller Firmware

The firmware, written in C and compiled with gcc, consists of a main file in conjunction with peripheral driver files. The dependency graph is shown in Figure 4.5 with file descriptions shown in Table 4.1. Several files were automatically generated by STM32CubeMX as well as the VL53L0X API, provided by STMicroelectronics, were modified for the specific application. Table 4.2 lists the firmware specifications.

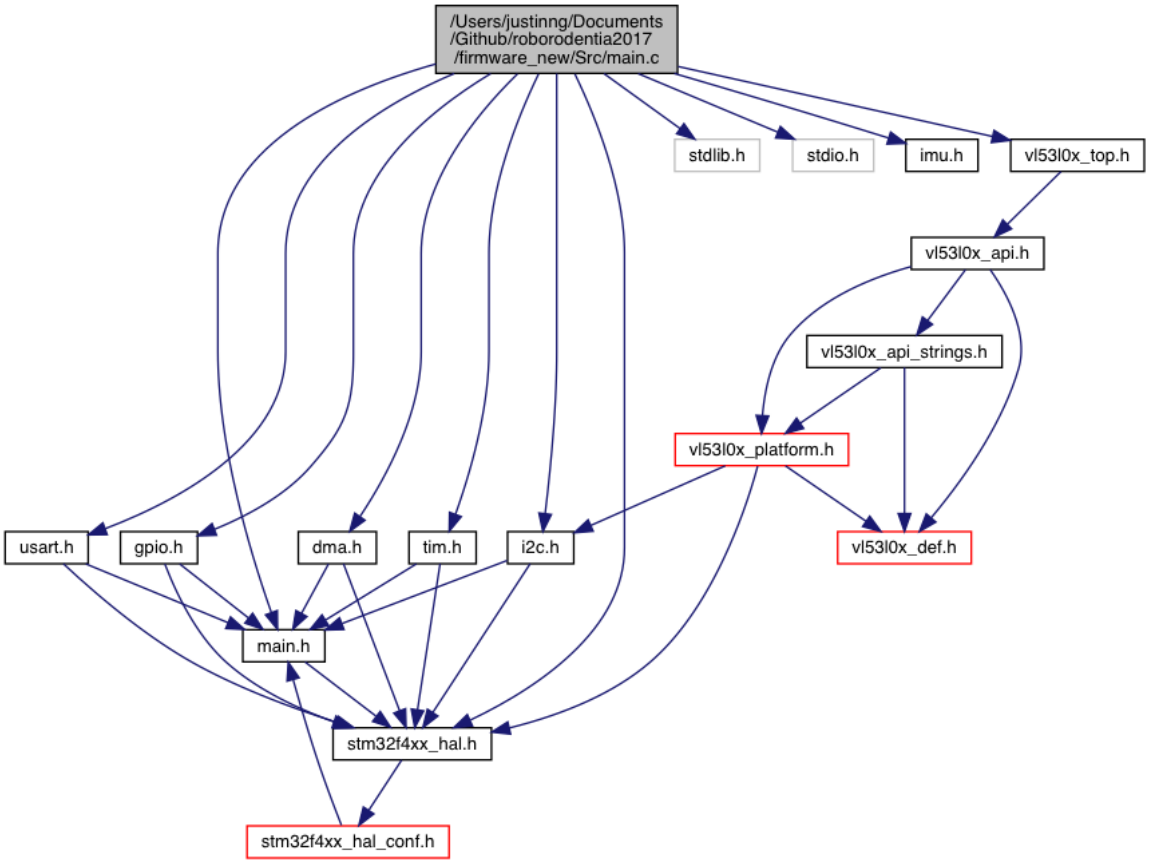


Figure 4.5: Firmware Dependency Graph for main.c

Table 4.1: Firmware – Source File Summary

File (.h/.c)	Description
main	Initialization and main loop.
stm32f4xx_hal	Hardware abstraction layer driver.
dma	Sets up DMA priorities and enables DMA interrupts.
i2c	Initializes I^2C buses, read/write, finds active devices.
tim	Initializes timers 3, 4, 5, and 12.
usart	Initializes UART, functions to write to bus, DMA callbacks.
gpio	Configures GPIO modes (in/out), speeds, and pull-ups/pull-downs.
vl53l0x_top	STMicroelectronics API for configuring and reading rangefinders.
imu	Initializes IMU; read gyroscope, accelerometer, and magnetometer.

Table 4.2: Firmware – General Specifications

	Specification
1	Initialize I^2C sensors including rangefinders and IMU.
2	Read and store data from sensors.
3	Filter data from sensors to reduce noise.
4	Generate PWM signal to drive servo.
5	Generate control signals to motor drivers.
6	Sequence launcher motors, blower fan, and servo to fire balls.
7	Flash LED to indicate error state.
8	Communicate with computer over UART link. Add'l specifications in 4.3.

4.2.1 UART Commands

The MCU responds to UART commands from a computer. The command syntax generally involves a sequence of one or more 8-bit characters. Table 4.3 lists the

available commands.

Table 4.3: Firmware – UART Commands

Syntax	Command
A	Start reading all rangefinders.
B	Returns sensor data in binary form (for speed).
D	Return sensor data in text form (human-readable).
I<BUS>	Scans I^2C bus and returns addresses of active devices. <BUS>valid values: 2 : Scan I2C2. 3 : Scan I2C3.
L<SIDE>	Launch balls from one side of the hopper. <SIDE>valid values: L : Launch balls from left side. R : Launch balls from right side. S : Stop all motors.
M<FL><BL> <FR>	Sets motor duty cycle calculated as <arg>divided by 2047. Each argument accepts a integer from 0 to 2047. <FL>: PWM value for front left motor. <BL>: PWM value for back left motor. : PWM value for back right motor. <FR>: PWM value for front right motor.
V	Returns firmware version info.
Z	Return debug button pressed flag.

UART Receiving

After receiving a character on the UART bus, the DMA places it into `rxBuffer` and fires an interrupt. The MCU enters the UART receive complete DMA callback, `HAL_UART_RxCpltCallback()` which collects the received characters into a command string stored in `stringBuffer`. If the UART receives a new line or line return character, the callback appends a null character to the string and calls `consoleCommand()` to parse and execute the command. The define `UART_RX_WRITEBACK` controls whether received characters and commands are echoed.

```
#define RX_BUFFER_MAX_LENGTH 32
#define UART_RX_WRITEBACK 0

uint8_t rxBuffer = '\000';
uint8_t stringBuffer[RX_BUFFER_MAX_LENGTH];
uint8_t stringBufferIndex;

void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
    uint8_t i;

    // Clear buffer
    if (stringBufferIndex == 0) {for (i = 0; i < RX_BUFFER_MAX_LENGTH; i++) stringBuffer[i] = 0;}

    if ((rxBuffer != 10) && (rxBuffer != 13))   //if received data different from ascii 13 (enter)
    {
        if (stringBufferIndex < RX_BUFFER_MAX_LENGTH){
            stringBuffer[stringBufferIndex++] = rxBuffer; //add data to stringBuffer
        }
    }
    else // If received data = 10 or 13
    {
        if (stringBufferIndex < RX_BUFFER_MAX_LENGTH){
            stringBuffer[stringBufferIndex++] = '\0'; //add null char
        } else {
            stringBuffer[RX_BUFFER_MAX_LENGTH] = '\0'; //add null char
        }

        if (UART_RX_WRITEBACK){
            HAL_UART_Transmit(&huart2, (uint8_t *)&stringBuffer, stringBufferIndex, 0xFFFF);
            printf("\r\n");
        }
        consoleCommand((uint8_t *)&stringBuffer, stringBufferIndex);
        stringBufferIndex = 0;
    }

    if (UART_RX_WRITEBACK){
        HAL_UART_Transmit(&huart2, (uint8_t *)&rxBuffer, 1, 1);
    }
}
```

UART Transmitting

The `_write()` function is overridden to redirect `printf()` output to UART using the code below.

```
// Redirect printf to UART
int _write (int fd, char *ptr, int len)
```

```
{
    transmitUART(ptr, len);
    return len;
}
```

To avoid spending CPU cycles waiting for UART transmission to complete, the DMA buffers the character array directly to the peripheral. A flag, `txInProg`, prevents subsequent transmission until the UART transmit DMA callback, `HAL_UART_TxCpltCallback()`, clears the flag when the current transmission completes.

```
#define TX_BUFFER_MAX_LENGTH 2000

uint8_t txBuffer[TX_BUFFER_MAX_LENGTH];
volatile uint8_t txInProg = 0;

// Transmit characters over UART
void transmitUART(char *ptr, int len)
{
    // Check that the input isn't longer than our buffer
    if (len > TX_BUFFER_MAX_LENGTH){
        _Error_Handler(__FILE__, __LINE__);
    }
    // Wait until UART TX is finished
    while(txInProg == 1){}
    txInProg = 1;
    // Transfer the data to be sent into the txBuffer
    memcpy(txBuffer,(uint8_t *)ptr,len);
    HAL_UART_Transmit_DMA(&huart2, txBuffer, len);
}

void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart)
{
    // Clear TX in progress flag when complete with transfer
    txInProg= 0;
}
```

4.2.2 MCU Execution Flow

The execution flow is as follows:

1. Define and initialize a struct for motor pins and timer channels.

```
struct motor_t {
    GPIO_TypeDef *GPIOx;
    uint16_t GPIO_Pin;
    GPIO_PinState PinState;

    TIM_HandleTypeDef *TIM_Handle;
    TIM_OC_InitTypeDef sConfigOC;
    uint32_t TIM_Channel;
};

struct motor_t motorConfigs[4] = {
    {MOTOR_FL_DIR_GPIO_Port, MOTOR_FL_DIR_Pin, GPIO_PIN_RESET, 0, {0}, TIM_CHANNEL_2},
    {MOTOR_FR_DIR_GPIO_Port, MOTOR_FR_DIR_Pin, GPIO_PIN_RESET, 0, {0}, TIM_CHANNEL_4},
    {MOTOR_BR_DIR_GPIO_Port, MOTOR_BR_DIR_Pin, GPIO_PIN_RESET, 0, {0}, TIM_CHANNEL_1},
    {MOTOR_BL_DIR_GPIO_Port, MOTOR_BL_DIR_Pin, GPIO_PIN_RESET, 0, {0}, TIM_CHANNEL_3}};
struct motor_t* motorConfigsPtr = motorConfigs;
```

2. Instantiate some variables.

```
volatile uint8_t wd_reset = 0;
unsigned char data_packet[21];
volatile uint8_t sensor_update_req = 0;
volatile uint8_t dbg_btn_latch = 0;
```

3. Enable all clocks since power saving not necessary.

```
RCC->AHB1ENR |= 0xFFFFFFFF;
RCC->AHB2ENR |= 0xFFFFFFFF;
RCC->AHB3ENR |= 0xFFFFFFFF;
RCC->APB1ENR |= 0xFFFFFFFF;
RCC->APB2ENR |= 0xFFFFFFFF;
```

4. Reset peripherals, initialize SysTick, set up PLL and clocks.

```
HAL_Init();
SystemClock_Config();
```

5. Initialize configured peripherals.

```
MX_GPIO_Init();
MX_DMA_Init();
MX_I2C2_Init();
MX_I2C3_Init();
MX_TIM3_Init();
MX_TIM4_Init();
MX_TIM12_Init();
MX_USART2_UART_Init();
MX_TIM5_Init();
```

