Mouse design report

Max Grenie(11370) Harry Staplehurst(11587)
Eugene Levinson(11699)

Abstract

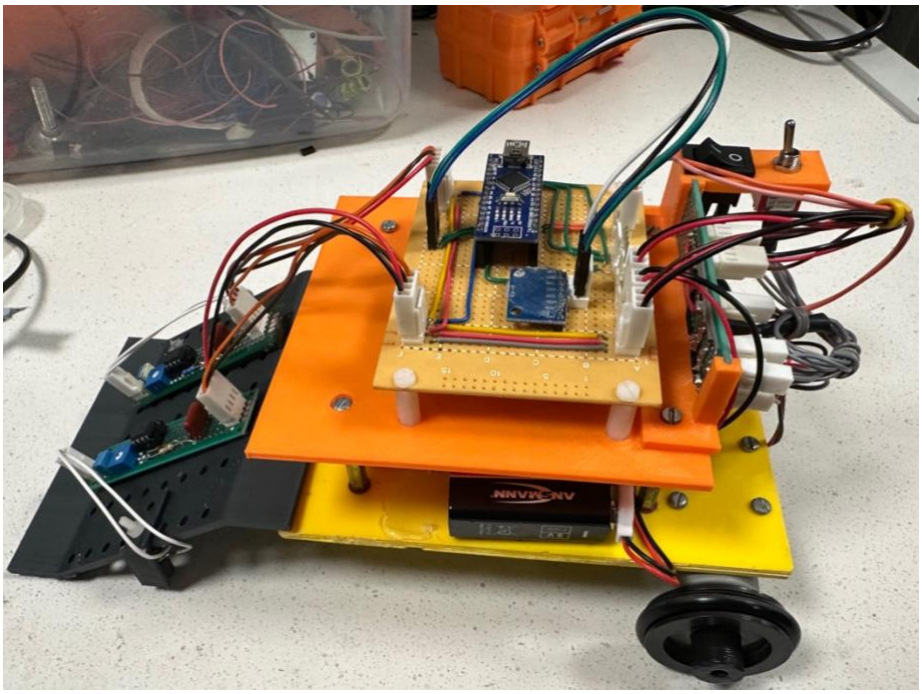
In this report the results of the group EEST 1's attempt to create a line following mouse with the information that was learnt in the current and previous semesters. The report includes explanation of the digital and analogue systems that the mouse used and what was learnt from the project about teamwork in a group of engineers. The project was difficult but created an immense learning experience and greatly advanced all the group member's knowledge about the subject of creating a control system and team management.

Introduction

Problem overview

The exercise was to design a line follower that could follow a track with a wire that emitted a 20kHz signal. The only restrictions were that only one set of batteries could be used on race day, the mouse had to steer using a differential steering system, that there could be no outside help and that the mouse had to cost less than 30£.

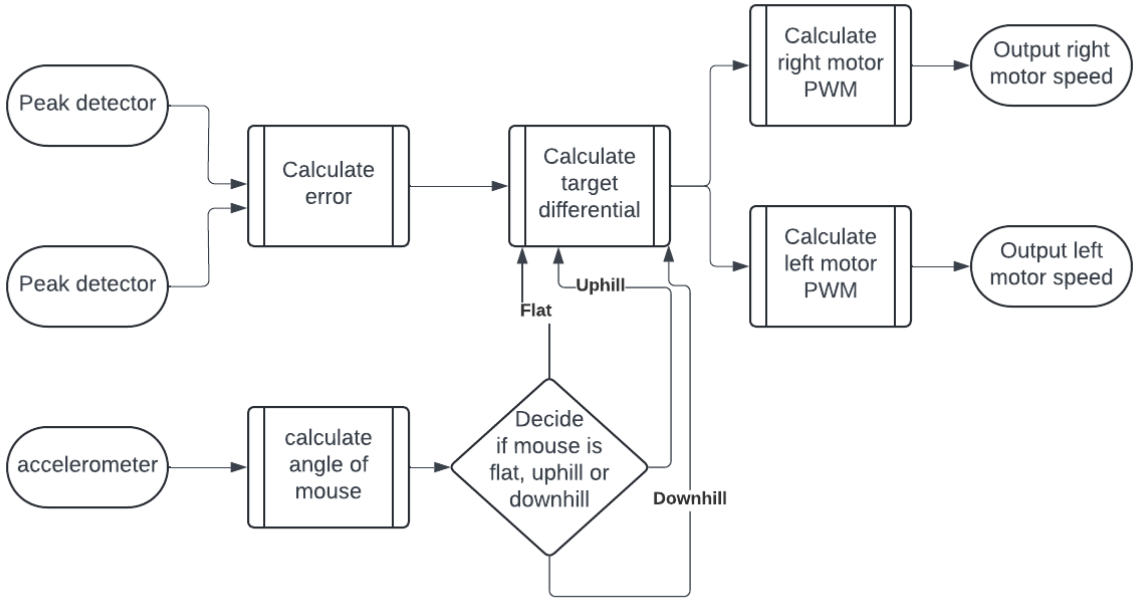
Goals set by group



Before any work started the group set a goal of having as little hard coding in the code as possible. Making the mouse theoretically able to run on any track that it was set on. This meant no timers to set different speeds or settings for the mouse. This goal was successfully completed as the mouse ended up steering around track, accelerated on the straight and out of corners and managed to change its speed according to if it was going uphill or downhill all based on information gathered from its components.
In terms of physical goals, the team wanted to the mouse to look clean and have all circuits available for easy tweaking or replacement. As can be seen in the image above all the goals were achieved.

General control and drive system architecture

Code flowchart



PID control system

For the steering of the vehicle, it was decided that PID control was to be used. PID control is an algorithm that uses 3 feedback variables to control in this case, the steering of a mouse with the input of two peak detectors. The system works by setting a base speed and having the values for P, I and D that were calculated change it depending on what is needed.

$$Target\ Differential = P + I + D$$

P is defined by the error between the sensors in the current clock cycle and is calibrated by a constant Kp. If Kp is too high the mouse will steer too hard when there are small changes in the error between the sensors and cause oscillations which slow down the mouse. If Kp is set too low, then the mouse will not steer enough and potentially lose the line that it is trying to follow.

$$P = Kp * (Right_{sensor\ value} - Left_{sensor\ value})$$

I is a variable that is proportional to the sum of all errors that have been recorded it is calibrated by a constant Ki. With a correct value of Ki, the mouse will be centred above the line when moving in a straight line. If Ki is set too high or low the mouse will trend over one side of the lin. The goal of Ki is to compensate for the peak detectors having different responses to distance from the line.

$$I = Ki * (Sum\ of\ Errors)$$

Lastly there is the parameter D, calculated by subtracting the difference between the current error and the last error and multiplying that by the calibrating factor Kd. If Kd is set too high, then the mouse will oscillate too much and if it is set too low the mouse will not steer enough. D is used to dampen the oscillations that arrive from increasing P to large values. This is

necessary as the mouse needs sharper steering at higher speeds but still needs to be able to control itself when moving in a straight line.

$$D = Kd * (Error - Last\ Error)$$

V component

The team found that the mouse had issues going around turns when at high speeds so an extra V variable was added to create a customised PIDV control system. V is variable that slows down the mouse when entering a turn and accelerates it when leaving the turn to give the vehicle better control. It is calculated by multiplying the target differential by a constant Kv. This addition was extremely effective as it allowed to team to greatly increase the speed of the mouse.

$$V = Kv * (Target\ Differential)$$

Finally, the speeds of the left and right motor are calculated by adding half of the calculated differential to one motor and subtracting the other half from the other motor.

$$Left\ Motor\ Speed = BasePWM + LeftMotorCalibration + \frac{TargetDifferential}{2} - V$$

$$Right\ Motor\ Speed = BasePWM + RightMotorCalibration - \frac{TargetDifferential}{2} - V$$

Adjusting for uphill and downhill sections

The main board of the mouse included an accelerometer that was attached to the Arduino. This allowed the mouse to know it's pitch and deduce if it was on flat ground or on an uphill or downhill section. The mouse changed to 80% speed when it went downhill to make sure that it didn't take up too much speed inducing oscillations which could cause the mouse lose control. When going uphill the mouse boosted to 180% speed to be able to make it up the hill as it was not able to do so without help. This system was very effective in increasing the chance that the mouse would go around the track smoothly especially when pushing its speed limit.

