SCHOOL OF ELECTRONIC AND ELECTRICAL ENGINEERING

UNIVERSITY OF BIRMINGHAM

Integrated Mechatronic Project Report

Group B

Group Member:

Lamis Alsuwaidi

Arushi Madan

Wenjun Ma

Anika Rahmam

Wen Wu

Hang Xu

I. Introduction

This report outlines the conceptual design, functioning, construction and testing of a current-following autonomous robot vehicle, as well as the science and technology behind it. A comprehensive narrative of a full design process along with the executed testing strategy is offered and physical deliverables are displayed in the form of photos and graphics. This robot is designed to follow a wire that is carrying a dual tone current which is a composition of two sinusoidal signals with different frequencies (1 kHz and 2 kHz). In addition, the wire is buried beneath an unknown track and the depths may vary within a range.

II. System Architecture

a) System Overview

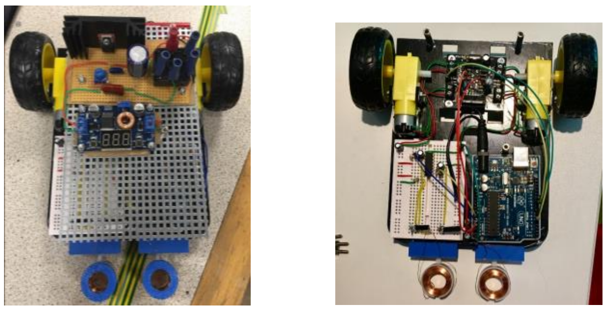


Figure.1 Robot Structure (two layers)

To start with, the chassis is built by acrylic sheet and mesh sheet. Two wheels and a ball caster are attached on the acrylic sheet to ensure mobility. We chose a plastic ball caster instead of a metal one as metal would influence the sensing of the coils. Each wheel is connected to a DC motor to drive the car while the ball caster plays the role of a driven wheel. Instead of using the motors given in the lab, we chose smaller motors which worked at 3-6 volts and a rated speed of 100 rounds per minute[1]. Comparing with the motors provided, this small DC motors has many advantages such as it is easier to drive. In the case of turns, it will be easier for the small wheels to turn for it has a relatively smaller friction and lighter weight. Besides, two adjustable sensor holders are made through 3D printing and are attached in front of the car as shown in Figure.1. The modular design of the robot is basically a simple two-layers top-down structure with the topper layer contributing to the whole power processing unit and the lower layer to signal processing.

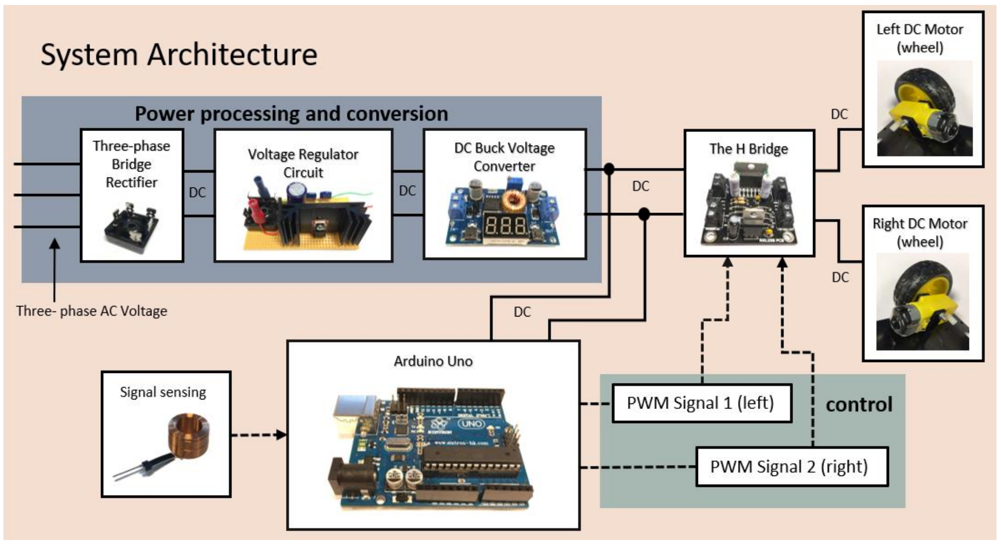


Figure 2. System Architecture

1 See reference [2]

The system architecture of the car is shown in Figure.2. It can be divided into several sub systems. First, the power processing and conversion system which consist of rectifier, voltage regulators are used to convert the three-phase AC power supply into a relatively smooth DC voltage to power other components including the H Bridge and Arduino chip. Instead of getting power supply directly from the power processing system, two DC motors gain voltage from the H bridge which can adjust the voltage level as the PWM signals change. Speaking of the control part, the PWM signals sent to the H Bridge is controlled by Arduino chip through coding. Changes in duty cycle of the PWM signal will result in changes in the average voltage provided to the motors. These changes are triggered by the signal sensed through the coil sensors in some way that will be discussed later.

b) Control System

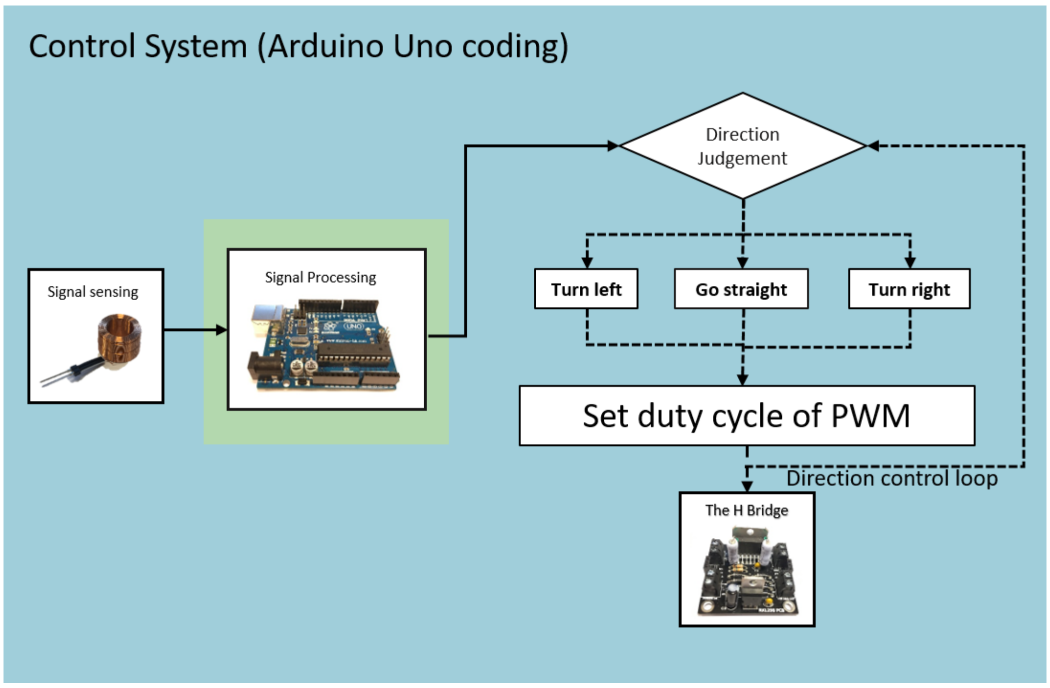


Figure.3 Control System

The way how sensed signal affect the changes of PWM signal can be described as Figure.3. After processing the signal sensed by the coil sensors in Arduino through coding, the position of the car compared to the wire is determined and a command of either turn left, go straight or turn right will be sent. Each command has a corresponding value of duty cycle already set in the program. Once the command is sent, the duty cycle of PWM signal will change to the corresponding value. After that, the direction judgment will be done again and again which forms a closed loop control. This is only a simplified version of the control system while the whole control part will be introduced later.

c) Signal Processing

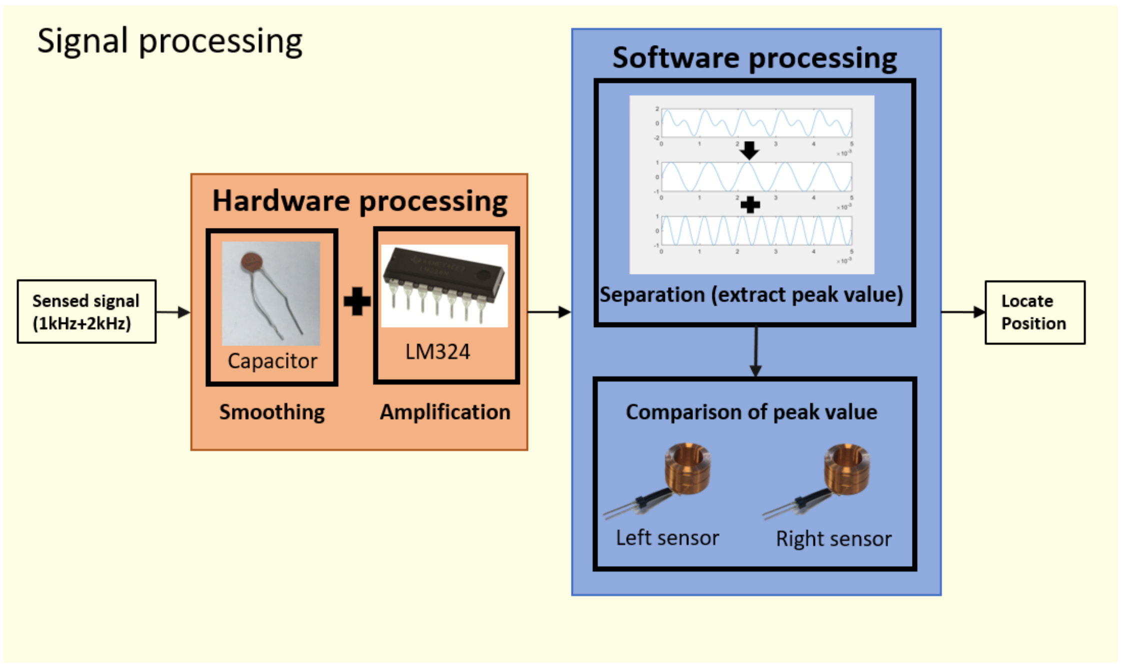


Figure.4 Signal Processing

The signal processing process is shown below in Figure.4. Two coil sensors are symmetrically placed in the front of the car deck to sense the magnetic field created by the current. The actual data sensed and processed is the induced voltage across the coils as there is a close relationship between the magnetic field and the induced voltage. Capacitors are connected to both the input and the output of an amplifier (LM324) to smooth the signal. The combination of capacitors and the LM324 op-amp forms the hardware processing process in order to get an amplified and relatively smoother signal. This is followed by the software processing which is realized through coding. First is to separate the two signals with different frequencies and extract the amplitude of each signal and by comparing the amplitude of signals got from two different sensors can the car be located. Details are discussed in later chapters. For signal processing, instead of using the PIC18F27K40 which is introduced before, Arduino chip is implemented for its faster processing speed. And there are tons of resources online which is really useful for self-studying. However, it turns out that Arduino coding may lead to some problems which may not happen under the circumstances of using PIC.

