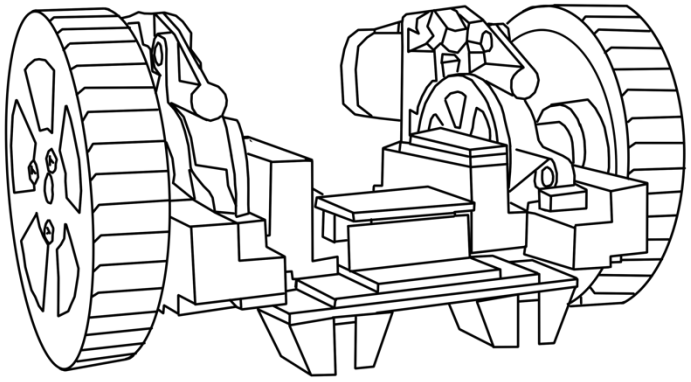
Integrated Mechatronic Project Final Report



Date 27/04/2019

Authors
Yuecheng Hong
Jingyi Chu
Yihao Wang
Jie Sun
Zhiyan Li
Ke Yang

Introduction

This Integrated Mechatronic Project (IMP) is aimed to improve our practical skills and help us bridge-the-gap between the underlying science and academic content of all of the second year modules. IMP requires us to design, build and document a wire-following autonomous vehicle. The tracking wire carries a peak current of between 10 mA and 1 A of a dual-tone (1KHz and 2KHz) mounted up to 30mm below the race track. The induced voltage across the sense coil will be the signal indicating the position of our vehicle.

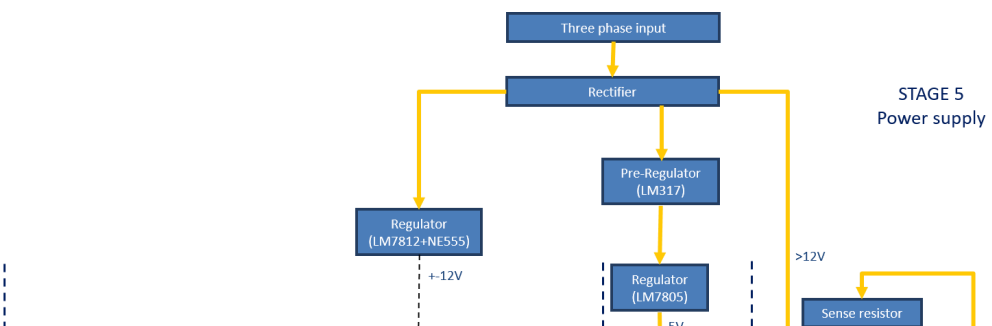
This project are divided into four main parts including wire-following sensor and navigation strategy (Yihao Wang and Jie Sun), motor controller (Zhiyan Li and Ke Yang), power supplies (Yuecheng Hong and Jingyi Chu), and Mechanical Implementation (all of the team members). LEGOs are chosen as the mechanical model of our vehicle due to its flexibility, so it is extremely convenient for us to modify the parameters during our design process. Specific structure and parameters are shown in the first part of this report.

In the power supplies part, the transferring circuit is needed to transfer 20V - 50V three phase AC voltage to 5V DC voltage supplying for the PIC source voltage and amplifier circuit bias voltage and +12V/-12V DC voltage for the amplifying source voltage. The diagram and corresponding analysis are shown in the second part of this report.

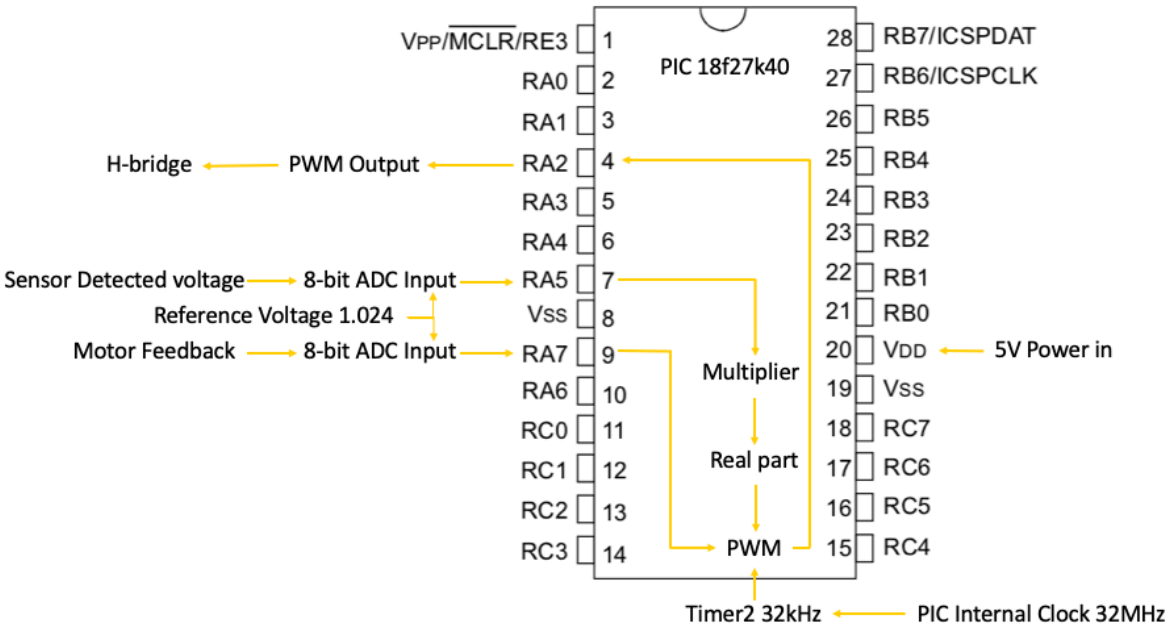
There are totally three wheels equipped with our vehicle, two of which are driving wheels controlled by motor directly and another one is a passive wheel in the front of vehicle deck. Two motors are connected to H bridge driver, where the enabling ports are connected to the Pulse Width Modulation output ports. The variance of PWM wave duty will change the velocity of motors. Detailed design process and corresponding control theory are listed in the third section of this report.

To build a stable intelligent vehicle tracking the hidden wire underground efficiently and effectively, two sense coils are used separately to control each of the wheels. The induced voltage across the coil is amplified and the amplifying signal will be sent to the Analogue to Digital input port of PIC chip. After I/Q demodulation and transversal filtering, a series of informative S values are calculated showing the position of vehicle. The value of the calculated S is positively correlated to the distance from sense coil to tracking wire. Through the comparison between S and threshold measured in advance, the velocity of wheels is adjusted to control vehicle's direction. Detailed algorithm and analysis are shown in the fourth part of this report.

System Architecture



Block Diagram of wire-following vehicle



PIC Implementation

Power Conversion and Supply Conditioning

A. Power supply

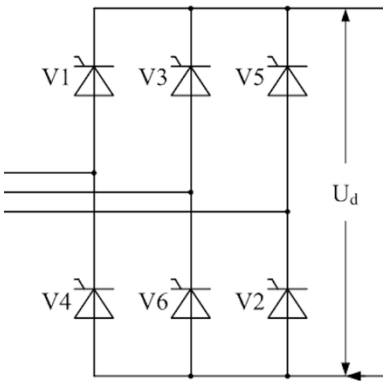
a) Overall Description

In power supply part, we need to transfer 20V ~ 50V three-phase AC voltage to +-12V DC voltage for the amplifier, 5V for the PIC18F27k40 and amplifier output bias and voltage less than 40V for the H-bridge Vss.

b) Theoretical Background

Firstly, we need to use a rectifier to transfer three-phase AC voltage to DC power we need to use the rectifier.

Figure 1.2.1 Three-phase rectifier



Six diodes will be used to get the DC power. V1, V3, V5 are used to get the positive value of the three-phase wave while V4, V6, V2 are for the negative value. The input and output voltage are shown in *Figure 1.2.2.*

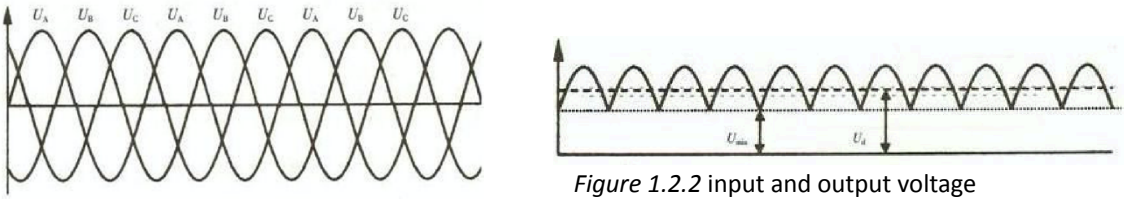
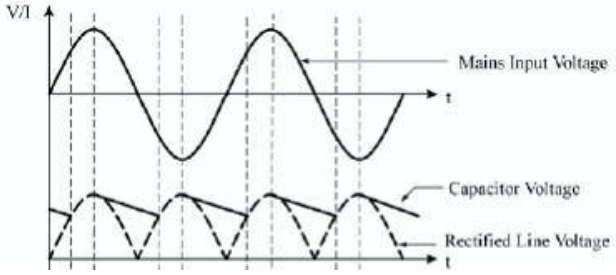


Figure 1.2.2 input and output voltage

To get a comparatively smooth output we need to add a capacitor parallel with the output.

Figure 1.2.3 output voltage after smoothing



As the maximum voltage tolerance of LM7805 and LM7812 is less than 50V, we need to decline the voltage first by using LM317 first, then using LM7805 to get 5V voltage and LM7812 and LM7912 to get +-12V voltage.

c) Practical Procedure

First, we designed the circuit providing +5 V voltage for PIC processor and voltage above 12 V for H-bridge. Three phase voltage first through the rectifier to convert it to DC voltage and then through the regulator modules to modify it to the voltages we want. H-bridge voltage was generated after LM317 and +5 V voltage was generated after LM317 and LM7805.

Second, two problems occurred after we finish the sensor part: we couldn't provide ±12 V voltages for the amplifier circuit. So, the second design was proposed by adding two models: LM7812 and NE555.

Third, we found a problem that the provide current for H-bridge after LM317 might be too small for the wheel to run. Therefore, we add a new wire directly after rectifier to provide the voltage for H-bridge.

Each stage was processed by simulation in Multisim first, test it on breadboard, and finally welded.