Linearisation of peak detector output

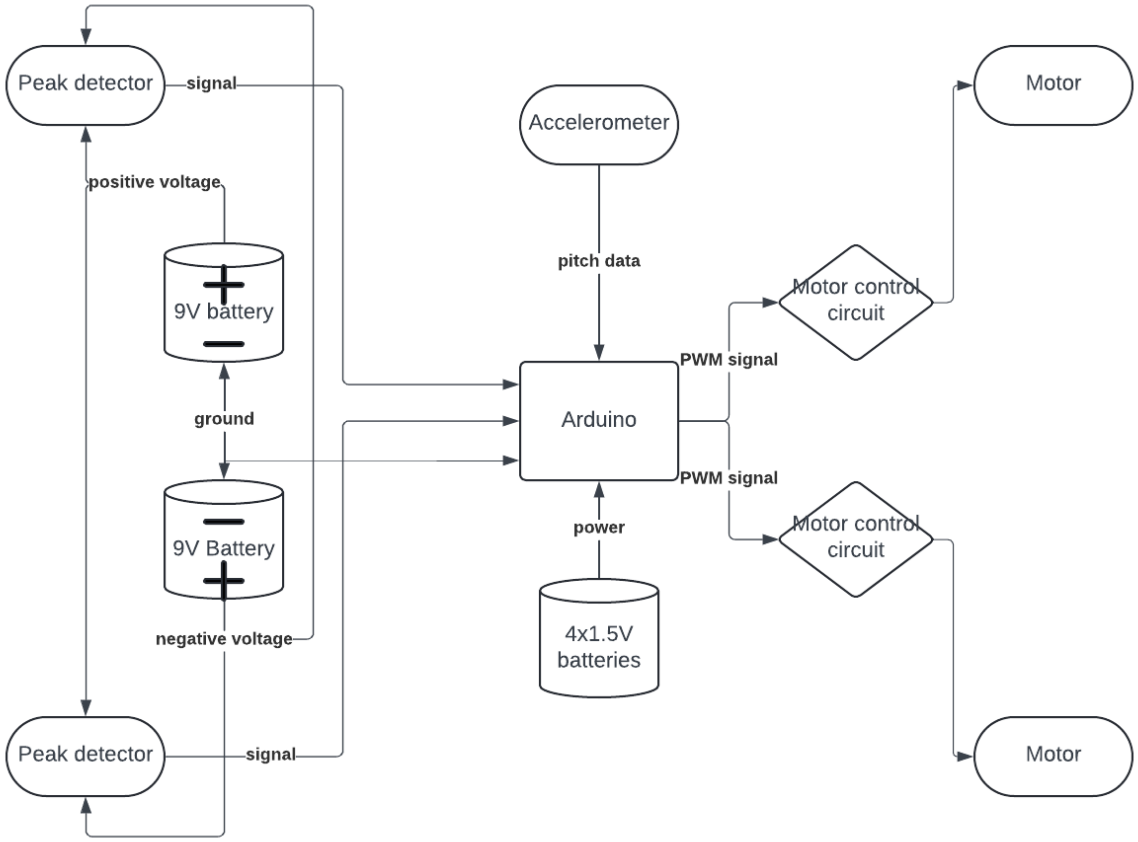
The team tested an idea of linearising the outputs of the peak detector. The theory was that if the mouse could know its exact distance from the line, it would be able to accurately follow it. This meant measuring the output of the peak detectors at known distances and finding a line of best fit that could relate the output to a distance. The team succeeded in finding the correct equation which could accurately achieve the goal of outputting the distance of the sensors from the line. Only this idea was scrapped as it returned marginal improvements for a far more complex control code which was difficult to debug.

Effects of batteries losing charge

One of the biggest challenges for the calibration of the mouse was the degradation of the batteries over time. The batteries losing charge meant that the mouse could not reach anywhere close to the maximum speed at full charge. The maximum turning angle was also reduced as the mouse would try to increase the motor speed to 100% on one side, but it would be less than what was expected so the mouse would go off course. The team attempted to account for this by switching out the batteries regularly. It also added large variability on the time that the mouse took to go around the track as it was recorded going around in 14.5 seconds before race day and during the race achieved a fastest time of 15.54 seconds.

Design of subsystem circuits and chassis

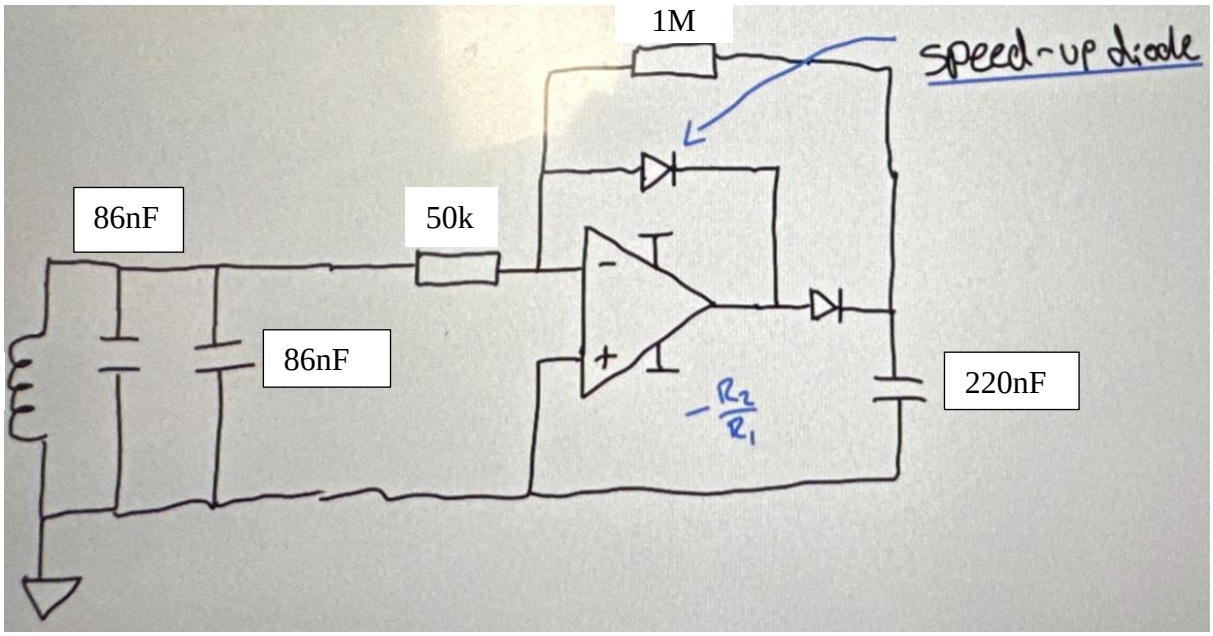
Analogue system



Above is a diagram of the mouse's analogue system. The peak detectors and accelerometer feed data into the Arduino which makes calculations and outputs the PWM signals of the left and right motors into the motor control circuits that drive the motors. The motors and Arduino are powered by four 1.5 volt batteries. The peak detectors' op-amps are powered by the two 9V batteries which have their terminals connected to create the Arduino's ground.

