Final Report

Autonomous Tractor Project

ECE 2804

Instructor: Peter Han

Spring 2023

Prepared by:

Rakesh Pillai

Sanjiv Rao

https://github.com/rxkesh/AT_Project.git

Introduction

Automation has been a cornerstone of technological development in the 21st century. Applications of software systems and new advancements, such as artificial intelligence, allow for the conversion of manual processes into fully automated systems. Tasks that were once achieved through large amounts of human labor can be executed more efficiently through the usage of technology. One field where the potential for automation cannot be understated is agriculture. Agricultural goods constitute the fundamental basis of the global food supply and are arguably one of the most critical industries on the planet. The large amounts of repetitive labor required for agriculture make it an ideal target for application of autonomous systems. In order to simulate the development of such a system, this project was conducted.

The overarching goal of this project was to design and create a prototype for an autonomous tractor that would be capable of maneuvering through a predefined course. The tractor needed to possess the ability to drive straight, autocorrect its pathing using a gyroscope, communicate through bluetooth, turn 90 degrees, detect black tapes on the ground, perform an e-stop when encountering an obstacle, as well as run all these tasks concurrently and autonomously. Alongside the tractor's base functionality, it also had to be able to communicate with a Python-based GUI which could send commands to the tractor and generate a report of relevant information. These goals were the focus of the design process and by the end of the project tenure, all relevant goals were achieved.

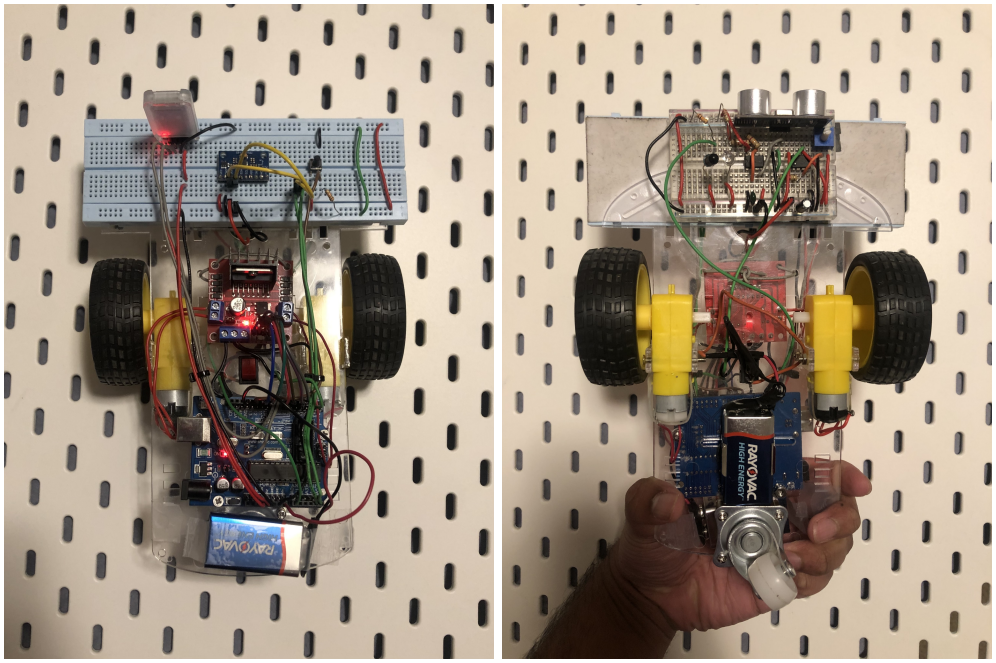


Figure 1 : Top view and bottom view of final tractor construction

High Level Design

Figure 2: Block Diagram of Entire System

The main parts to the our autonomous tractor are the: two motors, two wheels, the robot chassis, Arduino Uno R3 microcontroller, L298N Motor Controller, HC-SR04 Ultrasonic Sensor, HM-10 BLE Bluetooth module, MPU6050 gyroscope, Infrared Emitter and Receiver diodes, a push button, and two 9 Volt batteries. Other extraneous materials include two breadboards of varying size, 9V battery holders, 9V battery connectors, 3M tape, fastening hardware, a TLC555CP low power time, an LF356N op amp, resistors and capacitors of varying values, and jumper cables. Tools required for the construction of this robot may include a screwdriver, pliers, multimeter, computer, and soldering iron. Software required for this project include the Arduino IDE, VS Code for running Python scripts, Waveforms for circuit analysis, and the DSD Tech app for establishing a bluetooth connection during testing.

The Arduino acts as the main microcontroller responsible for organizing and instructing all aspects of the robot. Each subsystem is connected to the Arduino in some way or form, allowing all functionality of the robot to be controlled from one central unit. The AT_Base code provides the software foundation for all processes done in the robot. By uploading code to the Arduino, the Arduino receives the instructions for each of the subsystems and it is capable of communicating with every part of the tractor. The AT_GUI is the user interface where one can start and stop the robot with buttons as well as view a trip report after the robot is done moving. The GUI will be presented locally on the user's computer, but it will have a built-in bluetooth connection to the HM-10 Bluetooth module on the robot. If the robot is not on or the Bluetooth module is somehow unpowered, the GUI will not function as expected and the robot will not obey instructions.

Looking at physical aspects of the tractor, there's two main components: the movement and the functionality. The movement circuits are the motor controller and the motors and they are the basis for the robot's movement. Each motor is connected to the motor controller which then receives instructions from the Arduino about speed and direction for the wheels. The functional circuits are the "accessories" that enable the autonomous movement of the robot. The gyroscope is a passive component which constantly provides the robot a navigational sense of its position. This is used to ensure the robot can correct itself when driving straight and when making accurate turns. The push button and HM-10 Bluetooth module are active components that function as a peripheral for the tractor. If the button is pressed or the Bluetooth module receives a command, the robot will respond to those external stimuli. Similarly, the ultrasonic sensor and IR sensor function as inputs for the tractor and they provide sensory information about its environment. The ultrasonic sensor detects obstacles ahead of the robot, to prevent it from having collisions with obstacles in its environment. This module specifically is adjustable so that the user can manually select the range that which obstacles should be detected. The IR sensor circuit allows the robot to have a relative approximation of the color of surface it is driving above. When the robot detects it is driving over a dark or black surface, it will turn accordingly.