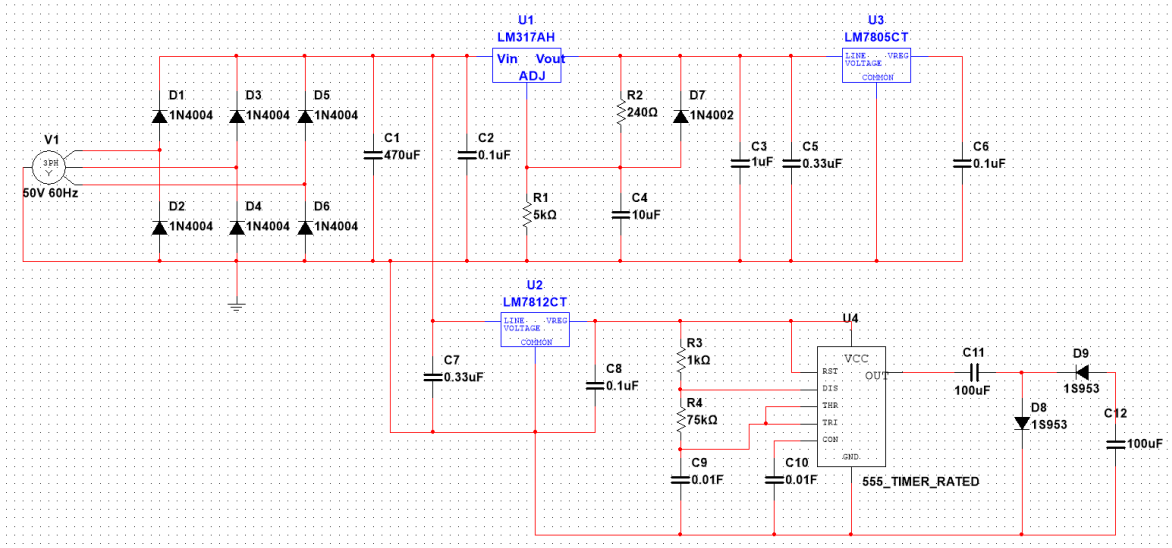


Figure 1.4 final simulation circuit

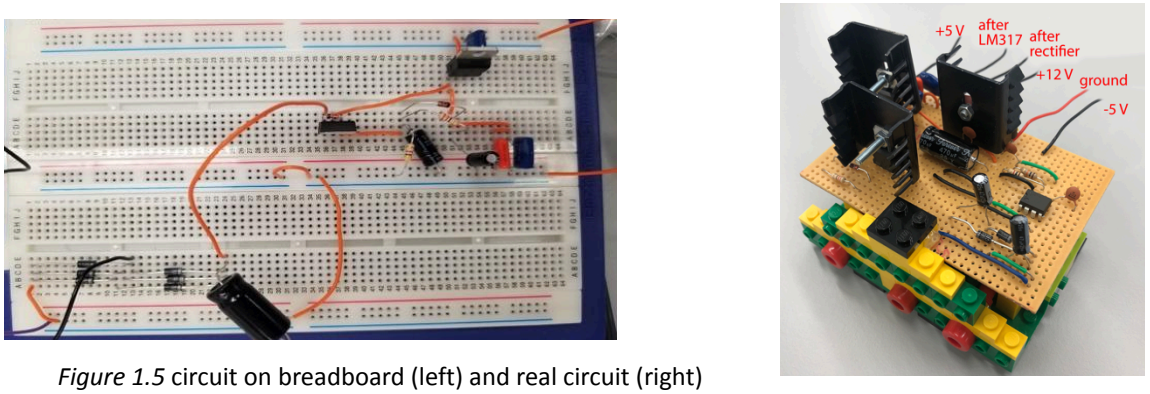


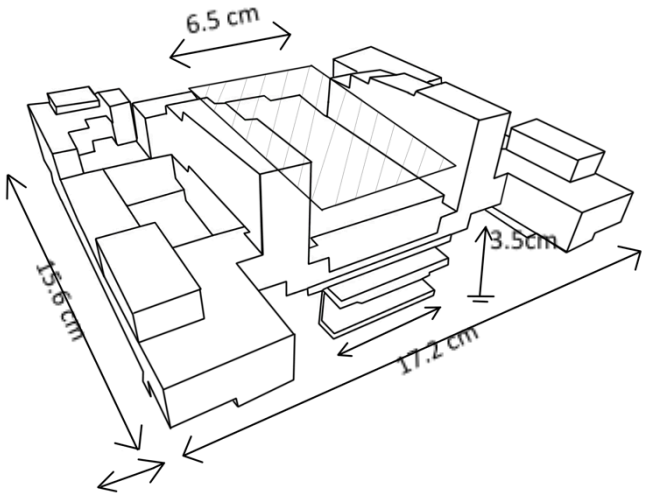
Figure 1.5 circuit on breadboard (left) and real circuit (right)

Then we structured our circuit on the breadboard to check whether it works well. Finally, we welded them to the circuit boards.

B. Mech structure

a) Overall Description

As the model needs to be flexible in case of the change and the placement of the circuits, we chose to use LEGO instead of 3D printer.



2.5 cm

Figure 1.6 cutaway view of wire-following vehicle

b) Practical Procedure

First, we made two parts of the structure used to stall the motors. Then we changed the gap between motors to fit the width of the race road and the broad. After that, we add omni-directional wheel on the back of our model in case of interfacing with the sense coils. Finally, we use toy glue to enhance some connection points in order to improve the strength.

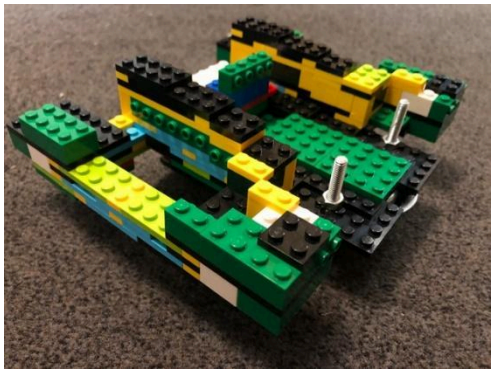


Figure 1.7 LEGO model of wire-following vehicle

Motor Control

A. Overall Description

The wire-following robot has two DC motors to drive two wheels moving forward. A circuit is built to take signals detected by two sense coils then control the speed of wheels' rotation. The different rotation speed can make the robot follow the invisible line closely by changing its direction. For example, if the detected magnetic filed shows the line is placed at the right side of the robot, the right wheel will slow down and the robot will turn right. Also, a close loop control of current and direction is done by processing data collected from sensor resistor.

B. Theoretical Background

a) Hardware

DC motor is widely used in variable speed drive applications because it's variable characteristics (ref1). In this experiment, we used the LM298, a dual H-bridge deriver which

can control two motors at up to 2A per motor as DC motor driver. The circuit diagram is shown in Figure 2-1.

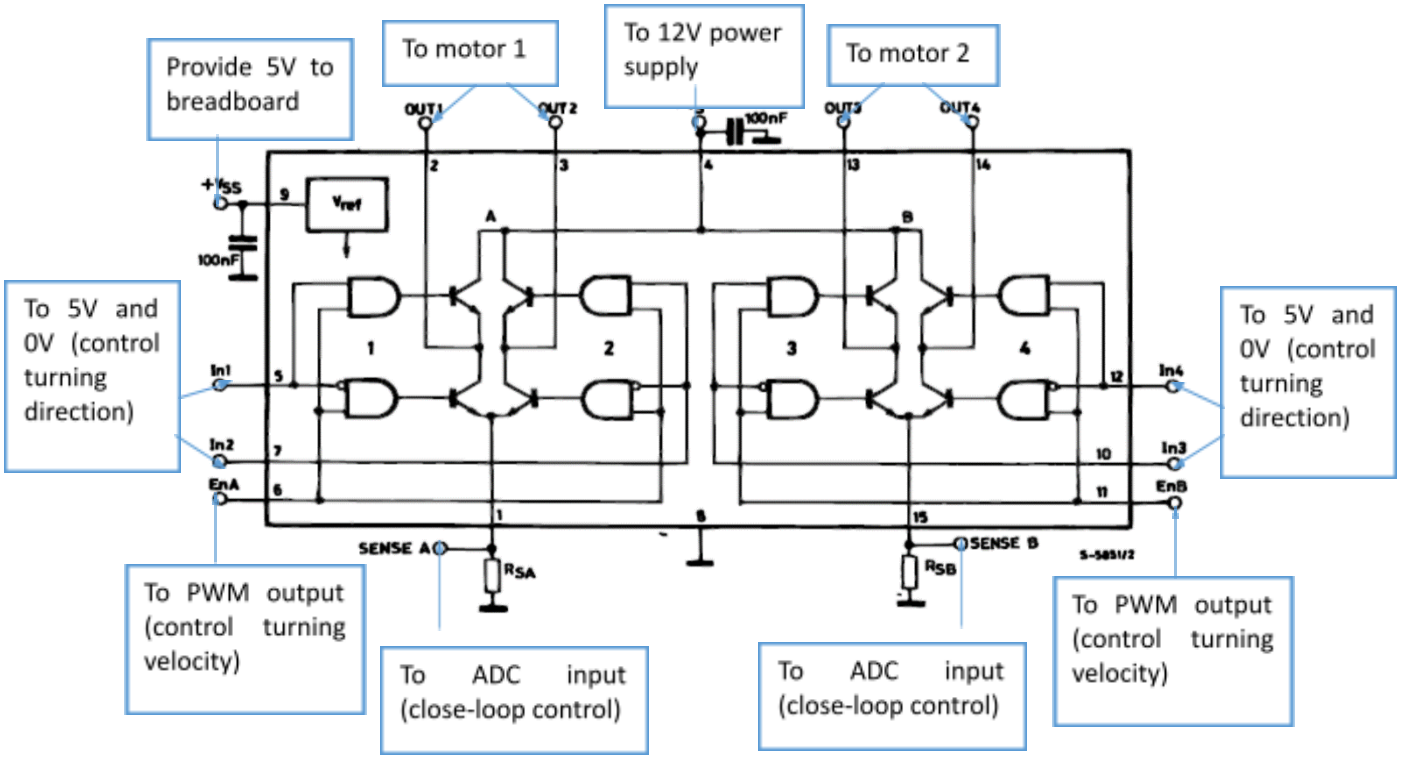


Figure 2.1 LM298 Dual H-bridge driver

Connect two motors to 'OUT1' , 'OUT2' and 'OUT3' ,OUT4 respectively. Apply the PWM signals created by PIC18F27K40 to 'EnA' and 'EnB'. The voltage level applied at 'In1' , 'In2' and 'In3' , 'In4' decides the rotation of each wheel. Sense resistors ' RSA' and 'RSB' are connected between the lower part of the H-bridge and ground to measure the current flows in the motor as an input of ADC port of PIC18F27K40. The function table of LM298 Dual H-bridge is as follows (X means irrelevant state):

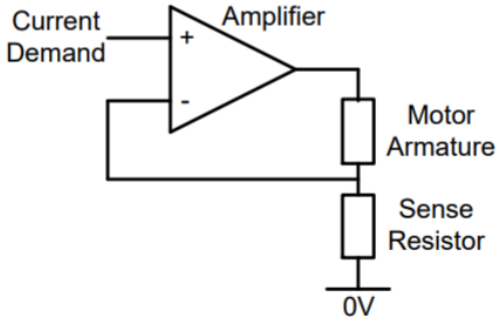
In1	In2	EnA	In3	In4	EnB	Motor1	Motor2
High	Low	High	High	Low	High	Clockwise	Clockwise
Low	High	High	Low	High	High	Anticlockwise	Anticlockwise
High	High	High	High	High	High	Brake	Brake
Low	Low	High	Low	Low	High	Brake	Brake
X	X	Low	X	X	Low	Stop	Stop

Table 2.1 Function table of LM298 Dual H-bridge

A high-power amplifier applies a voltage to the armature of a DC motor. To implement a closed -loop DC Motor Armature Current Control, measure the flowing current using a 0.5 ohm sense resistor. The potential difference appearing across the sense resistor will be compared to the appreciate demand signal and amplify the difference.