Peak detector design

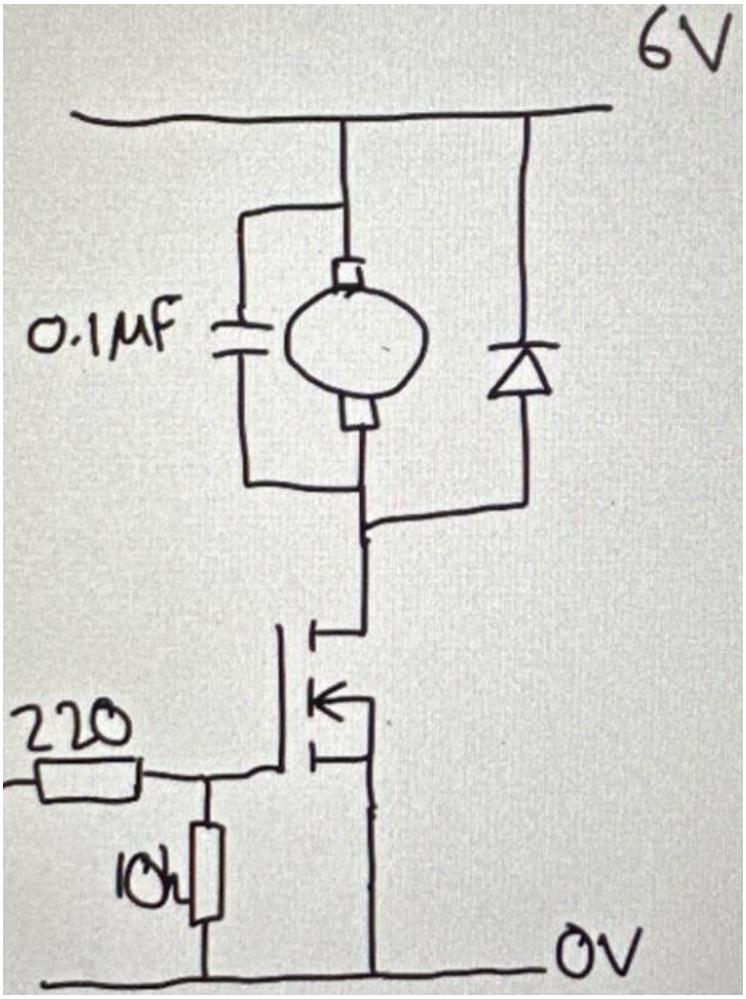
The peak detector circuit were designed as band pass filters that pick up any signal at a 20kHz frequency and amplify the signal that is detected.



The 220uH inductor and two 86nF capacitors is used as a coil to pick up the field produced by the 20kHz signal that is sent through the wire that the mouse needs to follow. There is an op-amp in an inverting amplifier configuration with a maximum amplification of -20. It was decided to replace the 50K resistor with a 50K potentiometer as it would allow the team to tweak how much each peak detector amplified the signal to try to minimise the difference in the response of the two peak detectors. The diodes in the circuit are used to reduce feedback in the circuit for a cleaner signal. The 220nF capacitor also had the goal of decoupling the output lines for a cleaner signal. The op-amps were powered by two nine volt batteries that had their negative nodes connected together to form a ground for the mouse and one positive node acted as the positive voltage and the other acted as the negative voltage in the mouse.

Motor control circuit design

For the motor circuits it was decided to put a 0.1uF capacitor and a diode in parallel with the motor. The diode was included to reduce the inductive kickback that occurs in the motor when the MOSFET is switched off. This kickback occurs as the motor has a coil in it which acts as an inductor and therefore impedes changes in current flow and therefore the diode allows a path for such a current to flow. The capacitor was used to slow down the transitions between the MOSFET being on and off. This would make the motors spin more smoothly and improve the performance of the car. For the MOSFET a 220 Ohm resistor was used to protect it from high currents and the 10K resistor was chosen as a pull-down resistor to make sure the transition between the high and low state of the MOSFET is very sharp.



Orientation of the coils

There were two options that the team thought would be best for the mouse. Either both coils pointing vertically down or both coils pointing horizontally towards each other. Both orientations were tested, and it was found that the vertical placement of the coils was the easiest to integrate with only one pair of peak detectors. This was because the peak detectors could be placed further apart from each other as they had more range to detect the coil and meant that it was harder for both to go over one side of the line and cause the mouse to go off track. There was a project to add a second pair of horizontal detectors between the already attached vertical detectors as it was believed that it would allow the mouse to have closer control with more information about the line but there was not enough time to implement this solution due to delays on other parts of the project.

Complete system testing, empirical development, and performance optimisation

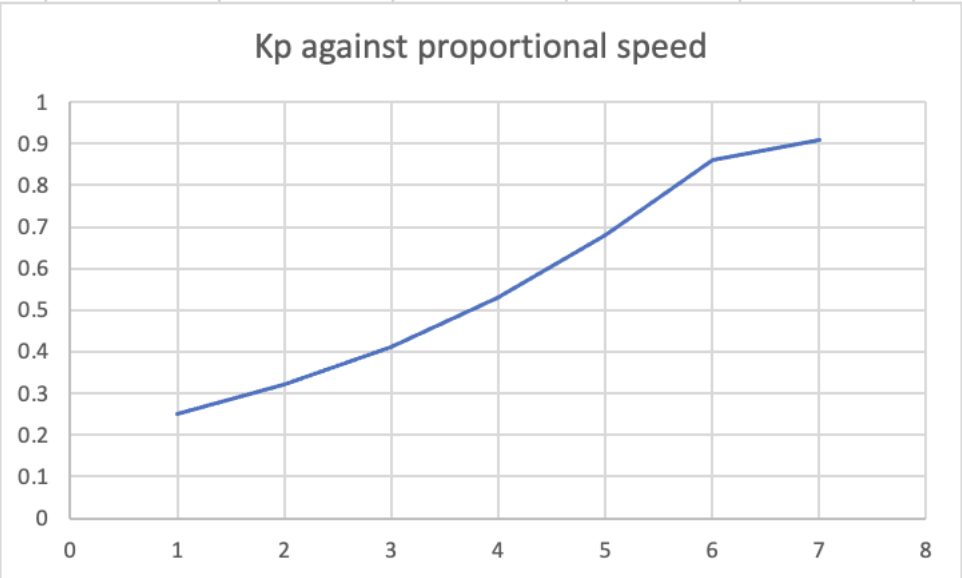
Tuning strategy

To be able to consistently get around the track at increasing speeds the values for Kp, Ki, Kd and Kv had to be tuned each step of the way. The strategy that was deployed to find the correct combination was to first observe how the car performed around the track. Then if the car didn't steer enough Kp would be increased slightly, if the car was oscillating too much on the straighter parts of the track, then Kd was increased. Finally, if increasing Kp and Kd didn't change how the car performed Kv was then increased. The value of Ki did not change along the course of the testing. This process was repeated every time speed was increased. Changing two variables or multiple parts of the code always led to confusion as to what had affected certain changes in the car's performance therefore, one variable or aspect of the control code was always changed one run at a time. This process seems more time consuming in the short term but over time it becomes easier to diagnose issues and progress becomes a lot faster.