In terms of design choices, the parts utilized in this project were chosen for a variety of reasons. Primarily, most of the individual components are made for low-power consumption and designed to function modularly. The IR sensor module for example could be placed on any side of the robot and still function, given it has a surface to bounce waves off of. It was sensible to place the ultrasonic sensor

Detailed Design

An image of the final overall circuit is displayed below. Detailed schematics of complicated circuits are provided in the following section.

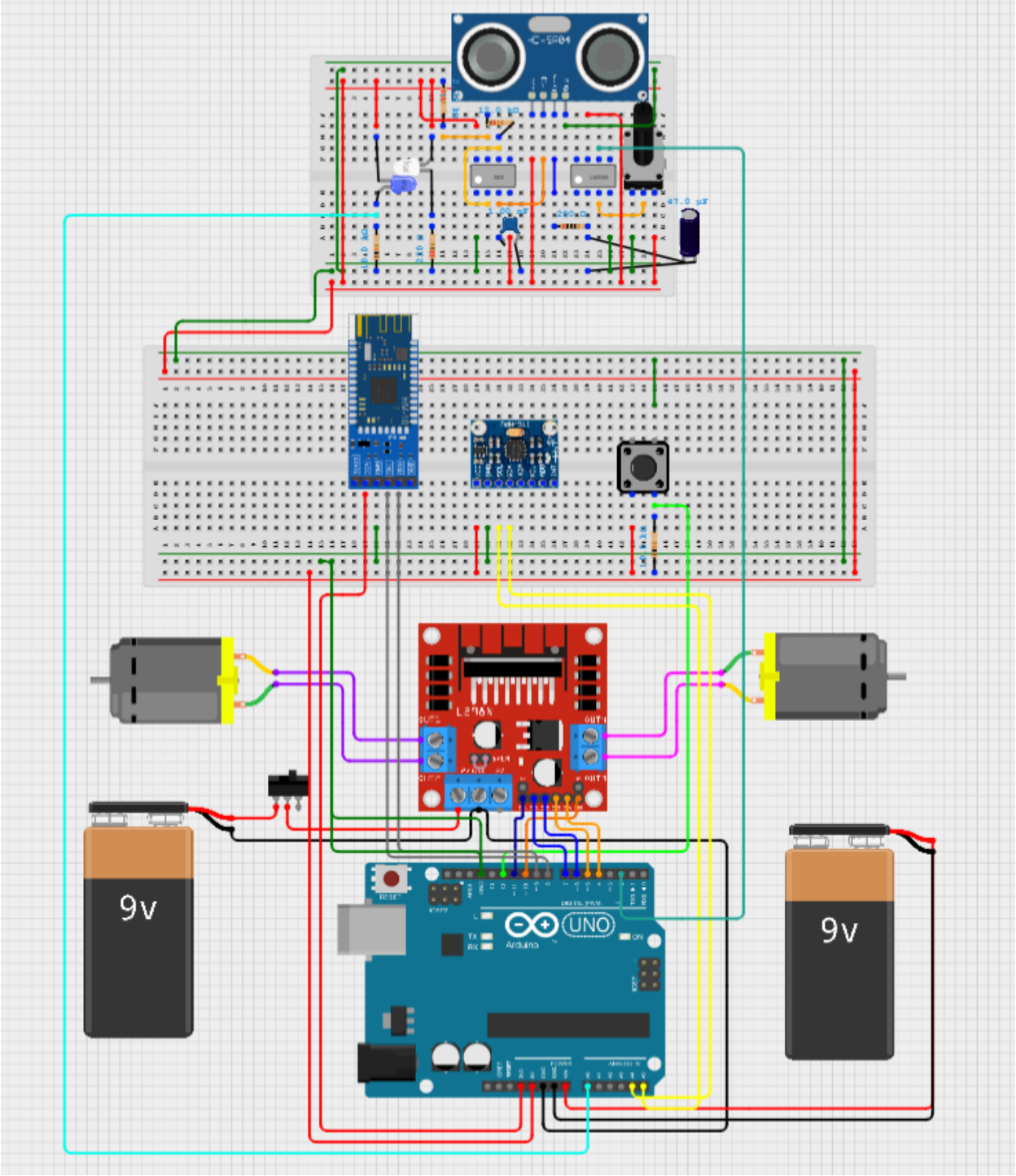


Figure 3: Overall integrated circuit diagram

In terms of power, a 9V battery is connected to the Vin pin of the Arduino to supply power to the microcontroller. This falls within the optimal 7-12V range for the Arduino Uno R3. Similarly, a 9V battery is connected to a switch and consequently the voltage in of the L298N motor controller. This powers the motors and the motor controller can handle up to 12V of voltage. The choice to utilize two 9V batteries was made to accommodate for the fact that a single 9V battery would provide insufficient power for the entire system. Initially, the project materials included a battery pack for 4 AA batteries. Provided with 1.2V AA batteries, the total voltage of 4.8V was not nearly enough to power the motor controller and Arduino. Thus, the decision was made to power the Arduino and motor controller separately, as the motor controller required a dedicated power supply due to its power consumption.

The main and most integral system on the robot is the motor control system. This consists of the L298N motor controller and the motors connected to it.

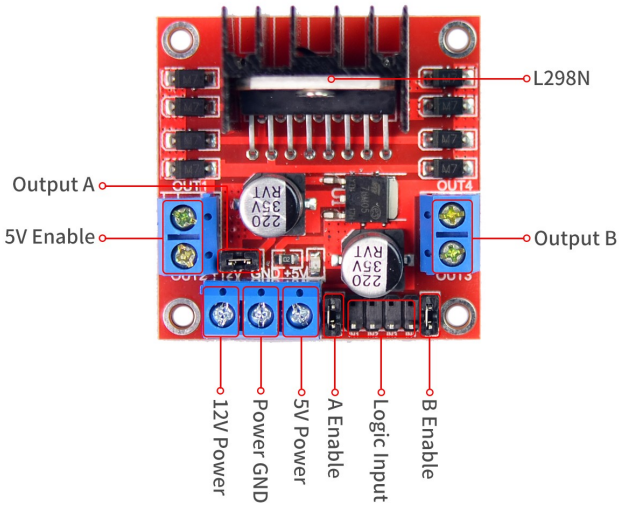


Figure 4: Pinout diagram of L298N motor controller
Credit:https://www.navestar.com/p/l298n-motor-drive-controller-board-module-dual-h-bridge-dc-stepper-motor-driver-for-arduino-raspberry-pi-smart-robot-car/

