Mouse Design Report

Written by Alexander Evans Moncloa: 13109, 13090, 13217

***Abstract*— The purpose of this task is to create an automated electric vehicle 'mouse' that follows a 20m electromagnetic track in the lowest time possible. This paper outlines how the mouse circuitry was created, implemented and modified along with a discussion of the Arduino code used to incorporate PID control and other time optimisations through software. Finally, mechanical designs for improved velocity are outlined and overall vehicle speed results are discussed.**

I. INTRODUCTION

The challenge given was to design a battery powered 'mouse' to automatically follow a specific pre-determined track. This 20m track was dictated by a 1A peak-to-peak wire, which ran at approximately 20kHz of AC current. This 'mouse' had a pre-determined chassis, motors, wheels and electromagnetic sensors however the remaining mechatronics were entirely modular as long as the total budget was under £30.

The track consisted of a declining ramp at the start, a multitude of sharp bends followed by a large gentle bend, a long straight section of track, an inclining ramp and a short gentle bend to finish the track.

The main priority was to create a mouse that could complete this track without external assistance, then total speed began to play a role in the design process. Weight, size and centre of mass were always accounted for along with friction and drag, although fine tuning the PID algorithm along with software improvements were most conducive to overall speed gains.

Having the option of choosing to create an analogue system or a digital system, we chose the latter due to the freedom software gave us in processing the electromagnetic signals and determining appropriate motor speeds.

II. SYSTEM ARCHITECTURE

Due to the mouse chassis being a rear-wheel drive, a system capable of powering two rear-motors was required in order to move the mouse. Four 1.5V batteries were placed in series in order to provide 6V to the op-amps and motors. A 9V battery was used to power the Arduino and another 9V battery was wired in reverse to power the op-amps.

In order for the mouse to follow the electromagnetic track, the mouse had to be capable of sensing the electromagnetic fields emitting from the active wire, and appropriately adjust the individual motor speed of each motor in order to remain on-track. This required 2 coil sensors to be present at the front of the mouse, placed approximately 4mm apart with the wire 2mm from each coil.

The signals from each coil are intrinsically sinusoidal, therefore a common point on each wave had to be chosen in order to compare values detected by each coil. Choosing any point other than the peak value would be harder to detect, therefore the signal processing circuit had to have a peak detector.

Here we had the option to stay analogue and feed the raw current into the transistors, however, this did not allow us to use code to alter input and output values. The freedom a digital system provided caused us to purchase an Arduino to process the peak coil values.

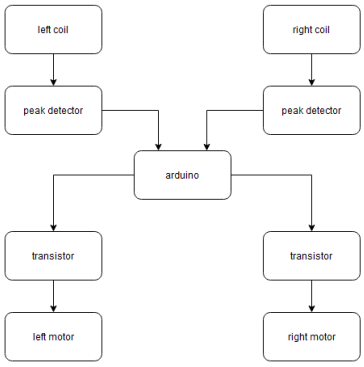

Figure 1: Control system diagram.

Within the Arduino, the software would run a loop every millisecond where the coil sensor values would be manipulated and an output would be generated. This process can be seen in *Figure 2*. Due to the inverse square law causing measured sensor values to drop off exponentially as the distance from the wire increases, the values were linearised using a mathematical function. This allowed for less extreme readings as the mouse deviated from the centreline, reducing oversteer which in turn increased stability.

Error was then calculated, which facilitated the creation of a PID algorithm. This algorithm calculated the appropriate output using error to estimate the exact amount of current each motor should receive in order to drive parallel to the track.

A tilt switch was to be included in order to signal to the Arduino when to increase motor speed regardless of sensor values, along with an in-built timer that was to signal that the general speed should be increased; this was desired as the track had a straight path near the end where there was more room for error. The timer was omitted from the second run however, due to inconsistencies in lap time during the race day.

The transistors were preventing the current from the batteries from flowing directly into the motors. From the Arduino, current was fed to each transistor base as a fraction of 255 using the PWM functionality. This would cause the motor to only receive a fraction of the maximum power available from the batteries per millisecond, controlling the speed of each wheel. The system can be seen in *Figure 1.*

It was assumed that this mouse would have to be custom made to the 20m track that was being used during the race. The boost timer was only an appropriate feature for the given racetrack due to the shape, however, the tilt switch would be useful for all racetracks with an incline. The decrease in speed any vehicle encounters at inclines is due to the added force of gravity pulling the vehicle backwards. The ability to counteract this force with a short burst of power from the batteries is an effective method to decrease time taken to complete any track.