6. Reset system timer, enable UART receive DMA, initialize sensors.

```
TimeStamp_Reset();
serviceUART();
printf("Enabling IMU...\r\n");
IMU_begin();
printf("Initializing rangefinders...\r\n");
VL53L0X_begin();
```

7. Set servo to starting position.

```
if (HAL_TIM_PWM_Start(&htim12, TIM_CHANNEL_1) != HAL_OK){ Error_Handler(); }
```

8. Initialize variables and begin main loop.

```
uint32_t wd_start = 0;      // Watchdog start time
uint32_t cur_time = 0;      // Current time
uint8_t wd_en = 0;           // Watchdog enable
data_packet[20] = '\n';     // 21 byte sensor data packet to send over UART

while (1)
{
```

9. Watchdog prevents motors from running when UART is inactive. Receiving any UART command sets `wd_reset` to 1.

```
   // Reset watchdog start time if flag is set
   if (wd_reset == 1){
      wd_start = TimeStamp_Get();  // Units of 0.1 ms based on Timer5
      wd_en = 1;
      wd_reset = 0;
   } else if (wd_en == 1){
      // Trigger if watchdog is enabled and expires
      cur_time = TimeStamp_Get();
      if ((cur_time - wd_start) > WD_LEN){
         wd_en = 0;

         // Disable motors
         int i;
         for (i=0; i<4; i++) {
            motorConfigs[i].PinState = GPIO_PIN_RESET;
            motorConfigs[i].sConfigOC.Pulse = 0;
         }
         for (i=0; i<4; i++) {
            // Alter the PWM duty cycle and start PWM again
            if (HAL_TIM_PWM_ConfigChannel(motorConfigs[i].TIM_Handle, &motorConfigs[i].sConfigOC,
               motorConfigs[i].TIM_Channel) != HAL_OK) { Error_Handler(); }
            if (HAL_TIM_PWM_Start(motorConfigs[i].TIM_Handle, motorConfigs[i].TIM_Channel)
               != HAL_OK){ Error_Handler(); }

            // Set the direction pin
            HAL_GPIO_WritePin(motorConfigs[i].GPIOx, motorConfigs[i].GPIO_Pin, motorConfigs[i].PinState);
         }
      }
   }
```

10. Read debug tactile button state and latch it until read by UART.

```
   // Read debug button
   dbg_btn_latch = HAL_GPIO_ReadPin(B1_GPIO_Port, B1_Pin) && dbg_btn_latch;
```

11. If requested over UART, read all four rangefinders and store the data in the `data_packet`.

```
   if (sensor_update_req == 1){
      int i;
      for (i = 0; i < 4; i++){
         if (rangefinderRead(i)){
            data_packet[i*2  ] = (rangeData[i] >> 8) & 0xFF;
            data_packet[i*2+1] =  rangeData[i]       & 0xFF;
         }
      }
      sensor_update_req = 0;
   }
```

12. Read the magnetometer and store the data in the data packet.

```
   magnetometer_read();
   data_packet[ 8] = (magData.x >> 8) & 0xFF;
   data_packet[ 9] =  magData.x       & 0xFF;
   data_packet[10] = (magData.y >> 8) & 0xFF;
   data_packet[11] =  magData.y       & 0xFF;
   data_packet[12] = (magData.z >> 8) & 0xFF;
   data_packet[13] =  magData.z       & 0xFF;
```

13. Read the accelerometer and store the data in the data packet.

```
   accelerometer_read();
   data_packet[14] = (accelData.x >> 8) & 0xFF;
   data_packet[15] =  accelData.x       & 0xFF;
   data_packet[16] = (accelData.y >> 8) & 0xFF;
   data_packet[17] =  accelData.y       & 0xFF;
   data_packet[18] = (accelData.z >> 8) & 0xFF;
   data_packet[19] =  accelData.z       & 0xFF;
}
```

14. Return to start of loop.

4.2.3 Error Handler

During errors or faults, the MCU enters the error handler, prints an error message, and flashes the debug LED at 5 Hz.

```
1  void _Error_Handler(char * file, int line)
2  {
3     printf("Entered error handler.\r\n");
4     while(1)
5     {
6        HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_1);
7        HAL_Delay(100);
8     }
9  }
```

Chapter 5

SOFTWARE DESIGN

5.1 Definitions and Assumptions

The robot is designed to only travel within a rectangular closed area of eight feet in the x direction and five feet in the y direction. The coordinate system is chosen as Cartesian with the origin placed at the bottom-left corner of the field. The position of the robot is always in the xy-plane since it cannot move vertically (z = 0). Therefore, x refers to the robot position along the x-axis and ranges from 0 to 8 feet, and y refers to position along the y-axis and ranges from 0 to 5 feet. Additionally, the robot can only rotate around the z-axis so θ refers to the angle of the robot in the xy-plane. Maintaining standard Cartesian coordinates, $\theta = 0^\circ$ is along the positive x direction while $\theta = 90^\circ$ is along the positive y direction.

5.2 Kalman Filters

The system relies on several different sensors to determine where it is within the environment, a problem commonly referred to as robot localization. The *Kalman Filter* (KF), an optimal state estimator, performs noise filtering and sensor fusion, the process of combining measurements from multiple sensors. The filter operates on the principles of Bayesian inference and uses statistically noisy measurements over time and knowledge of the system to produce a more accurate estimate of an unknown variable than with measurement alone.

5.2.1 Algorithm

The Kalman Filter algorithm and equations are reproduced here from Roger Labbe's excellent interactive online book [6]. The algorithm consists of two stages (not including initialization): prediction and update. During the first stage, the filter uses the current state and a process model (typically a function of time) to estimate the state in the next time step along with its uncertainty. The second stage uses sensor measurements to update the estimation by taking a weighted average based on the ratio of uncertainty between the prediction and measurement.

Initialization

Before the first run of the filter, initialize the estimated state ($\mathbf{x}$, also called the posterior) and estimated state covariance matrix ($\mathbf{P}$).

Predict

During the predict phase, the process model is used to predict the future state (known as the prior) ($\bar{\mathbf{x}}$) after one time step by summing the posterior ($\mathbf{x}$) multiplied by the *state transition function* ($\mathbf{F}$) with the control input model ($\mathbf{B}$) multiplied by the control input ($\mathbf{u}$). The covariance of prior ($\bar{\mathbf{P}}$) is larger than the posterior covariance ($\mathbf{P}$) due to uncertainty in the process model ($\mathbf{Q}$).

$$\bar{\mathbf{x}} = \mathbf{F}\mathbf{x} + \mathbf{B}\mathbf{u}$$

$$\bar{\mathbf{P}} = \mathbf{F}\mathbf{P}\mathbf{F}^T + \mathbf{Q}$$

Update

Make measurements ($\mathbf{z}$, measurement mean) and determine their accuracy ($\mathbf{R}$, measurement noise covariance). Calculate the residual (or difference) ($\mathbf{y}$) between the measurement and the product of the measurement function ($\mathbf{H}$) and the prior from the previous phase. $\mathbf{H}$ converts the prior from the state space to the measurement space. Calculate the weighting factor ($\mathbf{K}$, Kalman gain), valued between 0 and 1, based on the whether the measurement or prior is more accurate. Set the new posterior, $\mathbf{x}$, to an average of the measurement and prior, weighted by $\mathbf{K}$. Finally, update the posterior's covariance, $\mathbf{P}$, based on the measurement certainty. The algorithm then loops back to the predict phase using the newly-calculated posterior.

$$\mathbf{y} = \mathbf{z} - \mathbf{H}\bar{\mathbf{x}}$$

$$\mathbf{K} = \bar{\mathbf{P}}\mathbf{H}^T(\mathbf{H}\bar{\mathbf{P}}\mathbf{H}^T + \mathbf{R})^{-1}$$

$$\mathbf{x} = \bar{\mathbf{x}} + \mathbf{K}\mathbf{y}$$

$$\mathbf{P} = (\mathbf{I} - \mathbf{K}\mathbf{H})\bar{\mathbf{P}}$$

5.2.2 Design

The desired state variable $\mathbf{x}$ is chosen as the linear position, velocity, and acceleration in the x and y directions as well as the angular position, velocity, and acceleration about the z-axis:

$$\mathbf{x} = [x \ y \ \theta]^T$$

$$\dot{\mathbf{x}} = [\dot{x} \ \dot{y} \ \dot{\theta}]^T$$

$$\ddot{\mathbf{x}} = [\ddot{x} \ \ddot{y} \ \ddot{\theta}]^T$$

The process model for position and velocity:

$$\begin{cases} \bar{x} = x + \dot{x}\Delta t + 0.5\ddot{x}(\Delta t)^2 \\ \bar{\dot{x}} = \dot{x} + \ddot{x}\Delta t \\ \bar{\ddot{x}} = \ddot{x} \end{cases}$$

Which can be written in the form:

$$\begin{bmatrix} \bar{x} \\ \bar{\dot{x}} \\ \bar{\ddot{x}} \end{bmatrix} = \begin{bmatrix} 1 & \Delta t & 0.5(\Delta t)^2 \\ 0 & 1 & \Delta t \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \end{bmatrix}$$

$$\bar{\mathbf{x}} = \mathbf{F}\mathbf{x}$$

The measurement vector is chosen as:

$$\mathbf{z} = \begin{bmatrix} z_x & z_{\ddot{x}} & z_y & z_{\ddot{y}} & z_\theta & z_{\dot{\theta}} \end{bmatrix}^T$$

The measurement noise matrix is shown below. The off-diagonals are 0 because the noise between sensors is assumed to be uncorrelated.

$$\mathbf{R} = \begin{bmatrix} \sigma_x^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma_{\ddot{x}}^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma_y^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & \sigma_{\ddot{y}}^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sigma_\theta^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & \sigma_{\dot{\theta}}^2 \end{bmatrix}$$

Chapter 6

NEURAL NETWORK DESIGN

Chapter 7

CONCLUSION

BIBLIOGRAPHY

[1] . . , 2018. [Online; accessed May 11, 2018].

[2] Adafruit. Adafruit 9-DOF IMU Breakout. https://www.adafruit.com/product/1714, 2014. [Online; accessed May 15, 2018].

[3] AliExpress. DC-DC CC CV Buck Converter. https://www.aliexpress.com/item/DC-DC-CC-CV-Buck-Converter-Volt-Step-Down-12V-19V-24V-Car-Laptop-Power-Supply/32822603345.html?spm=2114.search0104.3.135.191f73dfqonWDU&ws_ab_test=searchweb0_0,searchweb201602_2_10152_10151_10065_10344_10130_10068_10324_10547_10342_10325_10546_10343_10340_10548_10341_10545_10696_10084_10083_10618_10307_10059_308_100031_10103_10624_10623_10622_10621_10620,searchweb201603_32,ppcSwitch_5&algo_expid=30a33cc9-3acf-4021-b514-6ba550085dd9-19&algo_pvid=30a33cc9-3acf-4021-b514-6ba550085dd9&priceBeautifyAB=0, 2018. [Online; accessed May 11, 2018].

[4] Cal Poly. Roborodentia - Cal Poly, San Luis Obispo. http://www.myurl.com, 2018. [Online; accessed May 11, 2018].

[5] Fabio Varesano. FreeIMU Magnetometer and Accelerometer Calibration GUI. http://www.varesano.net/blog/fabio/freeimu-magnetometer-and-accelerometer-calibration-gui-alpha-version-out, 2012. [Online; accessed May 11, 2018].