III. Power Conversion and Supply Conditioning

The power supply is provided through a three-phase umbilical in the voltage range 15-40 volts and 2 amps. As a result, a universal power supply module is required as the power supply is not a known value. Main function of the power conversion system is to convert the power supply into different levels of DC voltage to power other components in the whole system. There are two basic steps to realize the function which are voltage rectification, voltage regulation.

a) Voltage rectification

Figure.5 SBR3506 3-phase Bridge Rectifier

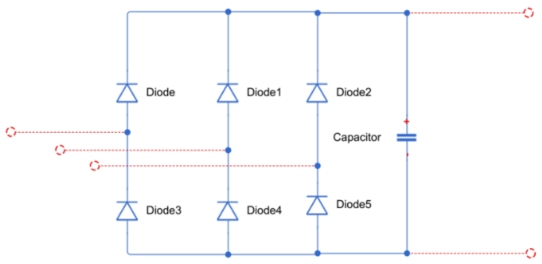


Figure.6 Working Circuit of a 3-phase Bridge Rectifier

All the components used, including the DC motors, are powered by DC voltage. As a result, the very first step is to convert the three-phase AC power into DC power. An integrated three-phase bridge rectifier (HY Electronic Corp SBR3506, 3-phase Bridge Rectifier, 35A 600V, 5-Pin SBR) shown in Figure.5 is implemented into the system. According to the datasheet, the minimum peak reverse voltage is 140 volts which is larger than input power supply. Thus the rectifier can function in its normal working range[2] and won't be broken. It is made up of six basic diodes, and the circuit within the shell looks like the diagram in Figure.6 (without the capacitor). There are five pins in total, three of which are input pins and the other two is the output pins. By using the properties of diode, the rectifier can convert three-phase voltage into DC voltage.

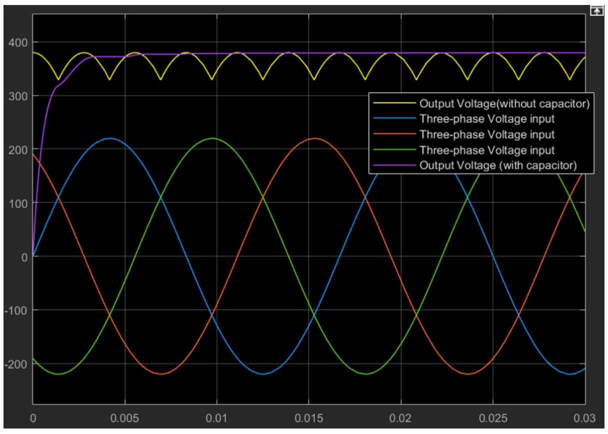


Figure.7 Input/Output Voltage of a Three-phase Rectifier

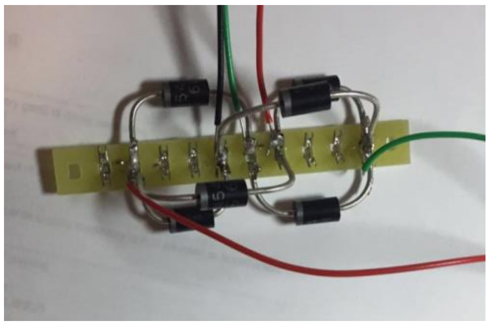


Figure.8 Three-phase Bridge Rectifier (first version)

Theoretically, the relationship of output and input voltage can be determined by $V_o = \frac{3V_{ll}}{\pi}$. Through calculation[3] can be known that the output of the three-phase rectifier is approximately range from 14.32 to 38.20 volts. The input/output voltage curves are shown in Figure.7. From $V_{ripple} = \frac{I_{load}}{2fC}$, we can derive a proper value for capacitor which is in parallel with the rectifier to smooth the voltage . In Figure.8 is the first version of the three-phase bridge rectifier which is soldered manually. After testing, the result of the first version of rectifier is not very ideal.

2 See reference [1]
3 See appendices b) 1

Also, either the structure of diodes or the position to put the rectifier can not be easily adjusted once the soldering is done which is not suitable for the modular design. For these reasons, an integrated rectifier is bought for a better result.

b) Voltage Regulation

In order to maintain a steady output DC voltage of certain value, a voltage regulation circuit based on the implementation of a LM317T Voltage Regulator (Figure.9) and a DC buck volt converter (Figure.10) is built in the system following the three-phase rectifier. Voltage regulation basically means removing the ripples and providing a stable DC output. The regulation circuit can be described in two parts which are the pre-regulation circuit and the step-down regulation circuit.

1. Pre-regulator Circuit

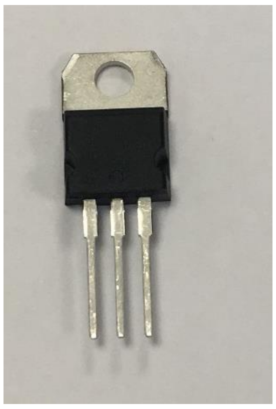


Figure.9 LM317T Voltage Regulator

Figure.10 DROK® Small DC Buck Volt Converter

The function of the pre-regulator circuit is first to regulate the DC voltage obtained from the rectifier as it is not stable enough to power the system. On the other hand, the maximum input voltage for the DC buck volt converter used for voltage step-down is 38 volts[4] while it has been calculated before that the output of the rectifier can reach 38.2 volts. Therefore, another function of the pre-regulator circuits is ensuring a suitable input voltage for the following circuit. As described in the datasheet, LM317T can bear an input up to 40 volts[5] which meets the need. The working circuit diagram[6] is shown in Figure.11. All parameters are determined by $V_o =$

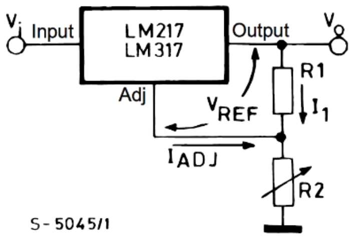


Figure.11 Pre-regulator Circuit

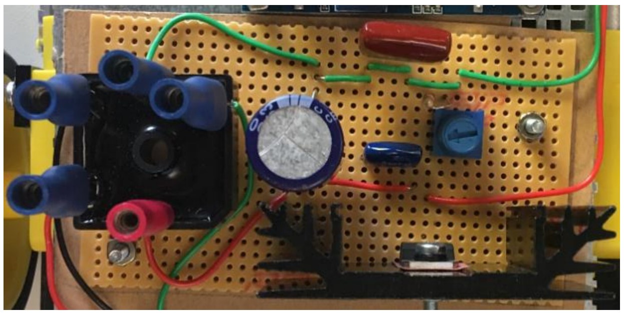


Figure.12 Pre-regulator Circuit on Veroboard (include rectifier)

4 See Reference [3]
5 see Reference [5]

$V_{REF}\left(1+\frac{R_2}{R_1}\right)+I_{ADJ}R_2$. Two capacitors can be connected in the circuit for smoothing. By inserting a potentiometer R2 in the circuit, LM317T can output different voltages range from 1.2 to 37 volts. After being tested on the breadboard to make sure the circuit can work, it is soldered on the Veroboard to enhance the connection (Figure.12). Since the LM317T voltage regulator will dissipate a large amount of heat during working, a cooling sink is added to prevent burns.

2. Step-down Regulation Circuit

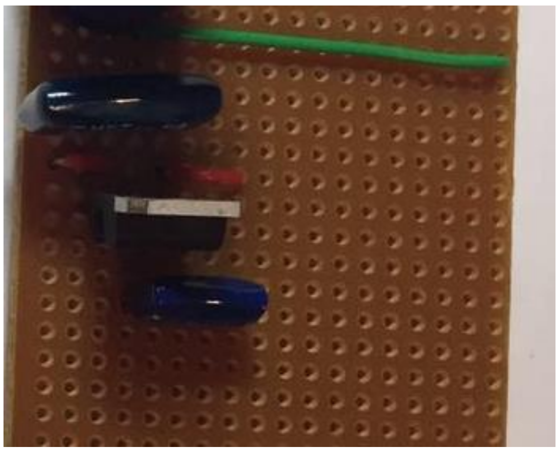


Figure.13 LM7805 Regulator Circuit

From the circuit built so far, steady DC voltage can be obtained, but the voltage value is not very accurate. As a result, A DC Buck volt Converter can be used to lower the voltage to a fixed voltage between 1.25 and 36 volts. It integrates the function of an autotransformer and a voltage regulator. The input and output voltage can be monitored in the LED digital display module. In practice, both the H Bridge and the Arduino chip need a power supply of 9-volt-DC voltage which can be provided by adjusting the winding ratio. And the H bridge will provide power for the DC motors and 5-volt-DC voltage for the op-amp. Above in Figure.13 is the initial design of the regulation circuit using LM7805, however, the result isn't very ideal as well and the output voltage is fixed at 5 volts[7] which is not adjustable. For these reasons, we choose to use the DROK® Small DC Buck Volt Converter instead of LM7805.

Ⅳ. Electromagnetic Position Sensors

In order to track the wire precisely, the most important steps are to figure out how the sensors work and how will the data sensed be analyzed and processed.

7 see reference [4]

a) Tracking Strategy

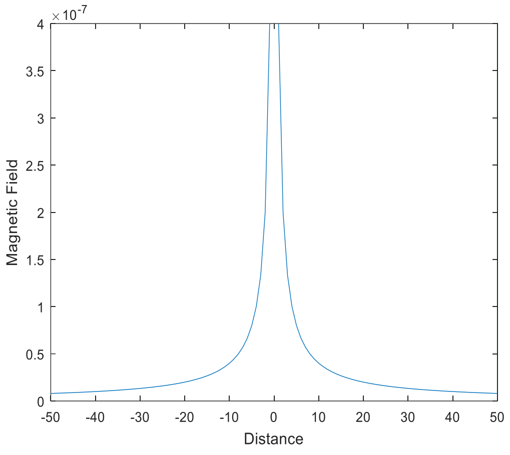


Figure.14 Magnetic Flux Density VS. Distance

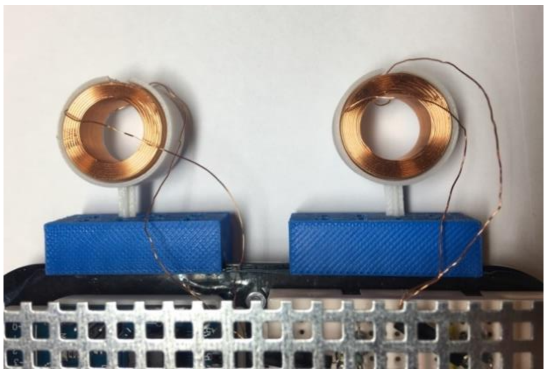


Figure.15 Two coil sensors placed in front of the car

The equation $B = \frac{\mu_0 I}{2\pi r}$ shows that distance is inversely proportional to magnetic field density. This implies that the closer to the wire carrying current (i.e. the smaller the value of 'r'), the bigger magnetic flux density we'll attain (i.e. larger value of 'B'). The relationship can be inferred from Figure.14. As a result, we can easily judge the position of the robot with respect to the wire by comparing signals obtained from two sides of the robot. Two coil sensors are placed in front of the car symmetrically to get data (Figure.15). The sensor holders and the holder bases are built through 3D printing which implement the technology of cross pin which can be seen in Figure.16. The holders are connected to a base-like module with cross holes in it which is attached to the bottom of the car (Figure.17). This enables the holder to change position as well as fixing it while testing.

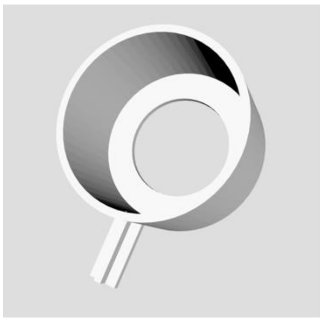


Figure.16 Sensor Holder Model

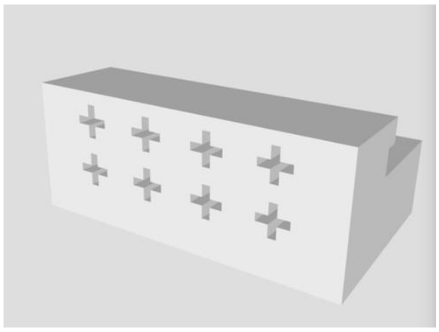


Figure.17 Sensor Holder Base

Instead of sensing the magnetic flux density directly, the coil sensor is actually sensing the induced voltage across them which should looks like Figure.18 as the current signal looks like Figure.19. It can be simply derived that the relationship between current signal and the sensed data is consistent with the equation $= -N\frac{\mu_0 S}{2\pi r}\frac{dI}{dt}$[8]. From Figure.18, we can easily extract the maximum and minimum value through coding and get the amplitude to do comparisons.

