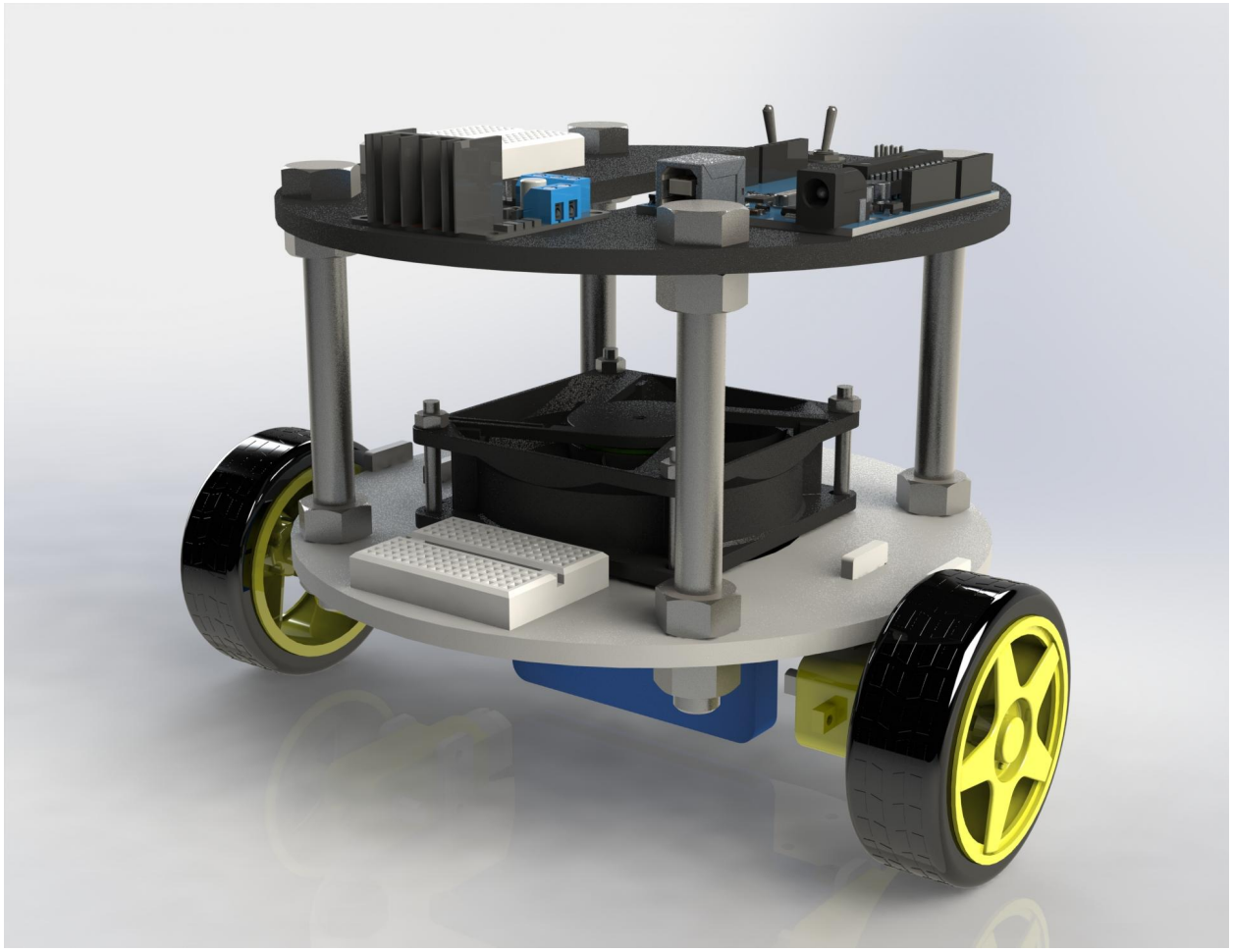
MIE438 Final Project -
Gyroscope Controlled Rover Vacuum



Project Demo Link: https://youtu.be/Dr4fFwFbMys

Ibath Rahman (1003484115)

Farhan Wadia (1003012606)

Date: Apr. 16, 2021

1.0 Introduction & Background

Our project proposal involved the development of a gyroscopically controlled rover vacuum, and was broken up into three stages: developing a gyroscopically controlled rover, adding a vacuum cleaning function, and adding sensory feedback to detect and override dangerous user commands that could cause the rover to crash into an obstacle or fall from a height.

We were successfully able to implement the first two stages, and were not able to implement the final stage due to a lack of time arising from software debugging issues discussed in Section 2.2.3. Our initial choices for microcontrollers were to use an Arduino Nano for the master gyroscopic controller and an Arduino Uno for the slave rover. We made this decision due to already having these microcontrollers, and because of the relatively large number of tutorials available online on how to use these microcontrollers. To control the motors, we used the L298N motor driver for largely similar reasons, but also because it met our control specifications exactly without us leaving any features unused; that is, it controls up to two motors using PWM to control speed and a H-Bridge circuit to control rotation direction. To implement the gyroscopic controller, we used the MPU-6050 sensor, which has a built in 3-axis accelerometer and gyroscope. To wirelessly send signals from the Arduino Nano on the controller to the Arduino Uno on the rover, we used the HC-05 Bluetooth module. To implement the vacuum cleaning function, a toggle switch connected to an input pin on the Arduino Nano master would determine the control signal sent to the Arduino Uno slave, which then passed through a transistor switching circuit to actuate a DC fan.

2.0 Detailed Design

In this section we discuss the design selections and implementations of the hardware devices, the software content, and the physical design of the rover.

2.1 Hardware Design

The hardware connection schematic of the gyroscopic controller and the rover is shown in Appendix A. The role of each hardware device in this design is discussed below.

2.1.1 Microcontroller Selection

Table 1 discusses the features used for the master Arduino Nano and slave Arduino Uno.

Table 1: Arduino Features and Usage in Project

Arduino Feature	Feature Use In Project
Digital Pins	These GPIO pins can be used to perform digital functions, inputting or outputting a 1 (for a 5V reference) or a 0 (for a 0V reference) which can be read or written in software using the *digitalWrite* or *digitalRead* command. This is used in our project with the Arduino Nano input pin 8 to identify the state of a push button (pressed or unpressed) for controlling the vacuum fan, and with the Arduino Uno output pins 4, 5, 6, 7 to control the direction of the two motors.

Analog Pins	These pins can be used as GPIO pins (as with the digital pins), but their main function is to read analog sensor values via an Analog to Digital converter with 10 bit resolution [1]. In software, this can be implemented through the *analogRead* command. We do not use any of these pins since we do not incorporate analog sensors into the design. However, it may have been useful to incorporate an analog temperature sensor near the motor driver to ensure it does not overheat when powering the motors.
Serial Pins	The Arduino has TX and RX pins which are used in our project to facilitate the communication between the Arduino and the Bluetooth modules.
PWM Pins	Pins 3, 5, 6, 9, 10, and 11 on the Arduino Uno can be used to generate PWM signals using the *analogWrite* command in the software implementation [2]. This essentially acts as a Digital to Analog converter. Pins 9 and 10 of the Arduino Uno are used in this project to send PWM commands to the motor driver module to control the speed of each motor.
SPI Pins	Pin 10 (Slave Select pin), Pin 11 (Master Out Slave In pin), Pin 12 (Master In Slave Out pin), and Pin 13 (Slave Acknowledge Pin) could be used for SPI communication [2]. This was not used in our project but could be used if it was required by the peripheral hardware devices or sensors.
SDA/SCL Pins	The SDA and SCL pins on the Arduino Nano (pins *A4* and *A5*, respectively) were used for I2C Communication with the MPU-6050.
Internal Pull-up Resistor	The Arduino contains internal pull-up resistors for some GPIO pins. This could have been used in the circuit design (particularly for the pushbutton) instead of an external resistor. This would have saved marginal space on the Gyroscopic Controller device. To use an internal pull up resistor we can implement in software a *pinMode(pin, INPUT_PULLUP)* command [3].
5V Output Voltage	The Arduino provided 5V pins that were used to power low powered peripheral hardware devices such as the Bluetooth modules and the MPU-6050 sensor.
On Board Voltage Regulator	The regulator in the Arduino was useful when using external power supplies. In our project we used a 9V battery as a power supply for the 5V Arduino Nano by connecting it to the *GND* and *Vin* pins. The on board voltage regulator regulated this to 5V to power the Nano.
External Interrupt Pins	Pins 2 and 3 of the Arduino are used to generate external interrupts. A polling method was used rather than an interrupt based method to read the sensor inputs because sensor updates from the MPU-6050 arrive rapidly as the Gyroscopic Controller device is tilted. In this case, an interrupt based method would add unnecessary overhead since it would require additional instructions for the compiler to implement. Furthermore the Arduino Nano's sole task is to process the MPU-6050 data and send it to the slave device; there are no other critical tasks this microcontroller does to warrant the need for an interrupt.

