Closed-Loop Electronic Controlled Mouse Design, Performance and Evaluation

William Powell

Student Code: wfp21, es2041, mdk40

Integrated control system design (EE20100)

Word Count: 3656

Referencing Type: IEEE

April 2021

Abstract

This paper explains the design process of an automated mouse vehicle which follows a track, and through closed-loop control corrects error in lateral positioning. Through simulation and practical testing, the bandpass filter demonstrated to effectively attenuate noise from the AC signal of the track, whilst employing passive gain to the signal. The motor driver circuit also demonstrated to be effective through real life testing, with a 50Hz PWM frequency demonstrating to be sufficiently large to smooth the motor's speed and ensure the Mosfet did not overheat. Using a microcontroller allowed for easy modification of variables such as the PID constants; motor or pick-up-coil offsets; reference voltage for average speed and turning speed. This ensured the versatility of the mouse and could be optimised for different race tracks. Through simulation, the final values for the PID constants are: kp = 5.2, ki = 0.3 and kd = 1.2, however, in addition to the reference voltage, they will likely require adjustment in practice, due to mechanical delays and component saturation.

Contents

1 Introduction

1.1 Context of investigation

Our brief is to design and assemble an automated vehicle, referred to as a 'mouse', which must drive unaided around a track in minimal time. Hence, a control system must be designed to manoeuvre the vehicle and correct its positioning on an 20kHz AC current carrying track.

1.2 Objectives

1. Design a closed loop system, that corrects error reliably and efficiently.
2. Use a pick-up-coil as a feedback sensor to detect the track's AC signal and correct positioning and speed.
3. Achieve a lap time of under 15s around a 20m track.

1.3 Requirements Specification

1. Filtering is required to eliminate noise from the pick-up-coil.
2. The two motors must be controlled individually to allow for cornering.
3. Amplification or motor drive circuits are required to drive the DC motors.
4. AA and PP3 batteries, prefabricated chassis, 20kHz pick-up-coil and DC motors must be used to control the system.
5. The electronics must be low-level mini-modules and the total cost of components must not exceed £30.

2 Background of Control System Methods

The two types of control systems are open-loop and closed-loop control, illustrated in Figure 1 below. With open-loop control, the output has no effect on the system control; it lacks feedback sensing and instead is often calibrated before use. As explained in Basic Control Systems [1], it is used when the system transfer functions are approximately linear, and are not significantly influenced by disturbances. In addition when feedback is not observable, such as plants operating in high-temperature or hostile environments, where the feedback is not accessible or cannot be measured. Open-loop has lower complexity, is inexpensive and suitable for a range of applications from volume on an audio amplifier to servo motors. Closed loop however, adapts the input signal through feedback sensing, using a summing junction, to correct error in the output. For example, a soldering iron or drone requires feedback to prevent an output from diverging and cause damage.

This project requires closed-loop control for the vehicle's lateral positioning since disturbances to the system are inevitable and must be corrected to complete the race. This could be through differences in the two motor's speeds, varying load torque and the turns of the track. It would also require memory to store each turn in the circuit, which would be impractical and unversatile for other circuits. Due to the linear relationship of voltage to RPM, demonstrated in figure 11, appendix 8.1, an open-loop is ample to control the speed of the motor.

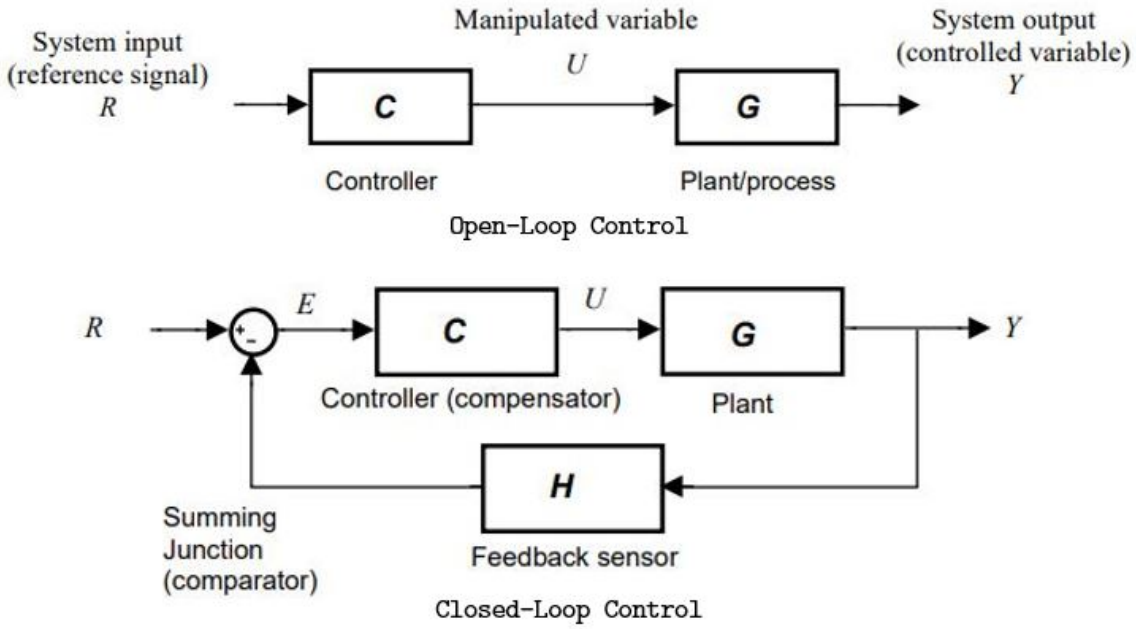

Figure 1: Generic Open and Closed Loop Control

The input R is connected to the feedback sensor via a summing junction to the plant. Continuous adjustment of the plant input is corrected by the electronic controller. The transfer function, defined by Frazzoli E,. [2], is the Laplacian transform of the impulse response as a ratio of a dynamic system's output to its input.

$$T(s) = \frac{Y(s)}{R(s)} \tag{1}$$

where R and Y are the Laplacian input and output respectively. This may be expressed in terms of the forward (plant and controller) and loop (feedback) gain.

$$T(s) = \frac{C(s) \cdot G(s)}{1 + C(s) \cdot G(s) \cdot H(s)} = \frac{Forward\,Gain}{1 + Loop\,Gain} \tag{2}$$

As the controller gain, C, increases, it reduces the significance of any error in the plant due to environmental factors such as operating temperature or load regulation. The output therefore tends towards the input, with less error and will be less subject to disturbances. However, there is a threshold, where if exceeded, causes an oscillatory and unstable response due to the introduction of signal phase-delay [1]. This results in increased frequency of the control loop, where the phase shifts to 180°, resulting in an inversed feedback gain. A compromise is therefore required between accuracy (high gain), and system stability (preventing significant phase shift).