8 Derivation see Appendices b) 3

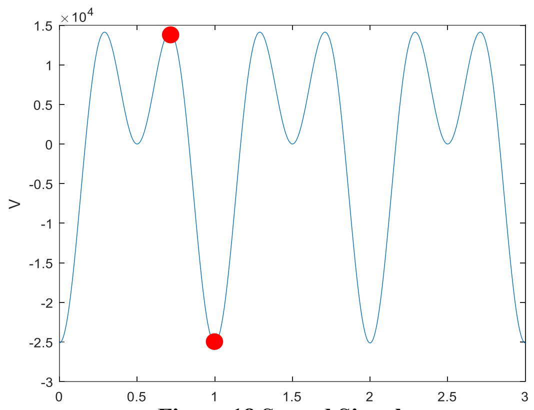


Figure.18 Sensed Signal

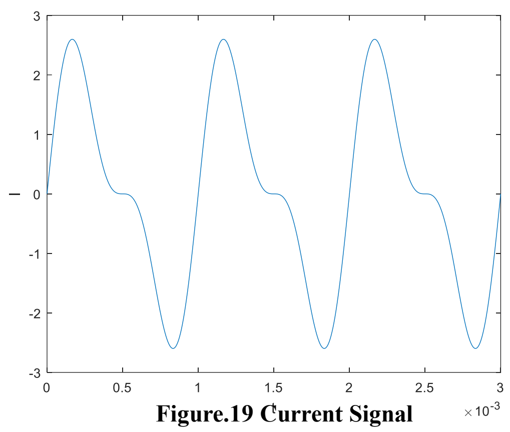


Figure.19 Current Signal

b) Hardware Signal processing

The signal processing unit consists of the hardware processing and the software processing. The hardware processing bit mainly contains the op-amp circuit with capacitors as filters.

1. Testing signal

```
SampleNumber=32;
Freq1=1e03;
Freq2=2e03;
t=(0:1/Freq1/32:1/Freq1);
S=sin(2*pi*Freq1*t)+sin(2*pi*Freq2*t);
```

Figure.20 Matlab Code for Look-up Tables

```
#int_timer2
void Timer2_Service_Routine()
{
    Port.debug = 0b1;
    Port.cs = 0b0;
    spi_xfer((SineTab1e[Select_Table][Look_Up_Table_Index])>>8);
    spi_xfer((+SineTab1e[Select_Table][Look_Up_Table_Index])&0x00FF);
    Port.cs = 0b1;
    Look_Up_Table_Index=++Look_Up_Table_Index % 32;
    Port.debug = 0b0;
}
```

Figure.21 PIC18F27K40 Code for Testing Signal

In order to carry out the strategy mentioned before, we built a testing signal which simulate the tracking wire first. It is built through Matlab and PIC18F27K40 coding which has been done before in EE2A labs. Matlab is used to generate a look-up table values of sinuslidal wave which contains frequency of 1 kHz and 2kHz (Figure.20). Within PIC18F27K40, we use SPI to output look-up table values to the digital-to-analogue converter. LDAC signal is produced by PWM module which is triggered by timer2, overflowing every 31.25 μ s (Figure.21). However, the result in Figure.22 is not an ideal and smooth signal which can cause glitches in sensed signal at a later stage. Therefore, a "LM324" op-amp coupled with resistors and capacitors (Figure.24) were used to smooth out the signal as shown in Figure.23. First two resistors are introduced to divide the voltage while the input voltage into the op-amp will be approximately ten times

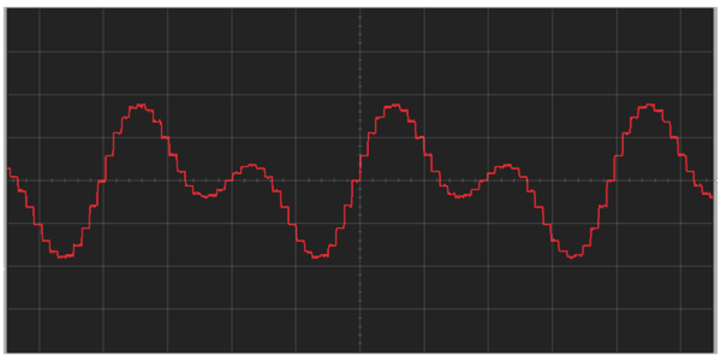


Figure.22 PIC18F27K40 code for testing signal

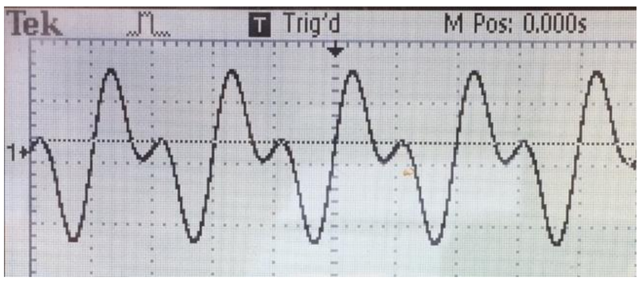


Figure.23 Testing signal after hardware processing

smaller. This will enable the 0.1 uF capacitor to smooth the signal. After that, through the amplification circuit can the amplitude of the signal be restored to the original level.

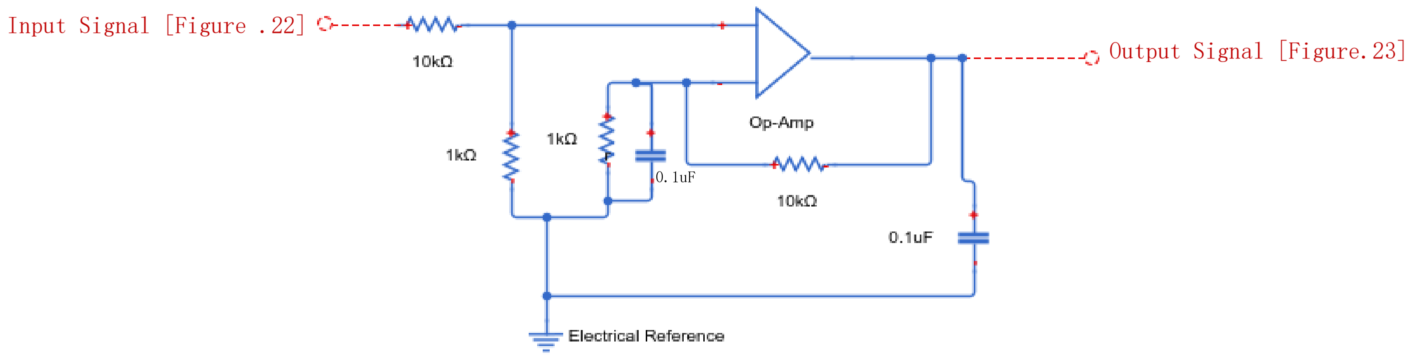


Figure.24 a “LM324” op-amp working circuit for smoothing testing signal

2. Sensed signal

As seen in Figure.18, the sensed signal theoretically has negative values, which causes an issue due to the fact that the “Arduino Uno” microcontroller used is incapable of reading negative volts. Therefore, an offset of 0.1 volt is applied to senser coils by connecting two resistors to one side of the coil to distribute the voltage as seen in Figure.25.

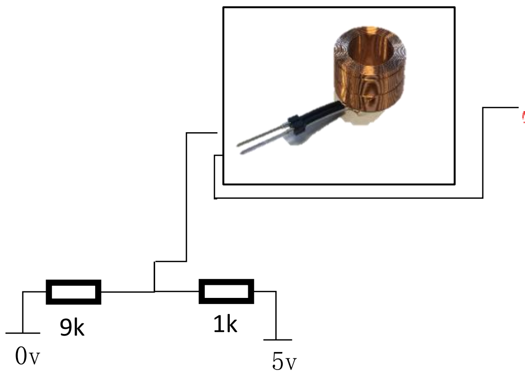


Figure.25 The offset applied to the sensing coil

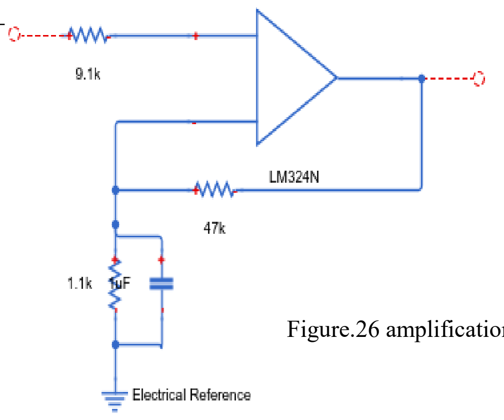


Figure.26 amplification of the sensed signal

As originally designed in the project, the wire is buried beneath the surface track. Considering that the signal sensed by the coil may be too weak, we build another amplification circuit (Figure.26). The testing wire has a current of 6 mA as measured. The peak value of the signal sensed is in the unit of 0.1 volts by a rough estimation. We wanted the signal to be as large as possible but still within the readable range of Arduino chip at the same time. Therefore, the amplification circuit is designed to amplify about 50 times[9].The sensing coil is then tested at different distances from the test signal generated previously. It can be observed in Figure.27

9 Derivation see Appendices b)4

and Figure.28 that the amplitude of the sensing coil signal is larger when the sensor coil is only a short distance from the testing signal and smaller when the distance is increased (The sensor shall not be directly on the top of the tracking wire). However, the signal is still very vague as the capacitor does not exist. When the capacitor is present, both signals are smoothed out as seen in Figure.29 and Figure.30.

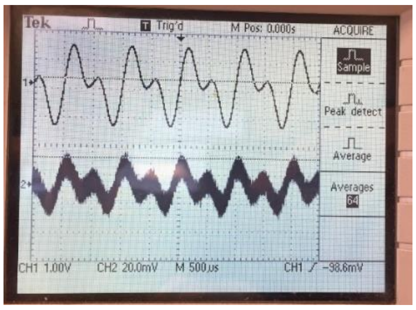


Figure.27 Signal without capacitor /away from the test signal wire.

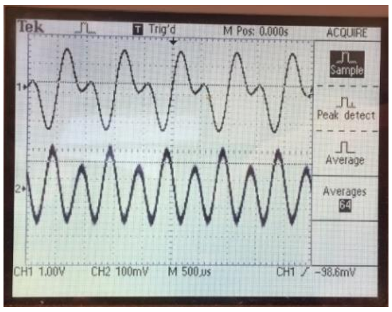


Figure.29 Signal with capacitor /away from the test signal wire.

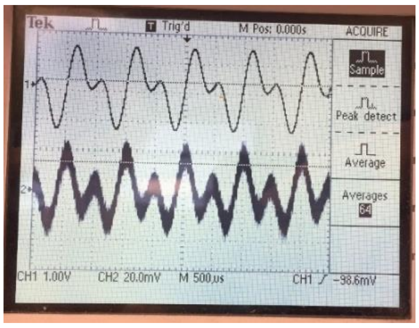


Figure.28 Signal without capacitor /close to the test signal wire.