2.1.2 Sensor and Peripheral Hardware Selections

Key sensors and peripheral hardware used in our project are discussed below.

2.1.2.1 L298N Motor Driver Module

In our design, this module is used to control the speed and rotation direction of two DC motors with voltage ratings of 3-9V. The input/output pins of this module are shown in Figure 1. The motor speed is varied by varying the duty cycle of the Pulse Width Modulation (PWM) signal sent from the Arduino to pin ENA (to control speed of motor A attached to OUT1 & OUT2) and ENB (to control speed of motor B attached to OUT3 & OUT4). Furthermore, the two integrated H-bridges in the motor driver allow for control of the direction for each motor. Motor A moves forward when pins IN1 and IN2 are set to low and high respectively, and Motor B moves forward when pins IN3 and IN4 are set to high and low respectively; for the backwards direction, each pin setting is the opposite of that described above [4].

This module was selected for our design's motor control functions because:

1. It is convenient since it already incorporates two integrated H-bridges, protection circuitry, and a 5V voltage regulator which can readily be used to power a microcontroller. This saves us work on building the circuitry ourselves.
2. It is easy to use. Other than the output motor and the ground/power pins, this module only consists of six control pins (3 pins for each motor) where 2 of the pins receive a simple PWM signal, and the other 4 pins receive a digital high/low signal for direction control. The rest is taken care of by the integrated circuitry.

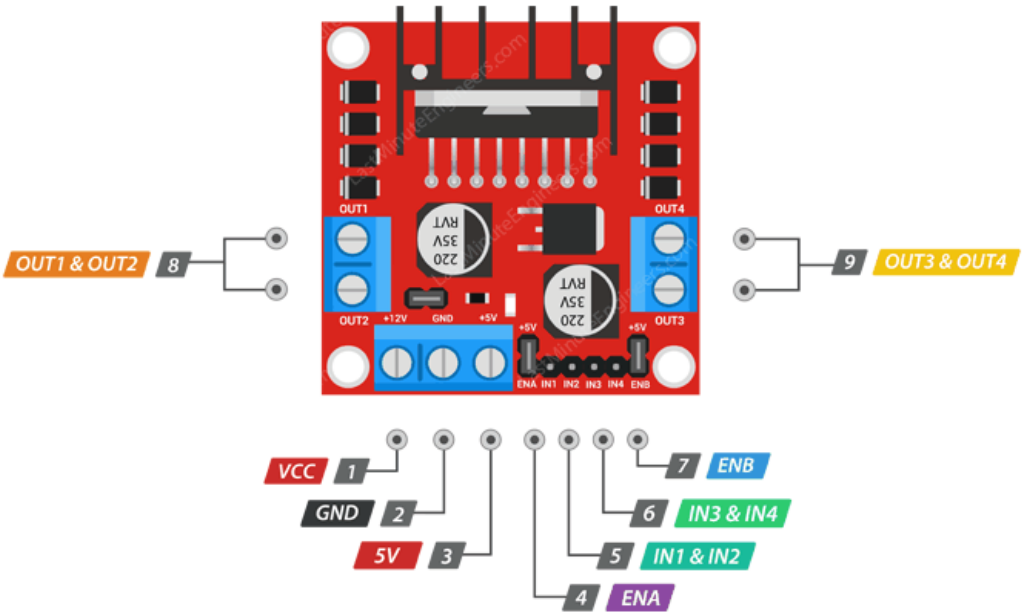


Figure 1: High-level schematic of L298N Motor Driver Module [5]

2.1.2.2 HC-05 Bluetooth Module

Two HC-05 Bluetooth modules were used in this design. One was used as the transmitter on the gyroscopic controller and the other was used as the receiver on the rover. Various sources were explored to obtain a better understanding on how to use and configure these modules [6] [7] [8].

This module was used to perform wireless communication of data from the transmitter to the receiver. Namely, the module on the master gyroscopic controller transmits the PWM signals with duty cycles corresponding to the tilt of the gyroscopic controller, and the state (unpressed or pressed) of the pushbutton. At the receiving end, the Arduino Uno slave passes the received PWM signals to the ENA and ENB pins of the L298N module which then actuates the attached motors to move forward, backward, right, or left and turns on/off the fan depending on the pushbutton state. This module can easily be interfaced with the selected Arduino microcontrollers through Universal Asynchronous Receiver/Transmitter (UART) [9]. This allows the Arduino to transmit/receive data to/from the module asynchronously and at a specified baud rate. This communication is done purely through hardware that is already integrated into the Arduino and Bluetooth module; essentially, we connect the Tx and Rx pins of the Arduino to the Rx and Tx pins of the Bluetooth module. This substantially decreased the workload of the team in software implementation and thus these modules were a desirable choice for the team to use to transmit/receive data wirelessly.

2.1.2.3 MPU-6050 Sensor

An integral piece to the gyroscopic controller is the MPU-6050 sensor which has an integrated 3-axis gyroscope and a 3-axis accelerometer, both of which can be used to determine the orientation of the controller. For our project, only the accelerometer was used because its processed sensor readings could be easily converted to the roll and pitch angles of the controller using algebraic equations; using the gyroscope sensor readings to calculate roll and pitch angles requires integration, making the estimates conducive to a drift in error over time. The optimal solution for future works would involve a software implementation of a complementary filter that uses both the gyroscope and accelerometer readings to more accurately estimate stable roll and pitch angles of the controller. A guide for receiving data in software from this sensor can be found in [10] and more detailed information on it can be found in its datasheet [11].

2.2 Software Implementation

Our software implementation used a high-level programming language on Arduino IDE, with syntax nearly identical to C++. High level flow diagrams of the master program running on the Arduino Nano and slave program running on the Arduino Uno are shown in Figures 2 and 3 respectively, with full code for each contained in Appendices B and C respectively. The programs were built with the aid of various sources [12] [13] [14] [15].