[6] R. R. Labbe. Kalman and Bayesian Filters in Python, Sep 2017.

[7] Leopoldo Armesto. How to generate a PPM signal with Arduino to control a

servo? http://robotica.webs.upv.es/en/how-to-generate-a-ppm-signal-with-arduino-to-control-a-servo/, 2015. [Online; accessed May 11, 2018].

[8] MechaMan. Working with L298N DC Motor Driver. http://fritzing.org/projects/working-with-l298n-dc-motor-driver, 2018. [Online; accessed May 13, 2018].

[9] R. Negenborn. *Robot Localization and Kalman Filters*. PhD thesis, Sep 2003.

[10] Nevada Mark. FAQ: Hard & Soft Iron Correction for Magnetometer Measurements. https://ez.analog.com/docs/DOC-2544, 2014. [Online; accessed May 11, 2018].

[11] D. H. Nguyen and B. Widrow. Neural Networks for Self-Learning Control Systems. *IEEE Control Systems Magazine*, Apr 1990.

[12] STMicroelectronics. LSM303DLHC Datasheet. http://www.st.com/resource/en/datasheet/DM00027543.pdf, 2013. [Online; accessed May 15, 2018].

[13] STMicroelectronics. STM32 Nucleo-64 User Manual. http://www.st.com/content/ccc/resource/technical/document/user_manual/98/2e/fa/4b/e0/82/43/b7/DM00105823.pdf/files/DM00105823.pdf/jcr:content/translations/en.DM00105823.pdf, 2017. [Online; accessed May 11, 2018].

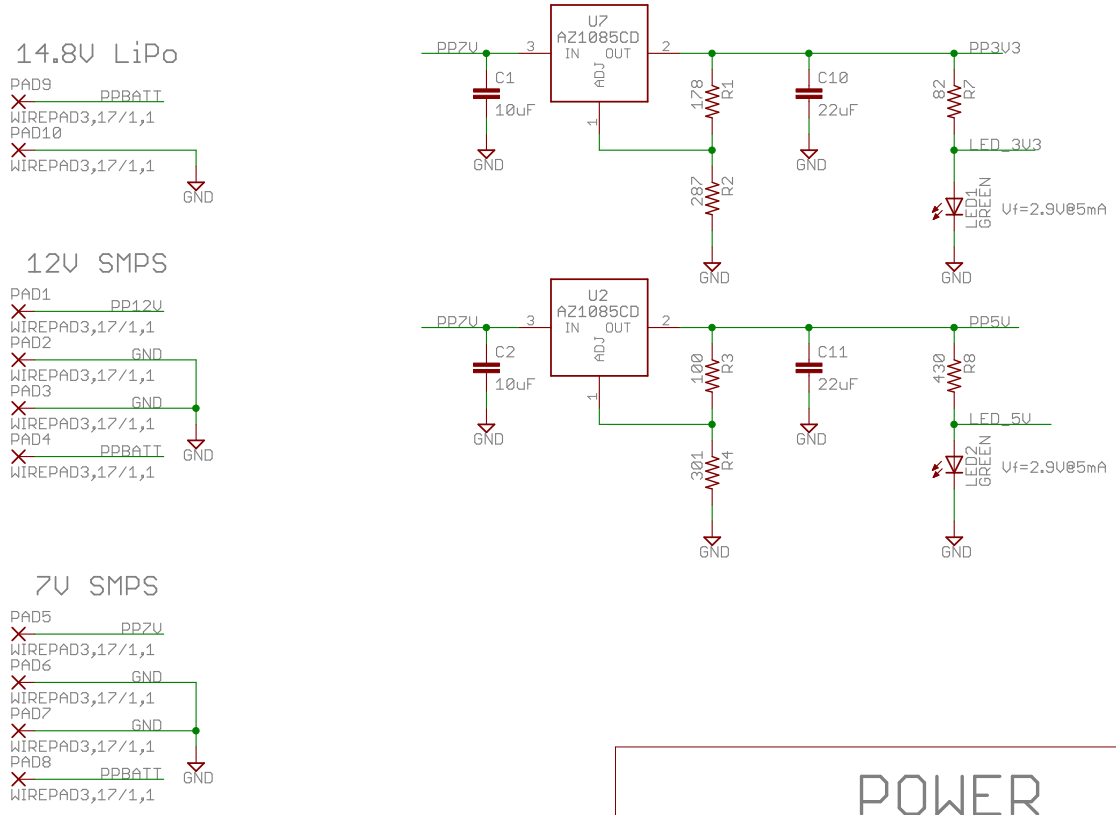
[14] STMicroelectronics. VL53L0X. http://www.st.com/en/imaging-and-photonics-solutions/vl53l0x.html, 2018. [Online; accessed May 15, 2018].

[15] STMicroelectronics. VL53L0X Datasheet.
http://www.st.com/content/ccc/resource/technical/document/datasheet/group3/b2/1e/33/77/c6/92/47/6b/DM00279086/files/DM00279086.pdf/jcr:content/translations/en.DM00279086.pdf, 2018. [Online; accessed May 12, 2018].

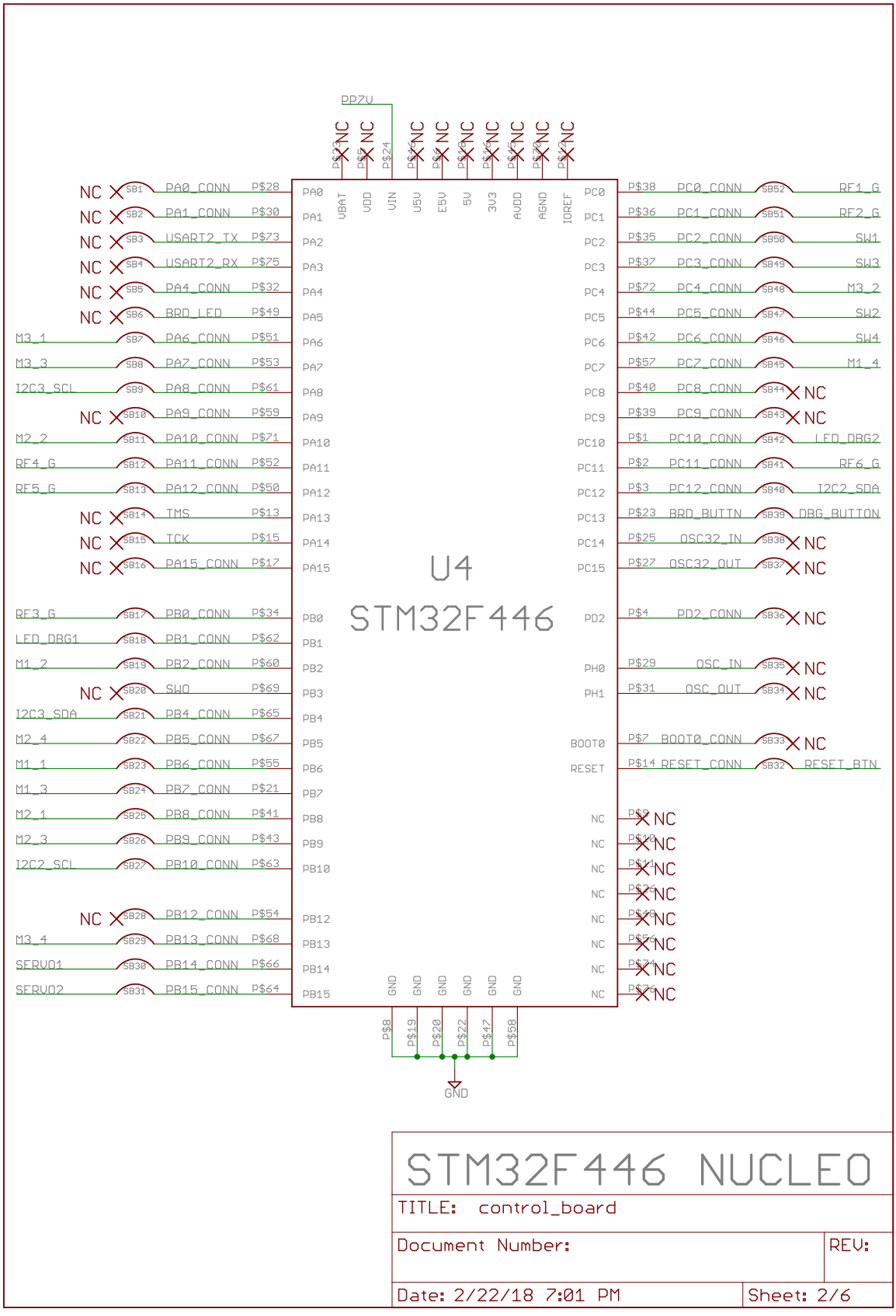
APPENDICES

Appendix A

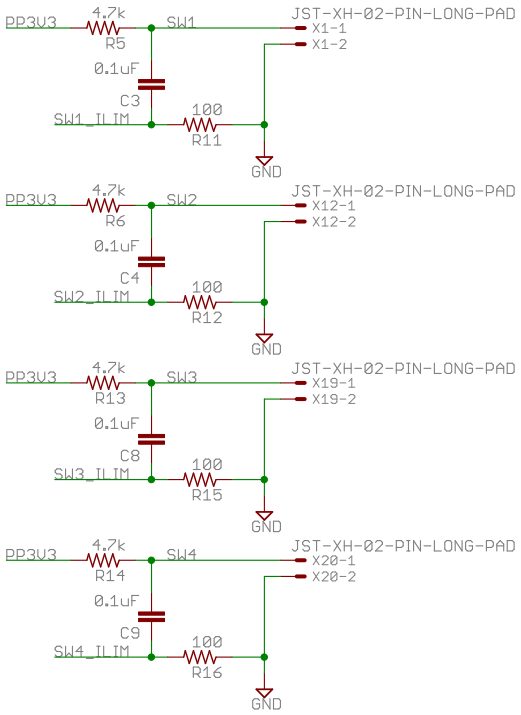
INTERCONNECT PCB SCHEMATIC



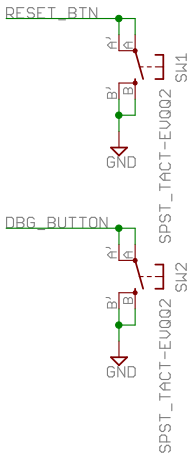
POWER	
TITLE: control_board	
Document Number:	REV:
Date: 2/22/18 7:01 PM	Sheet: 1/6