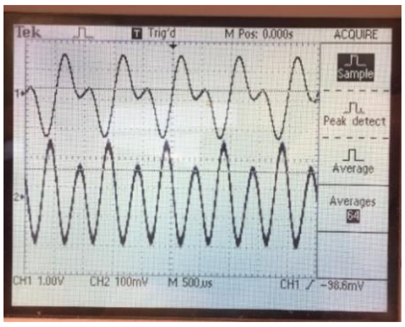


Figure.30 Signal with capacitor /away from the test signal wire.

c) Software Signal Processing

1. ADC sampling

Prescaler	Conversions/sec
2	615,385
4	307,692
8	153,846
16	76,923
32	38,462
64	19,231
128	9,615

Figure.31 Conversion rate correspond to prescaler value

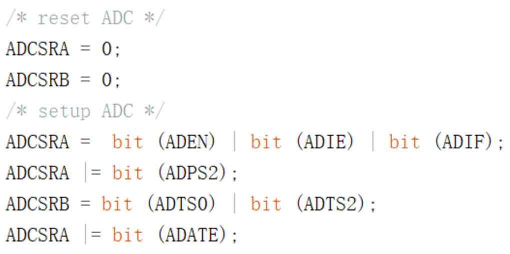
```
/* reset Timer1 */
TCCR1B = 0;
TCNT1 = 0;
/* setup Timer1 */
TCCR1B = bit (CS11) | bit (WGM12);
TIMSK1 = bit (OCIE1B);
OCR1B = 39;
```

Figure.32 Timer setting

In order to get enough samples to ensure a complete and usable waveform, we need the sampling frequency of ADC to be 50 kHz. Although by adjusting the prescaler value can we change the sampling frequency, the frequencies are limited to several certain value[10] shown in

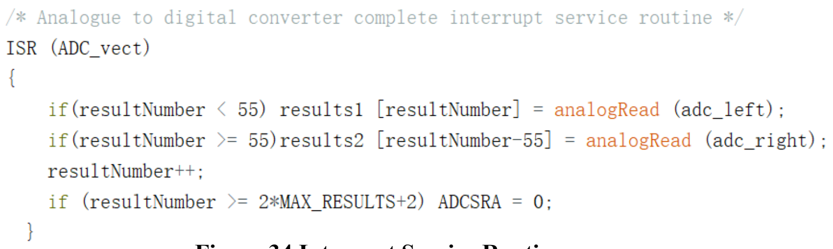
10 See reference[6]

Figure.31. To solve this problem, a timer and ADC interrupt routine are introduced. The timer is set to overflow every 20 μs (Figure.32) to trigger the ADC interrupt service routine (Figure.33). For there are two sensors need to be read, the ADC interrupt service routine set the reader to shift between two sampling channels (adc_left represents pin A0 which is connected to the left coil, adc_right represents pin A1 which is connected to the right coil) (Figure.34). In order to prevent disorder in samples caused by the time spent on shifting channel, ADC reader will wait until it gets 55 samples one side and then shift channel. The data are



```
/* reset ADC */
ADCSRA = 0;
ADCSRB = 0;
/* setup ADC */
ADCSRA =  bit (ADEN) | bit (ADIE) | bit (ADIF);
ADCSRA |= bit (ADPS2);
ADCSRB = bit (ADTS0) | bit (ADTS2);
ADCSRA |= bit (ADATE);
```

Figure.33 ADC setting



```
/* Analogue to digital converter complete interrupt service routine */
ISR (ADC_vect)
{
    if(resultNumber < 55) results1 [resultNumber] = analogRead (adc_left);
    if(resultNumber >= 55)results2 [resultNumber-55] = analogRead (adc_right);
    resultNumber++;
    if (resultNumber >= 2*MAX_RESULTS+2) ADCSRA = 0;
  }
```

Figure.34 Interrupt Service Routine

stored in a buffer which contains 55 numbers. It will clear its storage for new data once the number of the samples exceed 55.

2. Moving Average Filter

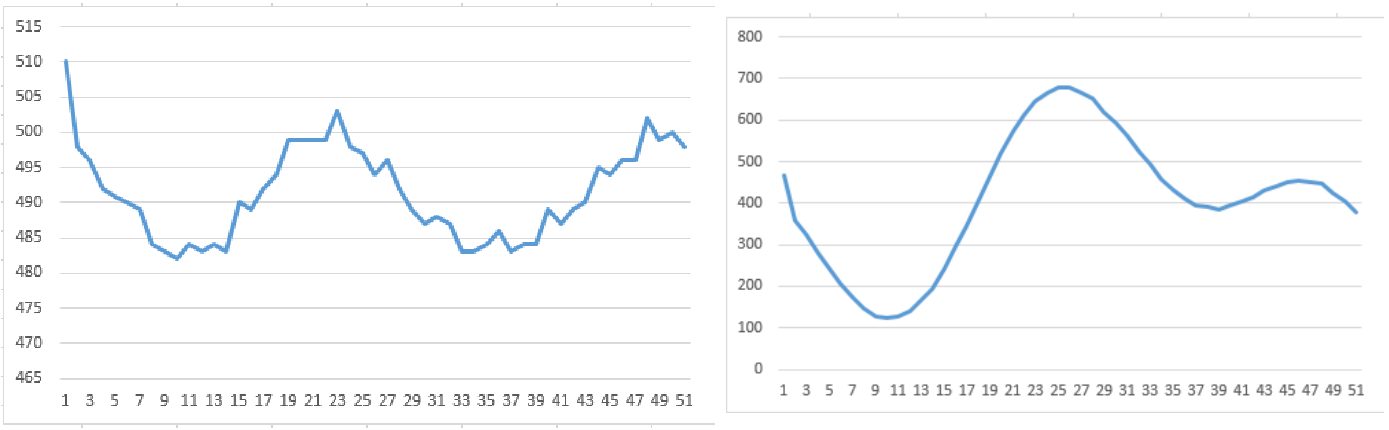


Figure.35 Unfiltered Raw Data (left) VS.   the Filtered Data(right)

After successfully obtaining the sampled data, we output the data to draw the curve of the signal. It can be seen in Figure.35 that the signal is very rough and has many glitches. A moving average filter is introduced to filter the signal. It simply divides the 55 numbers into 50 groups each of which contain 5 numbers (D_1-D_5, D_2-D_6, D_3-D_7 …, D_{50}-D_{55}). Instead of using the raw unfiltered data, we used the total value added up of each group thus forming a new group of 50 numbers which outlined the curve shown in Figure.35. The reason why we'd use the total value of each group rather than the average value is that it actually uses the software to amplify the signal for five times. The signal become smoother which made it easier for later processing. In Figure.36 is the code related.

```
/* apply moving average filter */
for (int i=0;i<50;i++)// taking average over every 5 samples
{
  filter1[i]=results1[i+1]+results1[i+2]+results1[i+3]+results1[i+4]+results1[i+5];
  filter2[i]=results2[i+1]+results2[i+2]+results2[i+3]+results2[i+4]+results2[i+5];
}
```

Figure.36 Application of Moving Average Filter

3. Extracting Maximum Possible Amplitude

```
/* find the maximum magnitude of data */
int filter1_min=filter1[0];
int filter1_max=filter1[0];
int filter2_min=filter2[0];
int filter2_max=filter2[0];
for (int i=0;i<50;i++)//find the max and min value
{
  if(filter1[i]<filter1_min) filter1_min=filter1[i];
  if(filter1[i]>filter1_max) filter1_max=filter1[i];
  if(filter2[i]<filter2_min) filter2_min=filter2[i];
  if(filter2[i]>filter2_max) filter2_max=filter2[i];
}
int magnitude_left=filter1_max-filter1_min;
int magnitude_right=filter2_max-filter2_min;
```

Figure.37 Extracting Maximum Possible Amplitude

As the amplitude of a signal is determined by the difference of its maximum and minimum value. In order to extract the maximum possible amplitude, we need to find the largest value and the smallest value in the signal sensed. This can be realized by doing comparison in a loop which shown above in Figure.37. In theory, when the robot is in the middle of the track, the amplitude of the signal sensed on each side should be almost the same. However, due to the slight structure and sensitivity difference of two coils, there's an error in the sensed signal.

right sensor			left sensor			
Position Deviation	Amplitude		Position Deviation	Amplitude		Quotient
-2	110		-2	40		2.75
-1	180		-1	100		1.8
-0.5	450		-0.5	270		1.666666667
0	520		0	300		1.733333333
1	250		1	160		1.5625
2	160		2	120		1.333333333
3	100		3	70		1.428571429

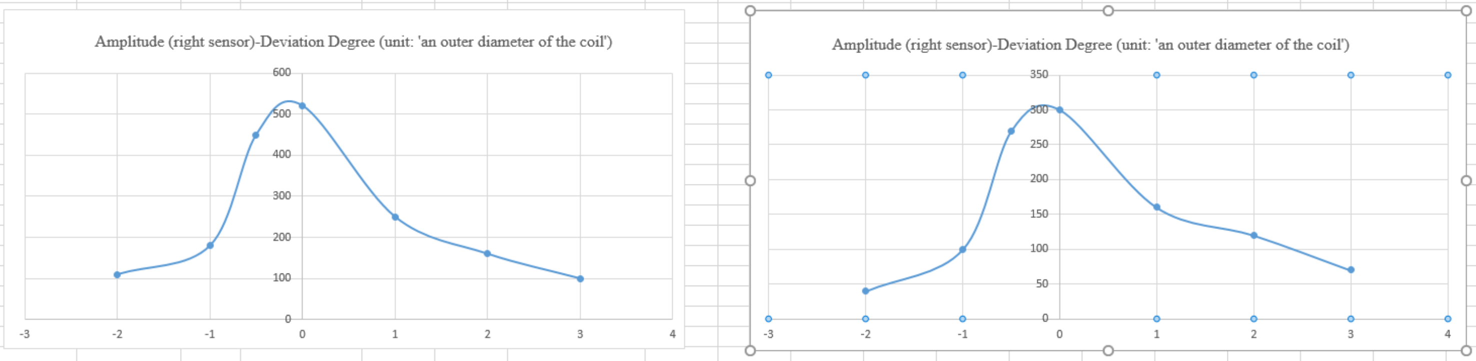
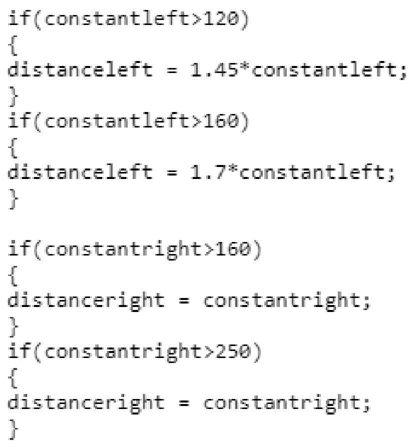


Figure.38 Amplitude VS. Deviation Degree

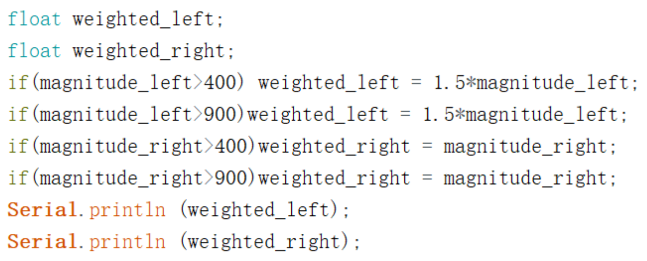
As shown in Figure.38, we collected the amplitude data of sensed signal on both sides and



```
if(constantleft>120)
{
distanceleft = 1.45*constantleft;
}
if(constantleft>160)
{
distanceleft = 1.7*constantleft;
}

if(constantright>160)
{
distanceright = constantright;
}
if(constantright>250)
{
distanceright = constantright;
}
```