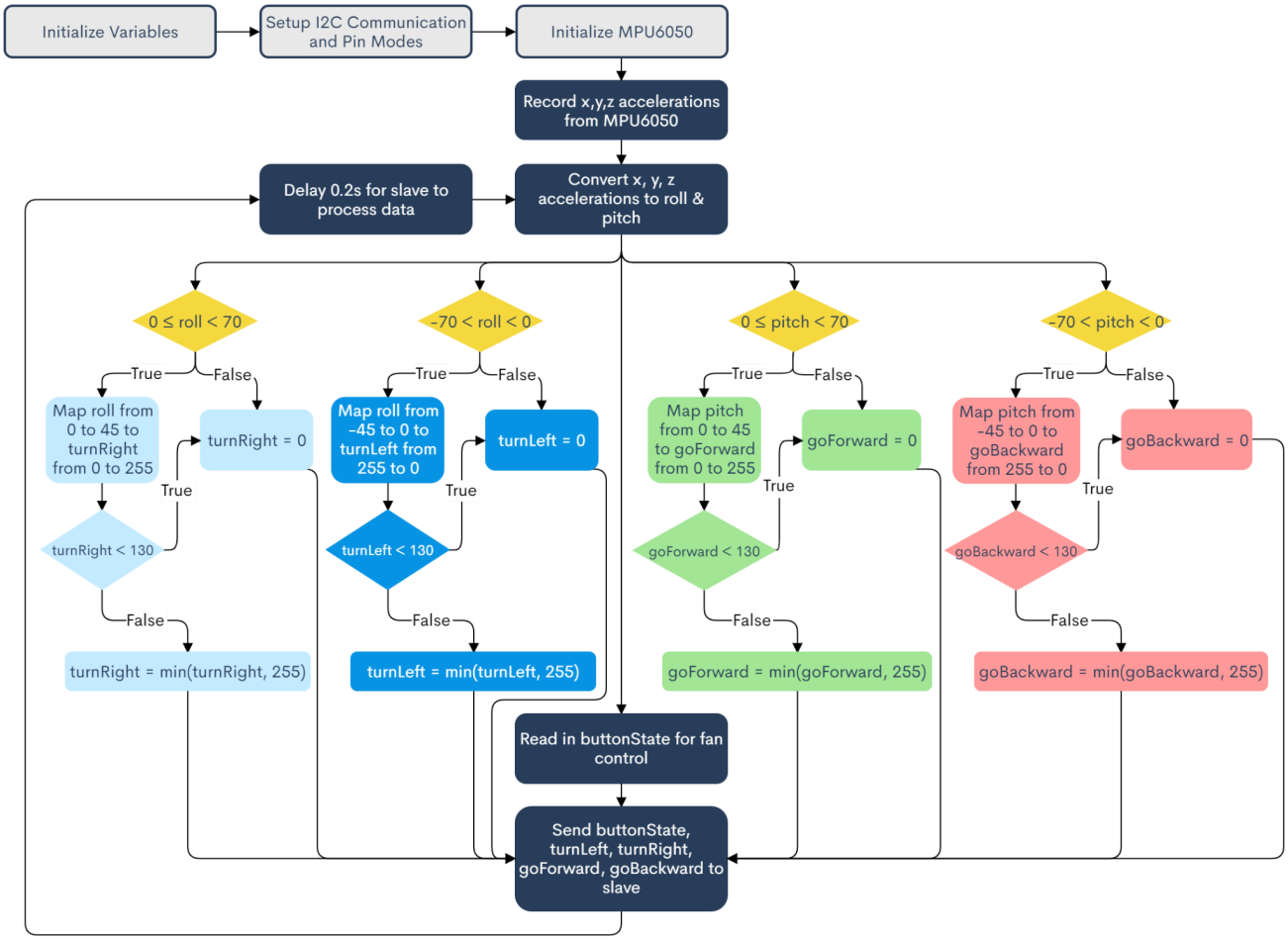


Figure 2: Flow diagram of master program on Arduino Nano

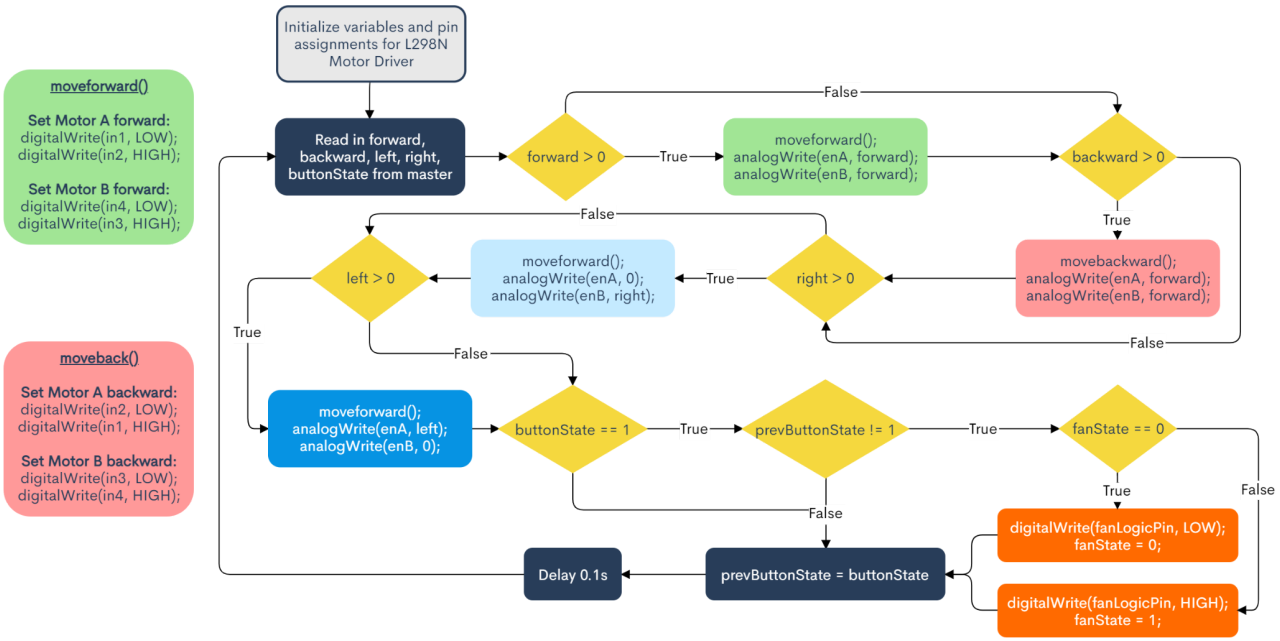


Figure 3: Flow diagram of slave program on Arduino Uno

In general, we wanted to balance code readability with optimizing the CPU cycles used by the program each iteration, so minor optimizations were made. For instance, earlier versions of the code incorporated various functions that did not appear frequently throughout the code (e.g. void setupMPU() and void recordAccelData()). To reduce overhead, these functions were changed to inline functions, which indicates to the compiler that the code within the function should be reproduced and directly called each iteration without the overhead of protecting contents of registers and producing a CALL or RET type instruction to execute the functions.

Control loops are not used in our program to control the PWM signals passed to the motors because during our prototype testing, we found that the existing functionality was adequate and that the rover navigated sufficiently well despite no feedback control being implemented. A state machine approach was not developed for our design due to its fairly simplistic nature in terms of user interaction; all users do is physically move the gyroscopic controller, and the processing steps within our code and as shown in Figures 2 and 3 convert that to desired actuations of the motors to move the rover. The vacuum cleaning function is similar in that the state of a pushbutton switch on the controller dictates whether or not the Arduino Uno on the rover outputs a control signal used to actuate the fan vacuum.

2.2.1 Master Code Structure

The master code is the software implementation of the gyroscopic controller. This code is divided into a high-level message layer, protocol layer, and hardware layer. Since the Arduino IDE contains many helper functions and libraries (such as delay, pinMode, analogWrite, and the Wire and Serial libraries), we were able to abstract away most of the low-level software implementation of the hardware and protocol layers in the code. The description of our software implementation in each layer is as follows:

1. The hardware layer of the master software implementation works to ensure that there are no timing related issues in the communication between devices, and that all I/O pins are defined. First, to ensure all devices communicate with the Arduino Nano at a consistent rate, the Serial command was used. Namely, in the main void setup() function of the code we use Serial.begin(38400) to set all data transmission rates at 38400 bits per second. This includes the communication with the MPU-6050 via I2C, and the HC-05 Bluetooth module via UART. Furthermore, the delay(100) command in Arduino was used throughout the code to solve timing issues in the communication between the master HC-05 Bluetooth module and the slave HC-05 Bluetooth module. This issue is discussed in more detail in Section 2.2.2. Lastly, we defined the pinMode of the Arduino pin 8, which is associated with the pushbutton, as an input since we will want to know the state of the button (pressed or unpressed). If the button is pressed, the Arduino pin 8 will read a voltage of 5V corresponding to a state of 1 and if the button is unpressed/released, the pin will read a voltage of 0V corresponding to a state of 0.
2. The protocol layer of the master software implementation works to facilitate the communication between devices in the gyroscopic controller. Namely, I2C

Communication protocol was used for the Arduino Nano to receive acceleration data from the MPU-6050. The I2C protocol was relatively simple to implement because the Arduino repository offers a convenient library, *<Wire.h>*, to abstract the low-level software implementation of the protocol [16]. For instance in the *void setupMPU()* function of our code implementation, *Wire.beginTransmission(MPU_addr)* of our code implements I2C start conditions and sends the MPU address to the bus to establish communication. *Wire.write(0x6B)* establishes a write to the power management register of the MPU-6050; from here we can use *Wire.write(0x00)* to write to the register and wake up the device, or *Wire.read()<<8|Wire.read()* to read the first byte of that register. *Wire.endTransmission()* then ends the I2C Communication by asserting a I2C stop condition. We developed the functions *void setupMPU()* and *recordAccelRegisters()* to further abstract the I2C Communication protocol. Therefore the *void loop()* of the message layer is completely independent of the type of communication protocol used between the Arduino and MPU-6050 as long as the sensor data is processed correctly.