Figure 2.2 Current Control Loop Circuit

b) PIC Implementation



1. PWM

Instead of applying a constant voltage across a DC motor, the robot is controlled by PWM to switch the DC motor on and off. As a result, the DC motor is applied the average voltage proportional to the duty cycles of the PWM pulses.

A Capture Compare Module is combined with a PWM module. With configurations set for PIC18F27K40 and requirement of controlling PWM, Timer 2 provides a timing event at 32 KHz rate from a 32MHz master clock.

The period of PWM4 is defined as:

$$PWM\ Period = [(PR2) + 1]\times 4\times TOSC\times (TMR2\ Prescale\ value) \quad (2.1)$$

Where PR2 is the value of the preset resister and TOSC Is the period of internal oscillator (can be calculated by 1/FOSC).

Pulse width (duty cycle of PWM4) is defined as:

$$Pulse\ Width = (CCPRxH{:}\,CCPRxL\ resister\ pair)\times TOSC\times (TMR2\ Prescale\ Vale)$$

Where CCPRxH: CCPRxL resister pair is the value entered into the CCPR4 resister.

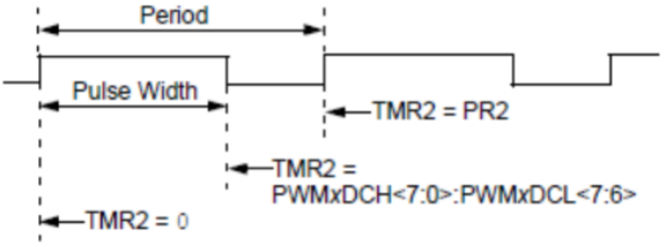


Figure 2.3 Pulse Width Modulator timing waveform

2. ADC

The Analog-to-digital (ADC) allows conversion of an analog input signal to a binary representation of signal. 18F27K40 uses analog inputs, which are multiplexed into a single sample and hold circuit. The output of the sample and hold is connected to the input of the converter. The converter generates a binary result via successive approximation and stores the conversion result into the ADC result register (ADRESH:ADRESL register pair).

In the experiment, it's necessary to connect the analogue signal detected on the sense resistor to the analogue-to-digital converter of the PIC18F27K40. It has a programmable reference voltage of 1.024V, 2.048V or 4.096V. The higher the reference voltage is, the greater the armature current that the software can handle without overloading, but the worse result.

3. Close-loop feedback control

Consider a simple model of a DC permanent magnet motor shown in Figure 2-4.

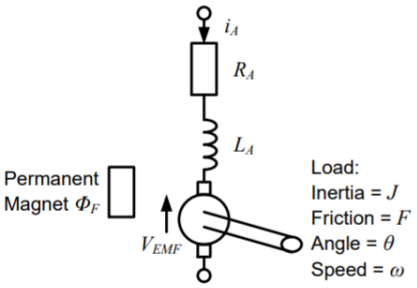


Figure 2.4 simple model of a DC permanent magnet motor

Spin the un-powered motor, it's able to supply a potential difference as a generator. Where

$$Vemf = K_b \omega(s) \tag{2.3}$$

As a motor, this voltage will be subtracted the applied armature voltage, $V_a(s)$

So, the armature current i_a(s) is

$$i_a(s) = \frac{V_a(s) - Vemf}{(R_A + L_A s)} \tag{2.4}$$

It shows that the armature current varies with time and the rotation speed. Since the armature will consume very high current when the motor is starting from stationary and small current when it's spinning under no-load conditions, it's crucial to implement the armature current control loop.

The torque T generated by the motor is defined as:

$$T(s) = K_A i_a(s) \tag{2.5}$$

The load torque for a rotating inertia J and Friction f is

$$T(s) = Js^2\theta(s) + fs\theta(s) \tag{2.6}$$

Overall, the transfer function of the motor and the load is

$$G(s) = \frac{\theta(s)}{V_a(s)} = \frac{K_A}{s(Js+f)(L_A s+R_A)+sK_A K_B} \tag{2.7}$$

Figure 2-5 shows the block-diagram of a simple DC permanent Magnet Motor.

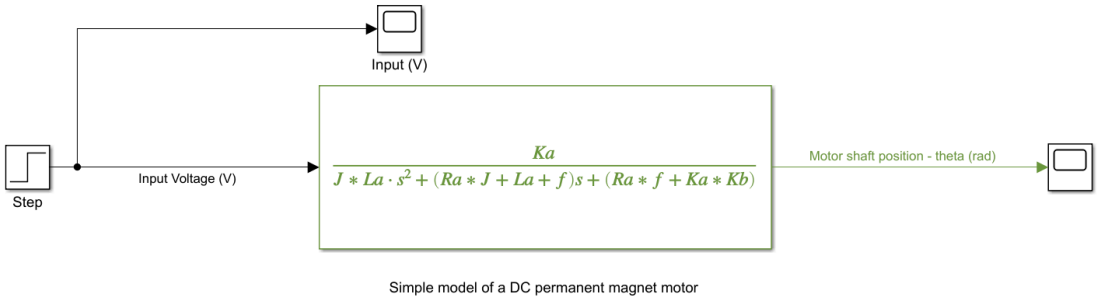


Figure 2.5 Block-diagram of a DC permanent magnet motor

A close loop DC motor Armature current control loop compares the measured current with the requirement and adjusts the drive voltage to stabilize the current then control the speed. The difference between two current values will be put into the PIC chip to change the duty cycle of the PWM pulses, which is equal to the voltage amplitude required to maintain the speed.

A proportional control loop can be mathematically expressed as

$$Pout = Kpe(t) + p0 \tag{2.8}$$

Figure 2-5 shows the block-diagram of a simple model of a close-loop control DC permanent magnet motor.

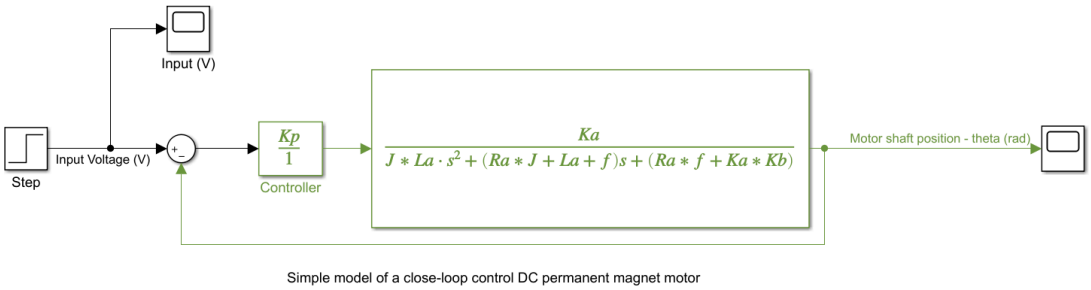


Figure 2.6 Simple model of a close-loop control DC permanent magnet motor

C. Practical Procedure

Firstly, plan A was proposed by controlling two wheel-motors using one microprocessor. To

achieve this target, PWM 3 and Timer 4 were added in the code with the former PWM 4 and Timer 2. The difference between PWM duty numbers were set to ± 200 when 'right' or 'left' was input in putty console window.

Second, connect two motors to the H-bridge for test. The wire connect way is shown in figure 2.1.

Third, installed those two wheels to the cabinet and power supply to modify the parameter more precise and test whether it work correctly after one-time power off. A default values of PWM duty was set previously to test 'turning left', 'turning right' and 'go straight' functions.

Fourth, considering two sensor processing time in one PIC, plan B was forwarded by using two PICs to control the direction detecting progress and control two wheel-motors separately.

D. Data and Discussion

a) Debugging process:

1. Motor didn't work when connected to the H-bridge and power supply. The bug was a common ground wire didn't connect between power and breadboard. The motor work with expectation after debugging.

2. One motor didn't work from beginning. We tested it by directly connecting it to the power supply and exchanging the side of H-bridge. The problem was one side of H-bridge was defunct. This problem solved after changing a new H-bridge.

3. In the third part of the test, we found the wheel couldn't work after we connect it to race power supply and all the node's voltages were right. The reason was we connected all 5V voltage through H-bridge. Theoretically there was no difference between connecting it from power supply or from H-bridge, but the practically it may cause microprocessor break down after switching off the power. This problem solved by connecting 5V from power supply regulator.

b) Data:

After testing and balancing, chose 2.048V as the reference voltage, therefore, the resulting transfer function will be:

$$\frac{0.5\ x\ 2^{10}}{2.048}\ integers\ per\ Amp$$

c) Discussion:

E. Conclusion

References :

Controlling Current.(2000).[online] Available at: http://irtfweb.ifa.hawaii.edu/~tcs3/tcs3/0405_Servo_review/onaka_docs/2000_articles_control.pdf [Accessed 26 Apr. 2019].

Electromagnetic Position Sensor

A. Overall Description

Designed vehicle is wire-following hence a sensor is needed to ensure that the vehicle is tracking along the buried wire. Basically the sensor is required to determine which side of the wire they are on and how far from the wire. This goal can be divided into three main parts: collecting induced signals, preliminary signal processing and main signal procession. Two sense coils can be used to collect induced signals. The preliminary processing includes filtering high-frequency noise and amplification of induced signals for facilitating further accurate signal processing. The most important and difficult part is to determine the vehicle position by collected data. This process is consisted of filtering dual-frequency signals by quadrature demodulator and transversal filters and calculations done by hardware.