U4
STM32F446
PA0 PA1 PA2 PA3 PA4 PA5 PA6 PA7 PA8 PA9 PA10 PA11 PA12 PA13 PA14 PA15
PB0 PB1 PB2 PB3 PB4 PB5 PB6 PB7 PB8 PB9 PB10 PB12 PB13 PB14 PB15
PC0 PC1 PC2 PC3 PC4 PC5 PC6 PC7 PC8 PC9 PC10 PC11 PC12 PC13 PC14 PC15
PD2
PH0 PH1
BOOT0
RESET
VBAT VDD VIN U5V E5V 5V 3V3 AVDD AGND IOREF
GND
PA0_CONN PA1_CONN USART2_TX USART2_RX PA4_CONN BRD_LED PA6_CONN PA7_CONN PA8_CONN PA9_CONN PA10_CONN PA11_CONN PA12_CONN TMS TCK PA15_CONN
PB0_CONN PB1_CONN PB2_CONN SWO PB4_CONN PB5_CONN PB6_CONN PB7_CONN PB8_CONN PB9_CONN PB10_CONN PB12_CONN PB13_CONN PB14_CONN PB15_CONN
PC0_CONN PC1_CONN PC2_CONN PC3_CONN PC4_CONN PC5_CONN PC6_CONN PC7_CONN PC8_CONN PC9_CONN PC10_CONN PC11_CONN PC12_CONN BRD_BUTTN OSC32_IN OSC32_OUT
PD2_CONN
OSC_IN OSC_OUT
BOOT0_CONN
RESET_CONN
M3_1 M3_3 I2C3_SCL M2_2 RF4_G RF5_G M3_4 SERVO1 SERVO2 RF3_G LED_DBG1 M1_2 I2C3_SDA M2_4 M1_1 M1_3 M2_1 M2_3 I2C2_SCL
RF1_G RF2_G SW1 SW3 M3_2 SW2 SW4 M1_4 LED_DBG2 RF6_G I2C2_SDA DBG_BUTTON RESET_BTN
STM32F446 NUCLEO
TITLE: control_board
Document Number:
REV:
Date: 2/22/18 7:01 PM
Sheet: 2/6

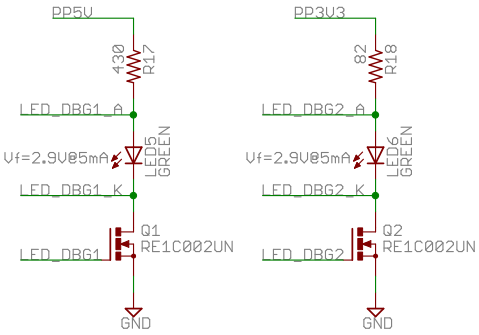
Microswitches



Debug Buttons

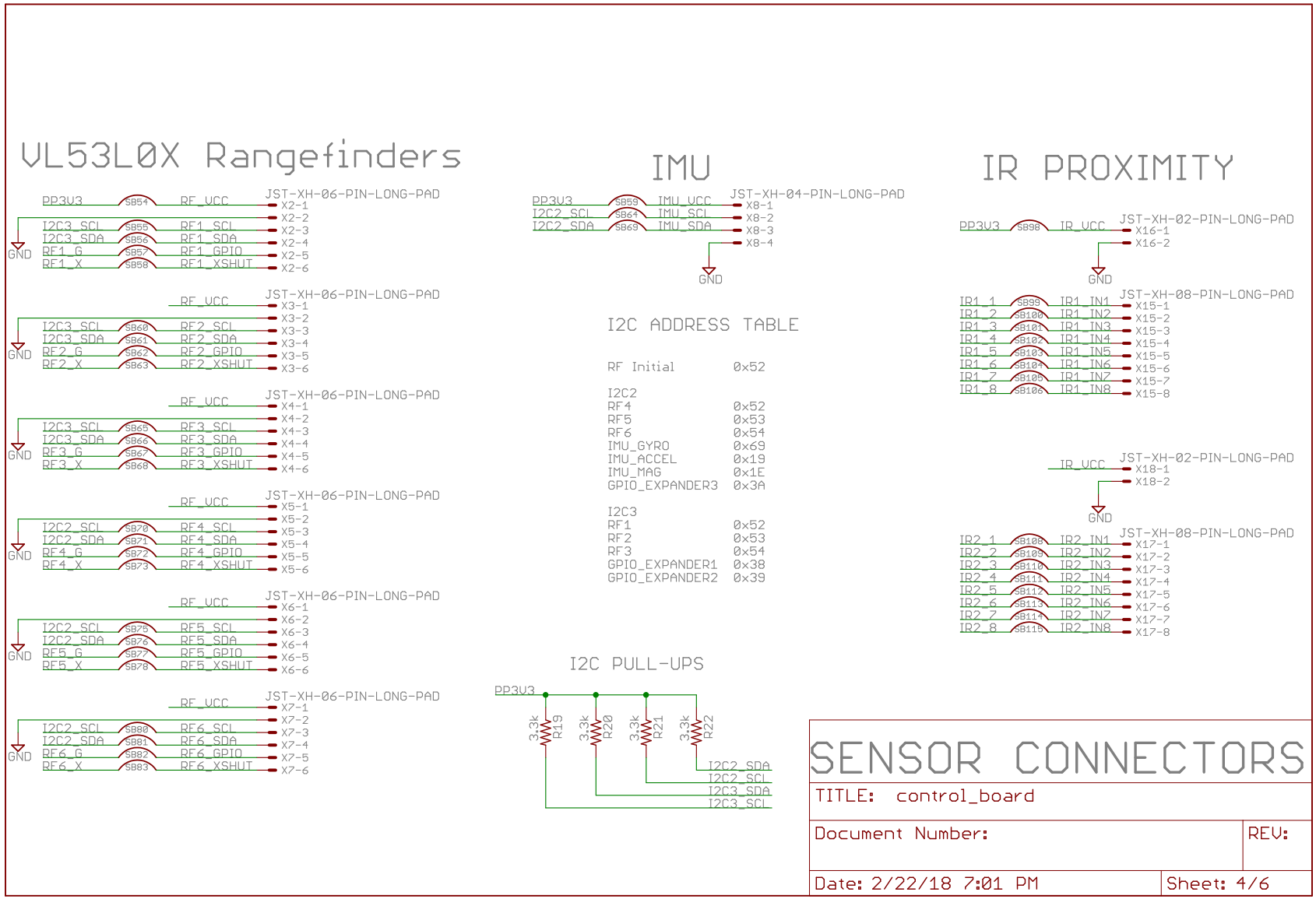


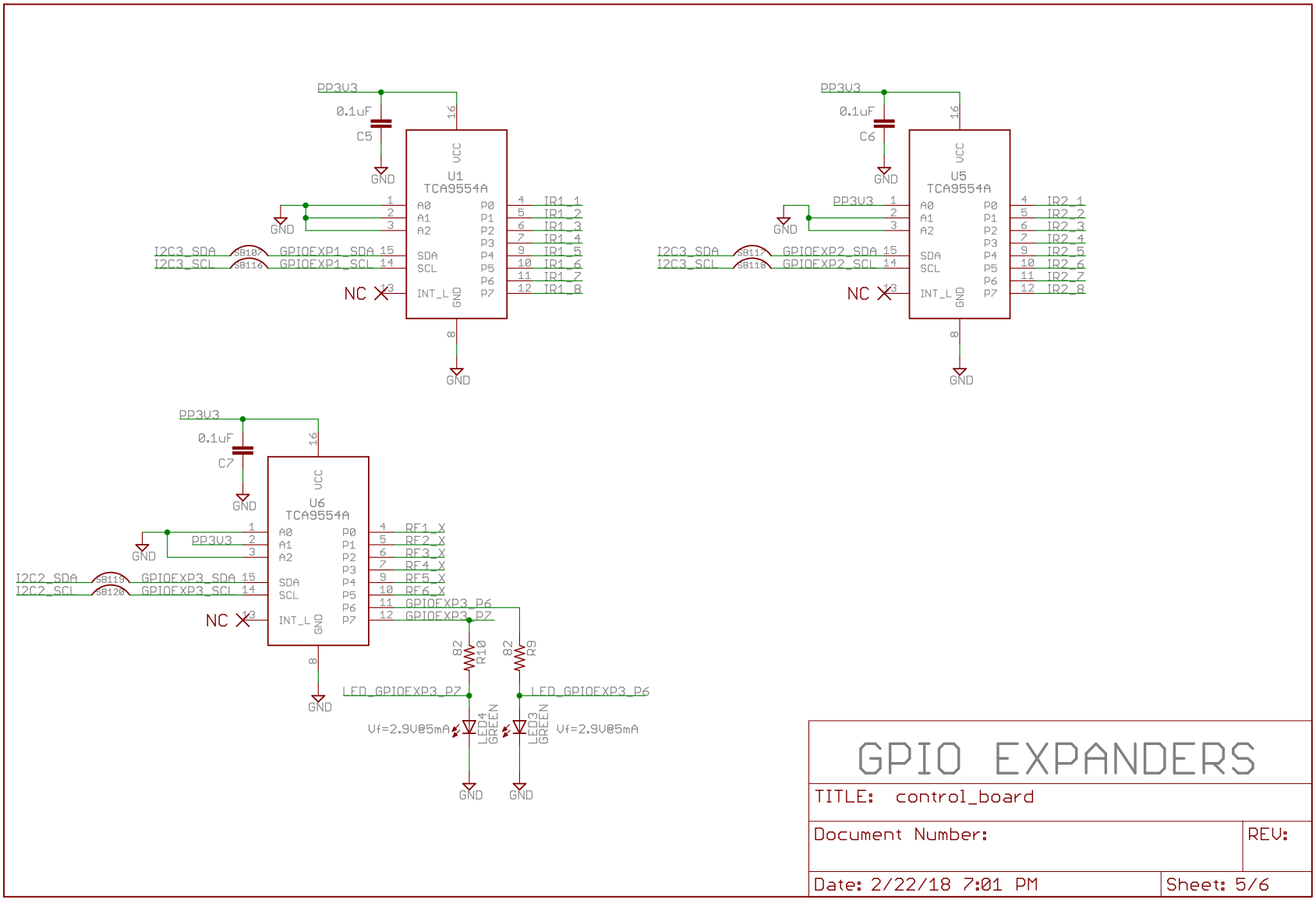
Debug LEDs



Switches & LEDs

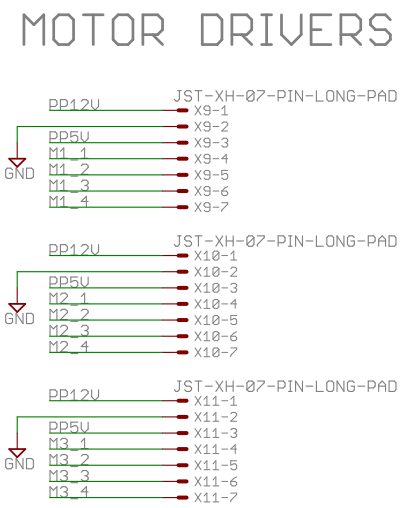
TITLE: control_board	
Document Number:	REV:
Date: 2/22/18 7:01 PM	Sheet: 3/6