The motor controller allows for the bidirectional control of 2 motors at up to 2 Amps. On the pinout diagram, the microcontroller has 4 control input pins and 2 enable pins. Inputs 1 and 2 correspond to Output A on motor A. Similarly, Inputs 3 and 4 correspond to Output B on motor B. These control pins are utilized for controlling the direction of the motors. If motor movement is desired, only one input can have a HIGH signal. Based on whether it is pin 1 or 2, the direction of rotation will be either clockwise or counter-clockwise. If both inputs are set to HIGH or LOW, the motors will not move. This principle also

applies to pins 3 and 4 respectively. Pins 1-4 on the motor controller will be connected to digital pins 7-4 on the Arduino. Alongside this, Enable A and B are utilized for motor speed control. By default, there is a jumper between the 5V source and enable with the desired effect of achieving maximum speed. Our goal was to have varied speed control, so this jumper was removed and digital PWM pins 11 and 10 were connected to enable A and enable B respectively. This allowed for PWM control of the motor speeds, which was an integral part of future algorithms. Initially, the 5V supply was also utilized to power the Arduino, but after further testing it was found that without a 12V supply the motor controller would effectively experience a 5V drop in supply. This resulted in the motor controller only realistically receiving around 4V, which was below what was expected. This was another important factor towards the reasoning behind electing to have 2 independent voltage sources.

Following the motor control subsystem came the design of the basic start-stop functionality of the robot. This came in the form of a physical push-button circuit. This circuit consisted of a 10K Ohm resistor and a tactile push button switch. These components are connected to a 5V supply and ground.

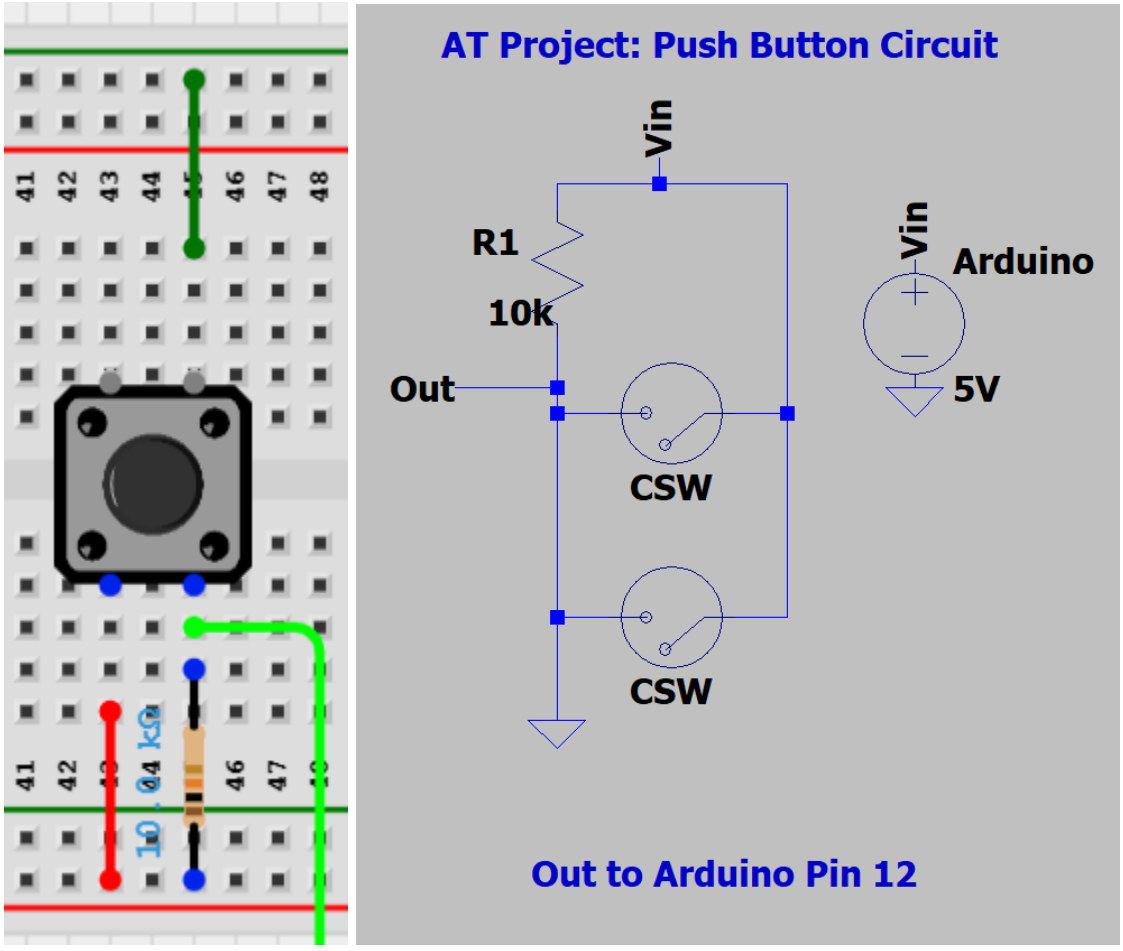


Figure 5 : Electronic depiction and schematic of the push button circuit

To understand the functionality of the circuit, one must first understand the functionality of the push button itself. Depicted below is a simple graphic exhibiting how the push button behaves. The button acts as two connected switches which are both activated when the switch is pressed. We designed the circuit around this and supplied 5V to one of the pins and no connection to the opposite pin. On the other end, a pull-down resistor was added with a ground on the opposite pin. The reasoning behind this circuit is that if a digital read is made at the resistor, while the button is released it will constantly read 0V since no voltage is applied on the pin; resulting in an output LOW. Once the switch is pressed though, the 5V is then electrically connected to the pull-down resistor by the switch, resulting in the digital read seeing 5V and thus output HIGH. There was not a specific component for the button switch in LTSpice, the software used for schematic generation, so it was simulated with two switches.