B. Theoretical Background

a) Signal Processing Algorithm

First of all, it is essential to apply effective filters to select these two signals in the buried wire from experiment noises and reject as much as interference as possible. In this case, the quadrature demodulator is applied to demodulate input signals which is achieved by shifting the input frequency using a baseband signal and then using a low-pass filter. And quadrature demodulation is not limited to signals that were originally created through quadrature modulation.

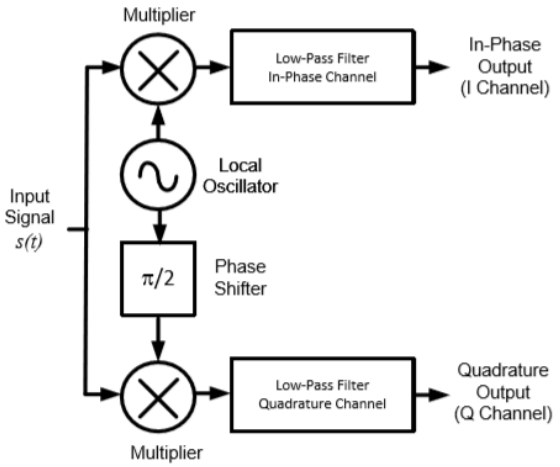


Figure 4.1 Quadrature Demodulator

As can be seen from figure1, the input signal is converted to the corresponding I/Q baseband signals. The input signal is connected to two multipliers which are local oscillator described as $\cos cos\ (\omega t)$ and a ninety degree phase-shifted version of the sinusoidal oscillator described as $-\ \sin sin\ (\omega t)$ respectively. And ω is the frequency of the input signal that is to be examined. The low-pass filters are needed because the quadrature multiplication applied to the received signal is no different from the multiplication employed in. Therefore the operations can be described as two equations:

$$I(t) = \int s(t) \cos cos\ (\omega t) dt \tag{4.1}$$

$$Q(t) = \int -\ s(t) dt \tag{4.2}$$

Thus this process is very conveniently described using complex numbers. The real component is the in-phase signal and the imaginary component carries the quadrature signal.

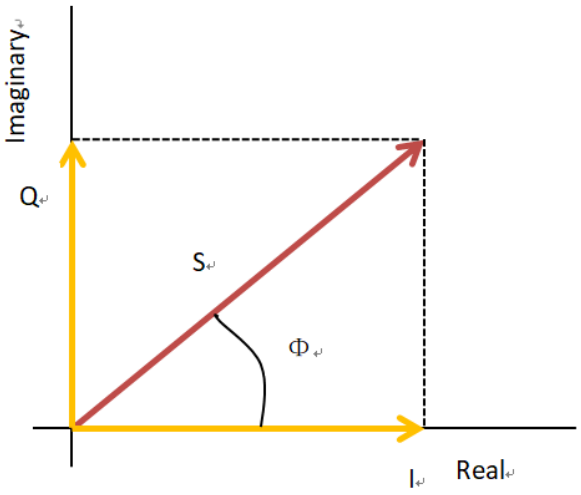


Figure 4.2 Q and I components

According to Euler's formula:

$$\cos cos\ (\omega t) - j \sin sin\ (\omega t) = e^{-j\omega t} \tag{4.3}$$

Then the two output signals could be combined into a single complex number:

$$S(\omega) = \int s(t) e^{-j\omega t}\, dt \tag{4.4}$$

Using a moving-average filter which is basically an integration process:

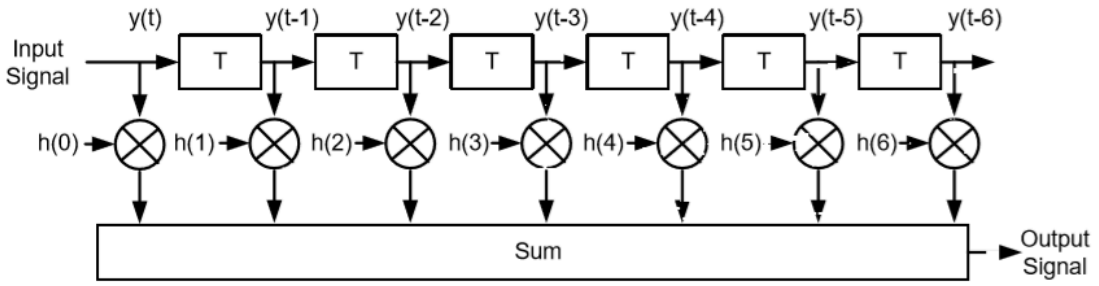


Figure 4.3 Transversal filter

As can be seen from figure3, $y(t)$ represents the input signal with time delay and $h(t)$ are the baseband signals used to multiply with input signals which will be that of a sampled 1 kHz and 2 kHz tone respectively. Basically the convolution operation can be considered as pattern matching.

Then the convolution operation in discrete form can be described as:

$$OutputSignal = \sum_{k=0}^{N-1} h(k) y(t - kT) \tag{4.5}$$

Thus the overall signal process can be described as:

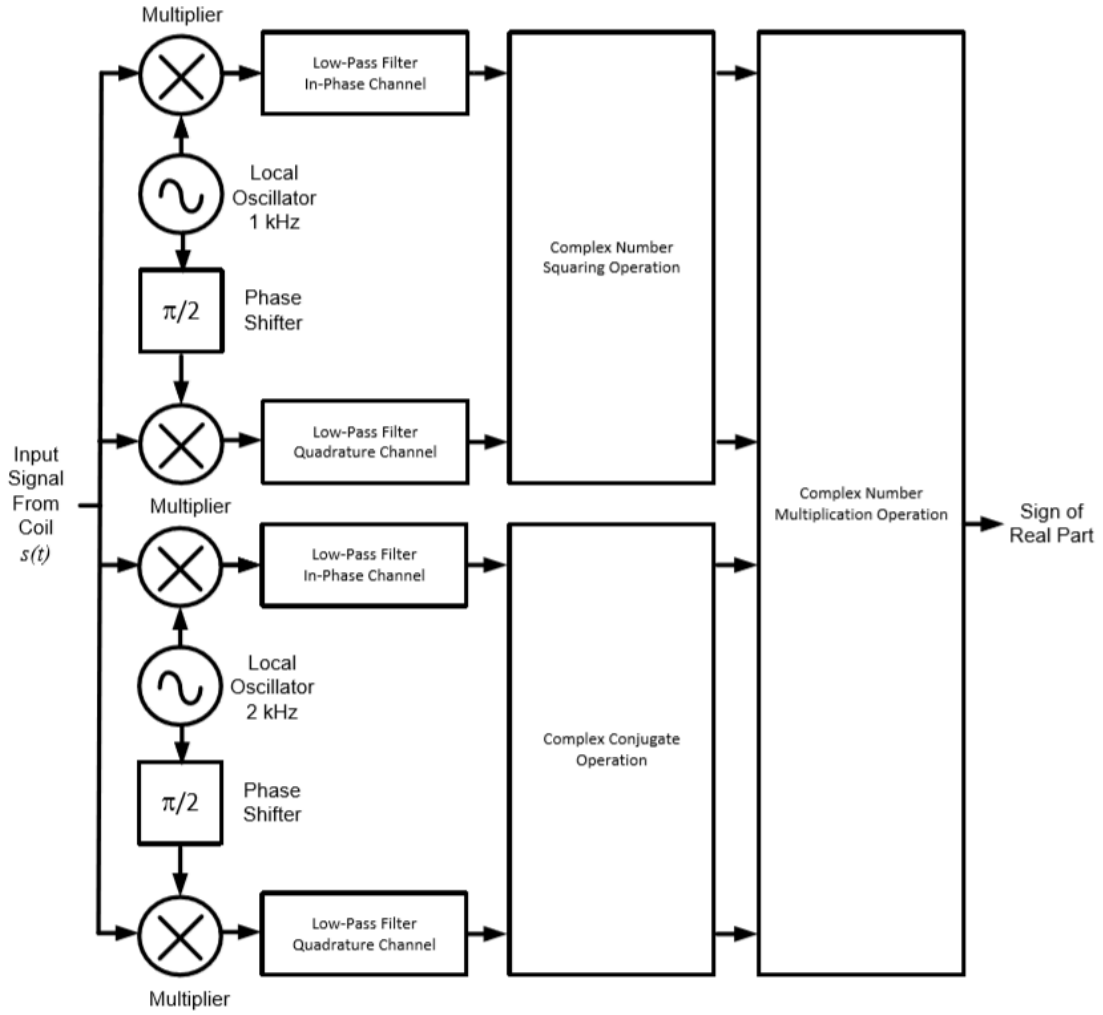


Figure 4.4 Signal processing flow diagram

Operation ‘Complex number squaring operation’ is to double the output frequency of the upper filter.

b) Hardware Amplifier Circuit

It is necessary to implement an amplifier circuit to amplifier signals captured by sense coils in case the signals directly collected by coils are too small to use. In this circuit, a SSM2019 differential amplifier chip is used.