3. Finally, the message layer of the master software implementation processes the content of the message and then sends it to the slave Arduino Uno device on the rover. Specifically, acceleration data in the form of forces (*xForce*, *yForce*, and *zForce* in the code) is obtained from the MPU-6050 sensor, and is converted to roll and pitch angles using Equations 1 and 2 below [17]. These angles then get mapped to analog values between 0 to 255 that control the duty cycles of the PWM signals sent to the motors. The *map* function built into the Arduino IDE is used to implement this to ensure that the values will be within the proper data representation limits before getting passed to *analogWrite* on the slave [18]. These values are then sent to the slave using the HC-05 Bluetooth module via the integrated UART by a *Serial.write(value)* command. This triggers the master HC-05 Bluetooth module to transmit its data (via electromagnetic waves) which is then decoded as it is read into the slave HC-05 Bluetooth module on the rover.

$$roll = \frac{180}{\pi} \cdot \arctan\left(\frac{G_y}{\sqrt{{G_x}^2 + {G_z}^2}}\right) \quad \textbf{Eq. } 1$$

$$pitch = \frac{180}{\pi} \cdot \arctan\left(\frac{-G_x}{\sqrt{{G_y}^2 + {G_z}^2}}\right) \quad \textbf{Eq. } 2$$

2.2.2 Slave Code Structure

For the slave software implementation on the rover, messages only get received (never sent), and all communications are implemented through hardware. For the slave HC-05 Bluetooth module, the Arduino Uno reads the transmitted message using the *Serial.read* command, and for the L298N Motor Driver module, the Arduino Uno sends voltage signals directly to its pins. The slave script is mostly an implementation of a hardware layer where pins on the Arduino are assigned generic names (e.g. *#define enA 9* in the code abstracts output pin number 9 by assigning it a name, enA, which represents the input pin of the same name on the L298N Motor

Driver). Pins are assigned as either an OUTPUT or INPUT using Arduino's *pinMode* command (e.g. *pinMode(enA, OUTPUT)*). Furthermore, pins IN1, IN2, IN3, and IN4 of the motor driver module are asserted a digital value (to control direction of the motors) using the *digitalWrite* command, and pins ENA and ENB of motor driver module are asserted an analog value (to control speed) using the *analogWrite* command. The *digitalWrite* command assigns a digital high at 5V and a digital low at 0V. In addition to setting motor directions, this is also used to provide a control signal to a transistor circuit to actuate the vacuum fan. The *analogWrite* command implements a PWM signal (via an integrated digital to analog converter within the Arduino) with a duty cycle that is defined by an analog value of between 0, with a 0% duty cycle, and 255, with a 100% duty cycle. This PWM signal controls the speed of each wheel; the larger the duty cycle, the faster the wheels rotate.

2.2.3 Software Testing

During development of the software implementation, we would develop the code parts at a time and test it periodically using the serial monitor of the Arduino IDE. Using the *Serial.println* command we could print variables onto the Serial monitor to see real time values. For instance, we tested the validity of our roll and pitch calculation - from the measured sensor data - by using *Serial.println(roll)* and *Serial.println(pitch)*. For example, Figure 4 shows the results of this in our testing when pitching the gyroscopic controller backward by approximately 40°, then forward by approximately 60°.

```
-------------------------------------------------------
Roll: 6.47
Pitch: -38.27
-------------------------------------------------------
Roll: 1.75
Pitch: -22.12
-------------------------------------------------------
Roll: 1.70
Pitch: -17.89
-------------------------------------------------------
Roll: -1.18
Pitch: -2.36
-------------------------------------------------------
Roll: 0.00
Pitch: 28.51
-------------------------------------------------------
Roll: -1.33
Pitch: 60.72
```

Figure 4: Roll and pitch calculation testing results displayed on serial monitor

As shown in Figure 4, the pitch value remains consistent with our real time motion. To ensure results are as expected, we implemented this same type of test for other variables used in our program. It is important that we conduct these types of tests using the serial monitor first before any hardware because if the software implementation does not behave as expected, this can damage the actual hardware used in our design or it can even cause the hardware to go haywire which could potentially harm people in the immediate area. Once we validated the software implementation of each section of the code, we moved on to testing with the physical hardware. A crucial step for this part of the test was to disconnect the Arduino from our computers since

motors are high powered devices that can draw a lot of current, potentially damaging the Arduino or computers. We ensured that the voltage used by the Arduino was regulated to safe voltage and current levels to prevent damage. For most part, testing the motor went smoothly; when we pitched or rolled the gyroscopic controller, the motor turned in the expected direction and speed to either turn forward, back, right, or left. However early on, we came across an issue regarding the communication between the two HC-05 Bluetooth modules.

One embedded related issue we came across is the timing issues between the communication of the master HC-05 Bluetooth module and the slave HC-05 Bluetooth module; the communication between the Bluetooth modules would experience a linearly increasing latency which resulted in delayed responses in the actuators. We believe the cause of this issue was because the master device is able to send information faster than the slave device can read, process, and implement it. The net effect would be that the data will get backed up on the slave device causing increased latency in the communication. To fix the timing related issues in the communication we implemented a delay of 200 milliseconds. This ensures that the slave device has sufficient time to process the communicated information properly before new information is received. Resolving this issue resulted in a delay in the timeline of this project, so we had to reduce the scope of the project by removing the implementation of a cliff sensor for the rover.

2.3 Mechanical Design

The mechanical design of our project consisted of 3D printing a rover body that could meet 3D printing size and resolution constraints while being sufficiently large enough to hold all desired components. It also consisted of designing a part that could be attached to the rover and capable of sucking up and storing dirt and small items.

We developed two iterations of our design; the former consists of bolts and nuts to support the layers of the rover, and the latter used 3D printed snap fits to attach components. The first design was designed for and printed using the Myhal Makerspace's Prusa MINI printer, and the second design was designed for and printed using the MIE MakerSpace's Dremel 3D40 printer after we realized that the MIE Makerspace does not charge for prints. However, the challenge with the second design was 3D printing annular snap fits required us to double the radius of the columns; the first design had holes for M10-1.5 bolts, but we calculated using [19] that 3D printing columns would require them to be 20 mm diameter (see Appendix D for calculation details). This issue was further compounded by us having to reduce the diameter of the rover body by 20 mm because the maximum width printable in a Dremel 3D40 is 20 mm less than for a Prusa Mini. For this reason, we decided to go with the first design, for which a rendering can be found on the title page, and also linked in Appendix D.

The current pandemic restrictions presented a significant challenge in the 3D printing and procurement of other parts like nuts and bolts. Due to us submitting the 3D printing files a few weeks ago, we were able to pick them up one day before the most recent stay-at-home order came into effect. The last-minute shopping for bolts and nuts also forced us to quickly figure out other compatible bolt and screw sizes based on what we could find; for example, instead of the

original M10-1.5 x 80 bolts to support the rover body and M4 x 40 screws to secure the fan, which could not be found in stores, we used similar sized ⅜ x 3 bolts and #6 x 2 screws.