3 Control and Drive System Architecture

As explained above, the mouse requires a closed loop circuit for lateral positioning, which is corrected through increasing or decreasing the reference voltage of the motor through the controller.

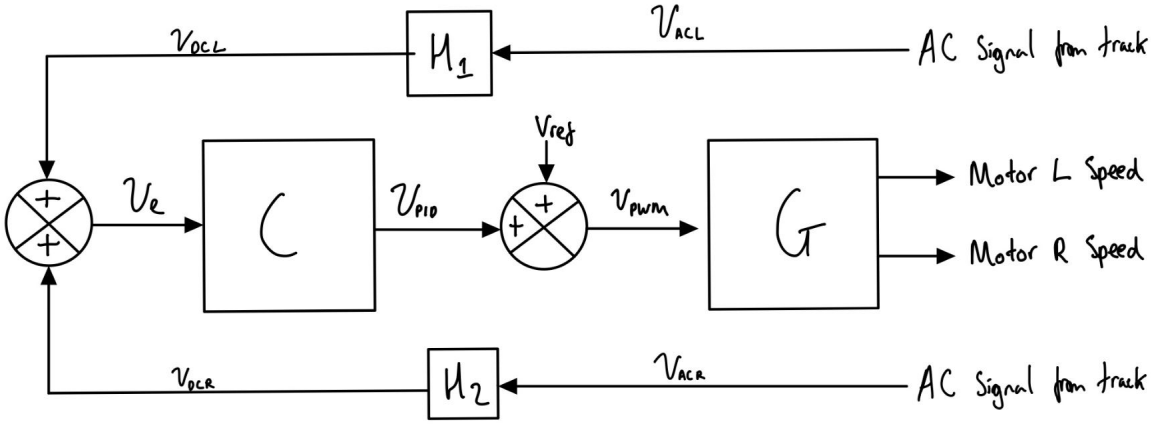

Figure 2: Mouse Closed Loop Control System to Maintain Lateral Positioning of the Mouse

Following the same notation as Figure 1. The output is the voltage to the left and right motor and the input is the voltage from the AC signal of the track, which changes amplitude with mouse positioning. This figure can be expanded to show the subsystem circuits which will be used.

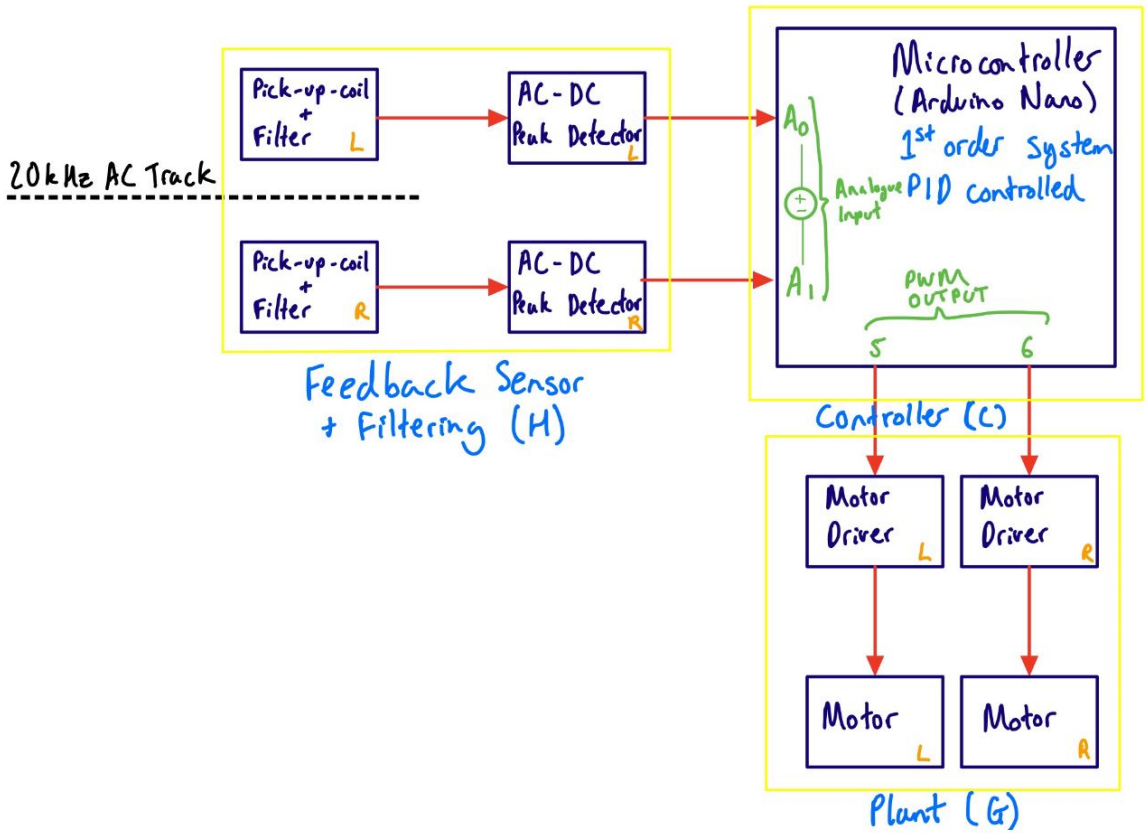

Figure 3: Overview Schematic of Mouse System Architecture

The feedback sensor contains the pick-up-coil to induce the signal from the track, V_{AC}; a bandpass filter to attenuate noise and disturbances that are outside the 20kHz bandwidth and an AC-DC peak detector circuit to convert the maximum of the sinusoidal wave to a digital signal, V_{DC}. The left and

right DC signals are then passed through a summing junction in the microcontroller, to produce an error voltage, V_e and adjusted with a gain from the PID control, V_{PID}. The PID voltage is summed with an average speed voltage, V_{ref}, to obtain a PWM voltage, V_{PWM}, to send to both the left and right motor independently. The PWM signal controls a motor driver circuit which will ensure sufficient current is delivered to the motor.

4 Design and Simulation of Subsystem Circuits

From equation (2) in 2.1, it is evident that when $C \cdot G \cdot H >> 1$, the transfer function tends to $\frac{1}{H}$. It is therefore imperative to have an accurately known feedback sensor and gain. Since the pick-up-coil is inherently subject to noise, signal conditioning through filtering and A/D conversion is required.

4.1 Bandpass filter Circuit

A series bandpass filter is used to pass through the track's 20kHz AC signal induced by the pick-up-coil, and attenuates noise by cutting off both high and low frequencies.