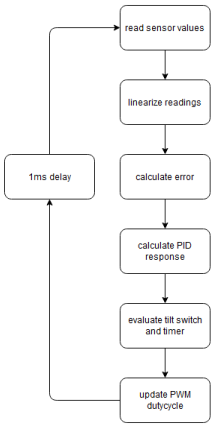

Figure 2: Data processing within Arduino.

III. SUBSYSTEM DESIGN

In order to create this system, the circuitry must be appropriate and perform the functions required. Firstly, it was noted that the input from the sensor coils was too small to be detected by the Arduino. The signal had to be boosted with an 071 op-amp per coil. The op-amps were configured with resistors in a manner to cause a 38x gain using a 4.7kΩ resistor coupled with a 180kΩ resistor.

Due to the signal being very erratic, a 68nF capacitor and a 220nF capacitor were connected in parallel to each sensor in order to smoothen out the outputs. The magnified signals were then sent from the op-amps through a diode then straight to pins A1 and A3 in the Arduino. The output was also grounded with a capacitor and resistor to retain the maximum peak value.

The Arduino then provides an output current at D5 and D6, going into the base of the transistors that control the power flow to the motors.

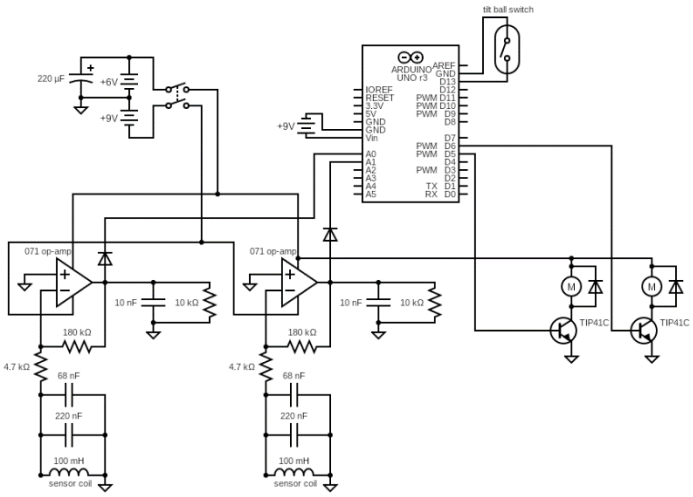

Figure 3: Full circuit diagram.

The Arduino had to be powered with a minimum of 5V and the 071 op-amps needed +5V and -5V to work optimally. A 9V battery powered the Arduino and a -9V battery was used for the op-amps. There was a switch to save energy and facilitate the racing process. The tilt ball switch was connected directly to the Arduino due to the sensor not needing any form of magnification or smoothening.

After the circuit diagram was created, a render of how the circuit would look on a prototyping board was made. The Arduino itself was excluded from the render along with switches, motors and batteries due to them not needing to be soldered onto the prototyping board. *Figure 4* shows the render with soldered wires going into any Arduino port or motor adequately labelled (Note: I and III refer to ground for motors 1 and 2. II and IV refer to the active component).

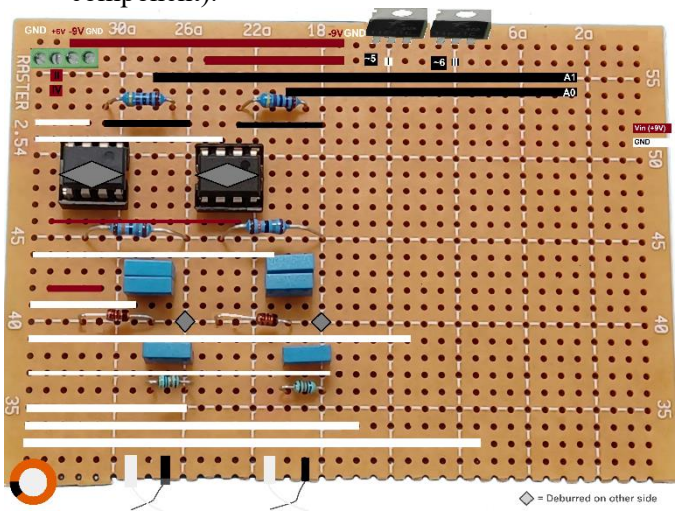

Figure 4: Colour-coded board render.