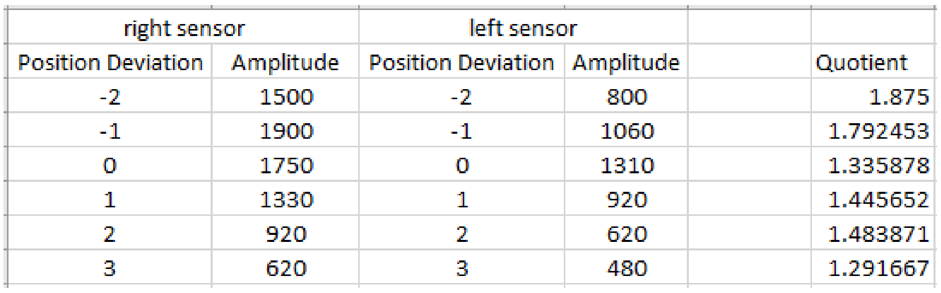
Figure.39 correction parameter



```
float weighted_left;
float weighted_right;
if(magnitude_left>400) weighted_left = 1.5*magnitude_left;
if(magnitude_left>900)weighted_left = 1.5*magnitude_left;
if(magnitude_right>400)weighted_right = magnitude_right;
if(magnitude_right>900)weighted_right = magnitude_right;
Serial.println (weighted_left);
Serial.println (weighted_right);
```

Figure.40 correction parameter for racing

plot a graph showing the relationship between the signal amplitude and degree of position deviation (distance between the outside of the wheel and the track). The position deviation is measured in the unit of ‘an outer diameter of the coil’. The strategy is to slightly pan the robot to right and then to left but meanwhile keep the testing signal wire in between two coil and measure the data sensed. It can be seen in the table that the data obtained from left sensor is always smaller than the theoretical value. To solve the problem caused by the different sensitivity of two coils, we multiply the data obtained from the left sensor with a correction parameter which can be seen in the code (Figure.39). By this correction method, the robot successfully run the testing track and pass through a crossover as can be seen in the video link[11]. .



right sensor		left sensor			
Position Deviation	Amplitude	Position Deviation	Amplitude		Quotient
-2	1500	-2	800		1.875
-1	1900	-1	1060		1.792453
0	1750	0	1310		1.335878
1	1330	1	920		1.445652
2	920	2	620		1.483871
3	620	3	480		1.291667

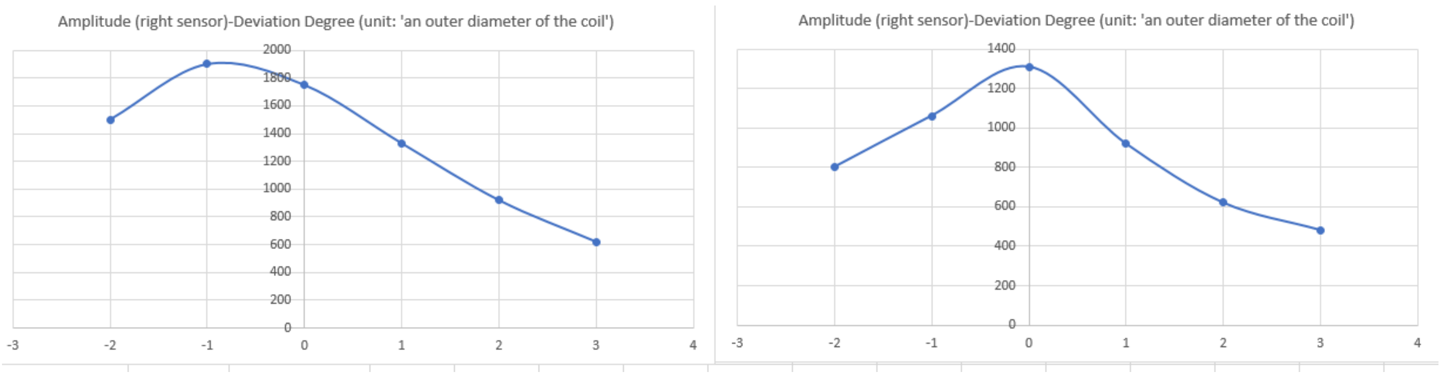


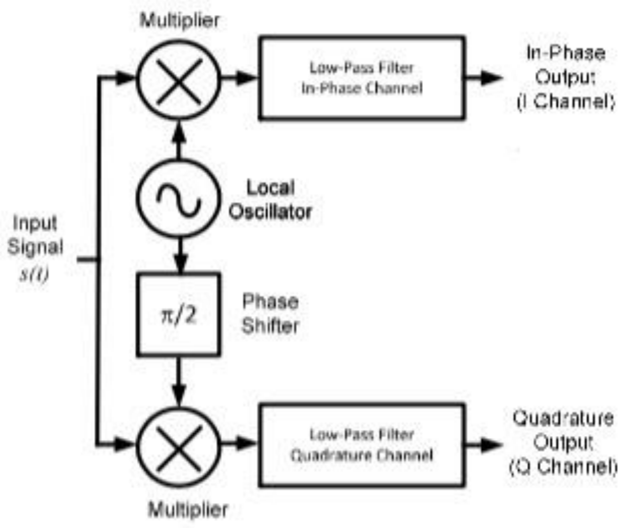
Figure.41 Amplitude VS. Deviation Degree (Racing day)

11 See Appendices a)[1]

Data must be collected in advance if applying this method. On the race day, we collected the data (Figure.41) and calculated the correction parameter before to make the robot more precise. It successfully pass the tracking wire as shown in the video link[12] . The code can be seen in Figure.40

However, since the given current signal has been changed for other group to test, the correction parameter might not be applied accurately. As a result, the robot tracked the wire less precisely when it ran the track for the second time.

4. More accurate method



Multiplier
Low-Pass Filter In-Phase Channel
In-Phase Output (I Channel)
Local Oscillator
Input Signal $s(t)$
$\pi/2$
Phase Shifter
Low-Pass Filter Quadrature Channel
Quadrature Output (Q Channel)
Multiplier

Figure.42 Q/I modulation

An improvement of signal processing is using Q/I modulation which means multiply the sensed signal by a 1kHz local oscillator to separate the 1kHz component. And to separate the 2kHz component by multiply the sensed signal by a 2kHz local oscillator. The simulation matlab

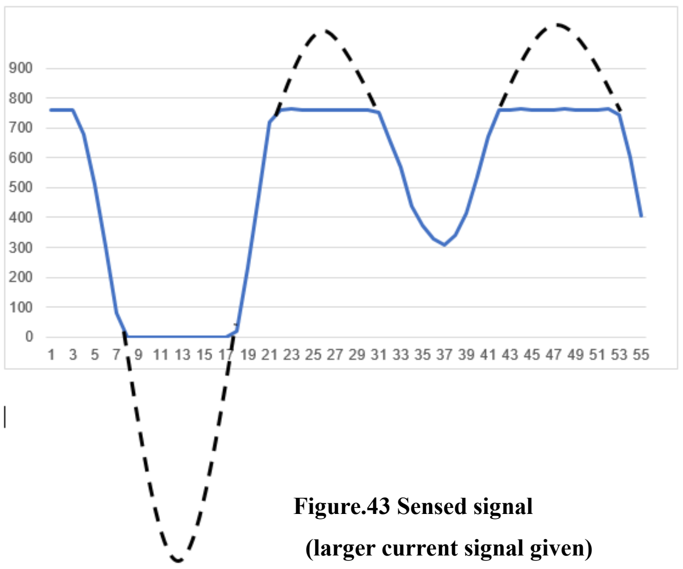


Figure.43 Sensed signal

(larger current signal given)

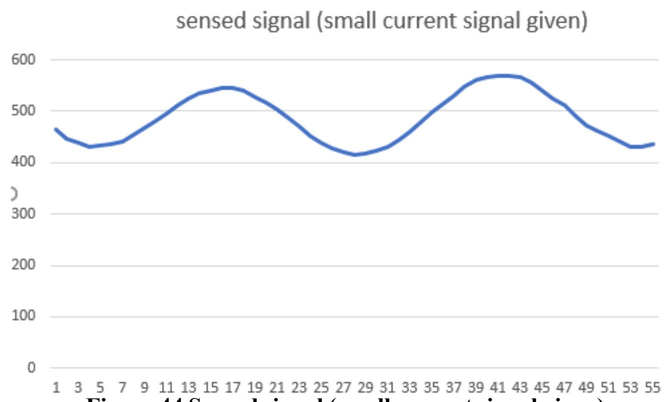


Figure.44 Sensed signal (small current signal given)

12 See Appendices a)[2]

code can be seen in the reference[13]. The essence of this modulation is basically doing Fourier Transform to directly get the amplitude of each component. However, the testing current signal we built is only of 6 mA (peak to peak) which is very small. Due to this reason, the sensed signal oscillates within a small range (Figure. 44) which make it difficult to use Fourier Transform. While testing on the racing day with a much larger current signal, the data sensed (Figure.43) is ideal enough to use Fourier Transform to do the separation.

V. Closed-loop DC Motor Armature Current Control

As illustrated in the architecture section, PWM signal is generated in Arduino and output to the H-bridge, which drives the motor, thus forming the control system of the robot. The system starts with a closed-loop current control, which is implemented to adjust the armature current according to feedback and keep it at the desired value. The closed-loop current controller takes the classical form as illustrated in Figure.45[14]. The plant includes gain, a pulse-width-modulator, an H-bridge driver, and the DC motor armature. And the feedback contains the sense resistor and the analogue-to-digital converter. The detailed control block diagram is shown in Figure. 46.

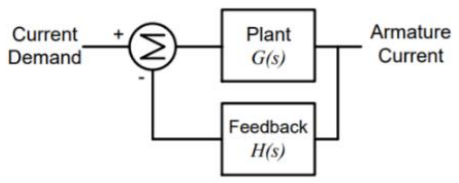


Figure.45 Classical closed-loop feedback control

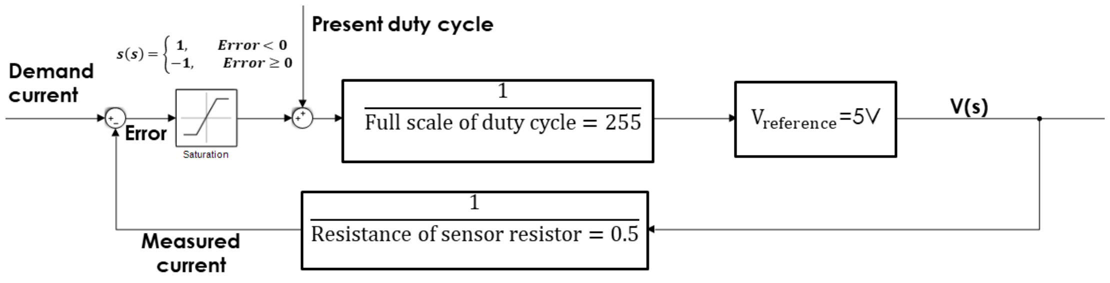

Present duty cycle
$s(s)=\begin{cases}1, & Error<0\\ -1, & Error\geq 0\end{cases}$
Demand current
Error
Saturation
$\frac{1}{\text{Full scale of duty cycle} = 255}$
$V_{reference}$=5V
V(s)
$\frac{1}{\text{Resistance of sensor resistor} = 0.5}$
Measured current

Figure.46 block diagram for closed-loop current control

The value of armature current flowing is measured using a low-value (0.5 Ω) sense resistor. [15]Dividing the potential difference appearing across this sense resistor by the resistance gives the measured armature current, which is then compared to the demand current value. The difference is fed into a saturation function where it gets amplified. If the error is positive, the duty cycle should decrease, otherwise it should increase. The duty cycle is than divided by the full scale which is 255 and multiplied by the reference voltage of H-bridge which is 5V. The final voltage is then used to drive the DC motors. Thus, the closed-loop current controller is formed. Figure.47 shows the code to implement the loop. Due to the difference in accuracy between measured value and demand value, a decision threshold of 2 (full scale of PWM is 255) is applied. Besides, a saturation detection judgment is added to prevent PWM duty cycle from going bigger than 255 or smaller than 0.