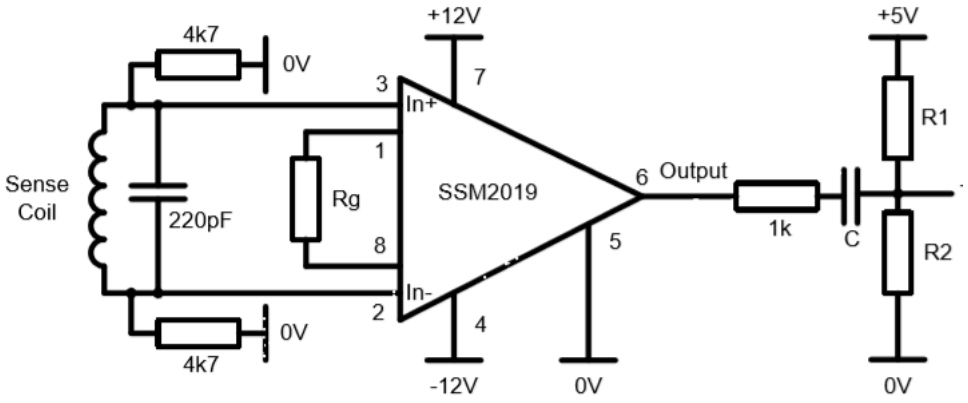


Figure 4.5 SSM2019 Differential amplifier circuit

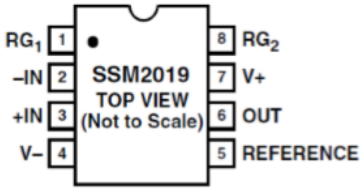


Figure 4.6 SSM2019

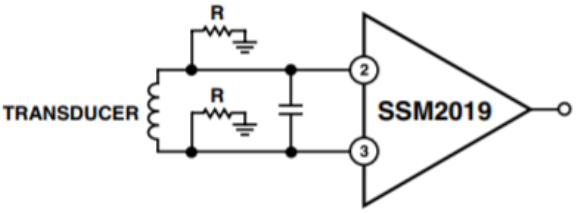


Figure 4.7 Interfacing transducers for high noise immunity

The experiment high-frequency noises need to be considered. Therefore, the 220pF capacitor in parallel with resistor R_g is to reduce high-frequency interference. Useful signals and noises that have not been filtered out are both amplified through this amplifier. Extra low-pass filter is required because of this. The resistor R_1 is used as pull-up resistor to protect PIC. The gain of the amplifier is set by resistor R_g :

$$Gain = \frac{10k\Omega}{R_g} + 1 \quad (4.6)$$

c) PIC Implementation

Signal processing is carried out in the hardware which is PIC 18F27K40 in this lab. Thus the ADC module of PIC is needed to sample induced signals before operation.

An 8 bit ADC module is used to collect the data from sense coil's voltage and set the reference voltage as 1.024V. The range of ADC results will be from 0 to 256 representing the value of input signal value.

C. Practical Procedure

a) Hardware circuit construction

Based on previous amplifier circuit template, the actual amplifier circuit:

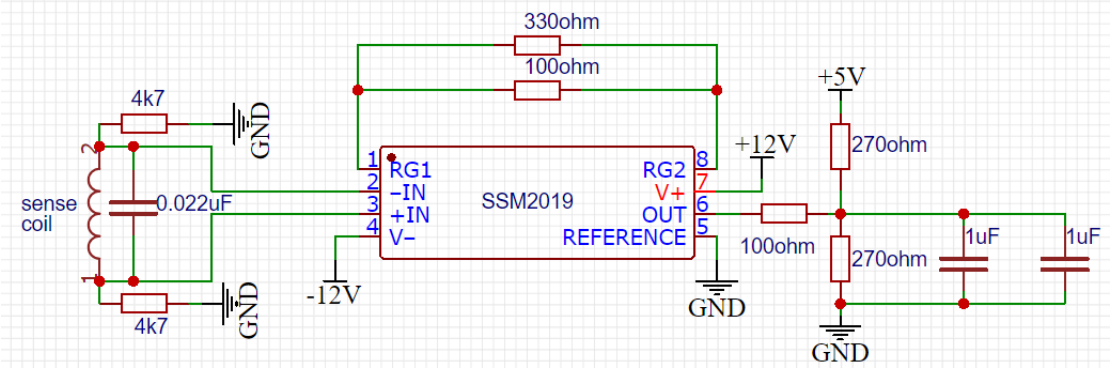


Figure 4.8 Theoretical Analog amplifier circuit

Compared with the previous one, the capacitor in series at pin6 is replaced by two in parallel capacitors. This substitution improves the filtering capability. The gain of this circuit is around 100, which is big enough for amplifying induced signals.

We test this amplifying circuit with a sinusoidal wave with the peak to peak value of 0.02V and frequency of 1 KHz. And the amplifying result is shown below:

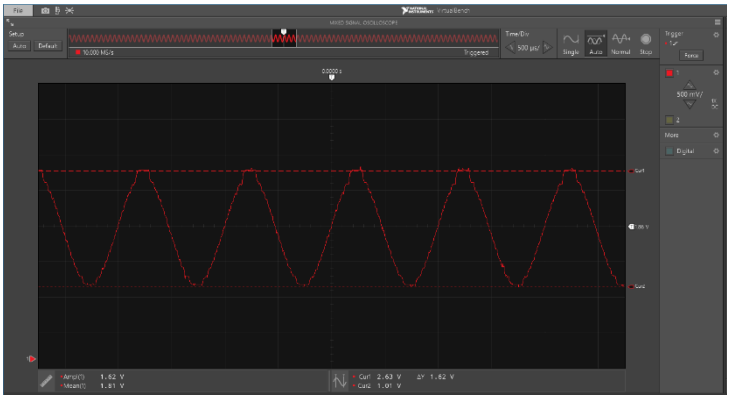


Figure 4.9 Amplifying signal after the actually building circuit

To ensure that this amplifying multiple is able to put the test signal in the correct range where the PIC chip can analyze it with enough correctness, we measured the inductive voltage of the censor near the tracking wire used in the final competition. The eventual value of our amplifying signal is around 1V which is reasonable for our algorithm.

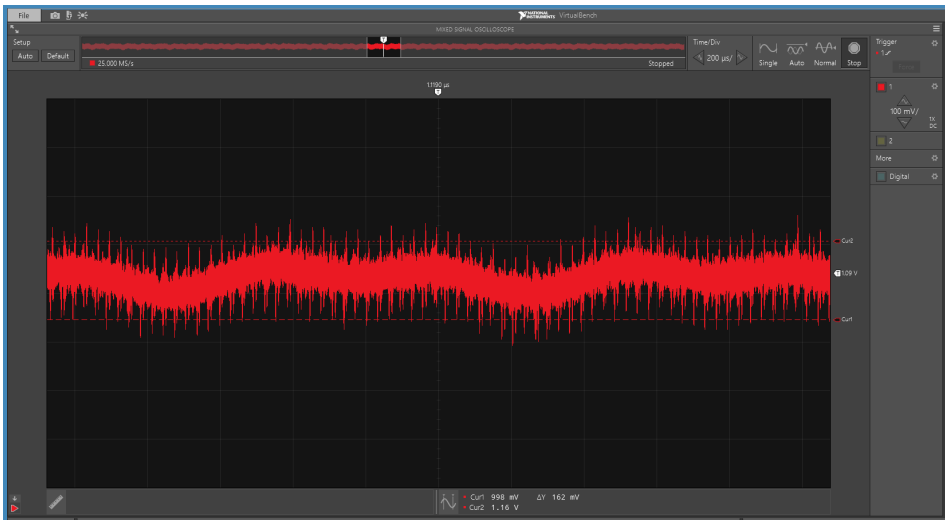


Figure 4.10 Amplifying signal of the sensor near the practical tracking wire

b) Hardware multiplier

In order to increase the efficiency of the multiplier, it is determined that the 8×8 signed hardware multiply will be used to calculate the final S indicating the position of our sensor.

Due to the equation (1.1), it is necessary to reverse the signal value after adc to make sure the range of signal values is symmetric with zero. Therefore, we collected a series of S values directly through adc and dealt with them with theoretical filter based on MATLAB code. It is found that the S bias value after adc is 63 and corresponding waveform is shown below.

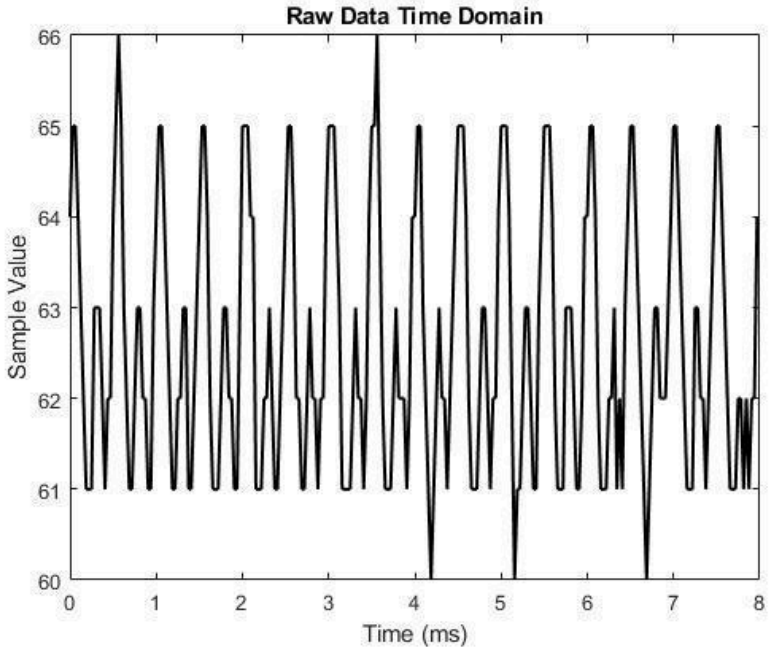


Figure 4.11 raw data time domain collected by ADC sampling

We only retained the first eight high bits of the final product we calculated to prevent from overflow when finally calculating the S value. The unsigned 8×8 multiplier will be used for four times through one loop to gain four values: the in-phase part of frequency response of 1 KHz I_1, the Quadrature part of frequency response of 1 KHz Q_1, the in-phase part of frequency response of 2 KHz I_2, the Quadrature part of frequency response of 2 KHz Q_2. In order not to calculate a S value beyond 2^{32}, I_1, Q_1, I_2, Q_2 have been divided with 2^8.

From the figure 4.3, it is shown that we needed to calculate the squaring of S_1 and the complex conjugate of S_2, so the final S value will be calculated by the following process:

$$S\_1^2 = (I\_1^2 - Q\_1^2) + (j2I\_1Q\_1) \tag{1.8}$$

$$S\_2^* = I\_2 - jQ\_2 \tag{1.9}$$

$$I(S\_1^2 S\_2^*) = I\_2(I\_1^2 - Q\_1^2) + 2I\_1Q\_1Q\_2 \tag{1.10}$$

$$Q(S\_1^2 S\_2^*) = 2I\_1Q\_1I\_2 - (I\_1^2 - Q\_1^2)Q\_2 \tag{1.11}$$

The in-phase value of S has been selected to indicate the position of our sense coil.

c) Navigation Strategy

1. The number of sensors

Due to the inaccuracy of internal ADC and the multiplier, the final S values in different sides of tracking wire are not symmetrical around zero but an unpredicted value can only be measured by test. Therefore, it is time-consuming and ineffective to use one sensor to control both of the motor simultaneously.

Two sensors will be applied to control left and right motor separately, which means two pic chips will be utilized and two S values will be calculated as well.