This render was then used to solder all the seen electronics on a prototyping board with adequately coloured and sized wires.

Op-amps were used due to their extremely high processing speeds, although the requirement of another 9V battery caused a significant increase in mass, which may have led to slower speeds.

Transistors were used due to the lack of power an Arduino can provide. The maximum current it can provide is 40mA, not enough to move the motors, therefore transistors were needed to preserve the high 2A current drawn from the four 1.5V batteries in series.

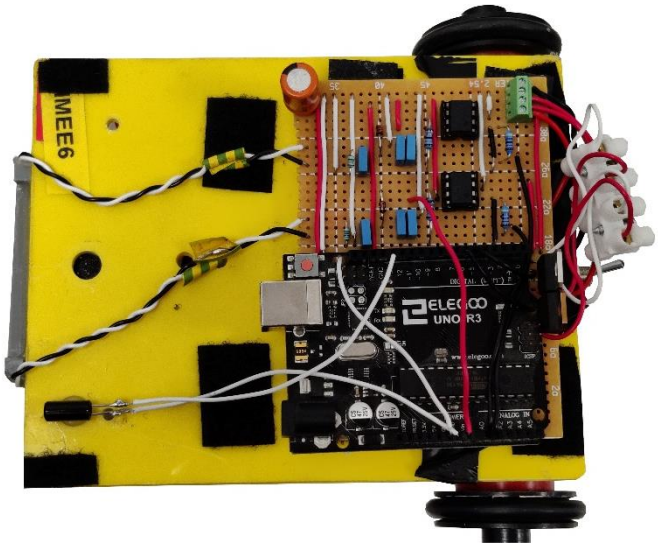

Figure 5: Initial system set-up before changes to the transistor placement, Arduino powering, transistor modularity and the implementation of 6V batteries (previously using 9V batteries).

The assembly choice of using Velcro to attach the prototyping board to the chassis and the Arduino to the prototyping board proved useful when a wire broke and had to be re-soldered. Alternatively, using the given metal rods and screws would be heavier and did not allow for as low a centre of mass as we desired.

The op-amps and transistors also were not soldered on directly, therefore any overheating malfunctions could be dealt with easily on race day with handheld removal and replacement.

Certain cables, such as those connecting the batteries to the prototyping board or the board to the motors, were not soldered on as to allow flexibility of replacements in the event of battery failure or motor failure.

The entire subsystem, bar the tilt switch and sensor coils, was placed as near to the wheels as possible. This was because its weight increased downforce on the wheels, which lead to greater traction on the track. Weight at the back of a rear-wheel drive led to less inertia from gravity when the mouse had to rapidly move at an angle away from its previous trajectory.

The front of the chassis originally had a heavy steel ball bearing which was replaced with a lightweight 3D-printed frame. The frame had aluminium strips on all points with contact to the track to decrease friction, increasing speed.