Unfortunately, an issue we had with the mechanical design is that the suction of debris did not work while the rover was moving because the suction cone was too far from the ground (about 15 mm). However, we did verify that for relatively close distances, the cone worked as intended to suck up debris such as hole-punched paper scraps. Unfortunately we did not videotape this, but we did videotape a receipt being able to stick to the bottom of the cone and stay there as we lifted the rover body higher. In a normal year, we would have printed another cone piece which was longer, but were unable to do so this year due to Ontario's stay-at-home order occurring before we discovered this issue.

3.0 Conclusion

Overall, we feel that our project turned out relatively successfully given the difficulties of physically prototyping while lockdowns were in place, and believe that this project was a great way for us to apply our learnings from the lectures on embedded software development, hardware integration, and communication protocols. With more time and full access to prototype facilities, we believe that we would have successfully been able to finish phase 2 of our design to implement vacuum cleaning while the rover moves; the only change we would need to make that we were unable to do was simply 3D print a new suction cone that was 10 mm longer, or alternatively, procure a more powerful fan. Furthermore, with broader access to prototyping facilities and material procurement, we believe that we could have begun stage 3 of our project to try and implement cliff sensors and distance measurement on the rover to be able to override dangerous user commands.

In terms of hardware changes, a change that we would like to try is to use two NodeMCU ESP32 microcontrollers rather than an Arduino Nano and Uno. This is because these modules are also programmable using the Arduino IDE, but have more memory, as well as built in WiFi and Bluetooth modules for communication between the microcontrollers, or between a microcontroller and other peripherals [20]. This would help reduce the complexity and increase the reliability of our design because it would mean both HC-05 Bluetooth modules in our existing design would not be needed, which would be beneficial given that we had some issues with the master HC-05 Bluetooth module physically disconnecting from the gyroscopic controller at certain tilt angles. Moreover, with access to prototype facilities, developing PCBs and soldering components onto them would be an excellent way to reduce space and increase the reliability of our design. Given that we have verified using breadboards and jumper wires that our circuits function as intended, we believe that developing PCBs to integrate our existing design components would be a great next step.

References

[1] Arduino.cc. 2021. analogRead() - Arduino Reference. [Online] Available at: https://www.arduino.cc/reference/en/language/functions/analog-io/analogread/ [Accessed: 15- Apr- 2021].

[2] ElProCus - Electronic Projects for Engineering Students. 2021. Arduino Uno Board : Features, Pin Configuration, and Its Applications. [Online] Available at: https://www.elprocus.com/atmega328-arduino-uno-board-working-and-its-applications/ [Accessed: 15- Apr- 2021].

[3] arduino. 2021. Digital Pins. [Online] Available at: https://www.arduino.cc/en/Tutorial/Foundations/DigitalPins [Accessed: 15- Apr- 2021].

[4] Lastminuteengineers.com. 2021. Interface L298N DC Motor Driver Module with Arduino. [Online] Available at: https://lastminuteengineers.com/l298n-dc-stepper-driver-arduino-tutorial/ [Accessed: 15- Apr- 2021].

[5] Lastminuteengineers.com. 2021. Interface L298N DC Motor Driver Module with Arduino. [Online] Available at: https://lastminuteengineers.com/l298n-dc-stepper-driver-arduino-tutorial/ [Accessed: 15- Apr- 2021].

[6] Components101. 2021. HC-05 Bluetooth Module Pinout, Specifications, Default Settings, Replacements & Datasheet. [Online] Available at: https://components101.com/wireless/hc-05-bluetooth-module [Accessed: 15- Apr- 2021].

[7] HowToMechatronics. 2021. Arduino and HC-05 Bluetooth Module Complete Tutorial. [Online] Available at: https://howtomechatronics.com/tutorials/arduino/arduino-and-hc-05-bluetooth-module-tutorial/ [Accessed: 15- Apr- 2021].

[8] HowToMechatronics. 2021. How To Configure and Pair Two HC-05 Bluetooth Modules as Master and Slave | AT Commands - HowToMechatronics. [Online] Available at: https://howtomechatronics.com/tutorials/arduino/how-to-configure-pair-two-hc-05-bluetooth-module-master-slave-commands/ [Accessed: 15- Apr- 2021].

[9] Campbell, S., 2021. Basics of UART Communication. [Online] Circuit Basics. Available at: https://www.circuitbasics.com/basics-uart-communication/ [Accessed: 15- Apr- 2021].

[10] HowToMechatronics. 2021. Arduino and MPU6050 Accelerometer and Gyroscope Tutorial. [Online] Available at: https://howtomechatronics.com/tutorials/arduino/arduino-and-mpu6050-accelerometer-and-gyroscope-tutorial/ [Accessed: 15- Apr- 2021].

[11] Invensense.tdk.com. 2021. MPU-6000 and MPU-6050 Product Specification Revision 3.4. [Online] Available at: https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf [Accessed: 15- Apr- 2021].

[12] HowToMechatronics. 2021. Arduino and MPU6050 Accelerometer and Gyroscope Tutorial. [Online] Available at: https://howtomechatronics.com/tutorials/arduino/arduino-and-mpu6050-accelerometer-and-gyroscope-tutorial/ [Accessed: 15- Apr- 2021].

[13] HowToMechatronics. 2021. How To Configure and Pair Two HC-05 Bluetooth Modules as Master and Slave | AT Commands - HowToMechatronics. [Online] Available at: https://howtomechatronics.com/tutorials/arduino/how-to-configure-pair-two-hc-05-bluetooth-module-master-slave-commands/ [Accessed: 15- Apr- 2021].

[14] n.d. ELEGOO 37 SENSOR KIT TUTORIAL. pp.278 - 289.

[15] HowToMechatronics. 2021. Arduino DC Motor Control Tutorial - L298N | PWM | H-Bridge. [Online] Available at: https://howtomechatronics.com/tutorials/arduino/arduino-dc-motor-control-tutorial-l298n-pwm-h-bridge/ [Accessed: 15- Apr- 2021].

[16] Arduino.cc. 2021. Arduino - Wire. [Online] Available at: https://www.arduino.cc/en/reference/wire [Accessed: 15- Apr- 2021].

[17] Nxp.com. 2021. Tilt Sensing Using a Three-Axis Accelerometer. [Online] Available at: https://www.nxp.com/docs/en/application-note/AN3461.pdf [Accessed: 15- Apr- 2021].

[18] Arduino.cc. 2021. map() - Arduino Reference. [Online] Available at: https://www.arduino.cc/reference/en/language/functions/math/map/ [Accessed: 15- Apr- 2021].

[19] Bayer MaterialScience, n.d. *Snap-Fit Joints for Plastic*. [Online] Available at: http://fab.cba.mit.edu/classes/S62.12/people/vernelle.noel/Plastic_Snap_fit_design.pdf> [Accessed: 15- Apr- 2021].

[20] S. Santos, "ESP32 Client-Server Wi-Fi Communication Between Two Boards", *Random Nerd Tutorials*, 2020. [Online]. Available: https://randomnerdtutorials.com/esp32-client-server-wi-fi/. [Accessed: 15- Apr- 2021].

[21] "PLA Technical Data Sheet", *Sd3d.com*. [Online]. Available: https://www.sd3d.com/wp-content/uploads/2017/06/MaterialTDS-PLA_01.pdf. [Accessed: 15- Apr- 2021].

[22] "Comparison of typical 3D printing material", *2015.igem.org*, 2015. [Online]. Available: http://2015.igem.org/wiki/images/2/24/CamJIC-Specs-Strength.pdf. [Accessed: 15- Apr- 2021].

[23] T. Rogers, "Everything You Need To Know About Polylactic Acid (PLA)", *Creativemechanisms.com*, 2015. [Online]. Available: https://www.creativemechanisms.com/blog/learn-about-polylactic-acid-pla-prototypes#:~:text=Plastics%20that%20are%20derived%20from,%2C%20or%20polystyrene%20(PS). [Accessed: 15- Apr- 2021].

Appendix A: Hardware Schematics

Circuit schematics for the controller and rover are shown in Figures A1 and A2 respectively.