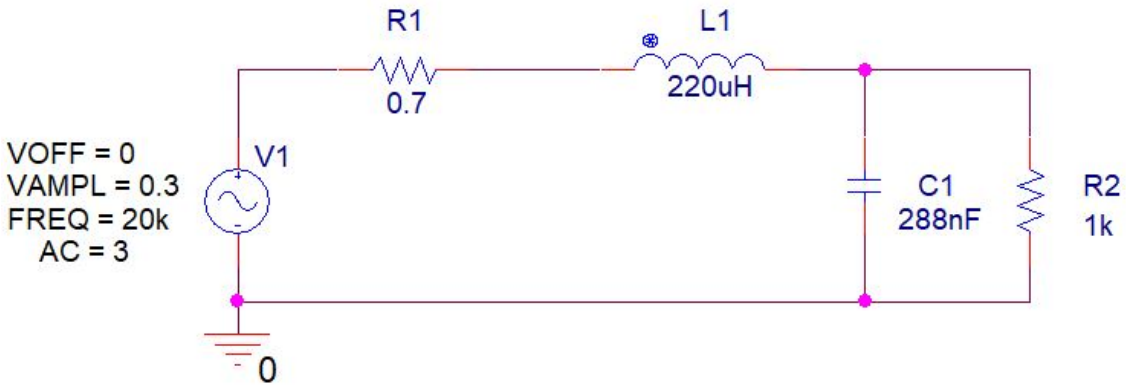

Figure 4: 20kHz Bandpass Filter, with output across a resistive load of 1k

The voltage source simulates the voltage gained through the pick up coil, with an output of the order of 100mV pk-pk. L1 and R1 are its inductance and resistance respectively. The pick-up-coil used is a 220µH inductor, with 0.7Ω resistance. Finding the capacitance for a bandpass filter with central frequency, f_r, of 20kHz:

$$f_r = \frac{1}{2\pi\sqrt{lC}} \tag{3}$$

hence,

$$C = \frac{(\frac{1}{2\pi \cdot 20k})^2}{220\mu} = 287.8nF \tag{4}$$

A 288nF capacitor will therefore be used for the RC filter. The Quality factor is calculated to be 39.48:

$$Q = \frac{\omega_r L}{R} = 39.48\omega_r = \frac{1}{\sqrt{LC}} \tag{5}$$

where, $\omega_r = \frac{1}{\sqrt{LC}}$

This results in a bandwidth of:

$$B = \frac{f_r}{Q} = 506Hz \tag{6}$$

This a small bandwidth and does not allow for sufficient tolerance for both the track frequency and the manufacturing tolerances of components. Thus, a resistor parallel to the load was inserted to decrease the Q-factor, and subsequently increase the bandwidth.

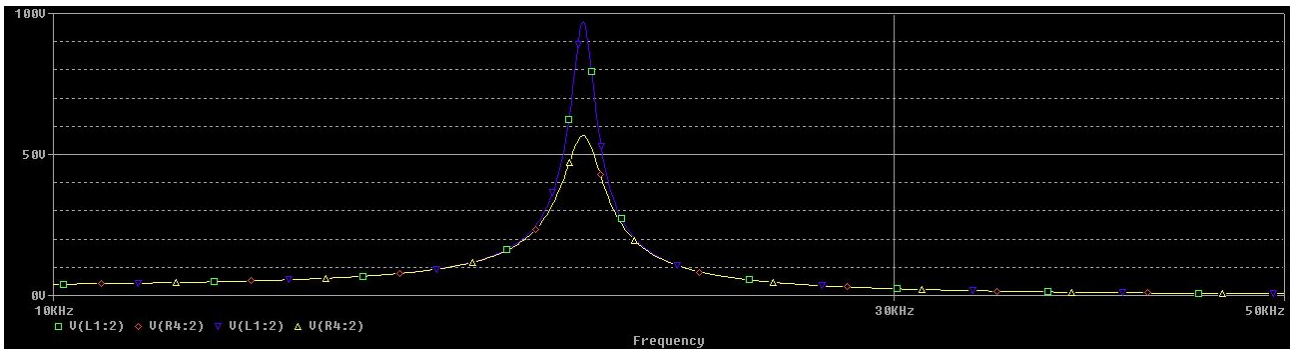

Figure 5: Logarithmic plot of Frequency vs Voltage for the Bandpass filter

With a resistive load, R2, of $5k\Omega$ and $1k\Omega$ for the blue and yellow plots respectively. The higher resistive load results in an increased Q factor, increasing the gain and lowering the bandwidth. Assuming a tolerance of $\pm 5\%$ for the 20kHz signal, an appropriate Q factor and thus bandwidth must be used to ensure there is no attenuation to the AC signal. As shown in the figure above, $5k\Omega$ does not allow sufficient tolerance for the AC signal. The 1k load has a peak voltage half that of the $5k\Omega$ load, allowing for a larger bandwidth.This resistance would tolerate a 5% change to the 20kHz signal, as well as the manufacturing tolerances of the resistor, capacitor and inductor, which could change the pass band location. The load resistor could be replaced with a variable resistor, allowing for quick adjustment of the bandwidth, dependent on the real tolerances for the mouse and track.

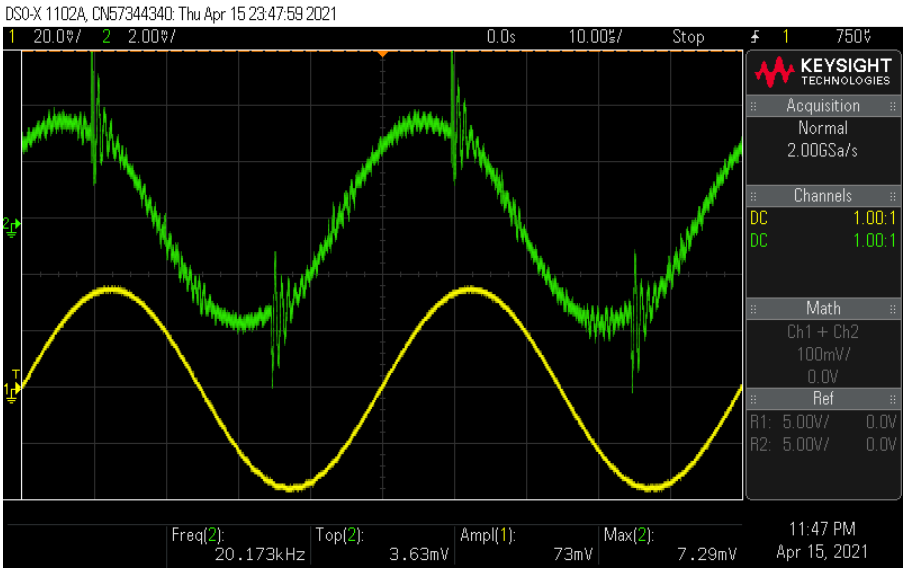

Figure 6: Generic Open and Closed Loop Control

The voltage increments are 2mV and 20mV respectively for an unfiltered and filtered pick-up-coil, marked as the green and yellow plots. Despite the passive nature of the RC filter, a gain of 10 is achieved, due to an inherently high Q factor from the capacitor and the $1k\Omega$ resistive load.