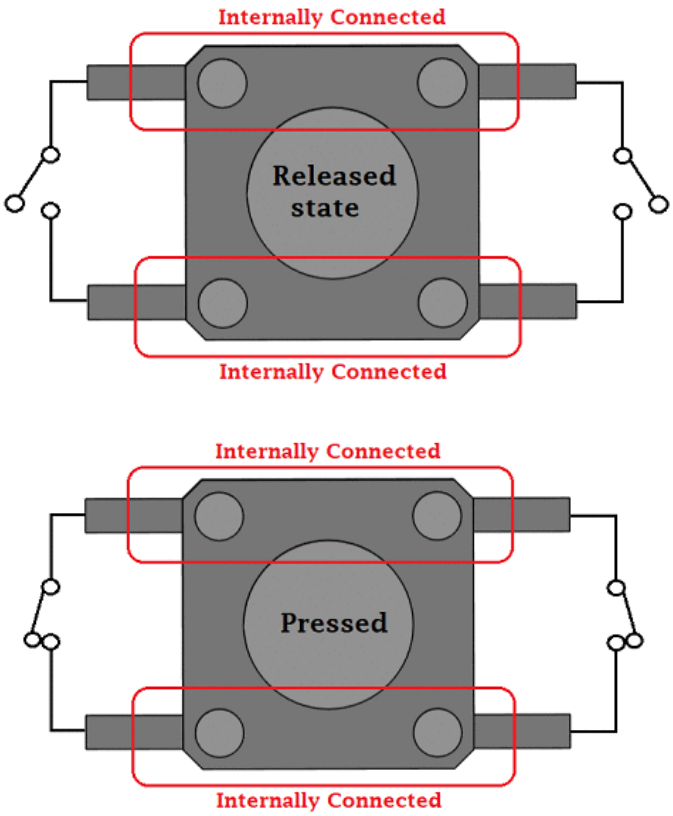


Figure 6 : Diagram of push button functionality
Credit: https://components101.com/switches/push-button

Using the fact that the Arduino pin will generate a high state whenever the button is pressed, the switch was debounced using software. In order to debounce the switch, a finite state machine was made with 2 different states that the switch was in, either PUSHED or RELEASED. The Arduino was coded to read pin 12 as a digital input and the value was stored in a variable labeled “buttonRead”. Based on the

value of “buttonRead”, the software identifies whether the switch is pressed down or not. In order to ensure the button only sends one command at a time, a boolean value labeled “buttonCommand” is established. This value is set to true when one button press is inputted and it does not switch to false until the overall tractor state machine transitions out of the state it is currently in. With this functionality in place, the robot was then able to start and stop based on button presses. Refer to Appendix A.

The next subsystem that was implemented was the gyroscope. Comparatively, the circuitry for this system was quite simple as it only consisted of soldering the pins to the gyroscope and inserting it into the top of the breadboard. There are only 4 relevant pins for the gyroscope, them being VCC, GND, SCL, and SDA. The VCC pin was connected to the 5V rail of the breadboard, the GND pin was connected to ground similarly, and the SCL and SDA pins were connected to Arduino pins A5 and A4 respectively. Note that the gyroscope needs to be soldered flat and parallel to the board, else the gyroscope will measure all the values with an offset that will not be accounted for by the software. This will generate error when taking angular measurements.

In order to implement the gyroscope into the code, we needed to identify a way to read data through Arduino. We did so by applying the “MPU6050_light.h” library and including it in our code base. This library, developed by Romain JL Fetick, facilitated fast and simple communication with the gyroscope by doing much of the calculations for the angles and tilt through built-in computation. The library accepts the raw data outputted by the gyroscope and computes the angular speed, acceleration, and angular displacement from initialization.

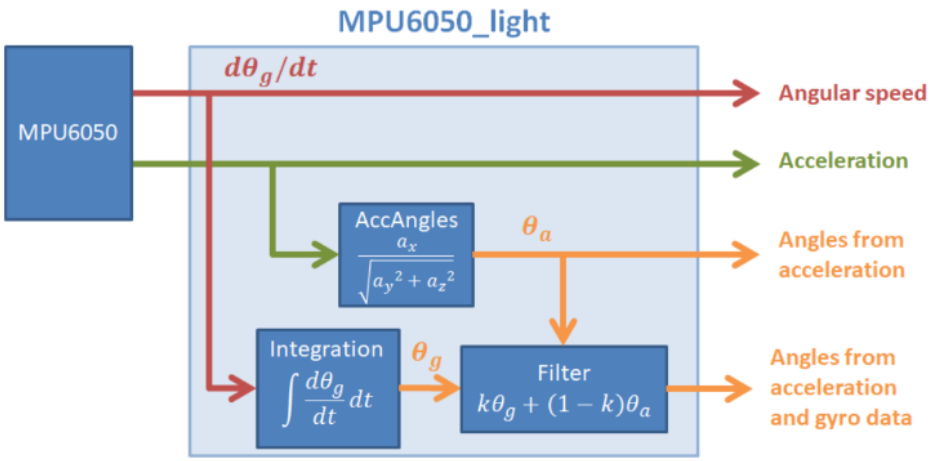


Figure 7 : Overview of the data accessible through the MPU6050_light library
Credit: https://github.com/rfetick/MPU6050_light/blob/master/documentation_MPU6050_light.pdf

By applying this library, we were able to easily acquire the angle about the z-axis at any time by calling the “getAngleZ()” function which returns a float value of the angular displacement. We measured the z-axis due to the orientation of our robot and the placement of the gyroscope on the breadboard.

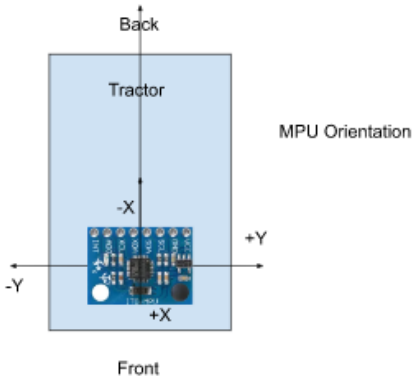


Figure 8 : Visualization of the gyroscope and orientation in terms of the tractor

In order to apply the library, certain setup functions need to be called to properly construct and initialize the gyroscope. The “MPU6050_light.h” library required the “Wire.h” library to function properly, so that was also included. This allowed the SCL and SDA analog pins to be read and converted into the data we were attempting to utilize. Upon calling the appropriate setup functions, the gyroscope was ready to be used in the code base. Refer to Appendix B.