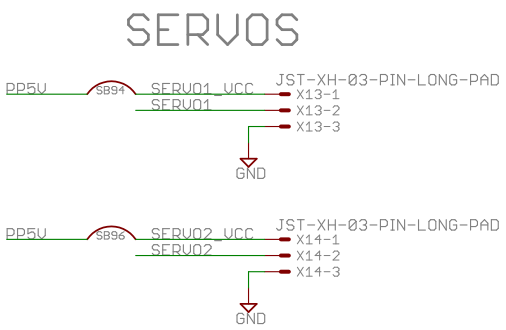

VL53L0X Rangefinders
PP3V3
SB54
RF_VCC
JST-XH-06-PIN-LONG-PAD
X2-1
X2-2
I2C3_SCL
SB55
RF1_SCL
X2-3
I2C3_SDA
SB56
RF1_SDA
X2-4
GND
RF1_G
SB57
RF1_GPIO
X2-5
RF1_X
SB58
RF1_XSHUT
X2-6
RF_VCC
JST-XH-06-PIN-LONG-PAD
X3-1
X3-2
I2C3_SCL
SB60
RF2_SCL
X3-3
I2C3_SDA
SB61
RF2_SDA
X3-4
GND
RF2_G
SB62
RF2_GPIO
X3-5
RF2_X
SB63
RF2_XSHUT
X3-6
RF_VCC
JST-XH-06-PIN-LONG-PAD
X4-1
X4-2
I2C3_SCL
SB65
RF3_SCL
X4-3
I2C3_SDA
SB66
RF3_SDA
X4-4
GND
RF3_G
SB67
RF3_GPIO
X4-5
RF3_X
SB68
RF3_XSHUT
X4-6
RF_VCC
JST-XH-06-PIN-LONG-PAD
X5-1
X5-2
I2C2_SCL
SB70
RF4_SCL
X5-3
I2C2_SDA
SB71
RF4_SDA
X5-4
GND
RF4_G
SB72
RF4_GPIO
X5-5
RF4_X
SB73
RF4_XSHUT
X5-6
RF_VCC
JST-XH-06-PIN-LONG-PAD
X6-1
X6-2
I2C2_SCL
SB75
RF5_SCL
X6-3
I2C2_SDA
SB76
RF5_SDA
X6-4
GND
RF5_G
SB77
RF5_GPIO
X6-5
RF5_X
SB78
RF5_XSHUT
X6-6
RF_VCC
JST-XH-06-PIN-LONG-PAD
X7-1
X7-2
I2C2_SCL
SB80
RF6_SCL
X7-3
I2C2_SDA
SB81
RF6_SDA
X7-4
GND
RF6_G
SB82
RF6_GPIO
X7-5
RF6_X
SB83
RF6_XSHUT
X7-6
IMU
PP3V3
SB59
IMU_VCC
JST-XH-04-PIN-LONG-PAD
X8-1
I2C2_SCL
SB64
IMU_SCL
X8-2
I2C2_SDA
SB69
IMU_SDA
X8-3
X8-4
GND
I2C ADDRESS TABLE
RF Initial 0x52
I2C2
RF4 0x52
RF5 0x53
RF6 0x54
IMU_GYRO 0x69
IMU_ACCEL 0x19
IMU_MAG 0x1E
GPIO_EXPANDER3 0x3A
I2C3
RF1 0x52
RF2 0x53
RF3 0x54
GPIO_EXPANDER1 0x38
GPIO_EXPANDER2 0x39
I2C PULL-UPS
PP3V3
3.3k
R19
3.3k
R20
3.3k
R21
3.3k
R22
I2C2_SDA
I2C2_SCL
I2C3_SDA
I2C3_SCL
IR PROXIMITY
PP3V3
SB98
IR_VCC
JST-XH-02-PIN-LONG-PAD
X16-1
X16-2
GND
IR1_1
SB99
IR1_IN1
JST-XH-08-PIN-LONG-PAD
X15-1
IR1_2
SB100
IR1_IN2
X15-2
IR1_3
SB101
IR1_IN3
X15-3
IR1_4
SB102
IR1_IN4
X15-4
IR1_5
SB103
IR1_IN5
X15-5
IR1_6
SB104
IR1_IN6
X15-6
IR1_7
SB105
IR1_IN7
X15-7
IR1_8
SB106
IR1_IN8
X15-8
IR_VCC
JST-XH-02-PIN-LONG-PAD
X18-1
X18-2
GND
IR2_1
SB108
IR2_IN1
JST-XH-08-PIN-LONG-PAD
X17-1
IR2_2
SB109
IR2_IN2
X17-2
IR2_3
SB110
IR2_IN3
X17-3
IR2_4
SB111
IR2_IN4
X17-4
IR2_5
SB112
IR2_IN5
X17-5
IR2_6
SB113
IR2_IN6
X17-6
IR2_7
SB114
IR2_IN7
X17-7
IR2_8
SB115
IR2_IN8
X17-8
SENSOR CONNECTORS
TITLE: control_board
Document Number:
REV:
Date: 2/22/18 7:01 PM
Sheet: 4/6



GPIO EXPANDERS

TITLE: control_board	
Document Number:	REV:
Date: 2/22/18 7:01 PM	Sheet: 5/6





MOTOR DRIVERS	
TITLE: control_board	
Document Number:	REV:
Date: 2/22/18 7:01 PM	Sheet: 6/6