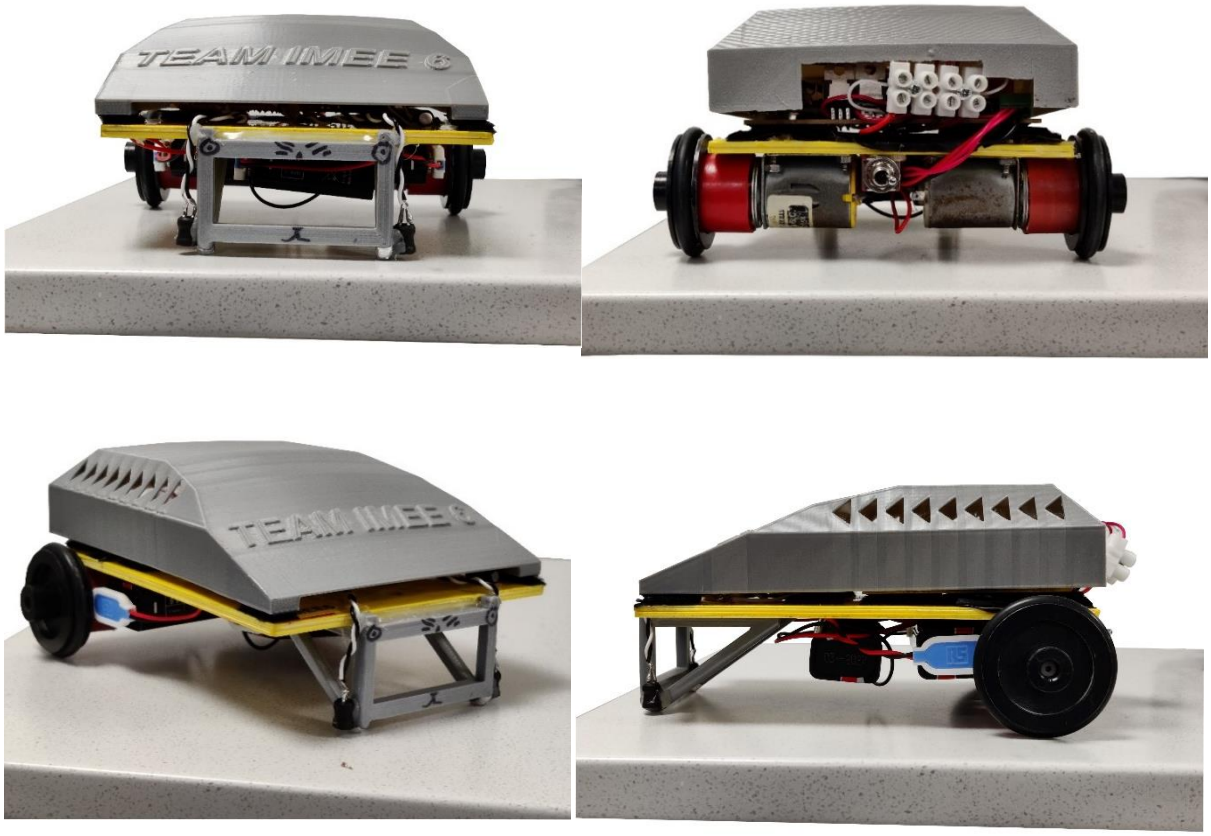

Figure 6: Multiple views of the completed mouse with chassis fully mounted.

Finally, the chassis had a protective 3D-printed exterior shell attached with Velcro. This provided safety for internal electric components during unwanted collisions, increasing the mouse's reliability on race day.

Small triangular vents were present to allow cold air to cool down internal components such as op-amps and transistors on race day.

The chassis also prevented drag to a degree and increased downforce on the rear-wheels due to cumulative air pressure, however, the speed the mouse was travelling at caused the change to be imperceptible.

IV. RESULTS

It was paramount that this mouse was able to stay on top of the track and not differ too greatly from it. When the PID values were adjusted and the proportional or derivative gain was too low or the integral gain was too high, the mouse would not turn fast enough to keep up with the initial bends in the racetrack. This issue caused the mouse to fall off the track in testing a few times (only when it was going too fast to catch). A particularly powerful collision caused severe damage to the entire system, which in turn inspired the 3D-printed chassis for protection, aerodynamism and aesthetics.

By increasing the proportional and derivative gain and decreasing the integral gain, the mouse was better at sticking to the racetrack. The downfall of this was that the mouse began oscillating powerfully during the turns and even more during the straight, causing the vehicle to often fall off the racetrack at the straight.

This was a serious issue which was at first mitigated but took innovation to truly eliminate. Fine tuning the PID further resulted in our first time-tested lap, with an impressive 20.5 seconds. By adding the boost feature at 11000 milliseconds (approximately when the mouse reached the straight) the time was reduced to 18.5 seconds. Multiple weeks of testing and fine tuning the overall speed and PID values only resulted in a small overall decrease in lap time, resulting in a best time of 17 seconds.

Changing the PID values were either not reducing the oscillations on the straight enough or not keeping the mouse over the track at the beginning meanders. We implemented a dual PID system where as the boost timer was activated, a new set of PID values would be introduced to cause less oscillations on the straight and allow for more velocity. This system reduced the overall time to 16.5 seconds, however it was not repeatable due to changes in battery power causing the exact finely tuned values to no longer work at keeping the mouse on track.