The next system was the bluetooth module. Similar to the gyroscope, this was relatively simple to wire. The bluetooth module communicates through UART communication with an operating voltage of 2-3.7V. The 3.3V output of the Arduino was connected to VCC of the bluetooth module, GND was wired to ground, and the TX and RX pins were connected to pins 9 and 8 respectively. In order to utilize them in the Arduino, the RX pin needed to be set as an input and the TX pin as an output. This was done because the RX pin is how the Arduino receives the data from the bluetooth module and the TX pin is how data is transmitted to the HM-10. Alongside that, a SoftwareSerial object was generated for the bluetooth module so helper functions could be called at any point. From there, a list of commands were defined that could be sent or received by the bluetooth module. These included “off”, which makes the tractor enter the OFF state, “on”, which makes the tractor enter the MOVE state, and “turnL”/”turnR” which will make the robot turn in the respective direction of right or left. To read commands from the bluetooth module, first the Arduino checks if a command is available using the “mySerial.available()” function. If there is one available, the Arduino then reads the command and stores it in a variable. A simple if statement is in place to filter known commands and consequently change the state machine. Refer to Appendix C for the setup and read function.

With the current subsystems in place, the robot was able to correct its course using the gyroscope, start and stop from phone commands, as well as start and stop with the push button. To continue fulfilling the guidelines for the project, the robot needed to be able to detect the black tapes of the course it was following. To do so, an infrared sensor circuit was devised and placed on the underside of the robot. This circuit functions by having an IR emitter and a receiver. The emitter sends out infrared waves which bounce off of objects and excite the photodiode allowing it to pass a small reverse-bias current. The reasoning behind this circuit is that infrared waves are sensitive to color. When waves are bounced off a light surface, the majority of the waves are reflected due to incident light bouncing off of light colored surfaces. If the surface they are bouncing off of is instead a darker color, a larger portion of the incident light is absorbed by the surface and less is reflected back. Due to this behavior, the photodiode will allow a larger current when placed above a lighter object compared to a darker surface. This principle is the basis for the IR sensor circuit.

This circuit consisted of an IR emitter LED, a photodiode, and two resistors of 10K and 220 Ohm resistances. The IR LED accepts around 30mA of maximum current, so a 220 Ohm resistor was placed in series to limit the current through the right branch to about 22.7mA. The photodiode resistor is slightly different where it instead decides how sensitive the photodiode is to infrared waves. By adjusting the resistance repeatedly, we experimentally derived that the ideal resistance for our experiment was approximately 10K Ohms.

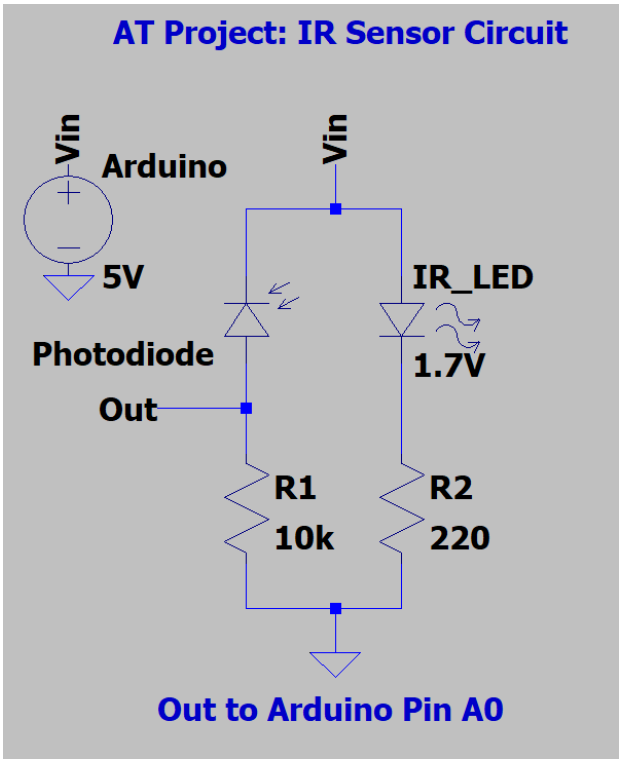


Figure 9 : Schematic of IR sensor circuit

Using this setup, a wire was connected in between the photodiode and resistor to the Arduino pin A0. Applying an analog read at this pin provided a value from approximately 0 to 1000. Above a light surface, this value would be approximately in the 800 to 900 range. Comparatively, above a darker surface this value would decrease to around 400 to 600 depending on the darkness of the surface. This allowed us to establish a threshold value that could be used to determine the color of the surface below the robot. Using a conditional statement and variable to read the value at any given point in time, the robot could then switch states to the appropriate turn state. The course we were provided with had an established start and stop point that the robot would have to progress through. The direction of each turn and the total number of turns was fixed. Due to this fact, a conditional within the conditional statement was implemented. Everytime the detection algorithm notices a black line, a variable storing the number of tapes detected thus far is incremented. Based on this value, the robot is instructed to either turn right or left. Refer to Appendix D for the IR sensor algorithm.

Our final subsystem is the ultrasonic sensor. To create the standalone obstacle detector, we used a 555 timer to send a pulse to the trigger pin of the ultrasonic sensor. We chose our resistor and capacitor values such that the pulse wave generated spends 12.3 milliseconds high and 10.4 milliseconds low. Using the following equations: $T_H = 0.693 \cdot (R_1 + R_2) \cdot C_1$ and $T_L = 0.693 \cdot R_1 \cdot C_1$, we determined $R_1 = 2.7k\,\Omega$, $R_2 = 12k\,\Omega$, and $C_1 = 47\mu F$. With the astable 555 timer sending pulses to the trigger pin, we needed to create a circuit to read the pulses being sent from the echo pin. We made a low-pass filter to attenuate the unnecessary high frequencies and convert the pulses into DC signals that can be compared. Using the formula, $D = (speed\ x\ time) / 2$, we found that the echo pulse would be high for approximately 1100 microseconds if it detected an object 20 centimeters away. Furthermore, we used the Arduino pulseIn() function to determine how long the signal would be low for (approximately 2180 microseconds) regardless of whether there's an object in front of it. This information allowed us to determine the period of the pulses being sent from the sensor. We estimated the largest frequency our tractor would need to read to be 0.4760 Hz. Using the equation $f_c = \frac{1}{2\pi RC}$, we calculated R to be approximately 7 thousand Ohms and C to be 47 micro Farads. This netted us a cutoff frequency of 0.4980 Hz.