2. The position of sensors

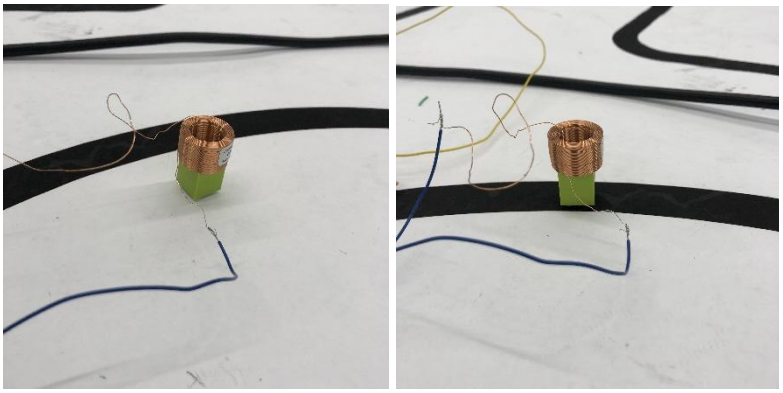


Figure 4.12 sensor position when the coil's section is parallel to the wire

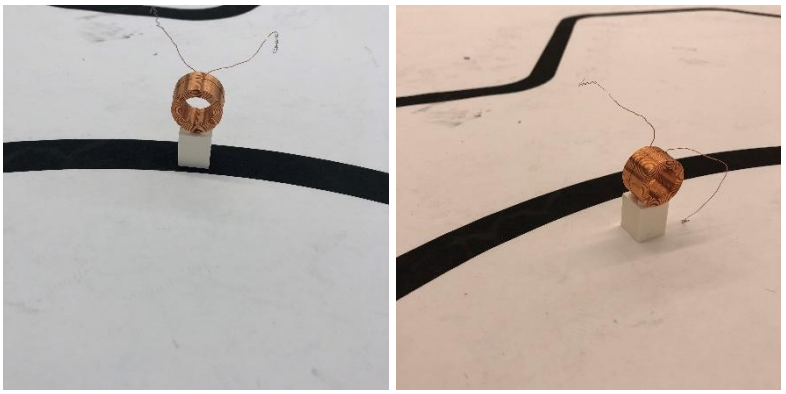


Figure 4.13 sensor position when the coil's section is vertical to the wire

As for the position of these two sensors, it is assumed that the height of them will be around two centimeters above the test surface. And we simulated the distance where the sensor is put in the middle of the wire and on the left side of the wire representing that the wheel controlled by this sensor is closed the tracking wire or away from it.

It is obvious that the changes of S value according to various locations of sensor are larger when the coil's section is vertical to the tracking wire surface. It will be easier to detect whether our vehicle should turn left or right. Figure 4.15 and figure 4.16 show the results of S at two positions.

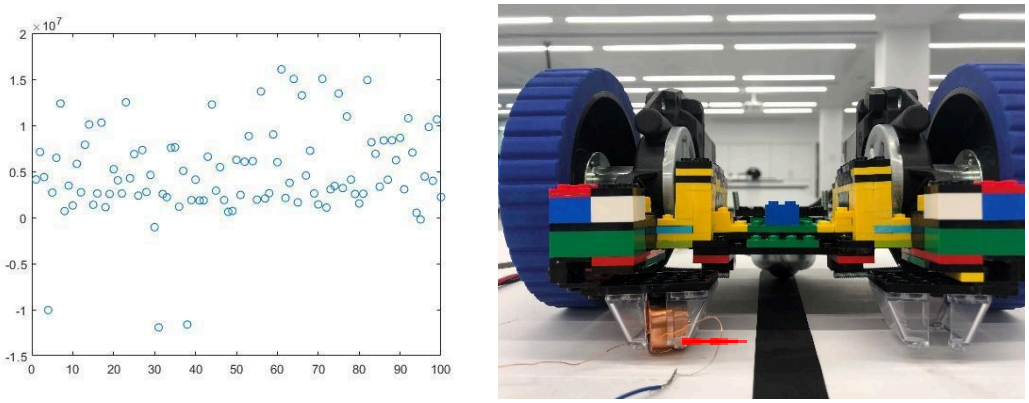


Figure 4.15 S values when coil's section is vertical to the tracking wire surface (closer distance)

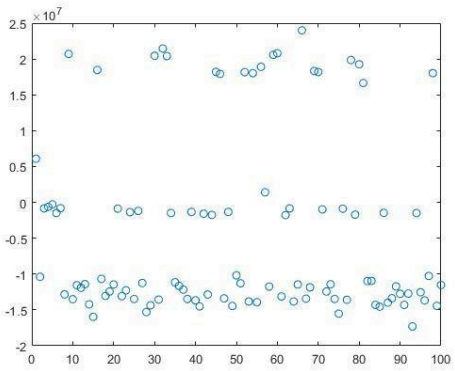


Figure 4.16 S values when coil's section is vertical to the tracking wire surface (further distance)

d) Method to control the velocity of motor

We can easily notice that when the sensor is positioned to different distance from tracking wires, the gap between those two values are limited to a relatively stable range. Therefore, we set a bound in advance and all of the calculated S values will be subtracted by this bound value. Therefore, S values are divided into two groups (positive and negative). If the treated S value is positive which means that the corresponding wheel is near the wire, the speed of motor will be slow down controlled by PWM wave and vice versa.

To reduce the effect of noise signal, we collected a group of 10 S values and then determine the position of sensor. More specifically, when S value is positive or negative, an indicator will be assigned a value of 1 or -1. Then a counter will be summed by the indicator. This operation will be repeated ten times and the final sum will be the evidence showing whether the sense coil is near or away from tracking wire. If this sum is positive, the duty of PWM wave will be set to a relatively low level. If this sum is negative, the duty of PWM wave will be set to a high level. Therefore, the velocity of the vehicle's wheels will be controlled correspondingly.

D. Conclusion

To build a reliable sensor to detect the tracking wire, we use two sensors to control each of the wheel based on the thesis of EE2A Lab 5. The induced voltage across the sense coil is amplified by SSM2019 amplifying circuit. Then the voltage is transferred by an 8-bit ADC inside the PIC chip and be filtered by transversal filter. With the specific value of frequency domain, we calculated S value indicating the position of sensor. As two sensors are applied separately, each wheel will modify its velocity due to the diverse S values. If the treated S is positive, the corresponding wheel will be slow down and vice versa. What needs to be improved is that the bound value is supposed to be decided in advance rather than set up

automatically. And the accuracy of this bound value will be extremely significant for our final success.