13 See Appendices a)[3]
14 Philip Atkins-EE2A USB Serial Interface Experiment 4-1.pdf
15 Philip Atkins-EE2A USB Serial Interface Experiment 4-1.pdf

```
if(result_number==MAX_RESULT)//negative feedback
// adjust duty cycle to modify set value according to feedback data
{
   adc_average = adcresult;
   if(DutyCycle>=255)  DutyCycle=255;// detect saturation
   if(Desire_Result-adc_average>=2) DutyCycle++;
   if(Desire_Result-adc_average<=-2)
   {
      if (DutyCycle<=2) DutyCycle=0;
      // detect saturation and add threshold
      else DutyCycle--;
   }
   analogWrite(PWM_output,DutyCycle);// output PWM signal
   result_number=0;
}
```

Figure.47 Code for armature current control

However, in practical, the collected feedback signal contains high frequency noise due to the use of PWM, which will cause error in comparison and calculation. In order to suppress noise and recover the original signal, we need to process data first to increase signal to noise ratio. There're many methods of reducing noise such as slowing down ADC sampling rate and applying a filter. In order to get sufficient samples to make accurate judgment, we could not decrease the sampling frequency. And the process should be simple and fast as it takes place within the ADC interrupt routine which is triggered every 20 μs. So, we could not apply a filter to the data as it contains loops which take too long. After trading off, the method of taking weighted average of collected data is selected. The block diagram and the code are shown in Figure. 48 and Figure.49 respectively.

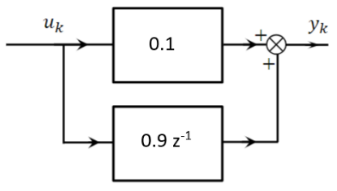
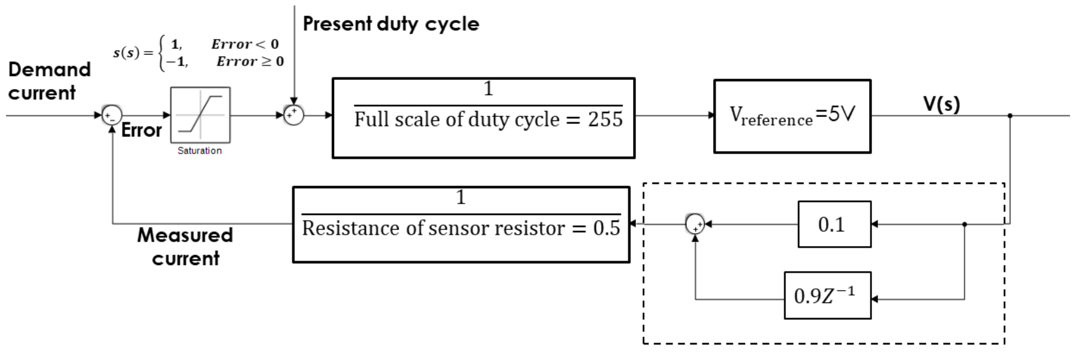


Figure.48 Block diagram for noise elimination

```
if(result_number<MAX_RESULT)
// If ADC result buffer isn't full
{
      adcresult = (0.9*adcresult)+(0.1*ADC);
      // reduce noise by taking weighted average
      result_number++;
}
```

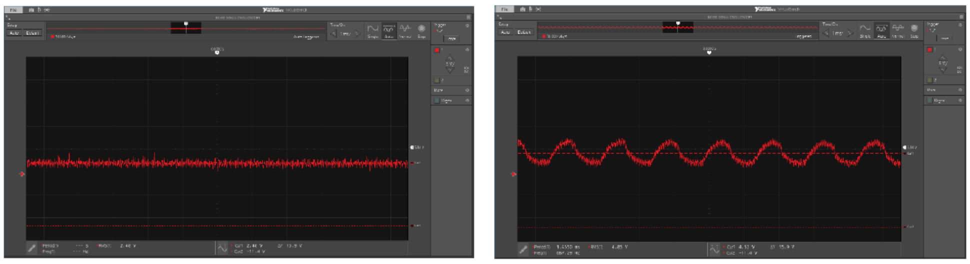
Figure.49 Code for noise reduction

At this point, the closed-loop DC motor armature current control is completed. The complete block diagram is shown in Figure.50. And the result is shown in Figure.51, where we can find that the actual current (I_{rms}) is approximately equal to the demand current with error less than 1%.



Present duty cycle
$s(s) = \begin{cases} 1, & Error < 0 \\ -1, & Error \geq 0 \end{cases}$
Demand current
Error
Saturation
$\frac{1}{\text{Full scale of duty cycle} = 255}$
$V_{reference}$=5V
V(s)
$\frac{1}{\text{Resistance of sensor resistor} = 0.5}$
0.1
$0.9Z^{-1}$
Measured current

Figure.50 block diagram for complete closed-loop



Demand armature current=500:
Vrms=2.48V, Irms=2.48/5=0.496A
Error=(0.496-0.5)/0.5=-0.8%

Demand armature current=900:
Vrms=4.52V, Irms=4.52/5=0.904A
Error=(0.904-0.9)/0.9=0.44%

Figure.51 Results of applying closed-loop armature current control

VI. Higher-Level Closed-loop Velocity Control

Controlling the movement of the vehicle mainly means to control the velocity. As velocity is a vector which has both magnitude and direction, we could divide the whole control system into two parts accordingly.

a) Speed control

Based on the armature current control, we could then build the speed control by adding the part shown in Figure.52 into the loop, where R is the armature resistance which is 0.66 Ω, L is the armature inductance which is 0.28 H, K is the motor constant which is 0.017 v*s/rad, and J is the inertia which is 0.0067 kgm2[16].Input of the system V(s) is the output voltage of the H-bridge and output of the system ω(s) is the angular velocity of the DC motor.

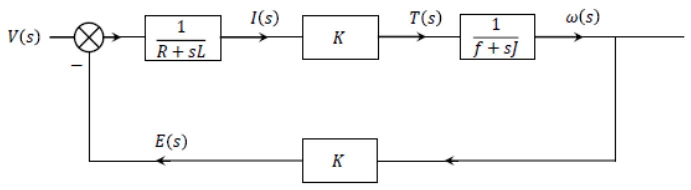


Figure.52 block diagram for speed control

As electrical time constant is much faster than mechanical time constant, armature inductance can be assumed as 0. Therefore, we could derive the transfer function of the system as follows:

$$\Psi(s) = \frac{\omega(s)}{V(s)} = \frac{K}{RJs + Rf + K^2}$$

The pole of the transfer function is $-\frac{RF+K^2}{JR}$.[17] Because all the parameters here are positive, the pole of the system will always be negative, which means that the system will always be stable.

16 Derivation see appendices b) 2
17 Derivation see Appendices b) 2

Combined with the closed-loop control of armature current, we can now derive the relationship between angular velocity of the motors and the command in Arduino. The complete speed control diagram is shown in Figure.53.

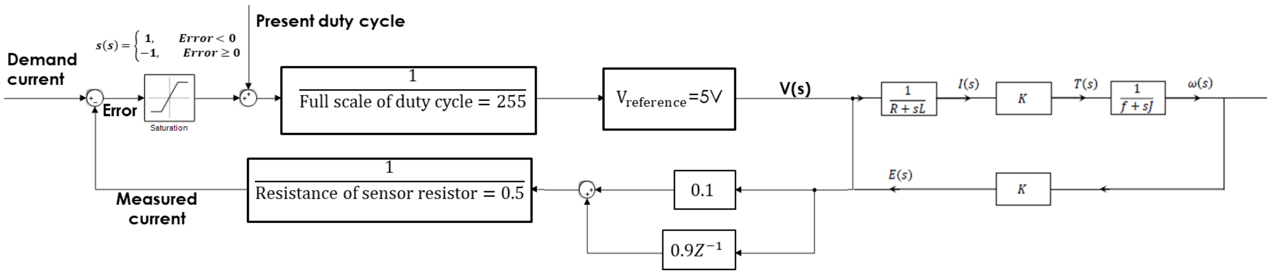

Present duty cycle
Demand current
$s(s) = \begin{cases} 1, & Error < 0 \\ -1, & Error \geq 0 \end{cases}$
Error
Saturation
$\frac{1}{\text{Full scale of duty cycle} = 255}$
$V_{reference}$=5V
V(s)
$\frac{1}{R+sL}$
$I(s)$
K
$T(s)$
$\frac{1}{f+sJ}$
$\omega(s)$
$E(s)$
K
0.1
$0.9Z^{-1}$
$\frac{1}{\text{Resistance of sensor resistor} = 0.5}$
Measured current

Figure.53 Block diagram for complete speed control

An incremental rotary encoder is needed to sense the angular speed. It converts the angular motion of the shaft of the motor to an analogue or digital signal which can be easily used for calculations. However, adding feedback control results in adding loops within both the ADC interrupt service routine and the main loop which means that it will slow down the speed of sampling, calculation, and judgment. The robot may get off track due to this. Besides, as the sampling frequency of ADC is 50kHz, the robot will almost keep adjusting the direction continuously. So, there's little chance for it to move at a constant speed. Furthermore, the magnetic poles inside the encoder may affect the sensing of magnetic field and cause error in position judgment. Therefore, after testing carefully, it's somewhat overkill, even counterproductive, to add speed control in this case.

b) Direction control

As mentioned in the tracking strategy section, the robot judges position by comparing the voltage sensed by the left sensor with the right one. There're simply three results of the comparison: the left one is bigger, the right one is bigger, or they are the same. If the voltage sensed by the left sensor is bigger than the right, it means that the robot is on the right side of the tracking wire. In this case, the robot should turn left. Similar analysis can be done for right turn, which is illustrated in the code shown in Figure.54.

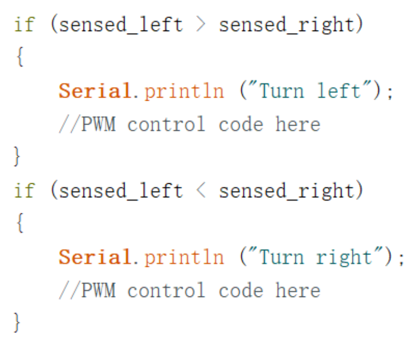

```
if (sensed_left > sensed_right)
{
    Serial.println ("Turn left");
    //PWM control code here
}
if (sensed_left < sensed_right)
{
    Serial.println ("Turn right");
    //PWM control code here
}
```

Figure.54 code for direction judgment

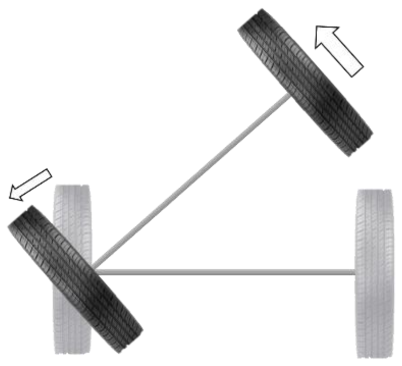


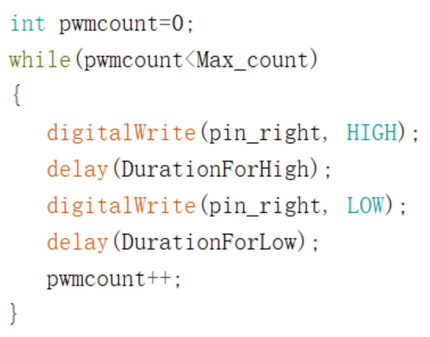
Figure.55 schematic diagram of turning

Ideally, if the sensed voltages are the same, the car should go straight. However, in practice, the tracking signal is not as stable as theoretically and the two sensors are not identical (after measurement, there is approximately a 1.5-fold nonlinear difference between them). So, the robot will constantly adjust its direction and move forward during the turn.