Using LTSpice, we simulated 3 different pulses being sent to the filter:

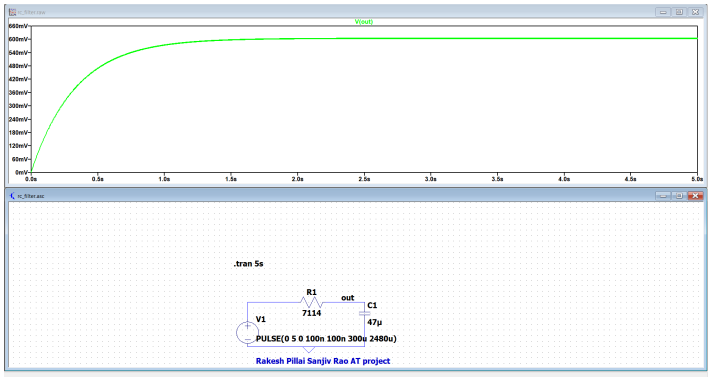


Figure 10.1: V(out) if ultrasonic sensor detects object very close and echo pin sends a pulse with a small duration

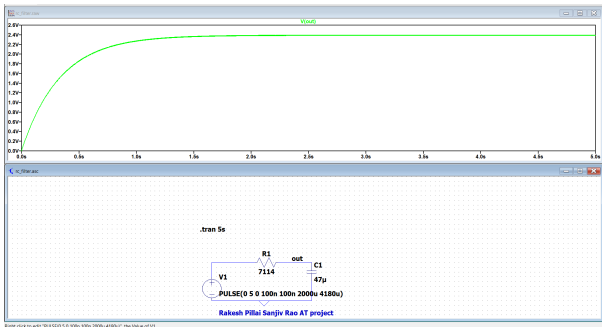


Figure 10.2: V(out) if ultrasonic sensor detects object with a approximately 30 centimeters away and echo pin sends a pulse with a medium-sized duration

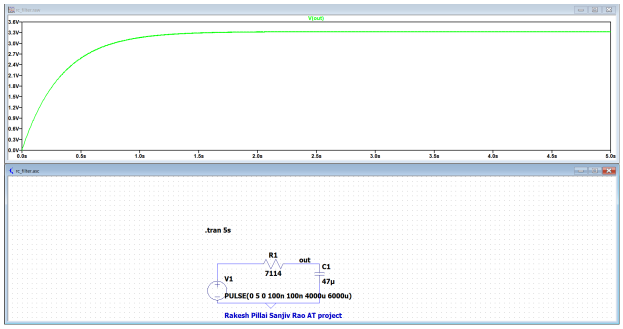


Figure 10.3: V(out) if ultrasonic sensor detects object very far away and echo pin sends a pulse with a large duration

As shown above, the low-pass filter converts the pulse into a signal resembling DC. With the low-pass filter created, we built a voltage comparator with a LF356 Op-Amp and a potentiometer as a voltage divider. The potentiometer is used to customize the voltage that is able to enter the positive input of the op-amp. This also allows us to change the distance at which the ultrasonic sensor will sense objects. Refer to Appendix E for circuit schematic.

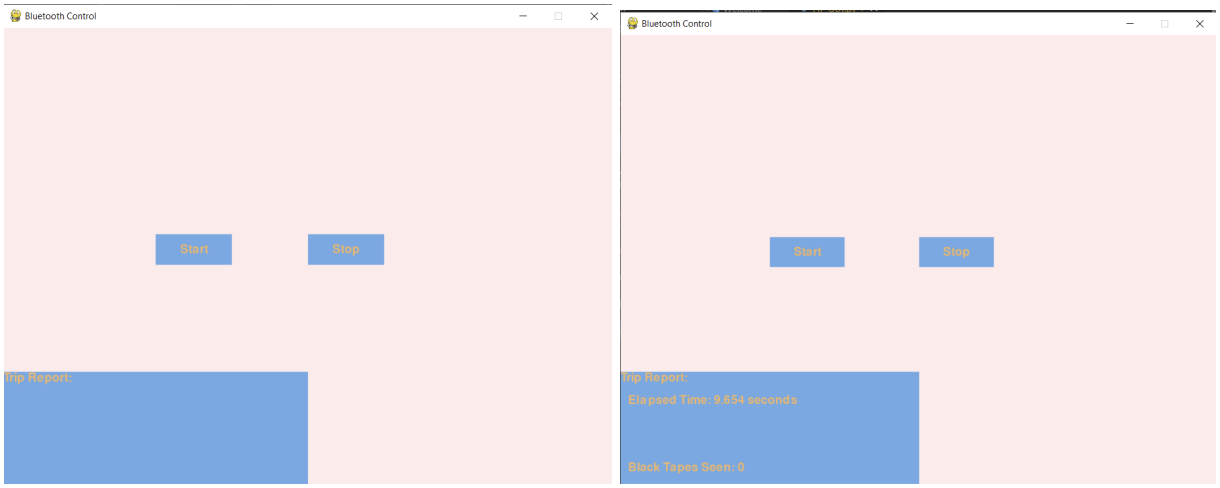


Figure 11: Screenshots of Python-based GUI (Black tapes are actively shown in the video)

Finally, for our Python-based GUI, we took a simple approach. There are two buttons and there is a section for the trip report. The trip report is generated after the robot is stopped and it displays the elapsed time and the number of black tapes the infrared sensor has passed. It took us multiple iterations to reach this product. We started with complex designs, but we trimmed our design to this simple screen since we found that PyGame and Bleak can easily cause conflicts with each other. We determined the MAC address of our module to be “94:A9:A8:3B:14:2F” and its characteristic UUID to be "0000FFE1-0000-1000-8000-00805F9B34FB". We used Python’s *asyncio.run()* function to call Bleak’s asynchronous functions. We used Bleak’s *connect()* function to connect to the bluetooth module and we used *write_gatt_char()* to send on and off commands in byte arrays to the Arduino. We also used *start_notify()* and *stop_notify()* to receive trip data (i.e. black lines seen) from the Arduino.