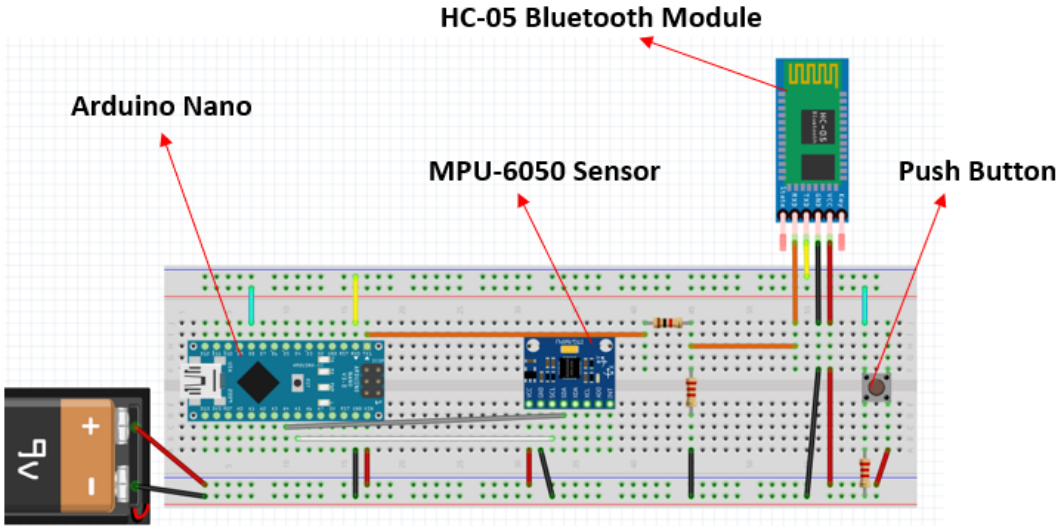


Figure A1: Gyroscopic Rover Controller Circuit Schematic

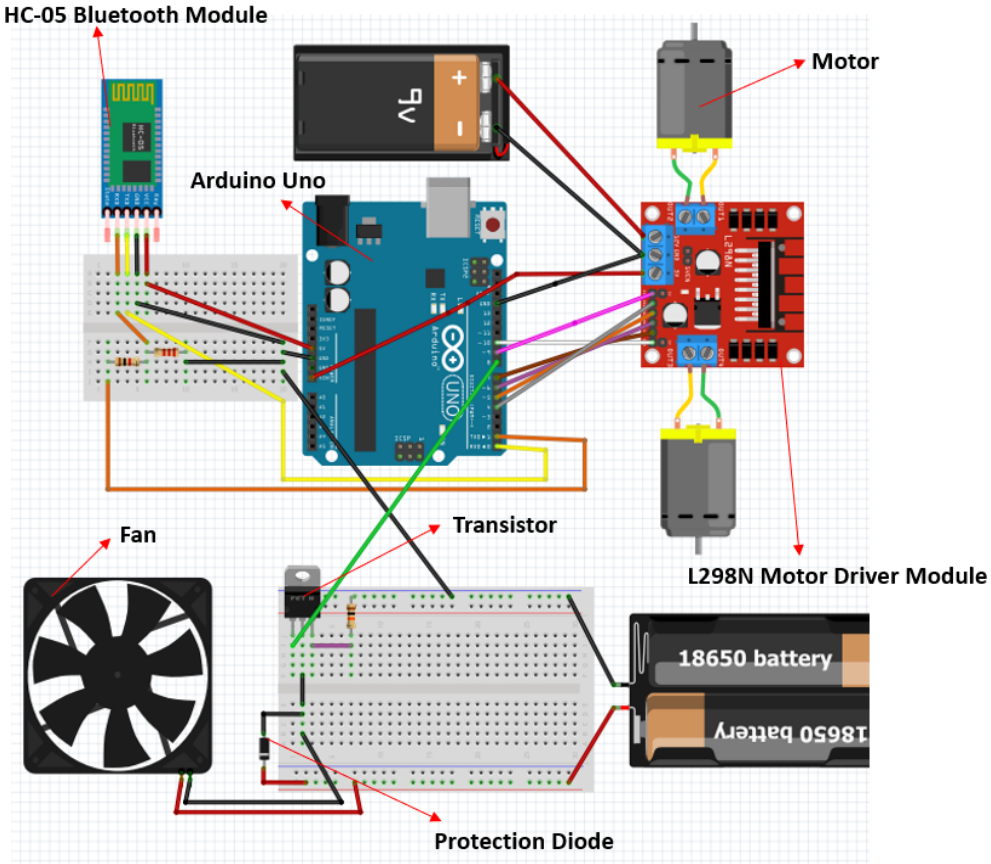


Figure A2: Rover Vehicle Motor and Fan Circuit Schematic

Appendix B: Master Gyroscopic Controller Code

A copy of the controller code can also be found in our team's Github repository.

```
// includes and variable definitions
#include <Wire.h>

#define buttonPin 8 // pin the push button will be connected to

int buttonState = 0; // state is either button pressed (1) or unpressed (0)

// Initialize variables that carry the PWM signal
int turnRight;
int turnLeft ;
int goForward;
int goBackward;

// Initialized Processed Sensor Variables
int xForce;
int yForce;
int zForce;

// I2C address of the MPU ( as shown on data sheet)
// If ADO is driven high then address changes to 0x45 in binary, which makes it
useful to link two of these modules to the same bus
const int MPU_addr = 0x68;

// Initialize pre-processed Sensor Variables
long accelX, accelY, accelZ;
float gForceX, gForceY, gForceZ;


void setup() {

  pinMode(buttonPin, INPUT); // Define the pushbutton pin as input

  Serial.begin(38400); // Default communication rate of the Bluetooth module

  // Begin I2C Comm
  Wire.begin();
  setupMPU();

}

void loop() {

////////////////////////////////////// Process Sensor Data to PWM Signal
/////////////////////////////////////////////////
```

```
  recordAccelRegisters(); //record accel sensor measurements

  // Scale up measured data
  xForce = long(gForceX*100);
  yForce = long(gForceY*100);
  zForce = long(gForceZ*100);


  // Calculate Roll and Pitch (rotation around X-axis, rotation around Y-axis)
  float roll = atan(yForce / sqrt(pow(xForce, 2) + pow(zForce, 2))) * 180 / PI;
  float pitch = atan(-1 * xForce / sqrt(pow(yForce, 2) + pow(zForce, 2))) * 180 /
PI;

  // turn right if tilted right
  if (0 < roll < 70){
    turnRight = map(roll, 0, 45, 0, 255);

    // if PWM signal is too low to actuate motors, set it to 0
    if (turnRight < 130){
      turnRight = 0;
      }

    // Cap values mapped outside PWM signal range
    else if (turnRight > 255){
      turnRight = 255;
      }
    }

  // if roll is outside acceptable range, there should be no PWM signal command
  else{
    turnRight = 0;
    }


  // turn left if tilted left
  if (roll < 0){
    turnLeft = map(roll, -45, 0, 255, 0);

    // if PWM signal is too low to actuate motors, set it to 0
    if (turnLeft < 130){
      turnLeft = 0;
      }

    // Cap values mapped outside PWM signal range
    else if (turnLeft > 255){
      turnLeft = 255;
```

```
    }
  }

// if roll is outside acceptable range, there should be no PWM signal command
else{
  turnLeft = 0;
  }


// Go forward if tilted forward
if (0 < pitch < 70){
  goForward = map(pitch, 0, 45, 0, 255);

  // if PWM signal is too low to actuate motors, set it to 0
  if (goForward < 130){
    goForward = 0;
  }

  // Cap values mapped outside PWM signal range
  else if (goForward > 255){
    goForward = 255;
  }
}

// if pitch is outside acceptable range, there should be no PWM signal command
else{
  goForward = 0;
  }


// go backwards if tilted backwards
if (pitch < 0){
  goBackward = map(pitch, -45, 0, 255, 0);

  // if PWM signal is too low to actuate motors, set it to 0
  if (goBackward < 130){
    goBackward = 0;
  }

  // Cap values mapped outside PWM signal range
  else if (goBackward > 255){
    goBackward = 255;
  }
}

// if pitch is outside acceptable range, there should be no PWM signal command
else{
```

```
    goBackward = 0;
    }


  buttonState = digitalRead(buttonPin); // last but not least, read the button
state


  // Send information to slave if values are different from previous values
 if ( goBackward|| goForward || turnLeft  || turnRight ){

   Serial.write(goForward);
   Serial.write(goBackward);
   Serial.write(turnRight);
   Serial.write(turnLeft);
   Serial.write(buttonState);

   delay(100); // delay so slave has time to receive and process the data

 }

 // Send information to stop the drive when the controller is leveled
 else if ( (goBackward == 0 && goForward == 0) && ( (turnLeft == 0 && turnRight ==
0)  ) ){


   Serial.write(0);
   Serial.write(0);
   Serial.write(0);
   Serial.write(0);
   Serial.write(buttonState);

   delay(100); // delay so slave has time to receive and process the data
  }

  delay(100); // slow down the loop

}


////////////// Establish Comm with MPU and setup registers////////////////////
inline void setupMPU(){

  // Begin transmission to the slave device on the I2C bus
  Wire.beginTransmission(MPU_addr);
  //Accessing the register 6B for Power Management
  Wire.write(0x6B);
```

```
  //Setting all bits to 0 to wake up device
  Wire.write(0x0);
  // End the transmission to device
  Wire.endTransmission();

  Wire.beginTransmission(MPU_addr);
  //Accessing the register 1C for Accelerometer Configuration
  Wire.write(0x1C);
  //Setting the accel to +/- 2g,Meaning gyro can only detect forces up to 2g
  //See data sheet to change the range if project requires detection of greater
force
  //The greater the range the lower the sensitivity
  Wire.write(0x0);
  Wire.endTransmission();
}

//////////////////////////////////////////////////Read Sensor Readings
////////////////////////////////////////////////////////////
inline void recordAccelRegisters() {
  Wire.beginTransmission(MPU_addr);
  //Starting register for Accel Readings
  Wire.write(0x3B);
  Wire.endTransmission();
  //Request Accel Registers (3B - 40)
  Wire.requestFrom(MPU_addr, 6);
  while(Wire.available() < 6);
  accelX = Wire.read()<<8|Wire.read(); //Store first two bytes into accelX
  accelY = Wire.read()<<8|Wire.read(); //Store middle two bytes into accelY
  accelZ = Wire.read()<<8|Wire.read(); //Store last two bytes into accelZ

  // process data in the registers
  gForceX = accelX / 16384.0;
  gForceY = accelY / 16384.0;
  gForceZ = accelZ / 16384.0;
}
```