Effect of speed on Kp, Kd, Ki and Kv around turns

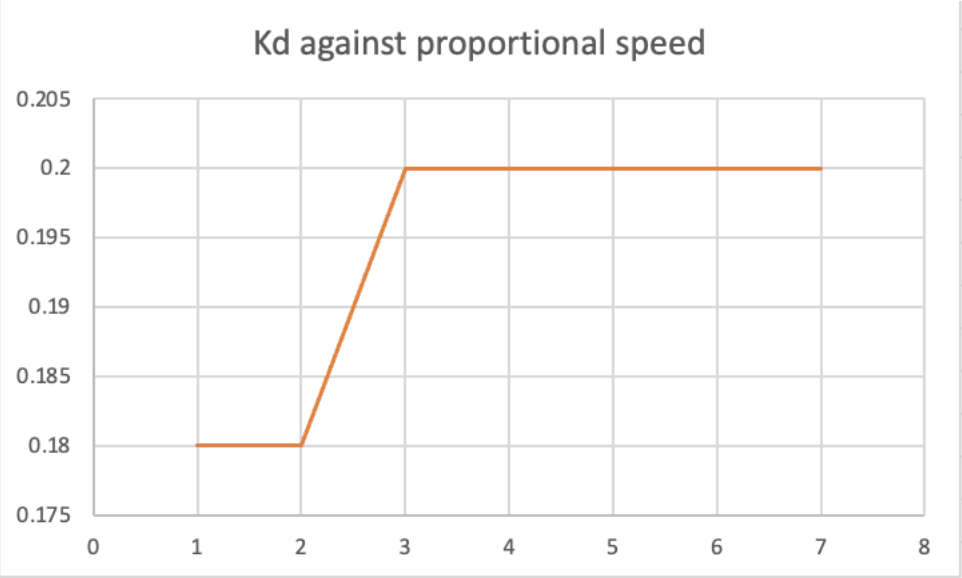
Kp

Kp was the variable that changed the most throughout the testing. It had to be increased every time speed to was increased by any amount. The amount that Kp had to change by increased at every increment of speed .



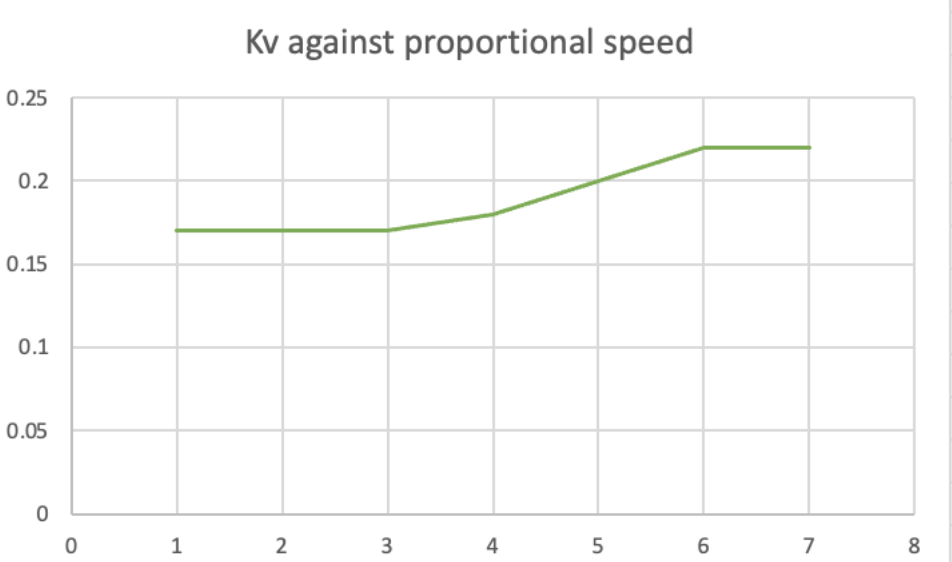
Kd

Kd had to change very little over the tuning process. This is because the team opted to use separate variables for when the track was mostly straight and in turns. Kd was far higher for example during the straight section at the end of the track.



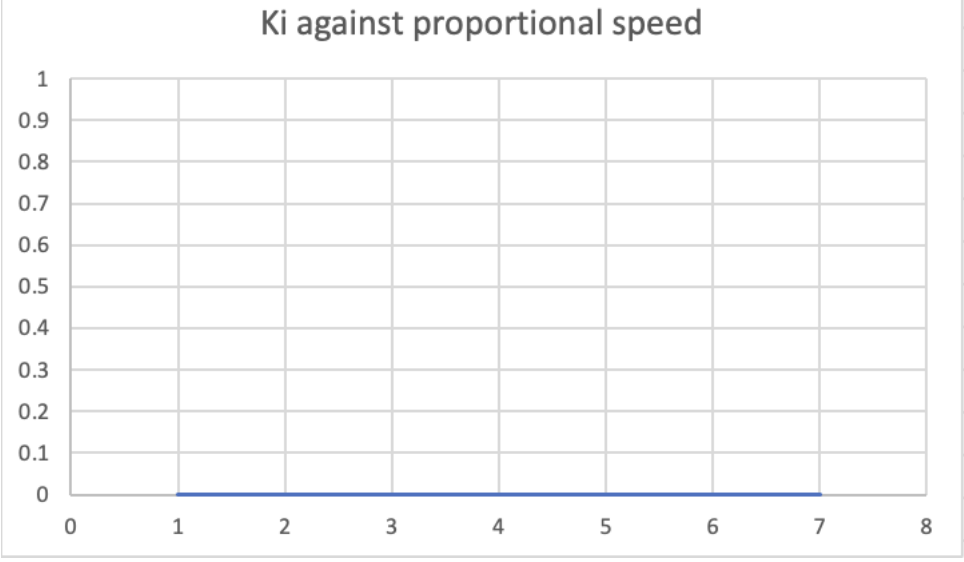
Kv

Kv also did not have to change much in comparison to Kp. This is because increasing Kv was the last option when trying to tune the car as slowing it down in corners meant a slower lap.



Ki

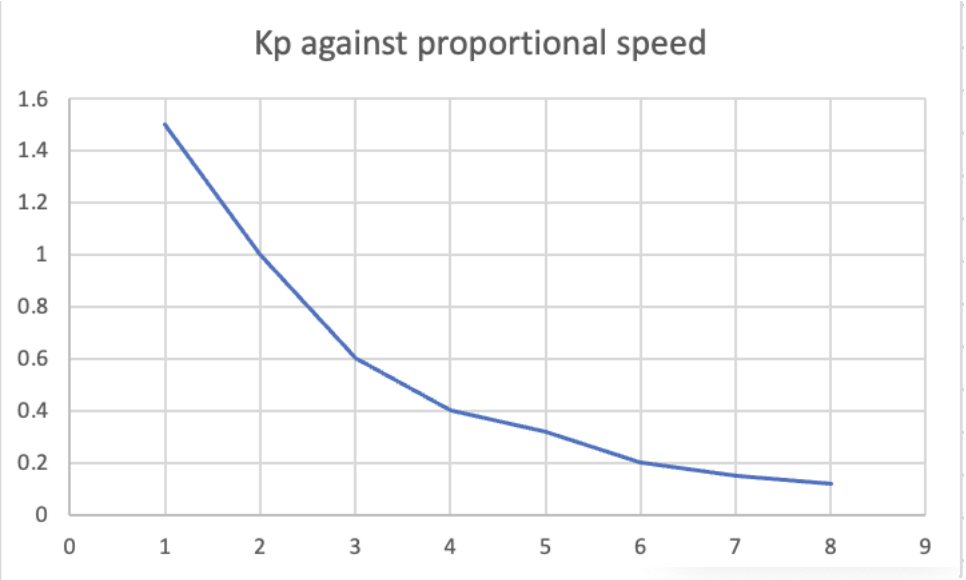
Ki remained at 0 as the peak the detectors were calibrated sufficiently by adjusting the potentiometer on the peak detector.