One of our initial goals at the beginning of the semester was to receive real-time data. However, we had a lot of trouble using *start_notify()* more than once. The trouble lies in Bleak’s use of asynchronous functions. It was difficult for us to find a point in our code where we can safely call an asynchronous function in a loop fashion without causing issues with PyGame. We spent hours with Hayden King, the Teaching Assistant for the autonomous tractor, but unfortunately, we were unable to access real-time data without causing a conflict with PyGame. We suspect the conflict arises from the way we configured our PyGame structure. With more time, we would certainly look into optimizing our PyGame structure and integrate real-time data access.

Validation

Circuit Position	Supply Voltage (V)
Breadboard Rail	4.99
Underside Breadboard Rail	4.99
HM-10 VCC Pin	3.306
Push Button (Pressed)	5
Push Button (Released)	0
IR Output (White)	4.81
IR Output (Black)	3.41
Ultrasonic Output (No Object)	1.42
Ultrasonic Output (Object Present)	3.53

Table 1 : Voltage readings from multimeter under various conditions

The voltage readings at various positions in the robot were measured utilizing a multimeter. The purpose of this test was to ensure that the expected voltages were being supplied at different parts of the tractor alongside measuring behavior of the circuit subsystems. The breadboard rails displayed the approximate 5V which was expected due to the precision of the Arduino +5V pin. Since both rails shared the same supply, it confirmed that all subsystems were receiving the appropriate voltage. The batteries were also tested, and with a small margin of error, both displayed slightly less than 9V. The VCC pin of the HM-10 Bluetooth module was measured to have approximately 3.3V, ensuring that the Bluetooth module would not receive an excessive amount of voltage. In testing the push button, the expected behavior was a HIGH output (5V) when the button was pressed and a LOW output (0V) when the button was not pressed. The measurements reinforced this behavior. In testing the IR circuit, there was a noticeable difference in voltage when the robot was above a white surface and a black surface. It is of note that the IR sensor circuit is volatile and can behave slightly differently depending on the conditions of the environment it is in. In a well-lit room it performs ideally, but in the absence of light or presence of other IR emitting devices it can behave unexpectedly. The ultrasonic sensor on the other hand was very consistent and provided a distinct drop in voltage from when an object was and was not present.

Now that the individual functionality of each subsystem had been verified, the overall tractor was validated through the course provided by the project requirements. Inserted below are several validation videos which demonstrate the robot's ability to fulfill the requirements.

https://drive.google.com/file/d/1QtbdTRGn_ydzwKj4cZ9VzRa_lsoHunQV/view?usp=share_link
Video 1: Robot sensing black tapes and completing course while self correcting. Also includes demonstration of GUI and trip report.

https://drive.google.com/file/d/1UGH_WWmJvYzLg-IKSM2mmCY2vmIfY3Qt/view?usp=share_link
Video 2: Demonstration of push button working as expected and pathfinding correcting the robot's course

https://drive.google.com/file/d/1bH482wK-3rTKM-SXizZXSJAC4fC9KuL1/view?usp=share_link
Video 3: Demonstration of tractor stopping when encountering an obstacle

Conclusion

This project was very instructive about the intricacies of the engineering design process. Every aspect of the tractor from its physical components, circuitry, and software contained their own set of nuances associated with the solution. The smallest oversight could result in dramatically different experimental results than what is expected. The fact that we had to delve into various aspects of the design process and were exposed to challenges of different forms was very educational. There was a strong sense of satisfaction whenever we spent hours on a subsystem and finally got it to work properly. It was very enjoyable to see the culmination of our work traverse the obstacle course we had been designing around for a semester. The unique aspect of this project is that we had to deal with challenges in terms of physical construction, electronics, and software bugs. It was generally one line of code or one misplaced wire which led to entire subsystems not working. This brought to light the importance of repetitive testing and a fine attention to detail in terms of every part of the project. If we had an opportunity to redo this project, the first thing we would have done was to properly understand each subsystem we wanted to implement and physically where we would place it on the tractor. Specifically when it came to the IR sensor, we had placed it somewhat on the left-hand side of the robot instead of centered underneath. This resulted in the robot sometimes missing the black tapes because the sensor was too far to the side. Reimplementing the sensor and making room around the ultrasonic sensor was a painstaking process which made us realize the importance of knowledgeable planning. Alongside this, we realized that we had to be very careful when working with a cloud repository for a collaborative project. Oftentimes we would find ourselves overwriting each other's code, and going through the edits to find the functioning code was unnecessary time we spent not working on other aspects of the project. Overall, this project provided us an opportunity to experience the struggles and triumph of the engineering design process.

Authorship

Introduction: Sanjiv Rao

High-Level Design: Rakesh Pillai

Detailed Design: Sanjiv Rao

Validation of Overall Project: Sanjiv Rao and Rakesh Pillai

Conclusion: Rakesh Pillai

Appendix

```
// Button switch debounce FSM
switch (buttonState) {
  case PUSHED:
    buttonRead = digitalRead(buttonPin);
    if (!buttonRead){
      buttonState=RELEASED;
    }
    break;
  case RELEASED:
    buttonRead = digitalRead(buttonPin);
    if (buttonRead){
      buttonState = PUSHED;
      buttonCommand = true;
    }
    break;
}
```

Appendix A : Code snippet of button debounce finite state machine

```
#include <MPU6050_light.h>
#include "Wire.h"

// Constructor for gyroscope
MPU6050 mpu(Wire);

/*=============================================================================================
Function: This function calls all the components of the gyroscope setup based on the
MPU6050_light.h library. A simple delay is included to allow the gyroscope enough time to
properly calibrate. If connected to the Arduino IDE, the function will display "Done!" in
the serial monitor once calibration is complete. It also will establish the initial value
of z.

Args: None

Return: None

Notes: This function is called only once in setup. It was put into a function for code
cleanliness and ease of debugging.
=============================================================================================*/
void gyroSetup(){
  Wire.begin();
  mpu.begin();
  Serial.println(("Calculating offsets, do not move robot"));
  delay(1000);
  mpu.calcOffsets(true,true); // gyro and accelero
  mpu.setFilterGyroCoef(0.98);
  Serial.println("Done!\n");
  z_init = mpu.getAngleZ();
}
```