Appendix B

INTERCONNECT PCB LAYOUT

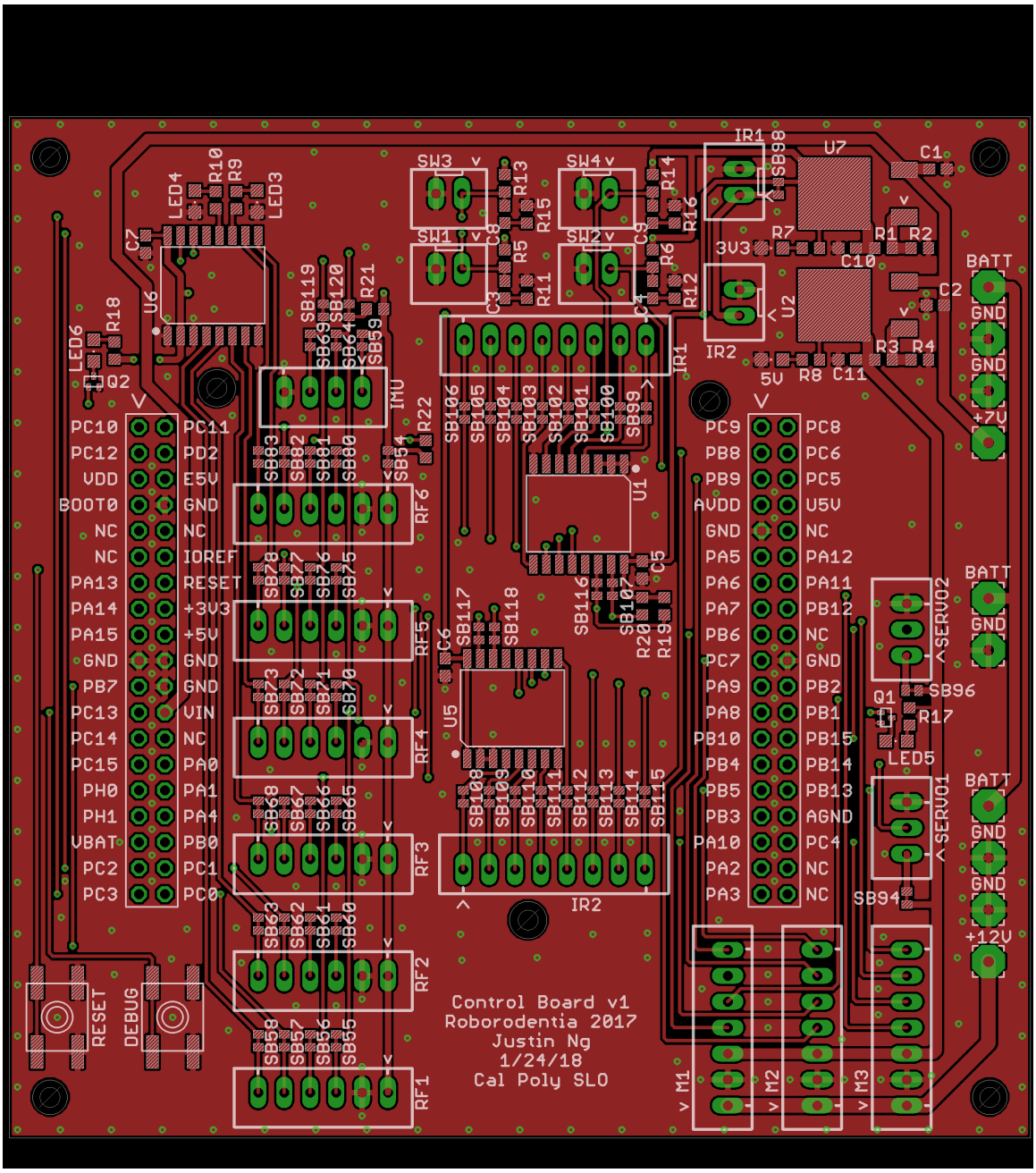


Figure B.1: Interconnect PCB Layout – Top Layer

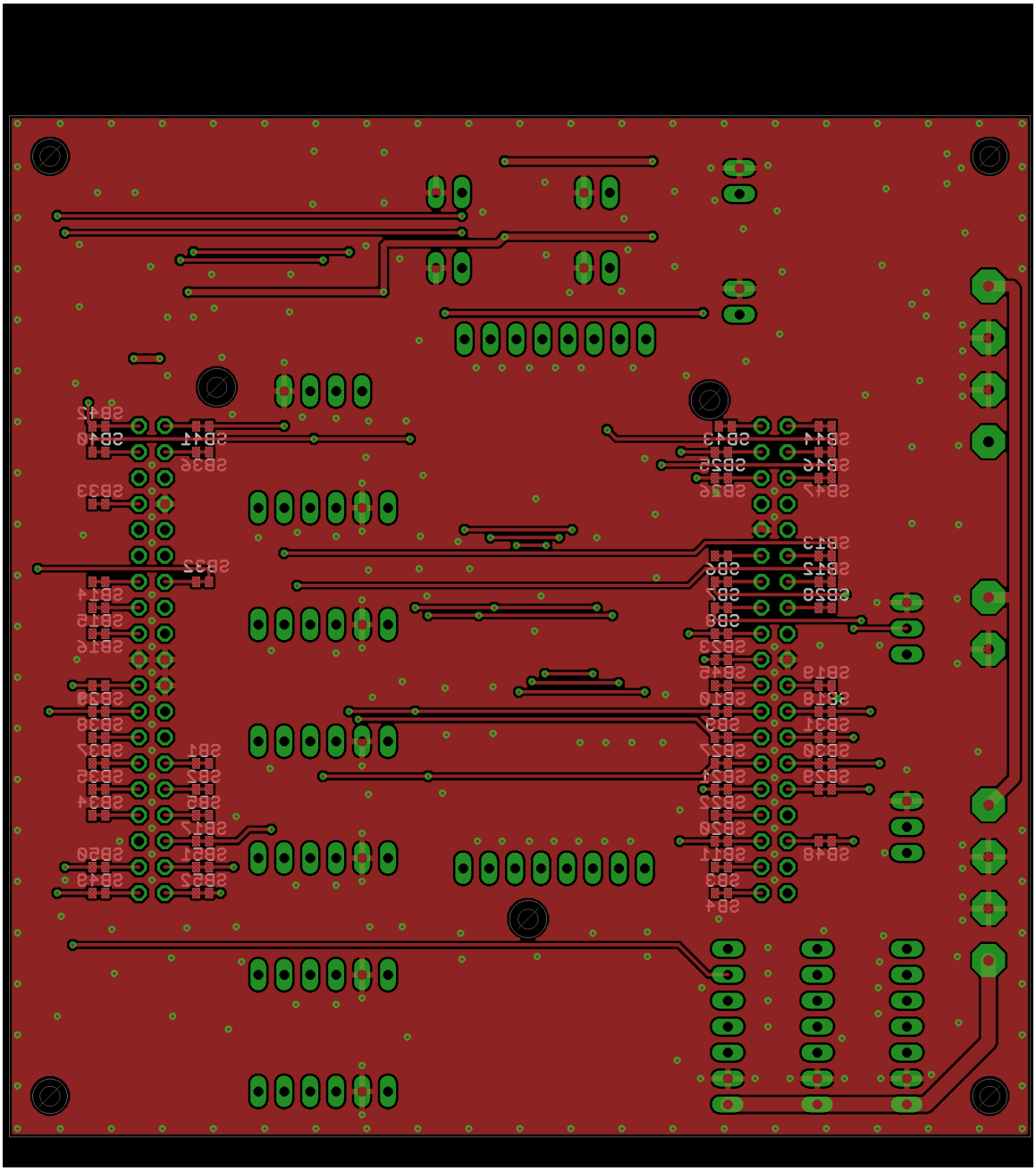


Figure B.2: Interconnect PCB Layout – Bottom Layer

Appendix C

INTERCONNECT PCB BILL OF MATERIALS

Qty	Value	Package	Parts	Description	DIGIKEY PN
2	AZ1085CD	TO252	U2, U7	AZ1085C	AZ1085CD-ADJTRG1DICT-ND
7	0.1uF	C0603	C3, C4, C5, C6, C7, C8, C9	CAPACITOR	445-5667-1-ND
2	10uF	C0603	C1, C2	CAPACITOR	490-13248-1-ND
2	22uF	C0603	C10, C11	CAPACITOR	490-10476-1-ND
3	TCA9554A	SOIC16	U1, U5, U6	I2C 8-bit GPIO Expander	296-45456-1-ND
6		JST-XH-02	X1, X12, X16, X18, X19, X20	JST XH Connector 2 Pin	455-2247-ND
2		JST-XH-03	X13, X14	JST XH Connector 3 Pin	455-2248-ND
1		JST-XH-04	X8	JST XH Connector 4 Pin	455-2249-ND
6		JST-XH-06	X2, X3, X4, X5, X6, X7	JST XH Connector 6 Pin	455-2271-ND
3		JST-XH-07	X9, X10, X11	JST XH Connector 7 Pin	455-2252-ND
2		JST-XH-08	X15, X17	JST XH Connector 8 Pin	455-2251-ND
6	GREEN	CHIPLED_0805	LED1, LED2, LED3, LED4, LED5, LED6	LED	732-4971-1-ND
2	RE1C002UN	SOT416FL	Q1, Q2	Logic-level N-FET	RE1C002UNTCLCT-ND
4	82	R0603	R7, R9, R10, R18	RESISTOR	311-82.0HRCT-ND
5	100	R0603	R3, R11, R12, R15, R16	RESISTOR	311-100HRCT-ND

1	178	R0603	R1	RESISTOR	311-178HRCT-ND
1	287	R0603	R2	RESISTOR	311-287HRCT-ND
1	301	R0603	R4	RESISTOR	311-301HRCT-ND
2	430	R0603	R8, R17	RESISTOR	311-430HRCT-ND
4	3.3k	R0603	R19, R20, R21, R22	RESISTOR	311-3.30KHRCT-ND
4	4.7k	R0603	R5, R6, R13, R14	RESISTOR	311-4.70KHRCT-ND
2	SPST	EVQ-Q2	SW1, SW2	SMT 6mm switch	P12955SCT-ND
1	STM32F446	NUCLEO	U4	STM32 NUCLEO-64	497-15882-ND
10		3,17/1,1	PAD1, PAD2, PAD3, PAD4, PAD5, PAD6, PAD7, PAD8, PAD9, PAD10	Wire PAD connect wire on PCB	n/a

105	BRIDGE	SB1, SB2, SB3, SB4, SB5, SB6, SB7, SB8, SB9, SB10, SB11, SB12, SB13, SB14, SB15, SB16, SB17, SB18, SB19, SB20, SB21, SB22, SB23, SB24, SB25, SB26, SB27, SB28, SB29, SB30, SB31, SB32, SB33, SB34, SB35, SB36, SB37, SB38, SB39, SB40, SB41, SB42, SB43, SB44, SB45, SB46, SB47, SB48, SB49, SB50, SB51, SB52, SB54, SB55, SB56, SB57, SB58, SB59, SB60, SB61, SB62, SB63, SB64, SB65, SB66, SB67, SB68, SB69, SB70, SB71, SB72, SB73, SB75, SB76, SB77, SB78, SB80, SB81, SB82, SB83, SB94, SB96, SB98, SB99, SB100, SB101, SB102, SB103, SB104, SB105, SB106, SB107, SB108, SB109, SB110, SB111, SB112, SB113, SB114, SB115, SB116, SB117, SB118, SB119, SB120	Solder bridge with knife-cuttable prebridged connection.	n/a

Appendix D

STM32CUBEMX REPORT

1. Description

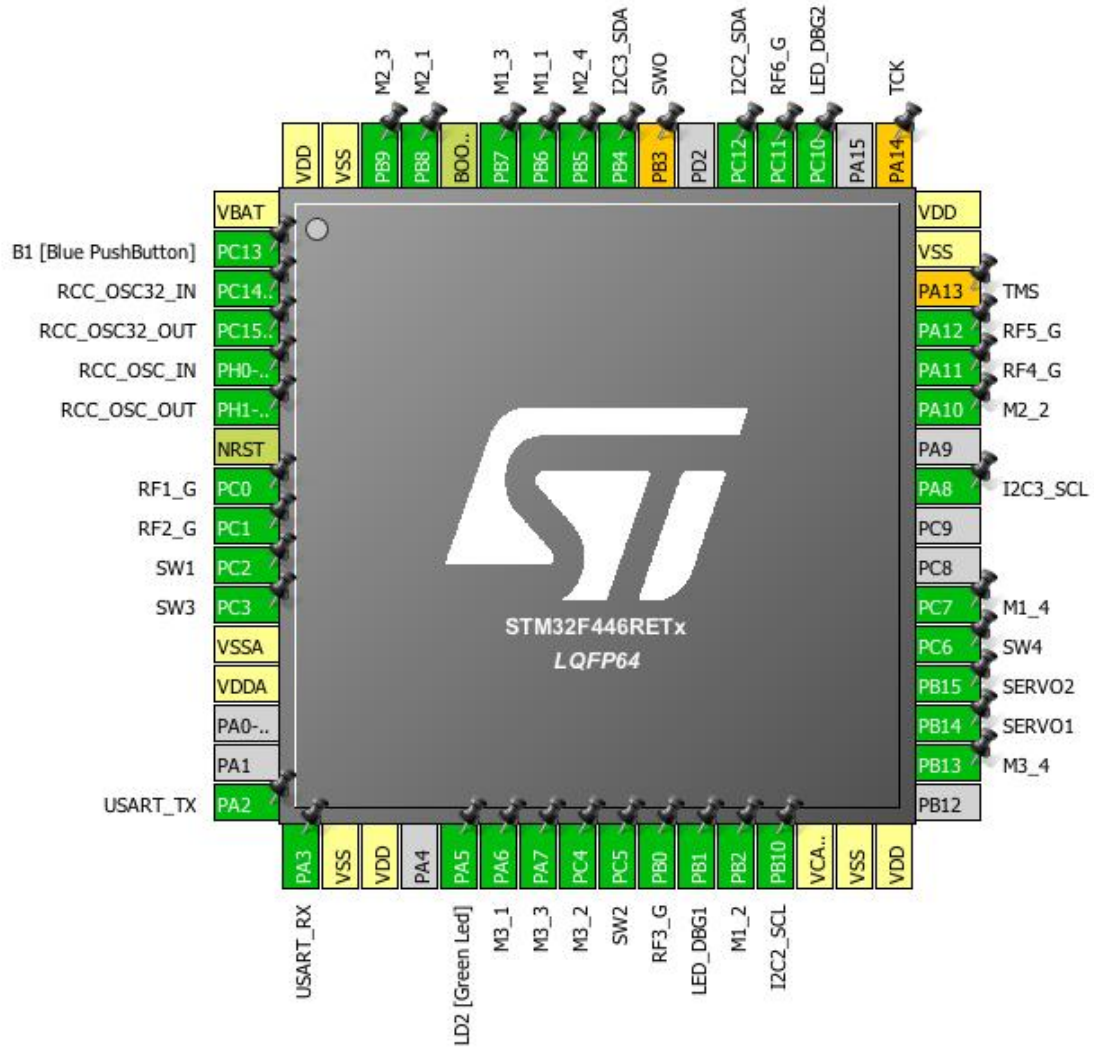
1.1. Project

Project Name	firmware_new
Board Name	NUCLEO-F446RE
Generated with:	STM32CubeMX 4.24.0
Date	05/19/2018

1.2. MCU

MCU Series	STM32F4
MCU Line	STM32F446
MCU name	STM32F446RETx
MCU Package	LQFP64
MCU Pin number	64