We were not sure how to improve overall speed from here, so we saved the code that gave us a reliable lap time of 20 seconds and decided to look into the signal processing algorithm in the Arduino more closely. Upon inspection, it was noted that the detected electromagnetic field from the wire decreased exponentially as the coil travelled further away from the wire. This followed the inverse square law. Only at this stage did we realise that linearising the amplified input should fix the oscillations, as these oscillations were due to the steep decrease in signal strength as distance increased linearly.

With the linearisation function active, the Arduino began processing smaller errors at points of great deviation from the wire. This caused the mouse to follow the track less aggressively, allowing for a higher general cruising speed and facilitating a much more meaningful boost speed at the now modified boost timer.

These changes caused both of the transistors to overheat during the testing phase, leading to a lack of motor speed control. The transistors were appeared damaged and were leading to much lower motor speeds. An initial solution to this was to desolder the transistors (as they were originally soldered on) and replace them with female pin inputs to allow modularity and easy transistor replacements. Unfortunately, new transistors did not solve the overheating problem we were having. Upon inspection, it appeared that the transistors were experiencing switching loss due to the Arduino PWM signals. The original circuitry (as can be seen in *Figure 5*) had high side switching, therefore the idea of changing to low side switching was proposed. A comparison of the two methods were made on a breadboard and there was evidently less leakage current with the low side switching set up. No transistors were heating up (determined by touch). Connected directly to 9V and 1A, the wheel speed was 1200rpm. With a high side switching circuit, the maximum wheel speed dropped to 650rpm. Low side switching maintained a top speed of 1200rpm therefore the subsystem was redesigned accordingly (see *Figure 4*).

The speed boost at the ramp in the original system also was malfunctioning. This was due to the current draw from the motors at 200/255 speed rising above 750mA. The Arduino required 250mA to operate and the two 9V batteries in parallel powering both the Arduino, op-amps and motors had a total current of roughly 1A. The ramp boost would therefore lead to the Arduino to momentarily switching off, causing an entire power system redesign to be needed.

This redesign consisted of four AA batteries in series to replace two 9V batteries. The current doubled to 2A, however the voltage was reduced to 6V which required an extra 9V battery purely for the Arduino. The new system had a total of 6 batteries underneath the vehicle, increasing the total weight and inertia. All previous finely tuned PID values had to be recalibrated 3 hours before the race itself.

The final performance on race day was a first run of 15.4s at stage 8, a 21.61s completed lap and 18.3s at stage 8.

The first run was much faster than expected for unknown reasons, which was an issue due to the timed speed boost being set between 12s and 17s. It reached stage 8 too early, causing the boost and altered PID values to be active for the last turn.

The boost function was removed entirely for the second run and a safer code that had consistently achieved a 19s time was used. For another unknown reason, the run was much slower than expected and ended at 21.61s.

The last run restored the boost from 12s to 14s.

Unfortunately, it ran much slower than the first run and suffered from a tilt switch failure due to the mechanical ball inside not falling properly.

The parts required to build this mouse cost under half the budget, as seen in *Figure 7*.

name	quantity	cost
perf board	1	1.1
resistor	6	0.0022
sensor coil	2	0.244
film cap	6	0.115
electrolytic cap	1	0.236
diode	4	0.0075
TIP41C	2	1.33
9v battery	2	1.65
PLA filament	1	0.1
Arduino uno	1	2.11
wire	1	0.1
switch	1	1.15
tilt switch	1	0.31
NiMH cells (8)	1	2.66
	total:	14.9472

Figure 7: Mouse parts list.

V. DISCUSSION

A key strength of our group was fantastic communication. We were dedicated to messaging to each other in detail at most times of the day for months, using a messaging service that facilitated file sharing. We also consistently met up in person to work on issues together, decreasing the time it would take to solve an issue if only one person was tackling it.

We were also good at task delegation, breaking up complex tasks into 3 smaller manageable tasks that each member of the group could work on individually between collective sessions. This was very useful as there were no conflicts on how a small task should or shouldn't be done as we had faith in the abilities of whoever was delegated that task.

If a delegated task was too challenging for an individual, we were all comfortable speaking up and asking the collective for help. There was no shame in not understanding something, therefore stalling or fixating on a small problem did not hold us back on an individual level.