Conclusion

Appendix

Appendices

A. Core code implemented on PIC

```
#include <18F27K40.h>
#DEVICE ADC = 8 // 8 bit ADC
#include <string.h>
#include <stdio.h>
#include <math.h>
#fuses NOWDT, NOPUT, NOPROTECT, NOMCLR, NOLVP, NOCPD, RSTOSC_HFINTRC_64MHZ
#use delay(internal = 32MHZ, clock_out) // the internal clock frequency is 32MHz
#pin_select U1TX = PIN_C4  //use the commands to ocntrol the peripheral pin select, USART transmit data
#pin_select U1RX = PIN_C5 // USART receive data
#pin_select PWM4 = PIN_A7 // set the PWM peripheral output
#use RS232(uart1, baud = 9600, ERRORS, STREAM = Serial_Stream) //to configure the MSSP interface
#define SERIALBUFFSIZE 32 // the depth of the buffer is 32

struct IO_Port_Def
{
  int unusedA1 : 2; // A0..1
  int1 IP3; // A2
  int1 unusedA2 ; // A3
  int1 adc_input2; //A4
  int1 adc_input; // A5
  int1 clockoutput; // A6 lock output
  int1 PWM4; // A7 analogue signal input
  int ExperimentSelection : 4; // B0..3
  int unusedA3 : 4; // B4..7
  int1 multi_bit_LED1; // C0
  int1 multi_bit_LED2;// C1
  int1 multi_bit_LED3;// C2
  int1 multi_bit_LED4;// C3
  int1 U1TX; // transmit the signal
  int1 U1RX; // receive the data
  int1 unused_3 ;
  int1 unused_2 ;
};

// make two copies of this structure
struct IO_Port_Def IO_Port;
struct IO_Port_Def IO_Port_Latch;
struct IO_Port_Def IO_Port_Direction;

// Lock these copies down to a particular location
#byte IO_Port = 0xF8D
#byte IO_Port_Latch = 0xF83
#byte IO_Port_Direction = 0xF88

// Global Variables
char RS232[SERIALBUFFSIZE]; // set the serial buffer
int RS232_next_in = 0; // set the input pointer
int RS232_next_out = 0; // set the input pointer
short int SerialCmdWaitFlg = False; // set the Serial Command In flag
short int CmdDoneFlag = False; // set the Command Read Done Flag
char command_string[30]; // Set the command string
int16 ADC_Buffer[256]; // set the ADC_Buffer
int16 Buffer_Index = 0; // Set the ADC buffer size
int Demand_current;
int Measured_current;
int Error_integer;
signed int32 S;
signed int8 Signal;
signed int16 Product_1_I;
signed int16 Product_1_Q;
signed int16 Product_2_I;
signed int16 Product_2_Q;
signed int32 I_1;
signed int32 Q_1;
signed int32 I_2;
signed int32 Q_2;
signed int8 i1;
signed int8 q1;
signed int8 i2;
signed int8 q2;
signed int counter = 0;
int index = 0;
signed int indicator;

signed int8 table_1KHz_I[256] =
{0,0,0,0,0,0,0,0,0,-1,-1,-2,-2,-3,-4,-5,-5,-6,-6,-6,-6,-5,-4,-2,0,2,4,7,10,12,15,17,18,
19,19,18,16,13,
9,5,0,-6,-12,-19,-24,-30,-34,-38,-40,-41,-40,-37,-33,-27,-19,-10,-1,10,20,31,40,49,
56,61,64,64,62,57,
49,39,28,14,0,-16,-31,-46,-59,-71,-80,-86,-89,-89,-85,-78,-67,-54,-38,-20,-1,19,39
,57,73,87,98,106,109,
108,102,93,80,63,44,22,0,-24,-46,-67,-86,-101,-113,-120,-123,-122,-115,-104,-89
,-70,-49,-25,-1,24,48,70,
89,105,117,125,127,125,117,105,89,70,48,24,-1,-25,-48,-70,-89,-104,-115,-121,-
123,-120,-112,-101,-85,-66,
-46,-23,-1,22,43,63,79,92,101,107,108,104,97,86,73,56,38,19,-1,-19,-37,-53,-66,-
77,-84,-87,-88,-85,-78,-69,
-58,-45,-30,-15,-1,14,27,39,48,55,60,62,62,59,54,48,39,30,20,10,-1,-10,-19,-26,-3
2,-36,-38,-39,-39,-37,-33,
-29,-24,-18,-12,-6,-1,5,9,13,15,17,18,18,17,16,14,12,9,7,4,2,0,-2,-4,-5,-5,-6,-6,-5,-
5,-4,-3,-3,-2,-2,-1,
-1,0,0,0,0,0,0,0,0};

signed int8 table_1KHz_Q[256] =
{0,0,0,0,0,0,0,0,1,1,1,1,1,1,0,0,-2,-3,-4,-6,-7,-9,-10,-11,-12,-12,-12,-11,-9,-7,-4,0
,3,8,12,
16,20,23,26,28,29,28,27,23,19,14,7,0,-9,-17,-25,-33,-40,-45,-50,-52,-53,-51,-47,-
41,-34,-24,-13,
-1,12,25,38,49,59,67,73,76,76,73,67,58,46,32,17,0,-18,-36,-52,-67,-80,-90,-97,-1
00,-99,-95,-86,
-74,-59,-41,-22,-1,21,42,62,80,95,106,113,117,115,109,99,85,67,46,23,0,-25,-48,
-70,-89,-105,-117,
-124,-127,-125,-118,-106,-90,-71,-49,-25,-1,24,48,70,89,105,117,123,126,123,11
5,103,88,68,47,23,
0,-24,-47,-67,-85,-99,-109,-115,-117,-114,-106,-95,-80,-62,-43,-22,-1,20,40,58,73
,84,93,97,98,95,
88,78,65,50,34,17,0,-17,-33,-46,-58,-67,-73,-76,-76,-73,-67,-59,-49,-38,-26,-13,-1
,11,22,32,39,45,
49,50,50,47,43,37,31,23,15,7,0,-8,-14,-19,-24,-26,-28,-28,-28,-26,-23,-20,-16,-12,
-8,-4,-1,3,5,8,9,
10,10,10,9,8,7,6,4,3,2,0,0,-1,-2,-2,-2,-2,-2,-2,-1,-1,-1,-1,-1,-1,-1,0};

signed int8 table_2KHz_I[256] =
{0,0,0,0,0,-1,-1,-1,-2,-2,-2,-1,0,1,2,3,4,5,4,2,0,-4,-7,-10,-11,-11,-9,-6,0,5,11,16,18,
18,14,8,0,-10,-19,-26,-29,-28,-23,-13,
-1,13,25,35,39,37,30,16,0,-19,-35,-47,-52,-50,-39,-22,-1,22,43,57,64,60,47,26,-1,
-28,-52,-70,-77,-73,-57,-32,-1,32,60,80,88,
83,64,35,-1,-37,-69,-91,-100,-94,-73,-40,-1,40,75,99,109,101,78,43,-1,-44,-82,-10
8,-118,-109,-84,-46,-1,46,86,112,122,114,87,
47,0,-48,-89,-117,-127,-118,-90,-49,-1,48,90,117,127,117,90,48,0,-49,-90,-117,-1
27,-117,-89,-48,-1,47,87,113,122,112,85,46,0,
-46,-84,-109,-117,-107,-81,-44,-1,42,78,100,108,98,74,40,0,-40,-72,-93,-99,-90,-6
8,-37,-1,35,63,81,87,79,59,31,0,-31,-56,-71,
-76,-68,-51,-27,-1,25,46,59,62,56,42,22,0,-21,-38,-48,-51,-45,-34,-18,-1,16,29,36,
```

```
38,33,24,12,0,-12,-22,-27,-28,-24,-18,-9,-1,
8,14,17,17,15,11,5,0,-5,-9,-11,-10,-9,-6,-3,-1,2,3,4,4,3,2,1,0,-1,-2,-2,-1,-1,-1,-1,0,0
,0,0};

signed int8 table_2KHz_Q[256] =
{0,0,0,0,0,0,0,0,1,1,1,1,1,1,0,0,-2,-3,-4,-6,-7,-9,-10,-11,-12,-12,-12,-11,-9,-7,-4,0
,3,8,12,16,20,
23,26,28,29,28,27,23,19,14,7,0,-9,-17,-25,-33,-40,-45,-50,-52,-53,-51,-47,-41,-34,
-24,-13,-1,12,25,38,49,59,67,73,76,76,
73,67,58,46,32,17,0,-18,-36,-52,-67,-80,-90,-97,-100,-99,-95,-86,-74,-59,-41,-22,-
1,21,42,62,80,95,106,113,117,115,109,
99,85,67,46,23,0,-25,-48,-70,-89,-105,-117,-124,-127,-125,-118,-106,-90,-71,-49,
-25,-1,24,48,70,89,105,117,123,126,123,
115,103,88,68,47,23,0,-24,-47,-67,-85,-99,-109,-115,-117,-114,-106,-95,-80,-62,-
43,-22,-1,20,40,58,73,84,93,97,98,95,88,
78,65,50,34,17,0,-17,-33,-46,-58,-67,-73,-76,-76,-73,-67,-59,-49,-38,-26,-13,-1,11
,22,32,39,45,49,50,50,47,43,37,31,23,15,
7,0,-8,-14,-19,-24,-26,-28,-28,-28,-26,-23,-20,-16,-12,-8,-4,-1,3,5,8,9,10,10,10,9,8
,7,6,4,3,2,0,0,-1,-2,-2,-2,-2,-2,-2,-1,-1,-1,-1,-1,-1,-1,0};



#INT_RDA
void serial_isr() // RS232 Receive Data Available
{
  // Place character in the buffer
  RS232[RS232_next_in] = fgetc(Serial_Stream);
  if (((RS232_next_in + 1) % SERIALBUFFSIZE) != RS232_next_out)
  {
    RS232_next_in = (RS232_next_in + 1) % SERIALBUFFSIZE;
  }

}

void GetCom()
{
  int length;
  char c;
  length = 0;
  RS232_next_out = 0;
  /*Reset command received flag*/
  /*Ignore leading spaces */
  do
  {
    c = RS232[RS232_next_out];
    RS232_next_out = (RS232_next_out + 1) % SERIALBUFFSIZE;

  } while (c == ' ');
            /*A non-space character has been entered*/
  command_string[length++] = c;
   /*get rest of command string - until space or CR termibated or input string is
too long*/
  while (c != 0x0D)
  {
    c = RS232[RS232_next_out]; // Get a character from the Serial buffer
    RS232_next_out = (RS232_next_out + 1) % SERIALBUFFSIZE;
    command_string[length++] = c;
  }
  command_string[--length] = 0;  // NULL terminate string
}

// clear the buffer when the command is done
void clear_buffer()
{
  int data_length;
  data_length = 0;
  while(data_length < SERIALBUFFSIZE)
 {
  RS232[data_length++] = 0; // set all of the data in the buffer t0 0
 }
  RS232_next_in = 0; //reset the input buffer pointer
  RS232_next_out = 0; //reset the output bufer pointer
}

// get the number from the command to control

#int_timer2
void Timer2_Service_Rountine(void)
{
}
// collect one buffer of 1024 samples
#int_AD
void ADC_Service_Rountine(void)
{
 if(Buffer_Index <256)
  {
   ADC_Buffer[Buffer_Index] = read_adc(ADC_READ_ONLY);
  }
  set_adc_channel(5);
  Signal = read_adc(ADC_READ_ONLY)  - 64;

  i1 = table_1KHz_I[Buffer_Index];

  q1 = table_1KHz_Q[Buffer_Index];

  i2 = table_2KHz_I[Buffer_Index];

  q2 = table_2KHz_Q[Buffer_Index];

  register _PRODH int8 Product_High_Byte;
  register _PRODH int8 Product_Low_Byte;

  #asm
  MOVF Signal, W;
  MULWF i1;
  BTFSC i1, 7;
  SUBWF Product_High_Byte, F;
  MOVF  i1,W;
  BTFSC Signal, 7;
  SUBWF Product_High_Byte,F;
  #endasm
  Product_1_I += Product_High_Byte;


  #asm
  MOVF Signal, W;
  MULWF q1;
  BTFSC q1, 7;
  SUBWF Product_High_Byte, F;
  MOVF  q1,W;
  BTFSC Signal, 7;
  SUBWF Product_High_Byte,F;
  #endasm
  Product_1_Q += Product_High_Byte;


  #asm
  MOVF Signal, W;
  MULWF i2;
  BTFSC i2, 7;
  SUBWF Product_High_Byte, F;
  MOVF i2,W;
  BTFSC Signal, 7;
  SUBWF Product_High_Byte,F;
  #endasm
  Product_2_I += Product_High_Byte;


  #asm
  MOVF Signal, W;
  MULWF q2;
  BTFSC q2, 7;
  SUBWF Product_High_Byte, F;
  MOVF q2,W;
  BTFSC Signal, 7;
  SUBWF Product_High_Byte,F;
  #endasm
  Product_2_Q += Product_High_Byte;


  Buffer_Index = Buffer_Index + 1;

  if (Buffer_Index == 256)
  {
    I_1 = (Product_1_I>>8);
    Q_1 = (Product_1_Q>>8);
    I_2 = (Product_2_I>>8);
    Q_2 = (Product_2_Q>>8);
    S = (I_1*I_1 - Q_1*Q_1)*I_2 + 2*I_1*Q_1*Q_2;
    Product_1_I = 0;
    Product_1_Q = 0;
    Product_2_I = 0;
    Product_2_Q = 0;
  }
}

void main()
{

  IO_Port_Direction.unusedA1 = 0b00;
  IO_Port_Direction.IP3 = 0b0; // output
  IO_Port_Direction.unusedA1 = 0b0;
  IO_Port_Direction.adc_input2 = 0b0; //adc port for feedback
  IO_Port_Direction.adc_input = 0b0; //PWM wave as an output
  IO_Port_Direction.clockoutput = 0b0;
  IO_Port_Direction.PWM4 = 0b0; // adc_analogue signal input
  IO_Port_Direction.ExperimentSelection = 0b1111;
  IO_Port_Direction.unusedA3 = 0b0000;
  IO_Port_Direction.multi_bit_LED1 = 0b0; // LED1 output
  IO_Port_Direction.multi_bit_LED2 = 0b0; // LED2 output
  IO_Port_Direction.multi_bit_LED3 = 0b0; // LED2 output
  IO_Port_Direction.multi_bit_LED4 = 0b0; // LED2 output
  IO_Port_Direction.U1TX = 0b0; // Transmit a signal
  IO_Port_Direction.U1RX = 0b1; //Receive a signal
  IO_Port_Direction.unused_3 = 0b0; //
  IO_Port_Direction.unused_2 = 0b0; //
```

```
   setup_ccp2(CCP_PWM|CCP_USE_TIMER1_AND_TIMER2); //configure CCP2 as a PWM
   setup_pwm4(PWM_ENABLED|PWM_ACTIVE_HIGH|PWM_TIMER2); //CCP2 is paried with Timer 2

   setup_timer_2(T2_CLK_INTERNAL|T2_DIV_BY_1,249,1); //32MHz/(4*250*1) = 32KHz

   setup_adc_ports(sAN5|VSS_FVR); //set up Pin A5 to be a analogue input and used the Fixed Voltage Reference
    setup_adc(ADC_LEGACY_MODE|ADC_CLOCK_DIV_32); // ADC_clk should be set to a frequency less than 1MHz
  setup_vref(VREF_ON|VREF_ADC_1v024); //-Vref = 0V and +Vref = 1.024V
    set_adc_trigger(ADC_TRIGGER_TIMER2);  // configure Timer 2 to trigger an ADC at a 32 kHz rate

   setup_adc_ports(sAN4|VSS_FVR); //set up Pin A7 to be a analogue input and used the Fixed Voltage Reference
    setup_adc(ADC_LEGACY_MODE|ADC_CLOCK_DIV_32); // ADC_clk should be set to a frequency less than 1MHz
  setup_vref(VREF_ON|VREF_ADC_1v024); //-Vref = 0V and +Vref = 1.024V
    set_adc_trigger(ADC_TRIGGER_TIMER2);  // configure Timer 2 to trigger an ADC at a 32 kHz rate


  port_a_pullups(0xFF); //enable a pull-up resistor
  port_b_pullups(0xFF);
  port_c_pullups(0xFF);

  // pre-set the input commands


  char Cmd11[] =  "COLLECTDATA"; // Collect the data of the analogue signal 00
  char Cmd12[] = "DIRECTION";
  char input_character;

  RS232_next_in = 0; // set the input pointer as 0
  RS232_next_out = 0; // set the output pointer as 0

  if (kbhit()) // get the commands from the keyboard
    fgetc(Serial_Stream);

  enable_interrupts(INT_RDA); // enable the preset interrupt
  enable_interrupts(INT_TIMER2);
  enable_interrupts(INT_AD);
  enable_interrupts(GLOBAL);

  while (TRUE)
  {
    if (RS232_next_in != RS232_next_out)  // when the output pointer is different from the input pointer
    {
        input_character = RS232[RS232_next_out];
      if (RS232[RS232_next_out] == 0x0D) // when the last input character is CR
      {
        SerialCmdWaitFlg = TRUE; //set the command waiting flag as true
        GetCom(); // transfer the input command
      }
        RS232_next_out = (RS232_next_out + 1) % SERIALBUFFSIZE; //  circulation pointer
         if (SerialCmdWaitFlg == TRUE) // begin to compare the pre-set commands with input commands
      {
        if (strcmp(Cmd11,command_string) == 0) // ADC
        {
          int16 i;
          i = 0;
          int16 a;
          while(i < 256)
          {
          a = ADC_Buffer[i];
}
          printf("%ld\r\n",a); // print the digital data after converted
          i = i + 1;
          }
        SerialCmdWaitFlg = FALSE;
        CmdDoneFlag = TRUE;
        Buffer_Index = 0;
       }
       if(strstr(command_string, Cmd12))
         {
           while(true)
           {
                 S = S - 5 * pow(10,6); // subtract S value by bound value for the following algorithm
            if(S > 0)
            {
             indicator = 1; // when S > 0, set the indicator as 1
            }
            if(S < 0)
            {
              indicator = -1; // when S < 0, set the indicator as 1
            }
            counter = counter + indicator; //sum up every indicator
            if(index == 10) // repeat the operation for ten times
            {
              if(counter >= 0)
              {
                set_adc_channel(4);
                     Measured_current = read_adc(ADC_READ_ONLY); // measured current of armature current
                Demand_current = 5; //pre-set value
                      Error_integer = (Demand_current - Measured_current)<<2 ; //proportional control for armature current
                  set_pwm4_duty(Error_integer); // if the sum is stiil positive, pwm duty is set to a small value
              }
              if(counter < 0)
              {
                set_adc_channel(4);
                Measured_current = read_adc(ADC_READ_ONLY);
                Demand_current = 6;
                Error_integer = (Signal - Measured_current)<<2 ;
                  set_pwm4_duty(Error_integer); // if the sum is stiil positive, pwm duty is set to a lager value
              }
              index = 0; // reset index and counter
              counter = 0;
            }
            index = index + 1;
            Buffer_Index = 0;
            printf("%ld\r\n",S);
            }
            SerialCmdWaitFlg = FALSE;
            CmdDoneFlag = TRUE;


          }

             if (CmdDoneFlag == TRUE) // when the command reading is done,print 'OK'
        {
          CmdDoneFlag = FALSE;
          SerialCmdWaitFlg = FALSE;
          clear_buffer(); // clear buffer to restore the command
        }
         else
        {
          SerialCmdWaitFlg = FALSE;
          clear_buffer();// clear buffer to restore the command
        }
        }
     }
   }
```