The perturbation of the unfiltered signal is due to the triangular voltage waveform in the signal generator. As shown, this and all other noise is attenuated by the filter, at the recorded frequency of 20.17kHz, demonstrating the effectiveness of the filter.

4.2 Peak Detector Circuit

To determine the lateral positioning of the mouse, the amplitude of the track signal, V_{AC} must be determined. Decreasing the distance between the pick-up-coil and the track, results in a power of two increase in peak voltage. A peak detector outputs a DC voltage equal to the peak value of the AC signal. This can be used to obtain an error for the mouse position.

With a simplistic peak detection circuit, see figure 13, section 8.2, the circuit with series diode and load capacitor, acts as a precision rectifier, outputting the exact value of the peak input voltage. The op-amp compensates for voltage loss in the diode through feedback to the non-inverting terminal, allowing for accurate quantisation. This is due to the fundamental principle of op-amps, that when unsaturated, the voltage difference between inverting and non-inverting terminals is negligible.

However, as explained by All About Circuits [3], this is not very precise as there will be increasing inaccuracies when the capacitor discharges due to leakage current from the negative bias of the diode when the input drops low. In addition any current draw at the output, due to the ADC, results in a discharge in capacitance and loss of accuracy.

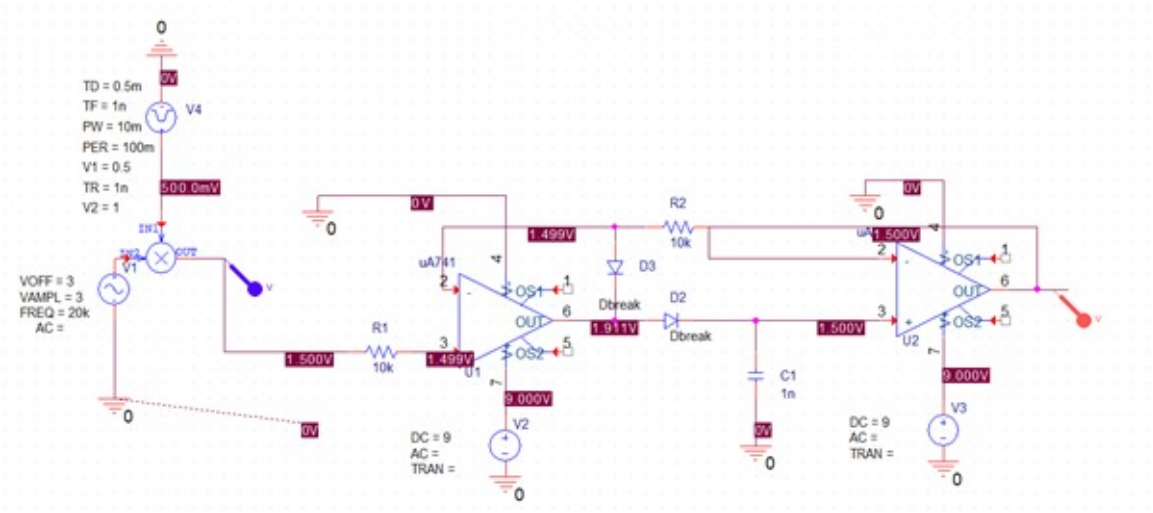

Figure 7: High Precision ADC Peak Detector

To prevent the capacitor from discharging due to current draw at the output, an additional op-amp is required, to act as a voltage follower or buffer amplifier between the load and capacitor.

An additional diode, D3 is added to restrict the output of the first op-amp from becoming negatively saturated. By using a Schottky diode, this reduces the forward voltage drop, increasing the charging current for C1, improving the slew rate and reducing the pedestal error, when the input signal drops [4]. The $10k\Omega$ resistor, R2, ensures there is no potential difference across D3, therefore no leakage current that would otherwise be present due to the reverse bias when the input voltage drops.

The capacitor C1 needs to be small to minimise the slew rate, which will add latency to the circuit, whilst ensuring the capacitor is large enough to discharge slowly. 1nF capacitance is an appropriate trade-off for this requirement. A teflon or polystyrene based capacitor should be used as they have low dielectric absorption, increasing the output precision [5].

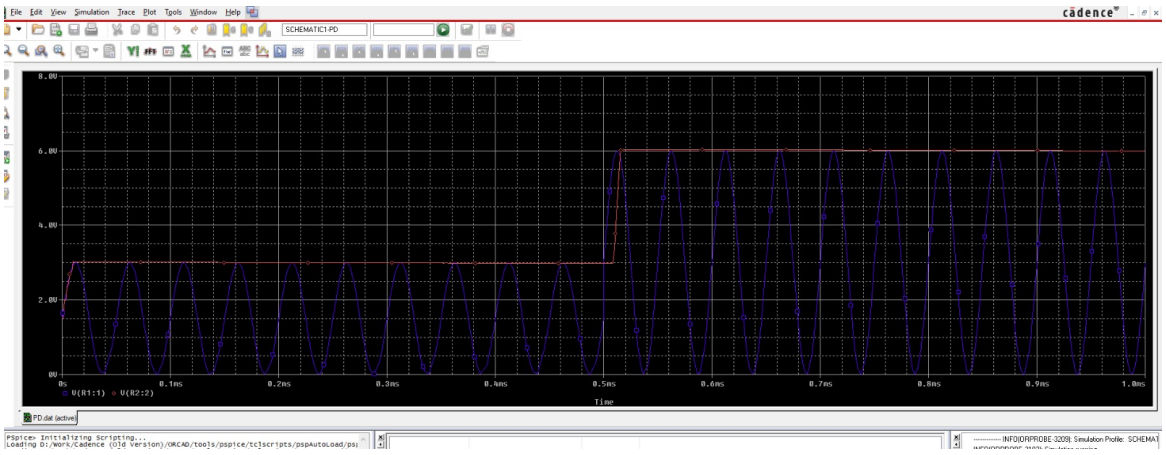

Figure 8: Peak Detection Simulation at 20kHz and 3V to 6V Peak-to-Peak

As shown below, the circuit performs well under simulation, precisely holding the maximum value of the AC input. The circuit shows excellent dynamic responsiveness. When increasing from a 3V to 6V input, the slew rate of the output is negligible, approximately 0.02ns to reach the new peak voltage, confirming the appropriate sizing of the 1nF capacitor. The output will then be inputted into the microcontroller, as illustrated in the schematic, Figure 3.