An issue in our group was that we would sometimes have an innovation block as a collective and as a group be stuck on a certain issue for very long. What we should have done was ask other colleagues or experts how to tackle these issues instead of keeping the problems inside our bubble. This may have led to us discovering the linearisation function earlier, which could have led to an even more optimised mouse than what we ended up having.

The Arduino had a 500Hz PWM default frequency. We did not know this could be altered and in hindsight we would have greatly increased this to allow for much greater speeds.

VI. CONCLUSION

The goal of the mouse design project was to create a small automated electric vehicle that, for under £30, was capable of following a charged wire for 20m in the lowest time possible. This task was completed successfully with an automated vehicle that cost £15 and travelled the distance in 21.61 seconds. What made our mouse stand out and excel was the combination of efficient, compact electronics with highly developed software that was streamlined with efficient hardware. This made the mouse good at processing data, accurate at turning with the right speed and deal with less inertia and drag whilst benefitting from more downforce on the rear wheels.

V. APPENDIX

```
// Define pins for left and right motors
const int leftMotorPin = 5; //1217 rpm
const int rightMotorPin = 6; //1203 rpm

// Define pins for left and right sensors
```

```
const int leftSensorPin = A0;
const int rightSensorPin = A1;

// Define variables for PID controller
float error = 0;
float previousError = 0;
float integral = 0;
float derivative = 0;
float pidOutput = 0;
int timer = 0;

// Define variables for sensor readings
float leftSensorValue = 0;
float rightSensorValue = 0;
bool tiltSwitch = 0;
bool firstSwitch = 0;

// define base movment speed
int cruseThrottle = 90; //80 is possable with fresh batteries and 150 on
the straight

//create system timer
int time = millis();

void setup() {
  // Set the motor pins as outputs
  pinMode(leftMotorPin, OUTPUT);
  pinMode(rightMotorPin, OUTPUT);
  pinMode(13, INPUT);
  // Initialize the serial monitor
  Serial.begin(9600);
}

void loop() {
  // Define constants for PID controller
  float kp = 0.15; //works at 7.8v
  float ki = 0.0;
  float kd = 0.3;
  // Read the sensor values
  //leftSensorValue =
int(sqrt(1024.0/float(analogRead(leftSensorPin)+1))*1024.0);
  //rightSensorValue =
int(sqrt(1024.0/float(analogRead(rightSensorPin)+1))*1024.0);
  leftSensorValue = sqrt(analogRead(leftSensorPin)+1)*40;
  rightSensorValue = sqrt(analogRead(rightSensorPin)+1)*40;
  tiltSwitch = ! digitalRead(13);

  // Calculate the error
  error = rightSensorValue - leftSensorValue + 47;

  // Calculate the integral and derivative
  integral += error;
  derivative = error - previousError;

  // Calculate the PID output
  pidOutput = kp * error + kd * derivative + ki * integral;

  // Update the previous error
  previousError = error;

  //set value of throttle for this loop
```

```
  int throttle = cruseThrottle;

  //tilt ball switch latching timer
  if (tiltSwitch){
    firstSwitch = 1;
    timer = 50;
  }

  //decrement tilt ball switch timer
  if (timer > 0){
    throttle = 200;
    timer = timer - 1;
  }

  // Set the motor speeds based on the PID output
  int leftMotorSpeed = (throttle - pidOutput);
  int rightMotorSpeed = (throttle + pidOutput);

  //ensure motor values are valid
  leftMotorSpeed = abs(leftMotorSpeed);
  rightMotorSpeed = abs(rightMotorSpeed);

  // Set the motor speeds using PWM
  analogWrite(leftMotorPin, leftMotorSpeed);
  analogWrite(rightMotorPin, rightMotorSpeed);

  // Print the sensor values and PID output to the serial monitor
  /*
  Serial.print("Left sensor value: ");
  Serial.print(leftSensorValue);
  Serial.print(",");
  Serial.print("\tRight sensor value: ");
  Serial.println(rightSensorValue);
  /*
  Serial.print(",PIDoutput ");
  Serial.print(pidOutput);
  Serial.print(",leftmotorspeed ");
  Serial.print(leftMotorSpeed);
  Serial.print(",rightmotorspeed ");
  Serial.print(rightMotorSpeed);
  Serial.println();
  Serial.println(error);
  */

  // Delay for a short period of time
  delay(1);
}
```