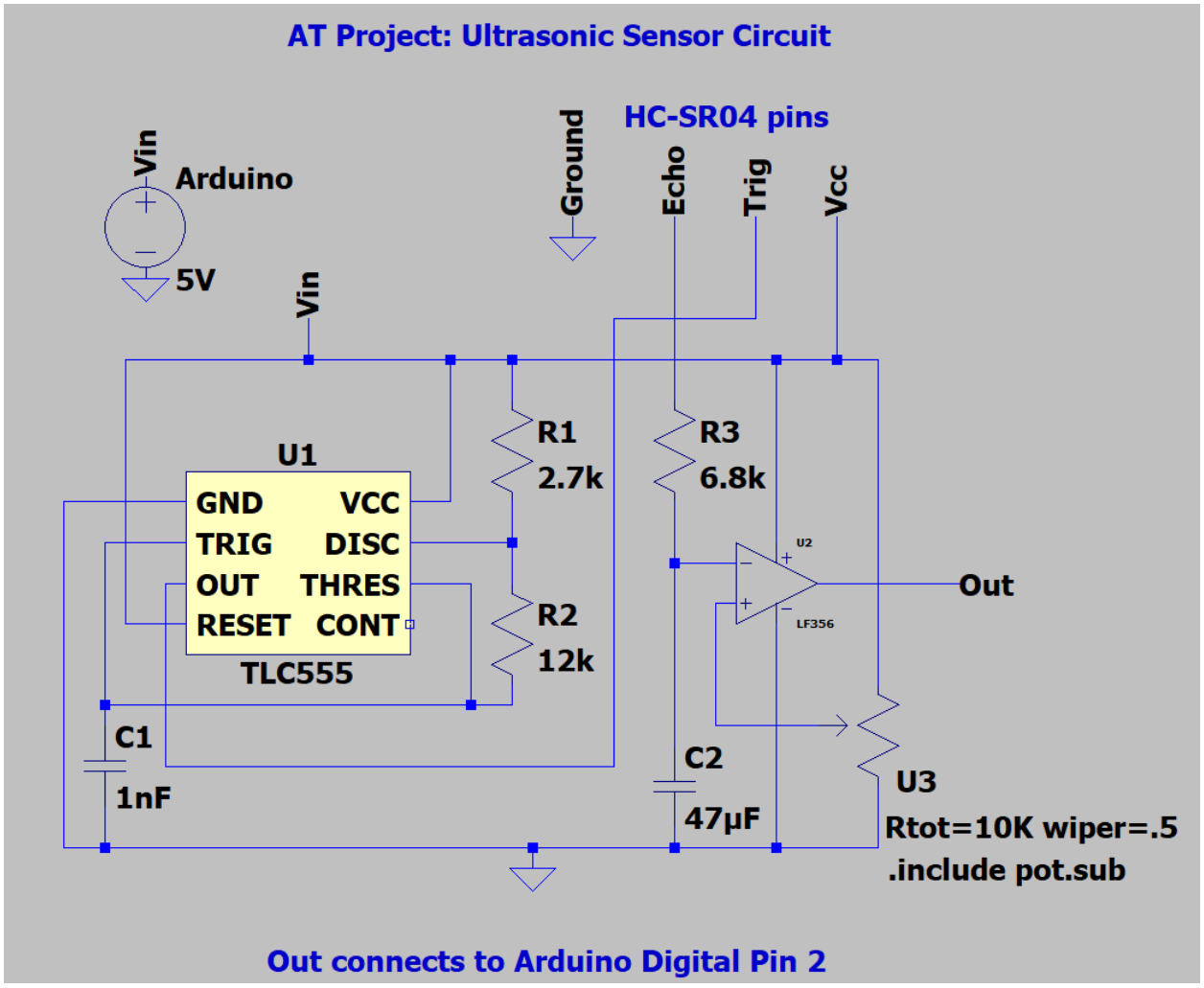
Appendix B : Snippets of gyroscope setup function and other necessary lines of code

```
#include <SoftwareSerial.h>

//bluetooth module communication pins
#define rxPin 8
#define txPin 9

// Creates an instance of a SoftwareSerial object
SoftwareSerial mySerial(rxPin,txPin);

/*=============================================================================================
Function: This function checks if a command is available in the bluetooth module. If so,
it reads the string and prints it to the serial monitor. Alongside this, it filters the
command to see if it is one of the predefined robot commands. If the command is the off
command, it will send the number of black tapes encountered to the bluetooth module.

Args: None

Return: None

Notes: This is how the bluetooth module sends the number of black tapes to the GUI.
=============================================================================================*/
void read_command() {
  if (mySerial.available()) {
    cmd = mySerial.readString();
    Serial.print("Command: ");

    if (cmd == "off") {
      Serial.println("Turning off Robot");
      mySerial.print(numTapes);
      //mySerial.print(distTraveled);
    }
    else if (cmd == "on") {
      Serial.println("Turning on Robot");
    }
    else if (cmd == "left") {
      Serial.println("Turning left");
    }
    else if (cmd == "right") {
      Serial.println("Turning right");
    }
    else {
      Serial.println("Invalid command");
    }

  }
}
```

Appendix C : Snippets of bluetooth module setup and application

```
/* Black line detection:
 * This section of code reads the value on the irPin and checks if it
 * is below a certain threshold for the black tape. If the value is
 * below the threshold, the robot registers the black tape and switches
 * to the appropriate state for the course provided. Based on the number
 * of tapes, the robot will turn right or left.
*/
irRead = analogRead(irPin);
// Detects the reflectivity of surface
if ((irRead < 700)){
  numTapes++;
  if (numTapes<=2){
    //turn left
    currentState = TURN_L;
    mySerial.println("LEFT");
    break;
  }
  else if (numTapes>=3 && numTapes<=4){
    //turn right
    currentState = TURN_R;
    mySerial.println("RIGHT");
    break;
  }
  else{
    //turn left
    currentState = TURN_L;
    mySerial.println("LEFT");
    break;
  }
  mpu.update();
  break;
}
```

Appendix D: Snippets of black line detection algorithm



Appendix E: Ultrasonic sensor circuit

Appendix F: Direct link to Python GUI codebase and Arduino codebase

Python GUI codebase: https://github.com/rxkesh/AT_Project/blob/main/python_gui/AT_GUI.py

Arduino Codebase: https://github.com/rxkesh/AT_Project/blob/main/AT_Base/AT_Base.ino