4.3 Motor Driver Circuit

Arduino output pins are rated for 20mA continuous current [6], hence to power the motor, additional current must be supplied from the control signal from the microcontroller. Initially, a power amplifier circuit was designed, see figure 14, appendix 8.2. The circuit used an operational amplifier to boost the voltage to a transistor, which would power the motor using AA batteries for the high current required. However, this was unnecessary since the 5V PWM signal from the Arduino was greater than the 3.3V turn-on voltage of the NPN transistor. Instead, a motor driver circuit is more suitable circuit for driving the motor, which does not require an op-amp. Low-side driver circuits require a lower gate voltage than high-side to turn on the Mosfet, since the voltage has not yet been dropped from the load.

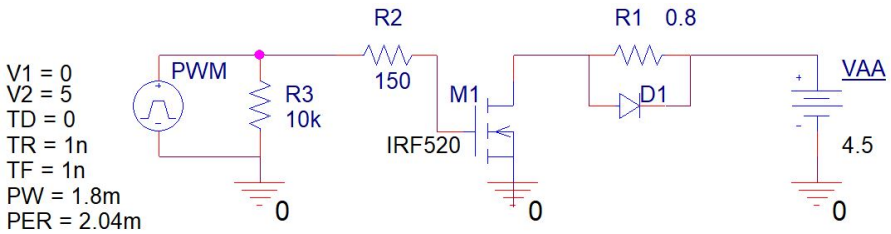

Figure 9: Peak Detector Circuit

A square wave signal is simulated at 50% or 128/256 duty cycle, with 1ns for rise and fall time, to represent the non-ideal nature of the Arduino PWM output. When the PWM is 5V, the Mosfet is turned on, closing the circuit and powering the motor through the AA battery supply. Due to the inductance and inertia of the motor, no smoothing is required for the motor to maintain at a constant speed. A practical test was conducted to determine what PWM frequency was required to keep the motor driving smoothly. 50Hz was sufficient and helped to minimise the heat generated from the Mosfet.

The resistance of the motor was determined through measuring the voltage and current whilst stalling the motor. From figure 12, appendix 8.1, the resistance was determined to be 0.806Ω and 0.699Ω for the two motors. The larger motor resistance is used in the simulation. To achieve a time below 15

seconds, the motor speed required is 637rpm. Interpolating from figure 11, this requires a voltage of 3.12V. When stalled, the motor requires an additional 0.8V, assuming a maximum current of 1A. Hence the total voltage required at maximum speed is 3.92V, therefore 3 AA batteries equating to 4.5V is sufficient and will be reached with a duty cycle of 87%.

An N-channel Mosfet is used to withstand the high current passing through the motor. A logic-level Mosfet, such as the IRF520 can comfortably withstand 1A, the maximum current draw that the motor will undertake, as evident in the motor experiment, see appendix 8.1. It has a low voltage drop between the drain and source, therefore it will consume a low amount of power, although a heat sink could be used due to the rapid switching that it will endure. The IRF520 has a V_{GS} of 3.3V, therefore it will turn on completely when the PWM is HIGH, preventing any power loss from insufficient turn-on voltage.

The 150Ω resistor to gate limits current surge when switched on to 33mA, preventing current loss in the Mosfet. The flyback diode is required to prevent back EMF caused by the motor's inductance when the voltage supply drops. The $10k\Omega$ pull down resistor, sinks the gate to zero voltage. This is precautionary since the Arduino is also configured with a pull-down resistor.

5 Design and Tuning of the Microcontroller

Microcontrollers prove more versatile and allow for quicker adjustments. The advantage of using a microcontroller is that it allows for continuous tweaking of the independent PID constants to optimise performance through rise-time, denoted as t_r, and steady-state error, $e(\infty)$. Hence, the physical mouse components and circuitry can be left unchanged and the code can be modified for change in requirements such as a change in weight, different circuit complexities, or adjusting the compromise between accuracy and speed. This would allow for large scale purchase of components which will significantly lower costs.

Microcontrollers are more costly than generic PI controllers or other type-1 or type-2 systems, however, as illustrated in figure X, a microcontroller does not require a summing junction between the two components. This reduces production time and allows for quick adjustment if one motor performs differently from the other, as evident in the initial experiment, see figure X, or if the pick-up-coil is not centred correctly.

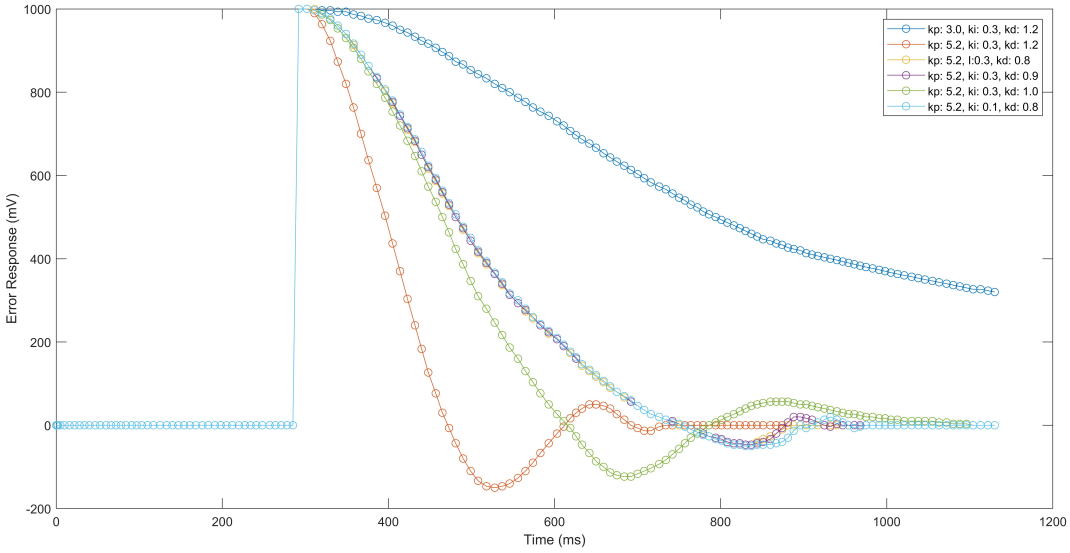

Figure 10: Simulated Error Response of the PID Control System for PID tuning

As shown in figure X, the microcontroller accepts two analogue inputs from the peak detector circuits. It is then mapped from a 10-bit integer to a value in volts, and summed with compensation to find an error value, which is used to correct the position of the mouse. See appendix 9.4 for the microcontroller source code.

The PID constants are tuned by setting a unit step error of 1 volt, simulating the pick-up-coil being off-centre from the track, and recording the response with different values for variables 'kp', 'ki' and 'kd'.