2. Pinout Configuration



3. Pins Configuration

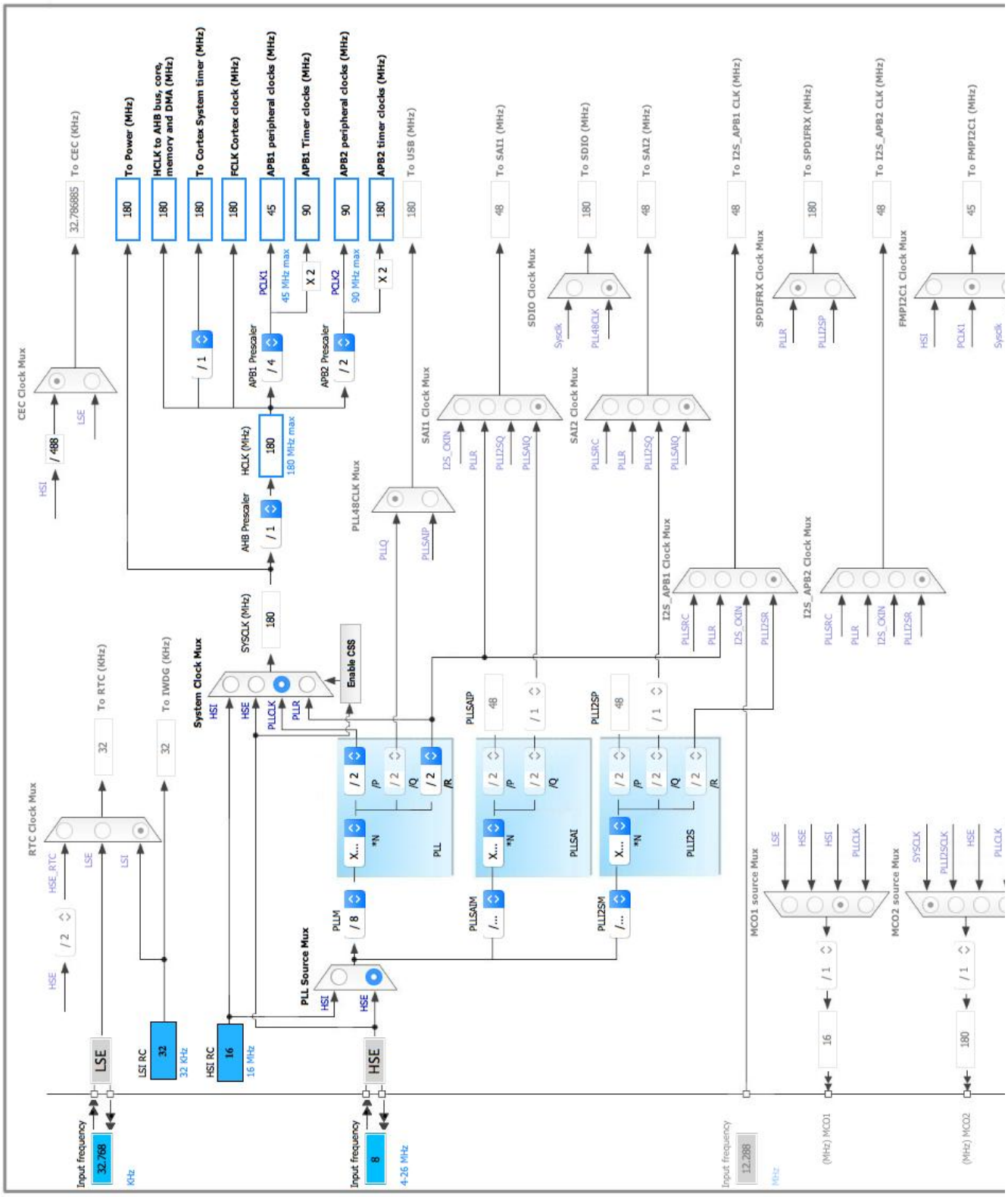
Pin Number LQFP64	Pin Name (function after reset)	Pin Type	Alternate Function(s)	Label
1	VBAT	Power		
2	PC13	I/O	GPIO_EXTI13	B1 [Blue PushButton]
3	PC14-OSC32_IN	I/O	RCC_OSC32_IN	
4	PC15-OSC32_OUT	I/O	RCC_OSC32_OUT	
5	PH0-OSC_IN	I/O	RCC_OSC_IN	
6	PH1-OSC_OUT	I/O	RCC_OSC_OUT	
7	NRST	Reset		
8	PC0	I/O	GPIO_EXTI0	RF1_G
9	PC1	I/O	GPIO_EXTI1	RF2_G
10	PC2 *	I/O	GPIO_Output	SW1
11	PC3 *	I/O	GPIO_Output	SW3
12	VSSA	Power		
13	VDDA	Power		
16	PA2	I/O	USART2_TX	USART_TX
17	PA3	I/O	USART2_RX	USART_RX
18	VSS	Power		
19	VDD	Power		
21	PA5 *	I/O	GPIO_Output	LD2 [Green Led]
22	PA6	I/O	TIM3_CH1	M3_1
23	PA7	I/O	TIM3_CH2	M3_3
24	PC4 *	I/O	GPIO_Output	M3_2
25	PC5 *	I/O	GPIO_Output	SW2
26	PB0 *	I/O	GPIO_Input	RF3_G
27	PB1 *	I/O	GPIO_Output	LED_DBG1
28	PB2 *	I/O	GPIO_Output	M1_2
29	PB10	I/O	I2C2_SCL	
30	VCAP_1	Power		
31	VSS	Power		
32	VDD	Power		
34	PB13 *	I/O	GPIO_Output	M3_4
35	PB14	I/O	TIM12_CH1	SERVO1
36	PB15	I/O	TIM12_CH2	SERVO2
37	PC6 *	I/O	GPIO_Output	SW4
38	PC7 *	I/O	GPIO_Output	M1_4
41	PA8	I/O	I2C3_SCL	
43	PA10 *	I/O	GPIO_Output	M2_2

Pin Number LQFP64	Pin Name (function after reset)	Pin Type	Alternate Function(s)	Label
44	PA11	I/O	GPIO_EXTI11	RF4_G
45	PA12	I/O	GPIO_EXTI12	RF5_G
46	PA13 **	I/O	SYS_JTMS-SWDIO	TMS
47	VSS	Power		
48	VDD	Power		
49	PA14 **	I/O	SYS_JTCK-SWCLK	TCK
51	PC10 *	I/O	GPIO_Output	LED_DBG2
52	PC11 *	I/O	GPIO_Input	RF6_G
53	PC12	I/O	I2C2_SDA	
55	PB3 **	I/O	SYS_JTDO-SWO	SWO
56	PB4	I/O	I2C3_SDA	
57	PB5 *	I/O	GPIO_Output	M2_4
58	PB6	I/O	TIM4_CH1	M1_1
59	PB7	I/O	TIM4_CH2	M1_3
60	BOOT0	Boot		
61	PB8	I/O	TIM4_CH3	M2_1
62	PB9	I/O	TIM4_CH4	M2_3
63	VSS	Power		
64	VDD	Power		

* The pin is affected with an I/O function

** The pin is affected with a peripheral function but no peripheral mode is activated

4. Clock Tree Configuration



5. IPs and Middleware Configuration

5.1. I2C2

I2C: I2C

5.1.1. Parameter Settings:

Master Features:

I2C Speed Mode	**Fast Mode ***
I2C Clock Speed (Hz)	400000
Fast Mode Duty Cycle	Duty cycle Tlow/Thigh = 2

Slave Features:

Clock No Stretch Mode	Disabled
Primary Address Length selection	7-bit
Dual Address Acknowledged	Disabled
Primary slave address	0
General Call address detection	Disabled

5.2. I2C3

I2C: I2C

5.2.1. Parameter Settings:

Master Features:

I2C Speed Mode	**Fast Mode ***
I2C Clock Speed (Hz)	400000
Fast Mode Duty Cycle	Duty cycle Tlow/Thigh = 2

Slave Features:

Clock No Stretch Mode	Disabled
Primary Address Length selection	7-bit
Dual Address Acknowledged	Disabled
Primary slave address	0
General Call address detection	Disabled

5.3. RCC

High Speed Clock (HSE): Crystal/Ceramic Resonator
Low Speed Clock (LSE) : Crystal/Ceramic Resonator

5.3.1. Parameter Settings:

System Parameters:

VDD voltage (V)	3.3
Instruction Cache	Enabled
Prefetch Buffer	Enabled
Data Cache	Enabled
Flash Latency(WS)	5 WS (6 CPU cycle)

RCC Parameters:

HSI Calibration Value	16
TIM Prescaler Selection	Disabled
HSE Startup Timout Value (ms)	100
LSE Startup Timout Value (ms)	5000

Power Parameters:

Power Regulatror Voltage Scale	Power Regulator Voltage Scale 1
Power Over Drive	Enabled

5.4. SYS

Timebase Source: SysTick

5.5. TIM3

Clock Source : Internal Clock
Channel1: PWM Generation CH1
Channel2: PWM Generation CH2

5.5.1. Parameter Settings:

Counter Settings:

Prescaler (PSC - 16 bits value)	**1 ***
Counter Mode	Up
Counter Period (AutoReload Register - 16 bits value)	**2047 ***
Internal Clock Division (CKD)	No Division

Trigger Output (TRGO) Parameters:

Master/Slave Mode	Disable (no sync between this TIM (Master) and its Slaves
Trigger Event Selection	Reset (UG bit from TIMx_EGR)

PWM Generation Channel 1:

Mode	PWM mode 1
Pulse (16 bits value)	0
Fast Mode	Disable
CH Polarity	High

PWM Generation Channel 2:

Mode	PWM mode 1
Pulse (16 bits value)	0
Fast Mode	Disable
CH Polarity	High

5.6. TIM4

mode: Clock Source

Channel1: PWM Generation CH1

Channel2: PWM Generation CH2

Channel3: PWM Generation CH3

Channel4: PWM Generation CH4

5.6.1. Parameter Settings:

Counter Settings:

Prescaler (PSC - 16 bits value)	**1 ***
Counter Mode	Up
Counter Period (AutoReload Register - 16 bits value)	**2047 ***
Internal Clock Division (CKD)	No Division

Trigger Output (TRGO) Parameters:

Master/Slave Mode	Disable (no sync between this TIM (Master) and its Slaves
Trigger Event Selection	Reset (UG bit from TIMx_EGR)

PWM Generation Channel 1:

Mode	PWM mode 1
Pulse (16 bits value)	0
Fast Mode	Disable
CH Polarity	High

PWM Generation Channel 2:

Mode	PWM mode 1

Pulse (16 bits value)	0
Fast Mode	Disable
CH Polarity	High

PWM Generation Channel 3:

Mode	PWM mode 1
Pulse (16 bits value)	0
Fast Mode	Disable
CH Polarity	High

PWM Generation Channel 4:

Mode	PWM mode 1
Pulse (16 bits value)	0
Fast Mode	Disable
CH Polarity	High

5.7. TIM5

mode: Clock Source

5.7.1. Parameter Settings:

Counter Settings:

Prescaler (PSC - 16 bits value)	**9000 ***
Counter Mode	Up
Counter Period (AutoReload Register - 32 bits value)	**0xFFFFFFFF ***
Internal Clock Division (CKD)	No Division

Trigger Output (TRGO) Parameters:

Master/Slave Mode	Disable (no sync between this TIM (Master) and its Slaves
Trigger Event Selection	Reset (UG bit from TIMx_EGR)

5.8. TIM12

mode: Clock Source
Channel1: PWM Generation CH1
Channel2: PWM Generation CH2

5.8.1. Parameter Settings:

Counter Settings:

Prescaler (PSC - 16 bits value)	**354 ***
Counter Mode	Up
Counter Period (AutoReload Register - 16 bits value)	**3178 ***
Internal Clock Division (CKD)	No Division

PWM Generation Channel 1:

Mode	PWM mode 1
Pulse (16 bits value)	**63 ***
Fast Mode	Disable
CH Polarity	High

PWM Generation Channel 2:

Mode	PWM mode 1
Pulse (16 bits value)	**63 ***
Fast Mode	Disable
CH Polarity	High

5.9. USART2

Mode: Asynchronous

5.9.1. Parameter Settings:

Basic Parameters:

Baud Rate	**921600 ***
Word Length	8 Bits (including Parity)
Parity	None
Stop Bits	1

Advanced Parameters:

Data Direction	Receive and Transmit
Over Sampling	16 Samples

*** User modified value**

6. System Configuration

6.1. GPIO configuration

IP	Pin	Signal	GPIO mode	GPIO pull/up pull down	Max Speed	User Label
I2C2	PB10	I2C2_SCL	Alternate Function Open Drain	Pull-up	**Very High** *	
	PC12	I2C2_SDA	Alternate Function Open Drain	Pull-up	**Very High** *	
I2C3	PA8	I2C3_SCL	Alternate Function Open Drain	Pull-up	**Very High** *	
	PB4	I2C3_SDA	Alternate Function Open Drain	Pull-up	**Very High** *	
RCC	PC14-OSC32_IN	RCC_OSC32_IN	n/a	n/a	n/a	
	PC15-OSC32_OUT	RCC_OSC32_OUT	n/a	n/a	n/a	
	PH0-OSC_IN	RCC_OSC_IN	n/a	n/a	n/a	
	PH1-OSC_OUT	RCC_OSC_OUT	n/a	n/a	n/a	
TIM3	PA6	TIM3_CH1	Alternate Function Push Pull	No pull-up and no pull-down	Low	M3_1
	PA7	TIM3_CH2	Alternate Function Push Pull	No pull-up and no pull-down	Low	M3_3
TIM4	PB6	TIM4_CH1	Alternate Function Push Pull	No pull-up and no pull-down	Low	M1_1
	PB7	TIM4_CH2	Alternate Function Push Pull	No pull-up and no pull-down	Low	M1_3
	PB8	TIM4_CH3	Alternate Function Push Pull	No pull-up and no pull-down	Low	M2_1
	PB9	TIM4_CH4	Alternate Function Push Pull	No pull-up and no pull-down	Low	M2_3
TIM12	PB14	TIM12_CH1	Alternate Function Push Pull	No pull-up and no pull-down	Low	SERVO1
	PB15	TIM12_CH2	Alternate Function Push Pull	No pull-up and no pull-down	Low	SERVO2
USART2	PA2	USART2_TX	Alternate Function Push Pull	Pull-up	**Very High** *	USART_TX
	PA3	USART2_RX	Alternate Function Push Pull	Pull-up	**Very High** *	USART_RX
Single Mapped Signals	PA13	SYS_JTMS-SWDIO	n/a	n/a	n/a	TMS
	PA14	SYS_JTCK-SWCLK	n/a	n/a	n/a	TCK
	PB3	SYS_JTDO-	n/a	n/a	n/a	SWO