Effect of speed on Kp, Kd, Ki and Kv in straight sections

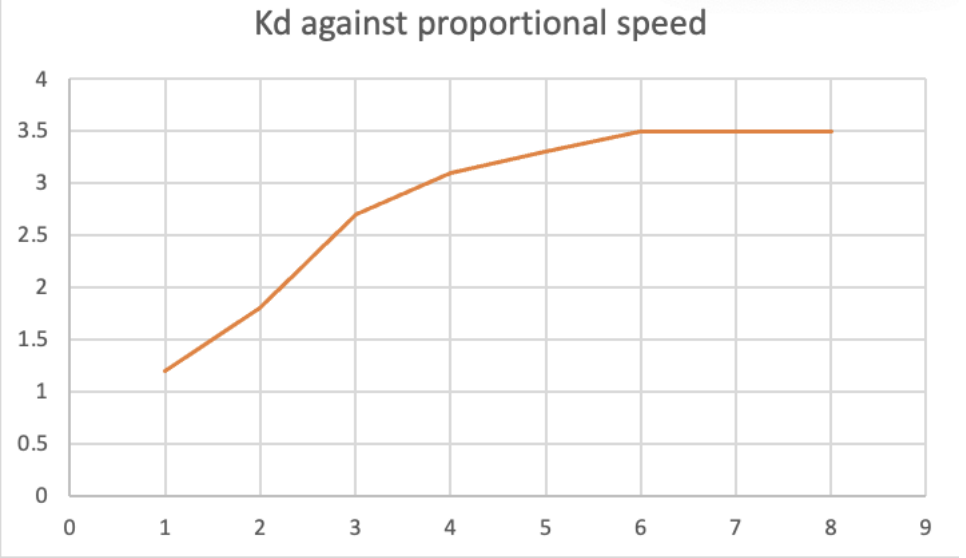
Kp

Kp had to be decreased as speed increased as the mouse needs to be less sensitive to the changes in error between the sensors. This is because any amount of steering slows down the mouse.



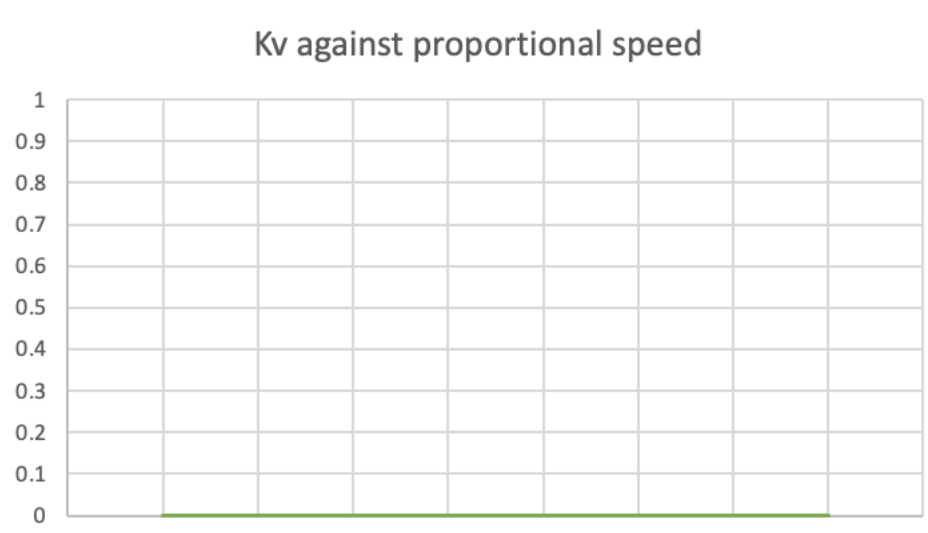
Kd

Kd had to be increased for the same reason that Kp was decreased. This is because it decreases oscillations and allows the mouse to smoothly go down the straight.



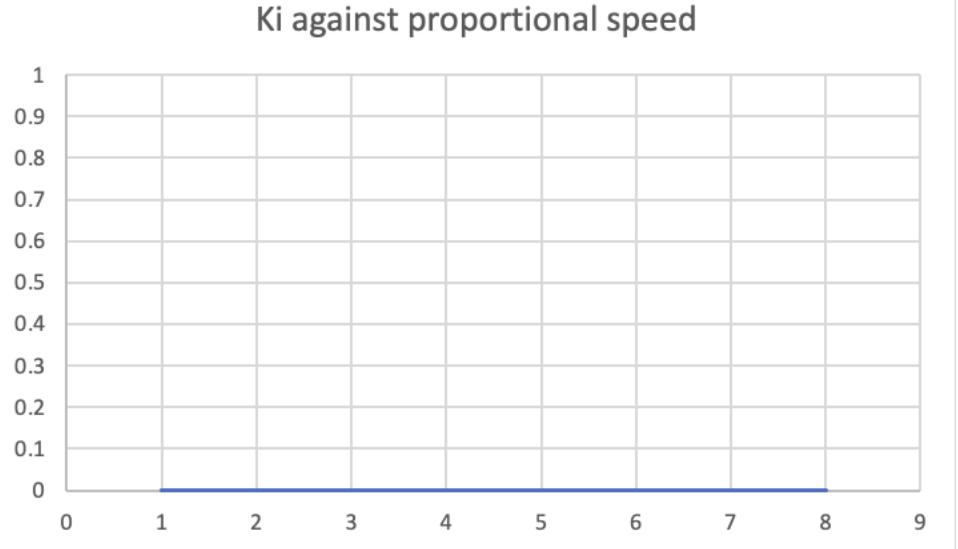
Kv

Kv was set to zero as the speed of the mouse needs to be maximised down the straight and there are no turns to struggle to go around.



Ki

Ki also remained at zero as the mouse was still tuned very well using the potentiometers on the peak detectors.



Final performance and limitations

Overall, the performance was quite good but still left some things to be improved. It achieved 10th place out of 52 in the race. As was mentioned before the batteries impeded the progress the tuning of the mouse for speed. At the highest speed in turns the mouse could not get around the large turn before the final straight.

Group work management

What worked and helped

The group worked very well together throughout the semester. There was no need to push people to do work and everyone was willing to stay longer in the lab if needed to get a part of the mouse working.

We had access to a 3D printer and one of our group members had extensive experience. Therefore, we were able to print parts for our mouse such as the front ramp, holders for the inductors and a mount for the batteries and switches. The fact that we could quickly remodel

and replace parts was extremely helpful towards the development of the mouse. See appendix A-D for pictures

Lastly, the idea to add a ramp. This allowed us to greatly increase the speed of the mouse as it increases the traction that it had around corners. Only Dr. Francis can be thanked for this idea due to his infinite wisdom in the subject of mouse optimization.

What impeded progress and could have been better

One thing that our mouse missed the mark on was power regulation. When four fully charged batteries were plugged into the mouse our Arduino had a hard time driving the motor signals for some reason that is not completely clear. This meant that two full and two half full batteries had to be used which greatly decreased the maximum potential of the mouse in terms of speed.

Another improvement to our mouse design could be to add more peak detectors. This would allow our mouse to have more accurate control but sadly our group had a few weeks of delay due to problems with the battery pack and processor that was used so this goal was not achieved in time.