Appendix C: Slave Rover Code

A copy of the rover code can also be found in our team's Github repository.

```
// Motor pins
#define enA 9
#define in1 4
#define in2 5
#define enB 10
#define in3 6
#define in4 7

#define fanLogicPin 2 // Fan Pin

// Received data from master controller
int forward = 0;
int backward = 0;
int right = 0;
int left = 0;
int buttonState = 0;

int fanState = 0; // 1 if the fan is on, 0 if the fan is off

int prevButtonState = 0; // the previous buttonState value

void setup() {

  pinMode(fanLogicPin, OUTPUT); // set fan pin as output

  // Motor control pins
  pinMode(enA, OUTPUT);
  pinMode(enB, OUTPUT);
  pinMode(in1, INPUT);
  pinMode(in2, INPUT);
  pinMode(in3, INPUT);
  pinMode(in4, INPUT);

  Serial.begin(38400); // Default communication rate of the Bluetooth module
}

void loop() {
 while (Serial.available() > 3){ // Checks whether data is coming from the serial
port
    forward = Serial.read(); // Reads the data from the serial port

    Serial.print("Forward: ");
    Serial.println(forward);
```

```
    backward = Serial.read(); // Reads the data from the serial

    Serial.print("Backward: ");
    Serial.println(backward);


    right = Serial.read(); // Reads the data from the serial

    Serial.print("Right: ");
    Serial.println(right);


    left = Serial.read(); // Reads the data from the serial

    Serial.print("Left: ");
    Serial.println(left);


    buttonState = Serial.read(); // Reads the data from the serial

    Serial.print("Button State: ");
    Serial.println(buttonState);


    delay(100);


    Serial.println("-----------------------------------------------------");

}
// Send low level control to motor driver to move forward
if (forward) {

    moveforward();
    analogWrite(enA, forward);
    analogWrite(enB, forward);
    Serial.println("Going Forward!");

 }

// Send low level control to motor driver to move backward
if (backward) {

    moveback();
    analogWrite(enA, backward);
    analogWrite(enB, backward);
```

```
    Serial.println("Going Backward!");

  }

// Send low level control to motor driver to move right
if (right) {

    moveforward();
    analogWrite(enA, right);
    analogWrite(enB, 0);
    Serial.println("Going right!");

  }

// Send low level control to motor driver to move left
if (left) {

    moveforward();
    analogWrite(enA, 0);
    analogWrite(enB, left);
    Serial.println("Going left!");

  }

  // Stop if the controller is leveled
  if (forward==0 && backward==0 && left==0 && right==0) {

    moveforward();
    analogWrite(enA, 0);
    analogWrite(enB, 0);
    Serial.println("Stop!");

  }

  // Turn the fan on if the button is pressed
  if (buttonState==1 && fanState==0 && prevButtonState != 1){

    Serial.println("Turn Fan on!");
    digitalWrite(fanLogicPin, HIGH);
    fanState = 1;

  }

  // Turn fan off if the button is pressed again
  else if (buttonState==1 && fanState==1 && prevButtonState != 1){

    Serial.println("Turn Fan off!");
```

```
    digitalWrite(fanLogicPin, LOW);
    fanState = 0;

  }
  prevButtonState = buttonState;

 delay(100);
}

void moveforward(){
    // Set Motor A forward
    digitalWrite(in1, LOW);
    digitalWrite(in2, HIGH);

    // Set Motor B forward
    digitalWrite(in3, HIGH);
    digitalWrite(in4, LOW);
}

void moveback(){
    // Set Motor A backward
    digitalWrite(in1, HIGH);
    digitalWrite(in2, LOW);

    // Set Motor B backward
    digitalWrite(in3, LOW);
    digitalWrite(in4, HIGH);
}
```

Appendix D: Mechanical Design & Related Figures

Renderings of our design in addition to the one on the title page can be found here. Figure D1 shows the 3D printed cone that was developed to suck dirt and small debris. SOLIDWORKS files for mechanical parts, the entire assembly, and stl files for 3D printing can be found here, with v3 referring to the design that uses bolts and nuts for supporting the rover, and v4 referring to the design that uses annular snap fits. Figures D2 and D3 show the assembled controller and rover. Finally, Table D1 shows the design calculations we performed for the snap-fit column design; the original spreadsheet used for these calculations can be found here. Videos of the rover moving and of the suction effect can be found here; unfortunately due to the inability to prototype further and modify the cone to be closer to the floor, suction was limited while the rover was moving.