With low kp values, and subsequent low controller gain, the system shows to be underdamped, as evident by the dark blue plot with kp = 3.0. Increasing the proportional gain, the larger the overshoot and subsequent error, however for this system, some overshoot is permitted, since worsened accuracy can be compromised for a faster response. By reducing the differential, kd to 0.9 or 0.8, the system appears less oscillatory. For kd greater than 0.9, the gain is large, resulting in a larger primary and secondary overshoots, due to the large gain at high rates of change in error, $\frac{de}{dt}$. Due to its summing nature, the integral ki becomes a larger component of the total controller gain when time increases. Increasing this value from 0.1 to 0.3, results in the integral reaching its final steady-state error sooner. However this is not of great significance, as the mouse can afford to be at small non-zero errors.

6 Practical Considerations

The components required for the build are listed in figure 15, appendix 9.3.

In addition to the parts required in the circuitry, a S.P.S.T switch will be placed in series between the microcontroller and the motor driver, to turn the whole system on or off. The 9V battery will be used in parallel to power the op-amps, where negligible current will be drawn and the microcontroller which has a base consumption of 100mW (20mA at 5V) [**?**] and a maximum of 165mW (33mA at 5V) for each PWM output, as discussed in 4.3. This gives a total of 430mW, which would allow the mouse to be driven, for approximately 10 hours, assuming the PP3 battery has a capacity of 500mAh or 4.5Wh. Three alkaline AA batteries will be used to drive each motor. Each AA battery has a current capability between 0.5 and 1A [8], however in practice they will only need to draw a maximum of a third of an amp to power the 1A motor. AA batteries have excellent energy density, between 2000-3000 mAh [8], thus the mouse at full speed could be powered for 1-1.5 hours.

Due to the inherent voltage decline when the batteries discharge with use. To compensate, the voltage reference should be increased by adding a potentiometer to control the 'coast speed' variable of the arduino. This will need to be adjusted to maximise speed, whilst holding position when cornering. An IMU sensor could be used to boost this coast speed for the rising ramps, to ensure sufficient torque. Similarly, for easy adjustment of the PID constants whilst testing on the track, potentiometers should be added to individually control the 'ki', 'kp' and 'kd' variables.

Since the motors are located at the rear of the chassis, when the mouse accelerates the car is likely to lift up at the front. To prevent this, allowing for quicker acceleration, the high mass components like the battery packs, should be housed at the front. For the pickup-coils, to ensure sufficient voltage is induced, should be as low to the track as possible. Lateral mounting is a compromise between maximising the voltage and ensuring both pick-up-coils are kept on either side of the track, otherwise this will significantly lower the difference between the two voltages, lowering V_e, despite the increase in error. A temporary mount, such as zip ties or blue-tack could be used whilst finding this balance.

7 Group Work Management

As evident by the structure of this report, the design process is split well into subsystems. This allowed for straightforward delegation of work for circuit design, where each member was responsible for a part of the circuitry. The key decisions, which affected the overall system design, were discussed in weekly teams meetings, which worked effectively despite communicating online. Practical experiments in addition to the simulations, cemented the circuit design for the filter, where subsequently a lower Q-factor was chosen. The motor driver circuit and was also improved by testing how the Mosfet reacted to different PWM frequencies. Through using different output pins with lower frequency, the Mosfet heated up less whilst maintaining constant speed. This demonstrates the importance of practical tests in addition to simulations, where more qualitative data could be obtained.

Frustratingly, a member of the group did not have OrCAD and thus could not construct circuits or run simulations. This resulted in an uneven work load for the other two members. The overall design proved to be critical to progress, where initially, an amplifier circuit was designed and simulated, before realising the circuit was redundant due to the later decision to use a microcontroller. Thus, better preparation of the system design in earlier weeks would have ensured we did not waste time on redundant sub-systems.

8 Conclusions

Evident through the practical experiments for the filter and motor driver circuit, real-life testing is of great importance to cement component sizing. In practice, the response of the system with loads, component saturation, load torque and other mechanical delays, the determined PID values from Figure X, will likely require modification. Adding simple modifications, such as using a variable load resistor parallel to the filter to change Q-factor, will allow quick adjustment based on the final tolerances of the components and AC track frequency, to ensure the signal is not attenuated due to low bandwidth.

The microcontroller allows for simple implementation of the type 2, PID control. The PID constants can be easily modified with potentiometers, to allow for quick adjustment whilst testing. The derivative constant, kd, proved most critical, where increasing the gain helped the mouse respond more quickly to rapid changes in error, lowering the rise-time, which is pivotal when cornering. However, a too large kd value resulted in oscillatory behaviour and took more time to reach steady-state. If the mouse were to be used on different tracks, with longer straights, or sharper corners for example, the average speed can be adjusted in addition to the cornering sensitivity, demonstrating the versatility of the microcontroller. By implementing a summing junction through code, an offset error can be accounted for if the left and right motors differ in performance with the same PWM signal, or if the pick-up-coils are off-centred on the track.

Due to inevitable delays in the dynamic system, such as mechanical delays or component saturation, the output does not immediately respond to the input. The system's response must be modelled using differential equations and transfer functions in the Laplace domain. Through using unit step and ramp inputs, the system can be tested to analyse how well it recovers to a sudden change, like the step of the PWM signal, or a gradual change, such as the pick-up-coil voltage as the mouse diverges from the centre of the track.

References

[1] Robinson F., 2020. *Basic Control Systems - Electrical Systems and Control* [Booklet, Offline]

[2] Frazzoli E., 2011. *Transfer Functions, Dynamic Systems and Control* [Online] Available at:https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-241j-dynamic-systems-and-control-spring-2011/lecture-notes/MIT6_241JS11_lec09.pdf [Accessed 16 April 2021]

[3] All About Circuits., 2016. *Peak Detector, Chapter 3 - Diodes and Rectifiers* [Online] Available at: https://www.allaboutcircuits.com/textbook/semiconductors/chpt-3/peak-detector/#:~:text=A%20peak%20detector%20is%20a,of%20the%20applied%20AC%20signal. [Accessed 21 April 2021]

[4] Alonso et Al., 2017. *LTC6244 High Speed Peak Detector* [Online] Available at: https://www.analog.com/en/technical-articles/ltc6244-high-speed-peak-detector.html [Accessed 18 April 2021]

[5] Elliott R., 2005. *Peak Detection Circuits* [Online] Available at: https://sound-au.com/appnotes/an014.htm [Accessed 19 April 2021]

[6] Gammon N., 2015. *Mosfet Driver Motor Circuits* [Online] Available at: https://www.gammon.com.au/motors [Accessed 21 April 2021]

[7] FTDI, 2020. *FT232R USB UART IC Datasheet* [Online] Available at: http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT232R.pdf [Accessed 22 April 2021]

[8] Pololu, 2010. *Understanding battery capacity: Ah is not A* [Online] Available at: https://www.pololu.com/blog/2/understanding-battery-capacity-ah-is-not-a [Accessed 22 April 2021]