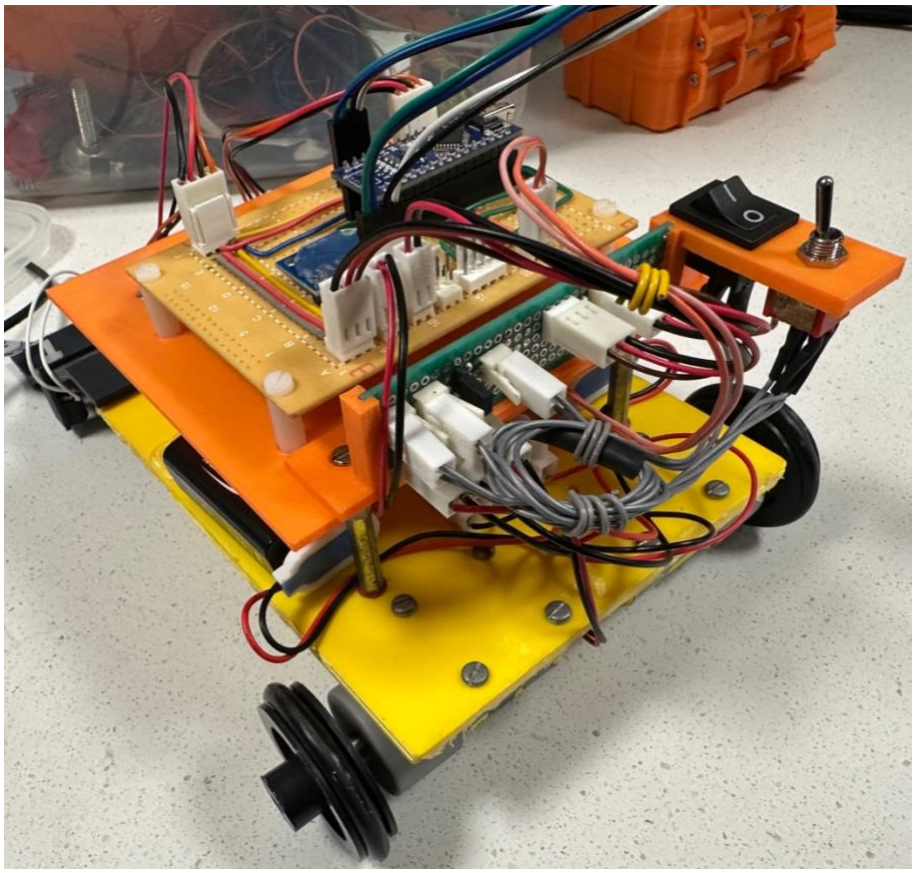
Finally, a better algorithm that could more accurately detect if we were in small turns, large turns or a straight with changing PID values for each of them could have allowed us to go around the track a lot faster than we did.

Conclusions

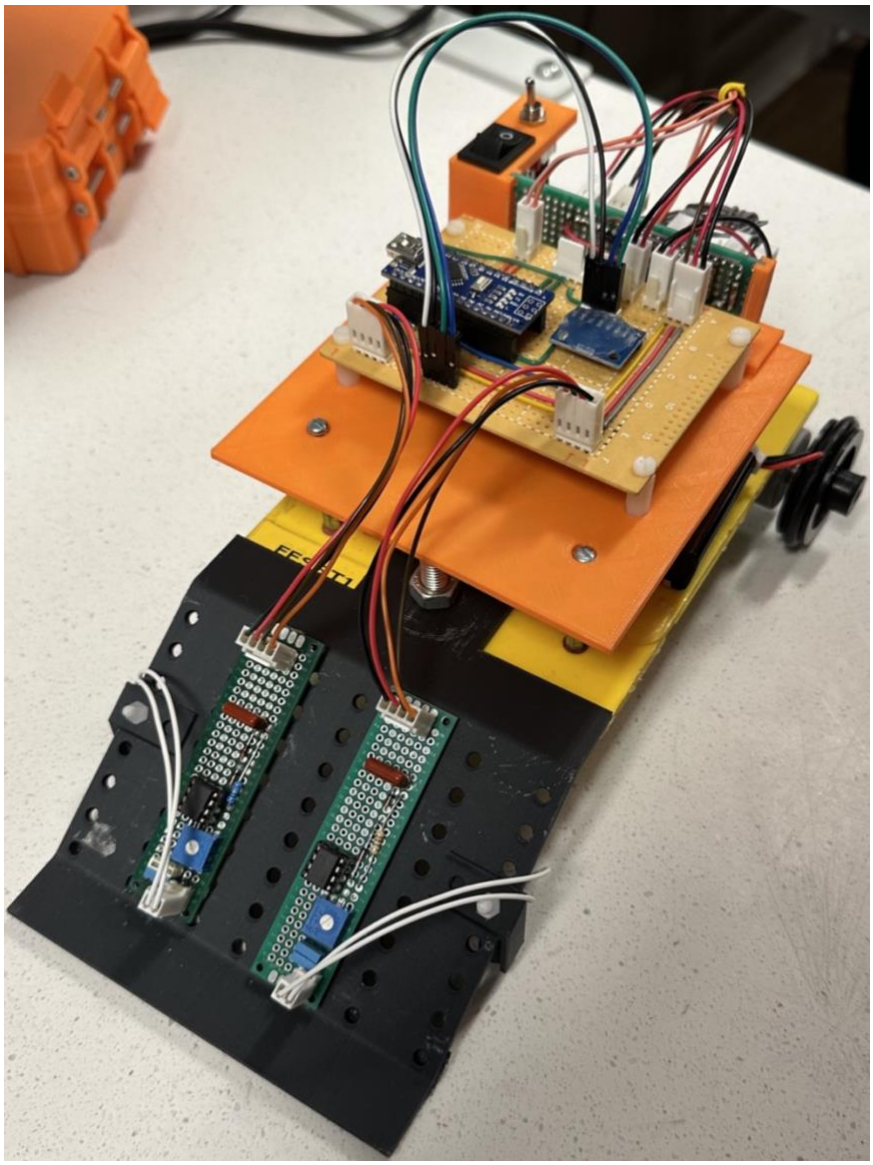
In conclusion, the project was a success with the mouse completing a fast lap during the race. This does not mean that the line follower was perfect as there are many improvements that can be made as highlighted in the parts discussing the digital and analogue parts of the mouse such as better code or more peak detectors. The team worked together and had good communication which created an enjoyable work environment and was a crucial factor in the success of the project. The budget restriction was met so all goals for the project were reached (Appendix E).