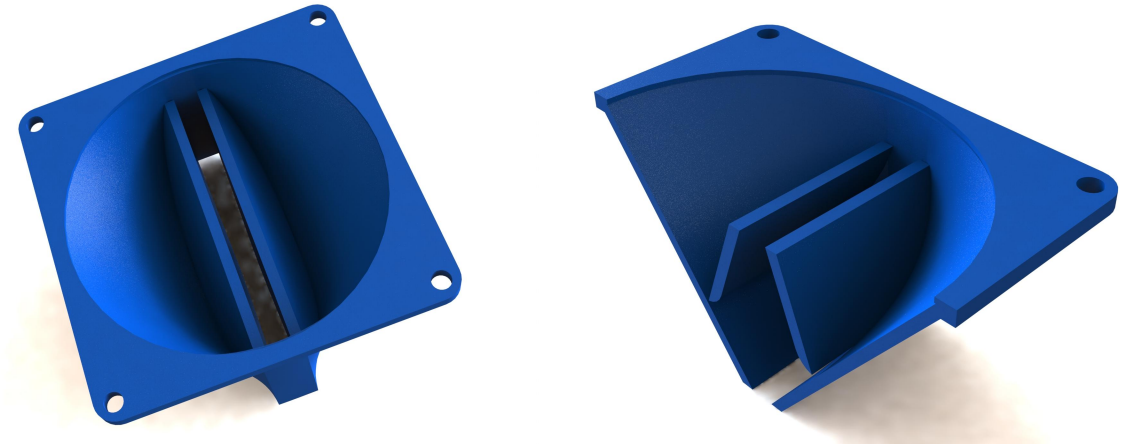


Figure D1: Cone attachment for sucking up debris (left) and cross-section (right)

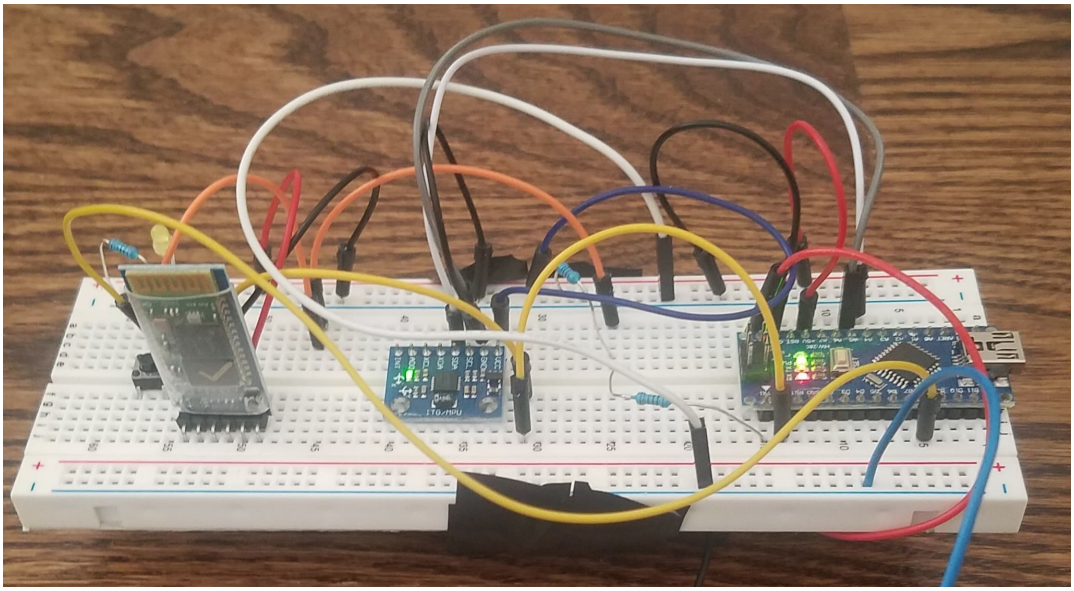


Figure D2: Assembled gyroscopic controller

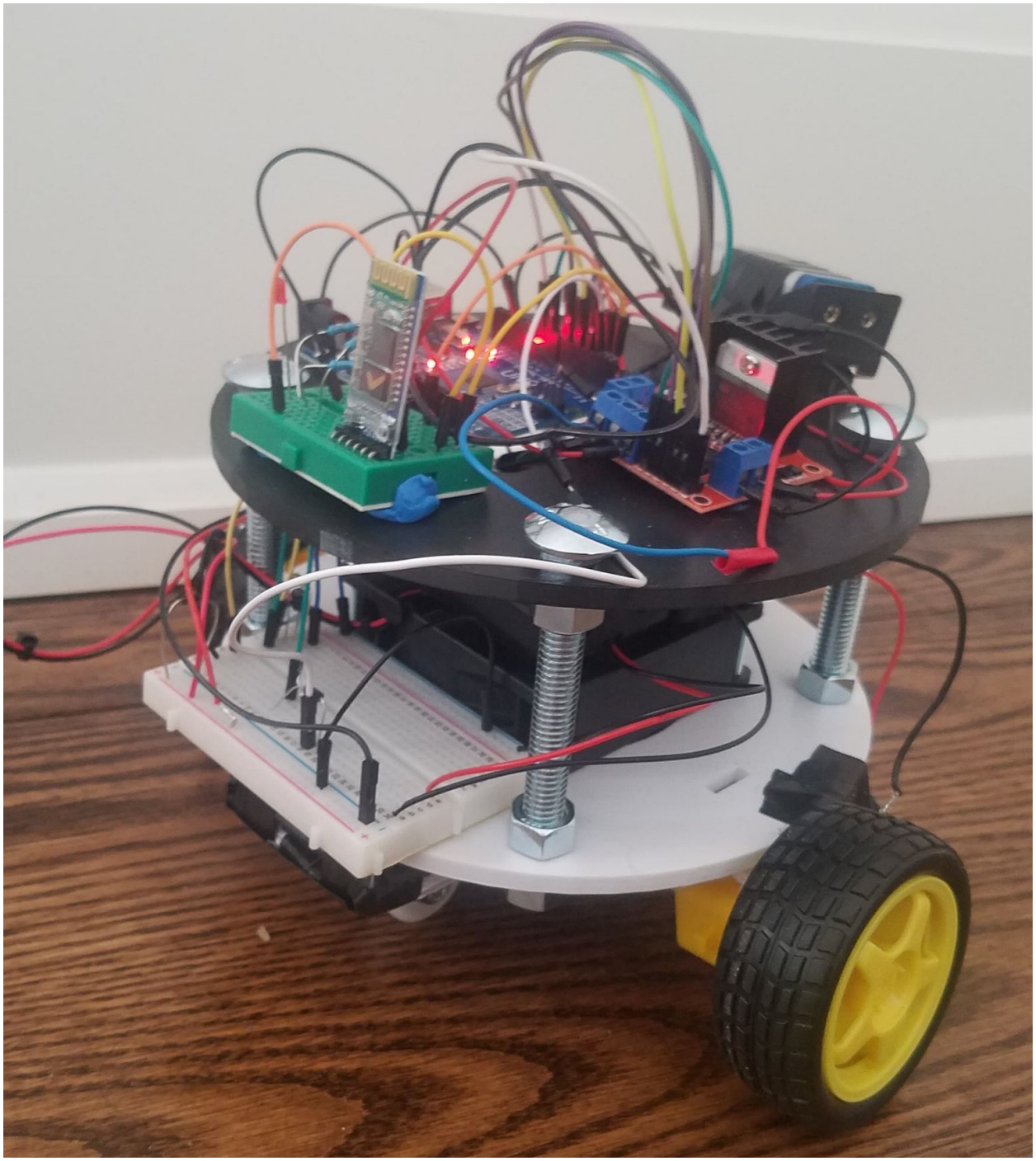


Figure D3: Assembled Rover

Table D1: Snap-Fit Design Calculations (bolded values are calculated values)

Variable	Value	Units	Notes/Assumptions
ε_pm	0.02		Use strain at yield to prevent plastic deformation. Value from [21]
d	0.02	m	
y_pm	**0.0004**	m	Values chosen such that y/2 satisfies 3D printer resolution (see Fig. 22 of [19])
E_s	3500000000	Pa	Assume secant modulus is elastic modulus to prevent permanent deformation. Value from [22]
X	0.019		From Fig. 27 of [19], d0/d = 11.5/10 = 1.15. Assuming rigid shaft, X_N is relevant geometric factor, which is approximately 0.019 from the chart
μ	0.4		Assume value is in between PP, PE, PS based on [23], which states PLA is similar to all of these. Coefficients are given in [19], and range from 0.2 to 0.6
α	30	deg	Designed as such because Fig. 22 of [19] shows it is permissible, and would need less overhang for 3D print
P	**532**	N	See Fig. 25 of [19], force corresponding to hoop stress in the materials
W	**676**	N	See Fig. 25 of [19], force required for mating