The original idea of how to execute a steering command is to move one of the wheels forward and the other backward. If so, the robot will rotate with the center of two wheels as the rotation axis, which means it will almost stay in place during the turn. But, as mentioned earlier, the robot will constantly adjust its direction. If it stays in place while turning, the robot could hardly reach the destination. Therefore, we decided to keep one of the wheels still while make the other one spin forward so that the robot could move forward during the turn. In order to implement this idea, we choose the wheels with lower friction coefficient and light weight to reduce friction. Because, while steering, the immobile wheel will have a slight turn with the rotating one, as illustrated in Figure.55. If the friction is too large, it will be difficult to turn and will thus hinder the whole vehicle's steering.

PWM signal is used to control the movement of wheels. Arduino gets a function called ‘analogWrite’ which can automatically write an PWM wave to the pin. However, ‘analogWrite’ function requires default reference voltage. Because we’ve already changed the reference voltage to meet ADC’s needs manually, the reference voltage is no longer the default. Since there is only one resistor in the Arduino to determine the reference voltage, we cannot provide them with two different reference voltages. To solve this problem, we chose to set up PWM signal manually by setting ‘digitalWrite’ high and low alternatively.

As shown in Figure.56, pin_right is connected to H-bridge to drive the right wheel. When the ‘while’ loop is being executed, the right wheel will be spining forward and the robot will be turing left. When the ‘while’ loop ends, the robot will stop and wait for a new turn command to come. This can aviod turning too much and tracking the wire more accurately. By increasing the ratio of ‘DurationForHigh’ to ‘DurationForLow’, we can increase the duty cycle of PWM to speed up the robot. By decreasing the sum of them, we can increase the frequency of PWM to get a more steady output. By changing the value of ‘Max_count’, we can change the timing of the PWM signal output to pin_right, thus changing the time for turning to achieve the balance between precision and elegance.



```
int pwmcount=0;
while(pwmcount<Max_count)
{
   digitalWrite(pin_right, HIGH);
   delay(DurationForHigh);
   digitalWrite(pin_right, LOW);
   delay(DurationForLow);
   pwmcount++;
}
```

Figure.56 code for PWM control

At this point, the robot already can track the wire quite elegantly. But there’s still room for improvement. In the above code, the duty cycle is predetermined and cannot be adjusted while the robot is working. Because we judge the position by comparison, the difference represents the degree of deviation from the center. The greater the deviation, the larger the required angle of turn, and the longer the turn takes and the bigger the ‘Max_count’ value. Therefore, through testing, we can derive the relationship between the difference of sensed voltage and the

‘Max_count’ and write ‘Max_count’ as a function of difference so that the robot will move more smoothly.

Please see the demonstration video in the attached website[18]

VII. Conclusions

Through the Integrated Mechatronic Project, we finally achieved our goal to design and program a robot who is able to follow the signal wire. When we review last two terms of learning, the relationship between circuit current and the magnetic field was mentioned in both EE2C transformer part and EE2B electrostatics and magnetostatics part. The signal amplification was taught in the open amplifier section of EE2B, while the filtering method was taught in EE2D lecture. Also, we learnt the signal processing method with Fourier Transform and closed-loop control in EE2C2. We gain the skill to design an embedded microprocessor system and write proper code for it from the EE2A module and experiment. And the knowledge about motor and rectifiers and converters were taught in EE2C. We apply all these above knowledges to build the sensor system, the signal processing system, the decision-making system and the power supply system of the robot and realize that there will be a gap between the theory and practice until you truly tried in the real world. Due to the limitation of our skill, time and funding, many great ideas were compromised and there is still huge space to improve to make the robot more accurate and efficient both in speed and power supply. After the whole project, we will pick the signal sensing part the most difficult part, because the current of our self-made tracking signal is limited and all we got at the beginning is the tiny signal with huge noise and there was no hint of using op-amp and capacitor to deal with the signal with hardware. The second difficulty is the Arduino board, we cannot deny there are many advantages of using an Arduino, but when the simplified software coding reference cannot meet our need, it is truly difficult to figure out how the hardware operating and how to program on hardware level by reading the datasheet of the ATmega328 chip in short time. After all, we learnt to work with each other as a team, be kind to each other and take each teammate’s advantage to succeed. The whole process will be our treasure and the experience will help us in our further learning or researching.

18 See Appendices a) 1

Appendices

a) Evidence of tangible deliverables

[1]https://bham-my.sharepoint.com/:v:/g/personal/wxm716_student_bham_ac_uk/Ec2ZXW-SNn1PrVeV3wjx17oBcYJuAIiMmm4sp95yiawupQ

[2] https://bham-my.sharepoint.com/:v:/g/personal/wxm716_student_bham_ac_uk/ERzbAeJG87dLoezCoyA5_JQBYcfjrR9gluA0zpbaYWLkmw?e=MNVohg

[3]

```
Local1_real=round(64*cos(2*pi*1000*t) );
Local1_imag=round(- 64*sin(2*pi*1000*t)); Local2_real=round(64*cos(2*pi*2000*t) );
Local2_imag=round(- 64*sin(2*pi*2000*t));
% multiplier Multi_1coscos=Local1_real.*X; Multi_1cossin=Local1_imag.*X;
Multi_2coscos=Local2_real.*X; Multi_2cossin=Local2_imag.*X;
% moving average filter Filter_Multi_1coscos=0;%initialize Filter_Multi_1cossin=0;
Filter_Multi_2coscos=0; Filter_Multi_2cossin=0;
%sample every 32 cycle for a=1:1:96 Filter_Multi_1coscos(a)=
(sum(Multi_1coscos(a:a+31))); Filter_Multi_1cossin(a)=
(sum(Multi_1cossin(a:a+31))); Filter_Multi_2coscos(a)=
(sum(Multi_2coscos(a:a+31))); Filter_Multi_2cossin(a)=
(sum(Multi_2cossin(a:a+31)));
end
% accumulation cos_1k = sum(Filter_Multi_1coscos(1:96))/96/65 536/2;
sin_1k = sum(Filter_Multi_1cossin(1:96))/96/65 536/2;
cos_2k = sum(Filter_Multi_2coscos(1:96))/96/65 536/2;
sin_2k = sum(Filter_Multi_2cossin(1:96))/96/65 536/2;
```

b) Mathematic derivation

1. $V_{omin} = \frac{3V_{ll}}{\pi} = \frac{3*15}{\pi} = 14.32\ \text{V}$ $\qquad$ $V_{omax} = \frac{3V_{ll}}{\pi} = \frac{3*40}{\pi} = 38.20\ \text{V}$

2. Derivation of transfer function:

	Time domain	S domain
Electrical equation	$V(t) = Ri(t) + L\frac{di\ (t)}{dt} + e(t)$	$V(s) = Ri(s) + sLi(s) + E(s)$ ①
Mechanical equation	$\tau(t) = f\omega(t) + J\frac{d\omega\ (t)}{dt}$	$\tau(s) = f(t)\Omega(s) + SJ\Omega(s)$ ③
Electrical power = mechanical power $ei = \tau\omega$ Define: $K = \frac{e}{\omega} = \frac{\tau}{i}$		

	$\tau = Ki$ $e = K\ \omega$ ②	$\tau(s) = K\ I(s)$ $E(s) = K\ \omega(s)$ ④

Block diagram of DC motor:

From ①②③④ => $I(s) = \frac{1}{R+sL}\left(V(s) - E(s)\right)$ $\omega(s) = \frac{1}{f+sJ} T(s)$

According to these, we can draw the block diagram as shown in Figure.20

The transfer function $\Psi(\mathrm{s}) = \frac{\omega(s)}{V(s)} = \frac{\frac{1}{R+sL}*K*\frac{1}{f+sJ}}{1+K*\frac{1}{R+sL}*K*\frac{1}{f+sJ}} = \frac{K}{(R+sL)(f+sJ)+K^2} = \frac{K}{LJs^2+(RJ+Lf)s+Rf+K^2}$

Assuming armature inductance L is 0.

$$\Psi(\mathrm{s}) = \frac{\omega(s)}{V(s)} = \frac{K}{RJs + Rf + K^2}$$

Let the denominator $RJs + Rf + K^2 = 0$ gives the pole $\mathrm{p} = -\frac{RF+K^2}{JR}$

3. $B = \frac{\mu_0 I}{2\pi r}, \Phi = \mathrm{BS}, v = -N\frac{d\Phi}{dt}$

$$v = -N\frac{d\Phi}{dt} = -N\frac{d\mathrm{BS}}{dt} = -\frac{\mu_0 SN}{2\pi r}\frac{dI}{dt}$$

4. $\frac{V_o - V_-}{47000} = \frac{V_o - V_i}{47000} = \frac{V_-}{1100} = \frac{V_i}{1100} \Rightarrow \frac{V_o}{Vin} = \frac{481}{11} = 44$