9 Appendix

9.1 Motor Experimental Results

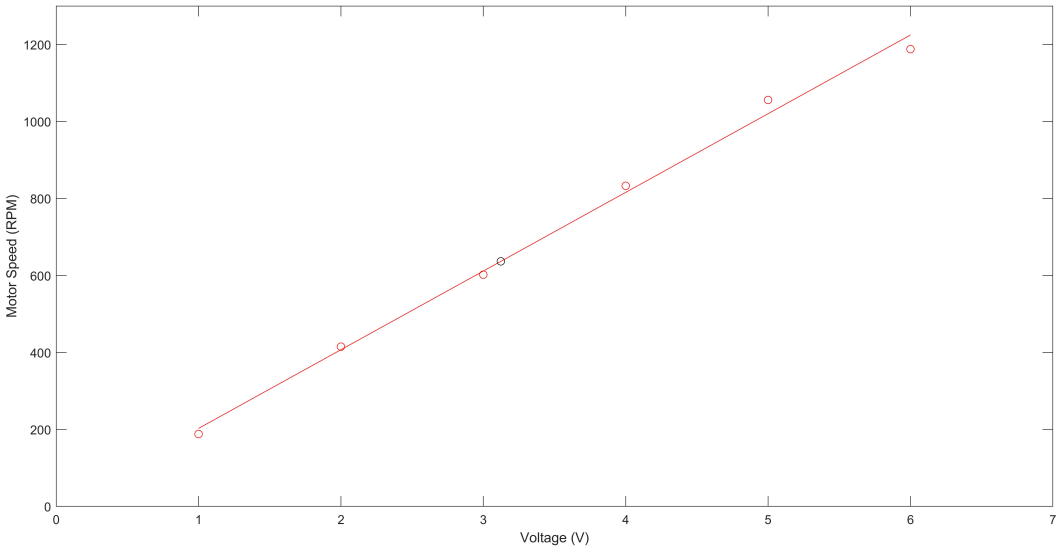

Figure 11: Relationship between Motor Voltage and Speed

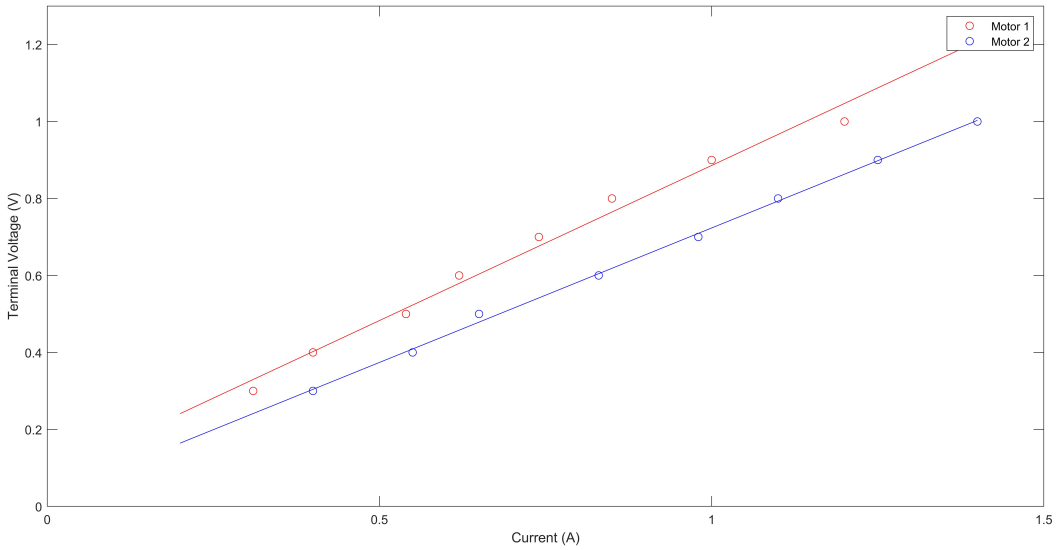

Figure 12: Relationship between Motor Voltage and Current when Stalled

9.2 Additional Circuitry

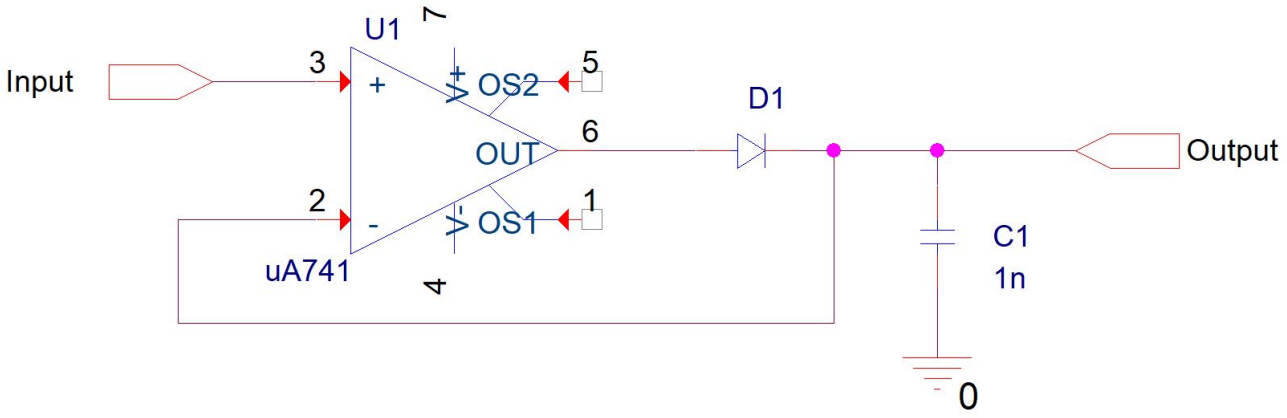

Figure 13: Generic, Low Precision Peak Detector

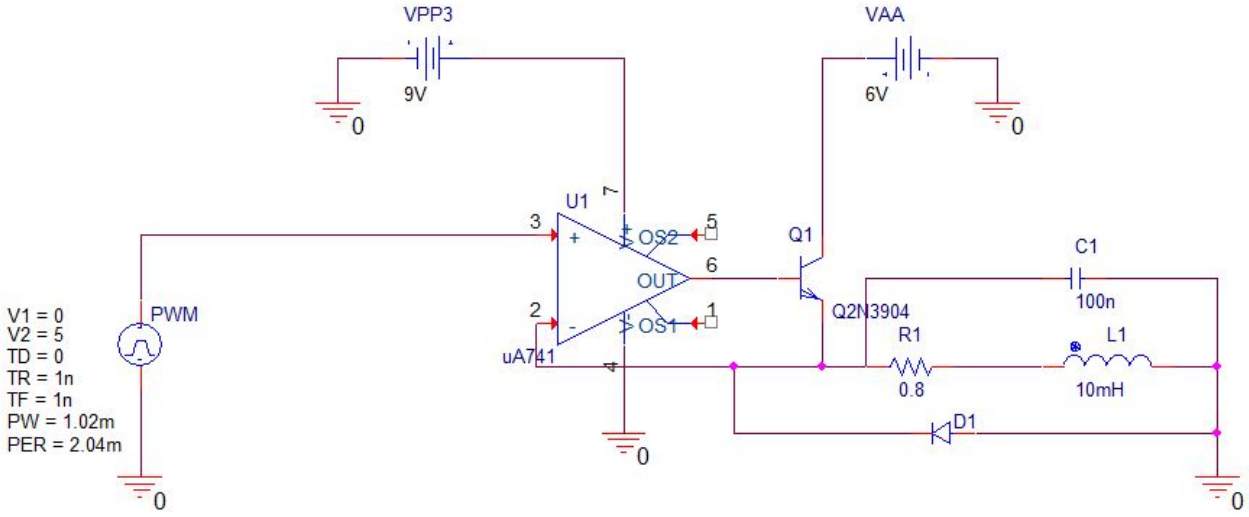

Figure 14: Amplifier Circuit to Drive the Motor

9.3 Parts List

Componet Location	Component Name	Description	Quantity
RC Filter	Resistor	0.7Ω, 10kΩ potentiometer	2
	Inductor	220µF Inductor, as pick up coil	2
	Capacitor	288nF	2
Peak Detector Circuit	9V Battery	Lithium PP3	1
	Resistor	10kΩ	4
	Op-Amp Circuit	uA741	4
	Diode	Schottky Diode, Low Dielectric Absorption	4
	Capacitor	1nF, PTFE/Teflon	2
Motor Driver Circuit	Microcontroller	Arduino Nano with Atmega328	1
	Switch	S.P.S.T switch	1
	AA Battery	Alkaline LR6	6
	IRF 540	N-Channel Power Mosfet	2
	DC Motor	5V, 0.8Ω DC Motor	2
	Resistor	10k, 150Ω	2
	Diode	Schottky Fly-back Diode	2

Figure 15: Component Specifications and Quantities for the Mouse Build

9.4 Source Code for Arduino Microcontroller

```
/*
 * MICROCONTROLLER FOR MOUSE DESIGN
 * TYPE 2 PID SYSTEM
 * Written by Will Powell, Mark Kogan and Ela Semaj
 */
/// PID values optimised by MatLab simulations
const int kp = 2;
const int ki = 2;
const int kd = 2;

/// Motor's PWM Values for Coast Speed (speed in a straight line) and Turn
Speed (speed when turning around a corner)
const int coastSpeed = 128;
const int turnSpeed = 40;

/// Pick up coil input ports, after filtering and amplification
const int PUCL = A0;
const int PUCR = A1;
/// Motor Output ports (pwm output)
const int motorL = 5;
const int motorR = 6;
/// PID Terms, initially set to zero
float PID = 0;
float P = 0;
long I = 0;
float D = 0;
// Timing variables
unsigned long loopTime, currentTime, previousTime = 0;
// Error variables
float error, previousError = 0;
int turnError = 0;


void setup()
{
  Serial.begin(115200); // set baud rate
  // sets input and output pins
  pinMode(PUCL, INPUT);
  pinMode(PUCR, INPUT);
  pinMode(motorL, OUTPUT);
  pinMode(motorR, OUTPUT);
}

void loop()
{
  currentTime = millis(); // reads current time
  loopTime = currentTime - previousTime; // calculate time taken for one
loop
  previousTime = currentTime; // sets previous time to the time at last
loop
  calcError(); // calculates current error

  if (error < -turnError){turnRight();} // if error is a large negative
value, turn mouse right
  else if (error > turnError){turnLeft();} // otherwise if error is a large
positive value turn mouse left
  else{
      calcPID(); // calculates current PID value