B. S display MATLAB code

```
clear;

ADC_Sampling_Rate = 32e3;       % PIC18F27K40 was programmed to collect data at a 30 kHz rate
ADC_Sampling_Points = 256;      % PIC18F27K40 was programmed to collect 1024 samples of data
B = zeros(1,100);

% Generate indeces for later display purposes
DisplayTime_ms =
1000*(0:(ADC_Sampling_Points-1))./ADC_Sampling_Rate;
% Time value of each sample in ms

DisplayFrequency_kHz =
ADC_Sampling_Rate.*(0:(ADC_Sampling_Points-1))./ADC_Sampling_Points ./ 1000;      % Frequency value of each transformed sample in kHz

% Configure port
% Check comms port on Windows control panel for com port number
COM_Port = 'COM7';      % This com port number will vary from machine-to=machine
COM_Baud = 9600;
```

```
% ***** Note: Terminator can be
'CR','LF','CR/LF','LF/CR' and must match your C code
RS232_Object =
serial(COM_Port,'BaudRate',COM_Baud,'DataBits',8,'Pari
ty','none','StopBits',2,'FlowControl','none','Terminat
or','LF');

fopen(RS232_Object);
% fopen will throw a Matlab error if the port is not
available, a typical error message will be:
% Error using serial/fopen (line 72)
% Open failed: Port: COM12 is not available. Available
ports: COM3, COM4, COM5, COM6, COM7.

%% Send commands using fwrite
for SnapShotIndex = 1:500
    % Send command to start data collection
    Command = ['DIRECTION' char(13)];             %
Send 'ADC' command string with a defined terminator
character
    fwrite(RS232_Object,Command,'int8');

    % Recover the data collected by the ADC
        ReturnedString = fscanf(RS232_Object);
% This may be an empty string causing the following
lines to crash
        fprintf("%c",ReturnedString)
        ReturnedSampleValue =
str2double(ReturnedString);
        B(1, SnapShotIndex) = ReturnedSampleValue;
        % This is done in multiple lines to ease
debugging
end

figure(4)
x = 1:500;
plot(x,B,'o')
fclose(RS232_Object);
```

C. Bias Voltage and raw data display MATLAB code

```
clear;

ADC_Sampling_Rate = 32e3;       % PIC18F27K40 was
programmed to collect data at a 30 kHz rate
ADC_Sampling_Points = 256;     % PIC18F27K40 was
programmed to collect 1024 samples of data
A = zeros(100,256);

% Generate indeces for later display purposes
DisplayTime_ms =
1000*(0:(ADC_Sampling_Points-1))./ADC_Sampling_Rate;
% Time value of each sample in ms
DisplayFrequency_kHz =
ADC_Sampling_Rate.*(0:(ADC_Sampling_Points-1))./ADC_Sa
mpling_Points ./ 1000;     % Frequency value of each
transformed sample in kHz

% Configure port
% Check comms port on Windows control panel for com
port number
COM_Port = 'COM7';     % This com port number will
vary from machine-to=machine
COM_Baud = 9600;

% ***** Note: Terminator can be
'CR','LF','CR/LF','LF/CR' and must match your C code
RS232_Object =
serial(COM_Port,'BaudRate',COM_Baud,'DataBits',8,'Pari
ty','none','StopBits',2,'FlowControl','none','Terminat
or','LF');

fopen(RS232_Object);
% fopen will throw a Matlab error if the port is not
available, a typical error message will be:
% Error using serial/fopen (line 72)
% Open failed: Port: COM12 is not available. Available
ports: COM3, COM4, COM5, COM6, COM7.

%% Send commands using fwrite
for SnapShotIndex = 1:50
    % Send command to start data collection
    Command = ['COLLECTDATA' char(13)];             %
Send 'ADC' command string with a defined terminator
character
    fwrite(RS232_Object,Command,'int8');

    % Recover the data collected by the ADC
    DataSnapShot = zeros(1,ADC_Sampling_Points);
    for SampleIndex = 1:ADC_Sampling_Points
        ReturnedString = fscanf(RS232_Object);
% This may be an empty string causing the following
lines to crash
        fprintf("%c",ReturnedString)
        ReturnedSampleValue =
str2double(ReturnedString);
%         B(1, SnapShotIndex) = ReturnedSampleValue;
        % This is done in multiple lines to ease
debugging
        DataSnapShot(SampleIndex) =
ReturnedSampleValue;
%         B(1, SampleIndex) = ReturnedSampleValue;
        A(SnapShotIndex,SampleIndex) =
ReturnedSampleValue;
    end
%
% Display the data in the time domain
    Figure1Handle = figure(1);
    Figure1Handle.Color = 'w';

plot(DisplayTime_ms,DataSnapShot,'k','LineWidth',1.5);
    xlabel('Time (ms)');
    ylabel('Sample Value');
    title('Raw Data Time Domain');
%
%     % Transform data to the frequency domain
    DataSnapShotFFT = fft(DataSnapShot);
    DataSnapShotFFTdB =
20*log10(abs(DataSnapShotFFT));
%
%     % Display the data in the time domain
    Figure2Handle = figure(2);
    Figure2Handle.Color = 'w';

plot(DisplayFrequency_kHz,DataSnapShotFFTdB,'k','LineW
idth',1.5);
    xlim([0 ADC_Sampling_Rate/2000]);
    xlabel('Frequency (kHz)');
    ylabel('Spectral Value (dB re 1 bit)');
    title('Raw Data Frequecy Domain');
%
end
fclose(RS232_Object);
```