Appendix

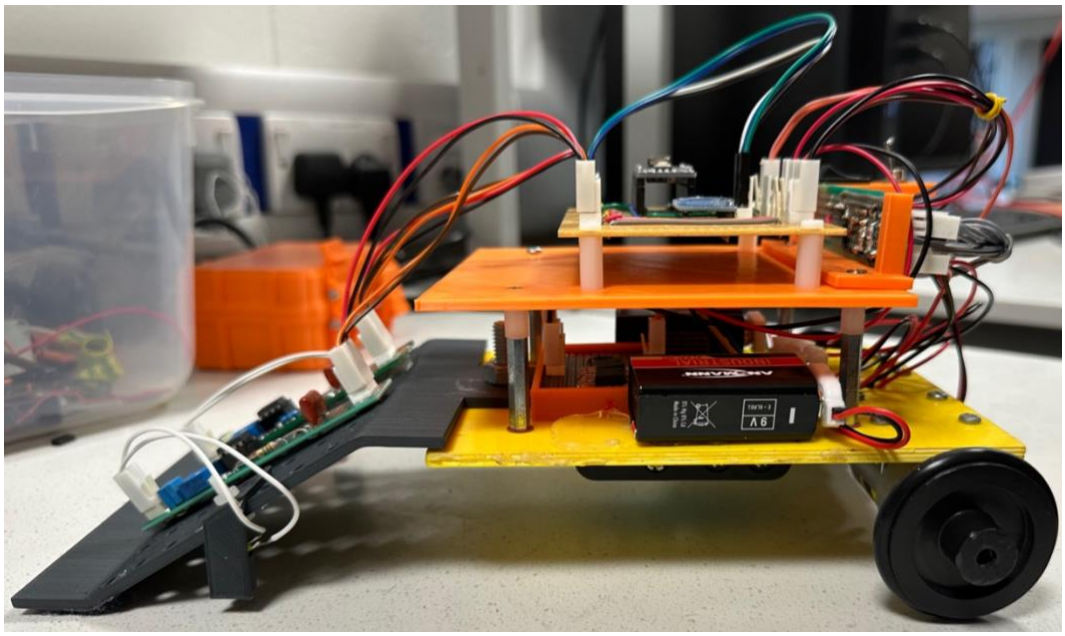
Appendix A



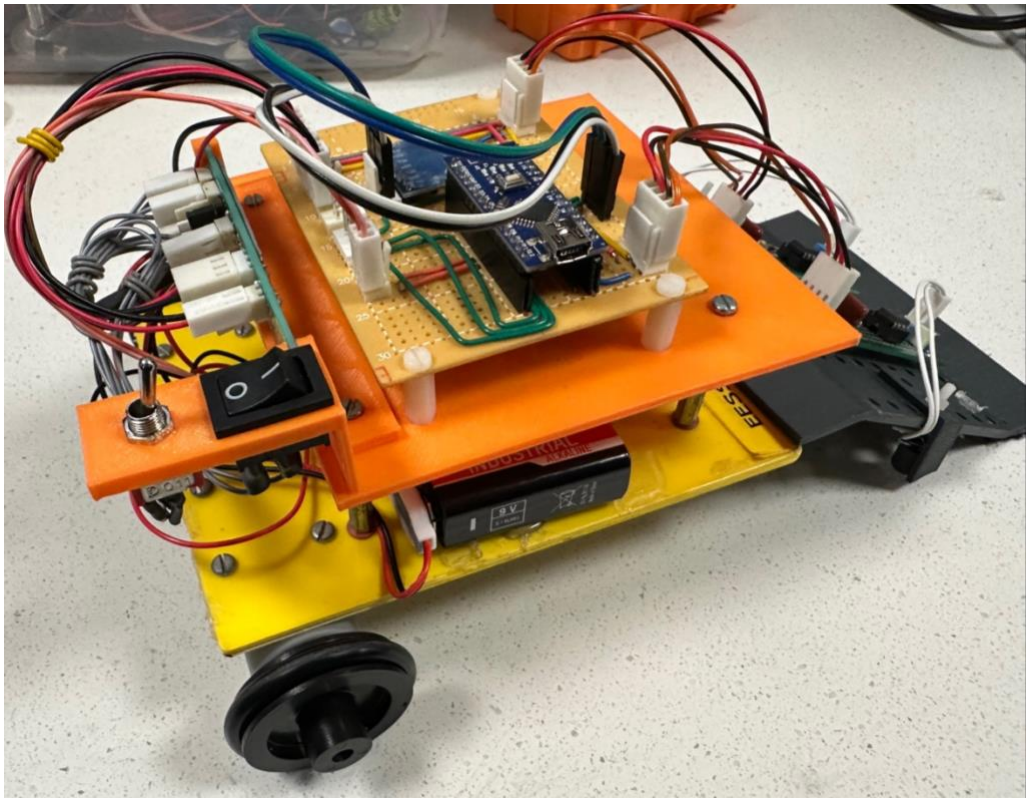
Appendix B



Appendix C



Appendix D



Appendix E

Item	Qt	Unit	Part number	Supplier	Price / Unit	Total Price
Arduino Nano	1	Item	Nano V3 ATmega328P Board	Aliexpress	1.1	1.1
Accelerometer & Gyroscope module	1	Item	MPU 6050	Ebay	1.48	1.48
220uH Inductor	2	Item	C400304	LCSC	0.03672	0.07344
86nF Capacitor	2	Item	C514193	LCSC	0.02696	0.05392
220nF Capacitor	4	Item	C393118	LCSC	0.04152	0.16608
10uF Capacitor	2	Item	C254094	LCSC	0.0092	0.0184
50k Potentiometer	2	Item	C118910	LCSC	0.10448	0.20896
TL071 Op-Amp	2	Item	C273645	LCSC	0.23776	0.47552
DIP-8 Socket	2	Item	C72124	LCSC	0.01824	0.03648
1M Resistor	2	Item	C5200501	LCSC	0.0772	0.1544
10k Resistor	2	Item	C410695	LCSC	0.00688	0.01376
220 Resistor	1	Item	C2848564	LCSC	0.00456	0.00456
1N4148 Diode	4	Item	C258182	LCSC	0.01664	0.06656
1N4004 Diode	2	item	C237267	LCSC	0.01144	0.02288
F1010ez MOSFET	2	item	C7568779	LCSC	0.54112	1.08224
JST EH Female Terminals	54	item	311-6439	RS	0.023	1.242
4 pin JST EH Female Plug 2.5mm Spaced	6	item	820-1618	RS	0.088	0.528
3 pin JST EH Female Plug 2.5mm Spaced	2	item	311-6215	RS	0.059	0.118
2 pin JST EH Female Plug 2.5mm Spaced	12	item	820-1611	RS	0.078	0.936
4 pin JST EH Male PCB Header 2.5mm Spaced	6	item	820-1434	RS	0.206	1.236
3 pin JST EH Male PCB Header 2.5mm Spaced	2	item	515-1399	RS	0.122	0.244
2 pin JST EH Male PCB Header 2.5mm Spaced	12	item	515-1383	RS	0.093	1.116
1x15 female pin headers 2.54mm spaced	2	item	C7499333	LCSC	0.1491	0.2982
Single core 22 awg wire	1	m			0.5	0.5
Multi-core 22 awg wire	1	m			0.7	0.7
Female-Female Jumper Wires	4	item	791-6450	RS	0.309	1.236
PRO AA NiMH Rechargeable AA Batteries	4	item	617-077	RS	2.758	11.032
9V <bat name> batteries	2	item		RS	1.909	3.818
AA x4 Battery Holder	1	item	SKU 201024	Switch Electronics	0.48	0.48
9V Battery Connector	2	item	185-4788	RS	0.64	1.28
M4 Nylon Screw	10	item			0.15	1.5
M4 Nylon Nut	10	item			0.05	0.5
Generic PLA for 3D Printing	100	g			0.015	1.5
Perf board	4	Item			0.5	2