      // sends PWM signal with PID control to motor
      analogWrite(motorL, coastSpeed + PID);
      analogWrite(motorR, coastSpeed - PID);
```

```
/*
 * MICROCONTROLLER FOR MOUSE DESIGN
 * TYPE 2 PID SYSTEM
 * Written by Will Powell, Mark Kogan and Ela Semaj
 */
/// PID values optimised by MatLab simulations
const int kp = 2;
const int ki = 2;
const int kd = 2;

/// Motor's PWM Values for Coast Speed (speed in a straight line) and Turn
Speed (speed when turning around a corner)
const int coastSpeed = 128;
const int turnSpeed = 40;

/// Pick up coil input ports, after filtering and amplification
const int PUCL = A0;
const int PUCR = A1;
/// Motor Output ports (pwm output)
const int motorL = 5;
const int motorR = 6;
/// PID Terms, initially set to zero
float PID = 0;
float P = 0;
long I = 0;
float D = 0;
// Timing variables
unsigned long loopTime, currentTime, previousTime = 0;
// Error variables
float error, previousError = 0;
int turnError = 0;


void setup()
{
  Serial.begin(115200); // set baud rate
  // sets input and output pins
  pinMode(PUCL, INPUT);
  pinMode(PUCR, INPUT);
  pinMode(motorL, OUTPUT);
  pinMode(motorR, OUTPUT);
}

void loop()
{
  currentTime = millis(); // reads current time
  loopTime = currentTime - previousTime; // calculate time taken for one
loop
  previousTime = currentTime; // sets previous time to the time at last
loop
  calcError(); // calculates current error

  if (error < -turnError){turnRight();} // if error is a large negative
value, turn mouse right
  else if (error > turnError){turnLeft();} // otherwise if error is a large
positive value turn mouse left
  else{
      calcPID(); // calculates current PID value
      // sends PWM signal with PID control to motor
      analogWrite(motorL, coastSpeed + PID);
      analogWrite(motorR, coastSpeed - PID);
```

```
  }

}

// function to calculate current error, where positive error is right of
line and negative is left
void calcError()
{
  previousError = error; // records previous error
  error = analogRead(PUCR) - analogRead(PUCL); // error is difference
between right PUC and left PUC
}

// function to calculate PID, updated each loop
void calcPID()
{
  P = error; // P is simply the error
  I += error * loopTime; // I is the integral or area of error-time graph
  D = (error - previousError)/loopTime; // D is the derivative or change in
error with respect to time
  PID = (kp * P) + (ki * I) + (kd * D); // PID is the sum of all terms
mutliplied by their constants.
}

void turnRight()
{
      analogWrite(motorL, coastSpeed - turnSpeed);
      analogWrite(motorR, coastSpeed + turnSpeed);
}

void turnLeft()
{
      analogWrite(motorL, coastSpeed + turnSpeed);
      analogWrite(motorR, coastSpeed - turnSpeed);
}
```