IP	Pin	Signal	GPIO mode	GPIO pull/up pull down	Max Speed	User Label
		SWO				
GPIO	PC13	GPIO_EXTI13	**External Interrupt Mode with Falling edge trigger detection**	No pull-up and no pull-down	n/a	B1 [Blue PushButton]
	PC0	GPIO_EXTI0	**External Interrupt Mode with Falling edge trigger detection**	**Pull-up ***	n/a	RF1_G
	PC1	GPIO_EXTI1	**External Interrupt Mode with Falling edge trigger detection**	**Pull-up ***	n/a	RF2_G
	PC2	GPIO_Output	Output Push Pull	No pull-up and no pull-down	Low	SW1
	PC3	GPIO_Output	Output Push Pull	No pull-up and no pull-down	Low	SW3
	PA5	GPIO_Output	Output Push Pull	No pull-up and no pull-down	Low	LD2 [Green Led]
	PC4	GPIO_Output	Output Push Pull	No pull-up and no pull-down	Low	M3_2
	PC5	GPIO_Output	Output Push Pull	No pull-up and no pull-down	Low	SW2
	PB0	GPIO_Input	Input mode	**Pull-up ***	n/a	RF3_G
	PB1	GPIO_Output	Output Push Pull	No pull-up and no pull-down	Low	LED_DBG1
	PB2	GPIO_Output	Output Push Pull	No pull-up and no pull-down	Low	M1_2
	PB13	GPIO_Output	Output Push Pull	No pull-up and no pull-down	Low	M3_4
	PC6	GPIO_Output	Output Push Pull	No pull-up and no pull-down	Low	SW4
	PC7	GPIO_Output	Output Push Pull	No pull-up and no pull-down	Low	M1_4
	PA10	GPIO_Output	Output Push Pull	No pull-up and no pull-down	Low	M2_2
	PA11	GPIO_EXTI11	**External Interrupt Mode with Falling edge trigger detection**	**Pull-up ***	n/a	RF4_G
	PA12	GPIO_EXTI12	**External Interrupt Mode with Falling edge trigger detection**	**Pull-up ***	n/a	RF5_G
	PC10	GPIO_Output	Output Push Pull	No pull-up and no pull-down	Low	LED_DBG2
	PC11	GPIO_Input	Input mode	No pull-up and no pull-down	n/a	RF6_G
	PB5	GPIO_Output	Output Push Pull	No pull-up and no pull-down	Low	M2_4

6.2. DMA configuration

DMA request	Stream	Direction	Priority
I2C2_RX	DMA1_Stream2	Peripheral To Memory	Low
I2C2_TX	DMA1_Stream7	Memory To Peripheral	**High ***
I2C3_RX	DMA1_Stream1	Peripheral To Memory	Low
I2C3_TX	DMA1_Stream4	Memory To Peripheral	**High ***
USART2_RX	DMA1_Stream5	Peripheral To Memory	**Very High ***
USART2_TX	DMA1_Stream6	Memory To Peripheral	**Very High ***

I2C2_RX: DMA1_Stream2 DMA request Settings:

Mode: Normal
Use fifo: Disable
Peripheral Increment: Disable
Memory Increment: **Enable ***
Peripheral Data Width: Byte
Memory Data Width: Byte

I2C2_TX: DMA1_Stream7 DMA request Settings:

Mode: Normal
Use fifo: Disable
Peripheral Increment: Disable
Memory Increment: **Enable ***
Peripheral Data Width: Byte
Memory Data Width: Byte

I2C3_RX: DMA1_Stream1 DMA request Settings:

Mode: Normal
Use fifo: Disable
Peripheral Increment: Disable
Memory Increment: **Enable ***
Peripheral Data Width: Byte
Memory Data Width: Byte

I2C3_TX: DMA1_Stream4 DMA request Settings:

Mode:	Normal
Use fifo:	Disable
Peripheral Increment:	Disable
Memory Increment:	**Enable ***
Peripheral Data Width:	Byte
Memory Data Width:	Byte

USART2_RX: DMA1_Stream5 DMA request Settings:

Mode:	**Circular ***
Use fifo:	Disable
Peripheral Increment:	Disable
Memory Increment:	**Enable ***
Peripheral Data Width:	Byte
Memory Data Width:	Byte

USART2_TX: DMA1_Stream6 DMA request Settings:

Mode:	Normal
Use fifo:	Disable
Peripheral Increment:	Disable
Memory Increment:	**Enable ***
Peripheral Data Width:	Byte
Memory Data Width:	Byte

6.3. NVIC configuration

Interrupt Table	Enable	Preenmption Priority	SubPriority
Non maskable interrupt	true	0	0
Hard fault interrupt	true	0	0
Memory management fault	true	0	0
Pre-fetch fault, memory access fault	true	0	0
Undefined instruction or illegal state	true	0	0
System service call via SWI instruction	true	0	0
Debug monitor	true	0	0
Pendable request for system service	true	0	0
System tick timer	true	0	0
DMA1 stream1 global interrupt	true	0	0
DMA1 stream2 global interrupt	true	0	0
DMA1 stream4 global interrupt	true	0	0
DMA1 stream5 global interrupt	true	0	0
DMA1 stream6 global interrupt	true	0	0
I2C2 event interrupt	true	0	0
I2C2 error interrupt	true	0	0
USART2 global interrupt	true	0	0
DMA1 stream7 global interrupt	true	0	0
I2C3 event interrupt	true	0	0
I2C3 error interrupt	true	0	0
PVD interrupt through EXTI line 16	unused		
Flash global interrupt	unused		
RCC global interrupt	unused		
EXTI line 0 interrupt	unused		
EXTI line 1 interrupt	unused		
TIM3 global interrupt	unused		
TIM4 global interrupt	unused		
EXTI line[15:10] interrupts	unused		
TIM8 break interrupt and TIM12 global interrupt	unused		
TIM5 global interrupt	unused		
FPU global interrupt	unused		

*** User modified value**

7. Power Consumption Calculator report

7.1. Microcontroller Selection

Series	STM32F4
Line	STM32F446
MCU	STM32F446RETx
Datasheet	027107_Rev6

7.2. Parameter Selection

Temperature	25
Vdd	3.3

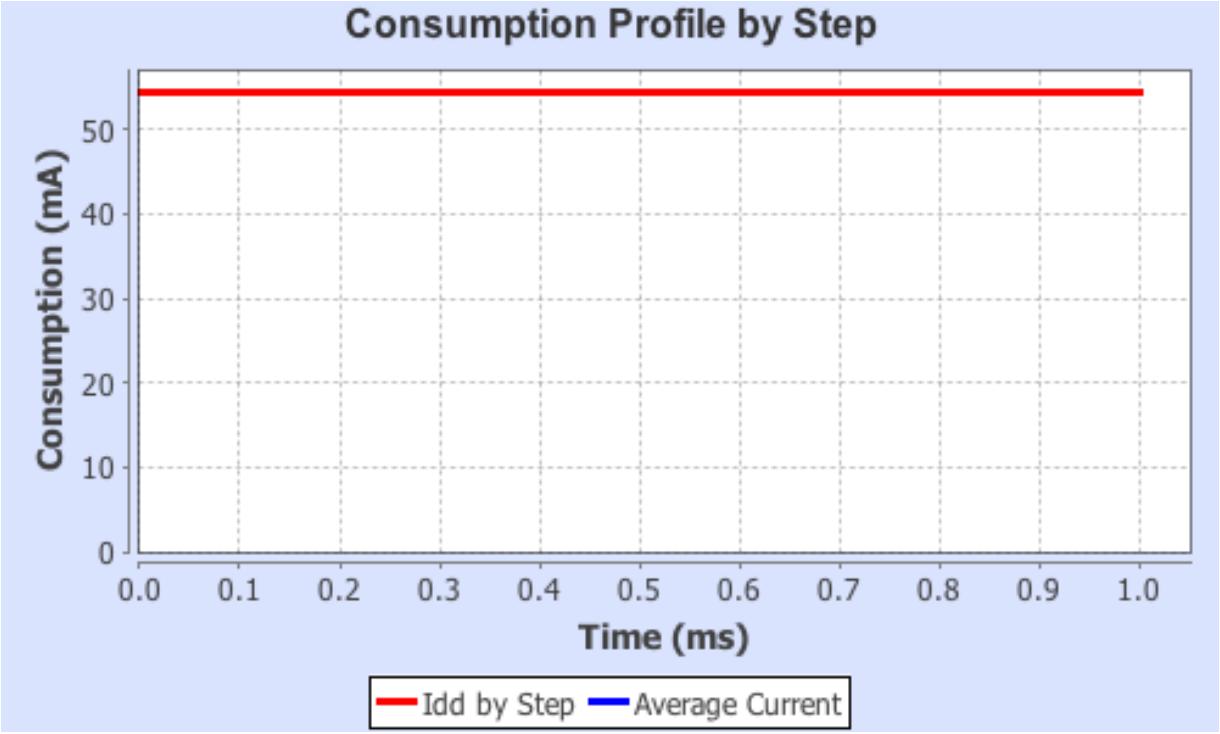
7.3. Sequence

Step	Step1
Mode	RUN
Vdd	3.3
Voltage Source	Vbus
Range	Scale1-High
Fetch Type	RAM/FLASH/REGON/ART/PREFETCH
Clock Configuration	HSE PLL
Clock Source Frequency	4 MHz
CPU Frequency	180 MHz
Peripherals	DMA1 DMA2 GPIOA GPIOB GPIOC I2C2 I2C3 SYS TIM1 TIM2 TIM3 TIM4 TIM12 USART2
Additional Cons.	0 mA
Average Current	54.31 mA
Duration	1 ms
DMIPS	225.0
Ta Max	96.76
Category	In DS Table

7.4. RESULTS

Sequence Time	1 ms	Average Current	54.31 mA
Battery Life	0	Average DMIPS	225.0 DMIPS

7.5. Chart



8. Software Project

8.1. Project Settings

Name	Value
Project Name	firmware_new
Project Folder	/Users/justinng/Documents/Github/roborodentia2017/firmware_new
Toolchain / IDE	Makefile
Firmware Package Name and Version	STM32Cube FW_F4 V1.18.0

8.2. Code Generation Settings

Name	Value
STM32Cube Firmware Library Package	Copy all used libraries into the project folder
Generate peripheral initialization as a pair of '.c/.h' files	Yes
Backup previously generated files when re-generating	No
Delete previously generated files when not re-generated	Yes
Set all free pins as analog (to optimize the power consumption)	No

9. Software Pack Report