c) Code

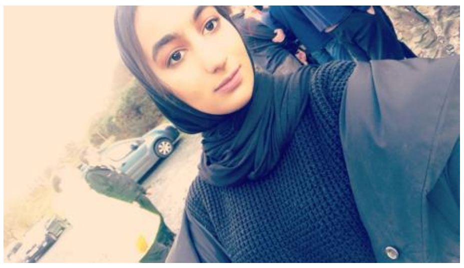
```
/*
Author: IMP Group B
Date complete: 26/04/2018
Filename: Current-tracking robot
Target device: Arduino Uno Rev3
Program function: to control the robot to track a
current-carrying wire
*/

const byte pin_left = 6;  // Pin6 controls the left
wheel
const byte pin_right = 9;  // Pin9 controls the right
wheel
const byte adc_left = 0;  // A0, connected to the left
coil
const byte adc_right = 1;  // A1, connected to the
right coil
const int MAX_RESULTS = 54;// buffer size
int k=0; //ensure ADC interrupt routine is off while
data gets processed
int results1 [MAX_RESULTS+1];// circular buffer for
data collected by left coil
int results2 [MAX_RESULTS+1];// circular buffer for
data collected by right coil
int filter1 [MAX_RESULTS-4];// filtered data of
result1
int filter2 [MAX_RESULTS-4];// filtered data of
result2
int resultNumber=0;// counter for ADC sampling

/* Analogue to digital converter complete interrupt
service routine */
ISR (ADC_vect)
{
    if(resultNumber < 55) results1 [resultNumber] =
analogRead (adc_left);
    // sampling channel 1,read and save signal sensed
by the left coil
    if(resultNumber >= 55)results2 [resultNumber-55]
= analogRead (adc_right);
    // shift to sampling channel 2,read and save
signal sensed by the right coil
    resultNumber++;
    if (resultNumber >= 2*MAX_RESULTS+2) ADCSRA = 0;
     // after finishing sampling, turn off ADC and
wait for data getting processed
  }
EMPTY_INTERRUPT (TIMER1_COMPB_vect);
// the interrupt of timer1 is empty as it is only used
to trigger ADC
 void setup ()
{
  Serial.begin (9600); // set baud rate to be 9600
for the interface between arduino and PC
  pinMode(5, OUTPUT); //connected to H-bridge for
backward rotation of the left wheel
  pinMode(6, OUTPUT); //connected to H-bridge for
forward rotation of the left wheel
  pinMode(9, OUTPUT); //connected to H-bridge for
forward rotation of the right wheel
  pinMode(10, OUTPUT); //connected to H-bridge for
backward rotation of the right wheel
  /* reset Timer1 */
  TCCR1B = 0;
  TCNT1 = 0;
  /* setup Timer1 */
  TCCR1B = bit (CS11) | bit (WGM12); // set prescaler
of CTC to be 8
  TIMSK1 = bit (OCIE1B); //interrupt enabled when the
counter matches B
  OCR1B = 39; //trigger the Timer1 interrupt when
count to 39//(calculated prescaler and counter for
triggering in 50kHz)
}
void loop ()
{
  if(k==1) //turn on ADC once in each decision making
loop
   {
    k=0;
    resultNumber=0;//reset index of ADC buffer
    /* reset ADC */
    ADCSRA = 0;
    ADCSRB = 0;
    /* setup ADC */
    ADCSRA =  bit (ADEN) | bit (ADIE) | bit (ADIF);
    // turn ADC on | enable ADC interrupt | wait and
turn interrupt off after the ADC service complete
    ADCSRA |= bit (ADPS2);
    // Prescaler of 16, ADC clock 1000kHz to ensure
the ADC reading is fast enough to complete in a 50kHz
Timer loop
    ADCSRB = bit (ADTS0) | bit (ADTS2);
    // Timer/Counter1 Compare Match B, every time when
timer  count  to  OCR1B=39(50kHz),  trigger  the  ADC
interrupt
    ADCSRA |= bit (ADATE);
    //  turn  on  automatic  triggering  -  sampling
frequency is 50 kHz
  }
  if (resultNumber == 2*MAX_RESULTS+2) //if buffers
are filled
  {
    /*pre-process data--subtract data by the minimum
value to prevent the processed data result from being
too large, exceeding the value range*/
    int result1_min=results1[0];
    int result2_min=results2[0];
    for  (int  i=1;i<MAX_RESULTS+1;i++)//find  minimum
value
    {
      if(results1[i]<result1_min)
result1_min=results1[i];
      if(results2[i]<result2_min)
result2_min=results2[i];
    }
    for (int i=1;i<MAX_RESULTS+1;i++) //subtract the
minimum value
    {
      results1[i]=results1[i]-result1_min;
      results2[i]=results2[i]-result2_min;
    }
    /* apply moving average filter */
    for (int i=0;i<50;i++)// taking average over every
5 samples
    {
      filter1[i]  =  results1[i+1]  +  results1[i+2]  +
results1[i+3] + results1[i+4] + results1[i+5];
      filter2[i]  =  results2[i+1]  +  results2[i+2]  +
results2[i+3] + results2[i+4] + results2[i+5];
    }
    /* find the maximum magnitude of data */
    int filter1_min=filter1[0];
    int filter1_max=filter1[0];
    int filter2_min=filter2[0];
    int filter2_max=filter2[0];
    for (int i=0;i<50;i++)//find the max and min value
    {
      if(filter1[i]<filter1_min)
filter1_min=filter1[i];
      if(filter1[i]>filter1_max)
filter1_max=filter1[i];
      if(filter2[i]<filter2_min)
filter2_min=filter2[i];
      if(filter2[i]>filter2_max)
filter2_max=filter2[i];
    }
    int magnitude_left=filter1_max-filter1_min;
    int magnitude_right=filter2_max-filter2_min;
    /* because the two sensor is not identical, and
the difference is non-linear, the filted data needs
to be segmented weighted */
    float weighted_left;
    float weighted_right;
    if(magnitude_left>400)       weighted_left       =
1.5*magnitude_left;
    if(magnitude_left>900)       weighted_left       =
1.5*magnitude_left;
    if(magnitude_right>400)      weighted_right      =
magnitude_right;
    if(magnitude_right>900)      weighted_right      =
magnitude_right;
    Serial.println (weighted_left);
    Serial.println (weighted_right);
    /* sidejudgement*/
    if (weighted_left >=450 || weighted_right >= 450)
    {
      if  (weighted_left  >  weighted_right)//  if  the
left voltage is bigger than the right, turn left
      {
        Serial.println ("Turn left");
        int pwmcount=0;
        while(pwmcount<15)
        {
          /* left wheel keep still and right wheel move forward */
          digitalWrite(pin_right, HIGH); //set PWM at 500Hz with duty cycle=50%
          delay(1);
          digitalWrite(pin_right, LOW);
          delay(1);
          pwmcount++;
        }
      }
      if (weighted_left < weighted_right)// if the left voltage is smaller than the right, turn right
      {
        Serial.println ("Turn right");
        int pwmcount=0;
        while(pwmcount<15)
        {
          /* right wheel keep still and left wheel move forward */
          digitalWrite(pin_left, HIGH);
          delay(1);
          digitalWrite(pin_left, LOW);
          delay(1);
          pwmcount++;
        }
      }
      delay(1);//delay between loops to ensure ADC be ready again
      k=1;//allow ADC to be turn on again after all decisions done
    }
  }
}
```

d) Personal Statement of Contribution

Lamis Alsuwaidi:

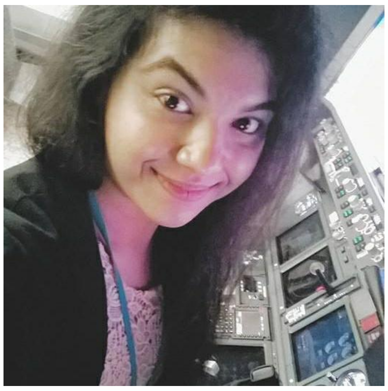


- Contributed towards the testing of the motors, the H-bridge, the three phase rectifier as well as an externally adjusted PWM signal applied to the H-bridge to change the speed of the motors with Hang.
- Cooperated towards the initial assembly of the Robot.
- Took part in the testing of the full assembled initial robot hardware (in which the Arduino was not connected) with a 9V battery pack with Hang.
- Contributed towards the building and soldering of the AC-DC converter circuit coupled with a voltage regulator and capacitors with Hang.
- Contributed to the hardware aspect of the report by writing the signal processing section.
- Documented and kept track of the hardware testing procedures in details.

Arushi Madan:

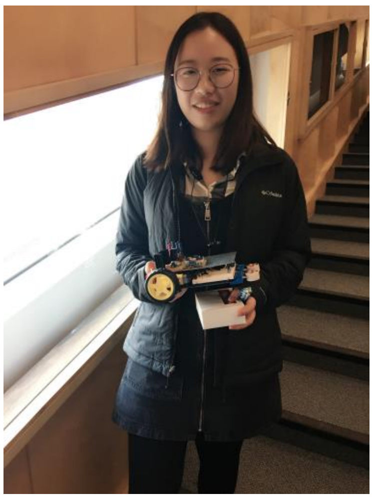
- Made Matlab graphs such as of frequency vs resistance etc. to make analysis easier.
- Along with other members, discussed, and worked upon tracking strategy and documented it.
- Helped in editing bench inspection presentations and speech.
- Assisted others in hardware building, along with in smoothening of signals.
- Helped others, in making report of various other sections.
- Keeping a detailed note, along with Lamis on the hardware and software developments of the robot, to make documenting easier.

Anika Rahmam:



- Keeping a detailed log book for the initial laboratory sessions
- Getting to know all parts of the Arduino board with the aid of a schematic diagram
- Learnt how to use power width modulation (PWM) and analogue to digital converter (ADC) port
- Developed a document to outline the PWM functions such as: analogWrite(), digitalWrite(), map()
- Wrote a reference code for the motors to go forward and backward at a given rate and for it to brake using input potentiometer data
- Contributed to the speed and direction control section of the report

Wenjun Ma:



During term time, I did:

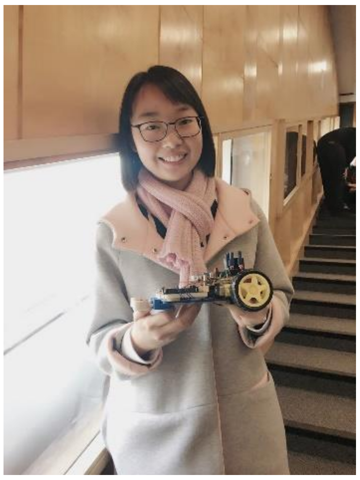
- Measure parameters of the DC motor
- help design the rectifier and regulator circuits and soldering.
- Be responsible for the progression on lab (mainly the power processing unit) in the mid semester bench inspection
- Help plan the shopping list with Wen and Hang.

- Do the risk analysis.

During Easter break, I did:

- Participate in designing the overall structure of the robot and tracking strategy.
- Soldering and layout design of the pre-regulator circuit
- Participate in hardware assembly
- Help with testing and fault finding.
- Be responsible for the power processing unit, system architecture, hardware signal processing of the 2nd bench inspection
- Responsible for the report (main writer): Including System Architecture, Power Conversion and Supply conditioning, Electromagnetic Position Sensors (include signal processing) and final formatting. (With the help of Wen writing the control bit and technical support from Hang).

Wen Wu:



During term time, I did:

- Measure parameters of the DC motor, including armature resistance, inductance, and motor constant.
- Design the rectifier and regulator circuits and help with soldering.
- Be responsible for the added value part of the mid semester bench inspection.
- Help plan the shopping list with Wenjun and Hang.
- Do the risk analysis.

During Easter break, I did:

- Participate in designing the overall structure of the robot and tracking strategy.
- Contribute to software coding.
- Help with hardware assembly and soldering.
- Help with testing and fault finding.
- Be responsible for the software signal processing and control system part of the 2nd bench inspection
- Write Closed-loop DC Motor Armature Current Control and Higher-Level Closed-loop Velocity Control part of the final report (with technical help from Hang).
- Add comments to the final code.

Hang Xu (Group Leader):

During term time, I did:

- Measuring parameters of DC motor with whole group
- Design and testing rectifier and regulator circuit with whole group
- Responsible for strategy part of the mid semester bench inspection
- Main planner of the shopping list

During Easter break, I did:

- Responsible for hardware: Theoretical calculation and testing, Hardware main designer, Sensing circuit designer, 3D print CAD, Robot construct with Wenjun, Wen, Lamis
- Responsible for software: Theoretical calculation and testing, Filtering and directing strategy maker, Software main programmer
- Software coding with the help of Wen (piece of coding), Anika (online reference searching)
- Group management: Timeline management, Meeting arrangement
- Technical & theoretical explanation for group mates
- Final assembling and testing with Wenjun
- Report technical & theoretical support
- Report conclusion

Reference:

[1] https://uk.rs-online.com/web/p/bridge-rectifiers/9179023/

[2] https://www.robotshop.com/en/wheel-dc-motor-kit.html

[3]https://www.amazon.co.uk/Converter-Adjustable-Stabilizer-Transformer-Motorcycle/dp/B00IJ353V6/ref=sr_1_5?ie=UTF8&qid=1524062392&sr=8-5&keywords=buck+dc+volt+converter

[4] http://www.ti.com/lit/ds/symlink/lm7800.pdf

[5]https://www.arrow.com/en/products/lm317t/stmicroelectronics?utm_source=google&utm_campaign=g-ppc-uk-dsa-stmicroelectronics&utm_medium=cpc&utm_term=DYNAMIC+SEARCH+ADS&utm_currency=GBP&gclid=EAIaIQobChMI5sq-laXO2gIV9hXTCh1aPgXBEAAYASAAEgL6L_D_BwE&gclsrc=aw.ds&dclid=CI3cgpelztoCFQNEGwodArYD1w

[6] https://www.gammon.com.au/adc

[7] ATmega328 Datasheet

http://ww1.microchip.com/downloads/en/DeviceDoc/ATmega328_P%20AVR%20MCU%20with%20picoPower%20Technology%20Data%20Sheet%2040001984A.pdf