Code

```
#include <Arduino.h>
#include <SPI.h>
#include <SD.h>
#include <Wire.h>
#include <MPU6050_light.h>
#include <Adafruit_Sensor.h>

// Pin Definitions
#define Right_Motor_Pin 6
#define Left_Motor_Pin 5

#define Left_Sensor_Pin A1
#define Right_Sensor_Pin A0

// Motor Calibration
#define Right_Motor_Traction_Calibration 0
```

```
#define Left_Motor_Traction_Calibration 25

#define Right_Moytor_Min_PWM_Calibration 0
#define Left_Motor_Min_PWM_Calibration 15

#define Right_Motor_Positive_Calibration Right_Motor_Traction_Calibration +
Right_Moytor_Min_PWM_Calibration
#define Left_Motor_Positive_Calibration Left_Motor_Traction_Calibration + Left_Motor_Min_PWM_Calibration

// Motor PWM values
// #define Min_Spin_PWM 50
// #define Min_Drive_PWM 75
#define TURN_PWM 90
#define STRAIGHT_PWM 255
#define UP_DRIVE_PWM 160
#define DOWN_DRIVE_PWM 65

// PID Definitions
#define TurnKp 0.91
//0.70
#define TurnKi 0.0
#define TurnKd 2
//0.50
//0.18
#define TurnKv 0.22
//0.17

#define StraightKp 0.12
#define StraightKi 0.0
#define StraightKd 3.5
#define StraighKv 0.0

// Hill detection definitions
#define UpAngle 11
#define DownAngle -11

// Straight line detection definitions
#define SmallWiggleMinTime 500

#define MinWiggleAngleStraight 12
```

```
#define MinWiggleAngleTurn 8

int MinWiggleAngle = 5.5;


// Enums for the states
enum HillState {
  Flat,
  Up,
  Down
};

enum WiggleState {
  Small,
  Large
};

enum TurnState {
  Turns,
  Straight
};

// Hill state variables
HillState CurrentHillState = Flat;
HillState LastHillState = Flat;
bool HasBeenDown = false;
unsigned long straightTimer;

// Straight line detection variables
unsigned long timer = 0;
unsigned long LastSmallWiggleTime = 0;
bool StraightLineDetectionEnabled = true;

WiggleState CurrentWiggleState = Large;
WiggleState LastWiggleState = Large;
TurnState CurrentTurnState = Turns;

// MPU Instance
const int MPU_addr=0x68;
int16_t AcX,AcY,AcZ,Tmp,GyX,GyY,GyZ;
```

```
int minVal=265;
int maxVal=402;
double x;
double y;
double z;
double last_z;
float FlatAngleOffset;

MPU6050 mpu(Wire);

// PID variables
double Setpoint = 0.0;
double Error = 0.0;
double ErrorSum = 0.0;
double ErrorSumThreadhold = 0.0;
double Target_Differential = 0.0;
double Last_Error = 0.0;
double SystemGain = 1;

// Motor variables
uint8_t DrivePWM = TURN_PWM;


// PID Variables
float Kp = TurnKp;
float Ki = TurnKi;
float Kd = TurnKd;
float Kv = TurnKv;

void setup() {
  Serial.begin(9600);

  // Pin setup
  pinMode(Right_Motor_Pin, OUTPUT);
  pinMode(Left_Motor_Pin, OUTPUT);
  pinMode(Left_Sensor_Pin, INPUT);
  pinMode(Right_Sensor_Pin, INPUT);

  // Set motor speed to 0 on startup
  analogWrite(Right_Motor_Pin, 0);
```

```
  analogWrite(Left_Motor_Pin, 0);

  // MPU setup
  Wire.begin();
  mpu.begin();
  mpu.calcGyroOffsets();
  mpu.setFilterAccCoef(1);
  mpu.setFilterGyroCoef(1);

  delay(20);
  FlatAngleOffset = mpu.getAngleY();

    // Set motor speed to 0 on startup
  // analogWrite(Right_Motor_Pin, 200);
  // analogWrite(Left_Motor_Pin, 240);

  // delay(200);


  straightTimer = millis();

}


void loop() {

  /// SENSOR PROCESSING ///

  // Read the sensor values
  int Left_Sensor_Value = analogRead(Left_Sensor_Pin);
  int Right_Sensor_Value = analogRead(Right_Sensor_Pin);

  mpu.update();
  y = mpu.getAngleY();
  z = mpu.getAngleZ();


  // STATE DETECTION //

  // Check if the mouse is on a hill
```

```
LastHillState = CurrentHillState;

if ((y > FlatAngleOffset + UpAngle)){
  CurrentHillState = Up;
  MinWiggleAngle = 2;
} else if (y < FlatAngleOffset + DownAngle){
  CurrentHillState = Down;
} else {
  CurrentHillState = Flat;
}


// Set the drive PWM based on the state
if (CurrentHillState == Up){
  DrivePWM = UP_DRIVE_PWM;
} else if (CurrentHillState == Down){
  DrivePWM = DOWN_DRIVE_PWM;
} else if (CurrentHillState == Flat){


  // Set appropriate drive PWM based on the turn state
  if (CurrentTurnState == Turns){
    DrivePWM = TURN_PWM;
  } else {
    DrivePWM = STRAIGHT_PWM;
  }

}


// Straight Line Detection
if (millis() - timer > 100){
  timer = millis();
  double z_diff = z - last_z;
  last_z = z;

  // Check if the difference is big
  LastWiggleState = CurrentWiggleState;
  if (abs(z_diff) > MinWiggleAngle){
    CurrentWiggleState = Large;
```

```
  } else {
    CurrentWiggleState = Small;
  }

  // If the last wiggle state was large and the current wiggle state is small start timer
  if (CurrentWiggleState == Large){
    LastSmallWiggleTime = millis();
  } else if (LastWiggleState == Large && CurrentWiggleState == Small){
    LastSmallWiggleTime = millis();
  } else if (CurrentHillState != LastHillState){
    LastSmallWiggleTime = millis();
  } else if (CurrentHillState == Up || CurrentHillState == Down){
    LastSmallWiggleTime = millis();
  }

  if (StraightLineDetectionEnabled && (millis() - straightTimer) > 10000){
    // If the last small wiggle time is more than 2s gas it
    if ((millis() - LastSmallWiggleTime) > SmallWiggleMinTime){
      CurrentTurnState = Straight;
      //MinWiggleAngle = MinWiggleAngleStraight;
    } else {
      CurrentTurnState = Turns;
      //MinWiggleAngle = MinWiggleAngleTurn;
    }

  } else {
    CurrentTurnState = Turns;
  }
}

// Adjusting PID values based on the turn state
if (CurrentTurnState == Turns){
  Kp = TurnKp;
  Ki = TurnKi;
  Kd = TurnKd;
  Kv = TurnKv;
} else {
  Kp = StraightKp;
  Ki = StraightKi;
  Kd = StraightKd;
```

```
  Kv = StraighKv;
}



///// PID CALCS /////

// Calculate the error and cast it to a double
Error = (double)Right_Sensor_Value - (double)Left_Sensor_Value;

ErrorSum += Error;

// Calculate the PID values
double P = Kp * Error;
double I = Ki * ErrorSum;
double D = Kd * (Error - Last_Error);

Target_Differential =  SystemGain * (P + I + D);
double V = Kv * abs(Target_Differential);

// Update the last error
Last_Error = Error;

// Calculate the motor speeds
double Left_Motor_Speed = DrivePWM + Left_Motor_Positive_Calibration + Target_Differential/2.0 - V;
double Right_Motor_Speed = DrivePWM + Right_Motor_Positive_Calibration - Target_Differential/2.0 - V;


/// MOTOR LOGIC ///

// Check if bellow 0 and set to 0 and adjust the other motor
if (Left_Motor_Speed < 0){
  double diff = abs(Left_Motor_Speed);
  Right_Motor_Speed += diff;
} else if (Left_Motor_Speed > 255){
  double diff = Left_Motor_Speed - 255;
  Right_Motor_Speed -= diff;
}

if (Right_Motor_Speed < 0){
```

```
    double diff = abs(Right_Motor_Speed);
    Left_Motor_Speed += diff;
  } else if (Right_Motor_Speed > 255){
    double diff = Right_Motor_Speed - 255;
    Left_Motor_Speed -= diff;
  }

  if (Left_Motor_Speed < 0){
    Left_Motor_Speed = 0;
  } else if (Left_Motor_Speed > 255){
    Left_Motor_Speed = 255;
  }

  if (Right_Motor_Speed < 0){
    Right_Motor_Speed = 0;
  } else if (Right_Motor_Speed > 255){
    Right_Motor_Speed = 255;
  }

  // Set the motor speeds
  analogWrite(Left_Motor_Pin, Left_Motor_Speed);
  analogWrite(Right_Motor_Pin, Right_Motor_Speed);

  // Serial.print("Left Motor Speed: ");
  // Serial.print(Left_Motor_Speed);
  // Serial.print("Right Motor Speed: ");
  // Serial.println(Right_Motor_Speed);
  delay(1.0);
  // Serial.print("Left Sensor Value: ");
  // Serial.print(Left_Sensor_Value);
  // Serial.print(" Right Sensor Value: ");
  // Serial.println(Right_Sensor_Value);

  //Serial.println(CurrentTurnState);

}
```