Table of Contents

1. Introduction

Objective

The objective of this project was to design a robot that can detect and pick up coins. The robot operates in a predefined perimeter which is created using AC current carrying wires. The robot is made using a folded chassis, servo motors, and servo wheels. It has an electromagnet on an arm controlled by servo motors that is used to pick up the coin. A microcontroller was used to control the various circuitry components to operate the robot.

Specifications

Hardware:

- Robot:
    - 2 x Solarbotics GM4
    - 2 x Servo Wheel 2.63" x 0.35" (pair)
    - Tamiya 70144 Ball Caster
    - 4 x AA Battery holder
    - 1 x 9V cable (9v Battery Clip)
    - 2 x KY61 or equivalent (Metal Gear Micro Servo Motor)
    - 1 x Coin picker assembly
    - 1 x Electromagnet
    - Folded Chassis
- Circuit:

    - BO230XS USB Adapter
    - PIC32MX130 Microcontroller
    - 2 x Optocoupler
    - 1 x Voltage regulator
    - N-FET MOSFETS
    - P-FET MOSFETS

Software:

- PIC32MX130 Functions
    - Coin detection
    - Perimeter detection
    - Moving the wheel motors
    - Servo actuation
    - Electromagnet activation

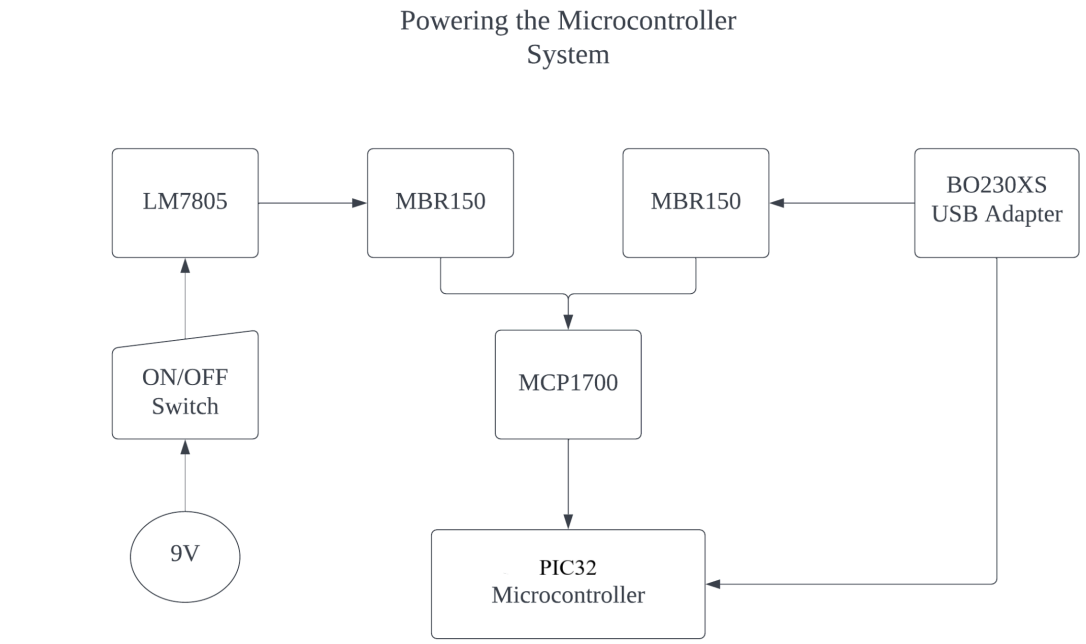

Figure 1: Hardware block diagram for powering the microcontroller system

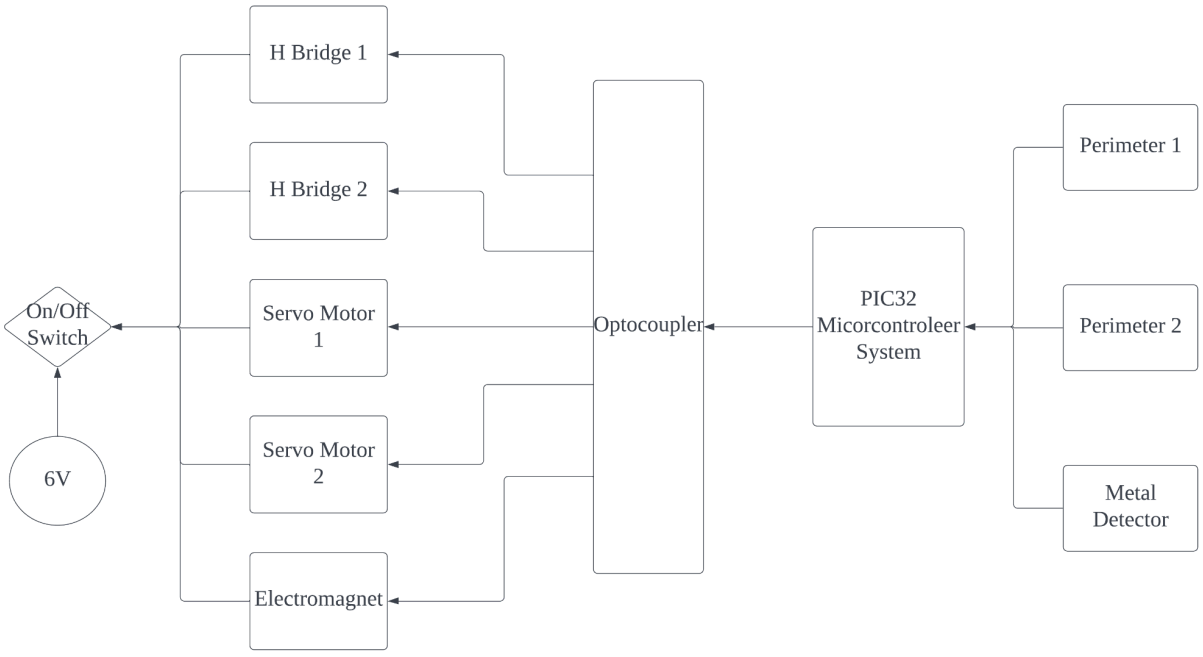

Figure 2: Hardware block diagram for powering the motors, servo and electromagnet

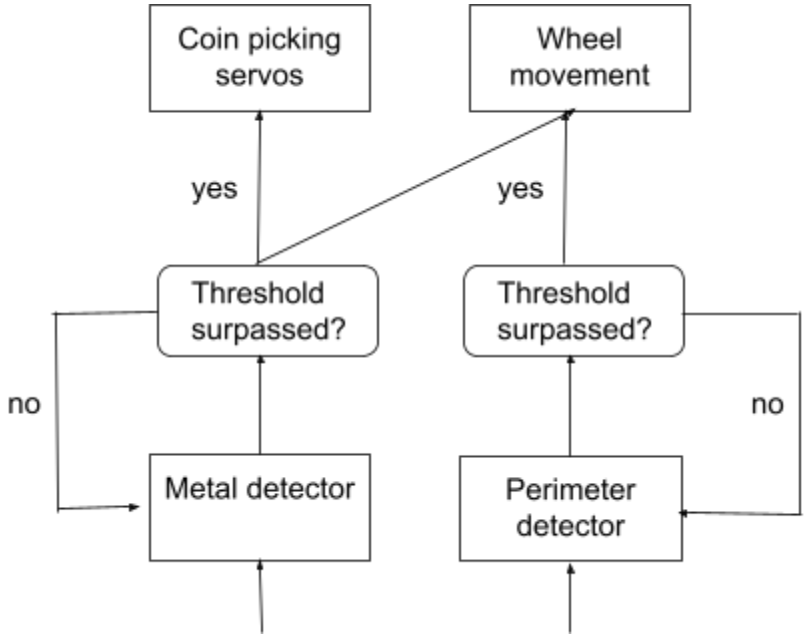

Figure 3 : Block diagram of software design

2. Investigation

Idea generation

Our team generated several ideas for the construction of the hardware necessary to construct the robot.To manage and debug the system, two breadboards will be used to spread out the circuit, which will be beneficial.

To implement the software design, the tasks necessary for the robot to pick up the coins were identified and divided. These tasks included the detection of coins, movement, perimeter detection, and activation of servo motors.

Figure 4 lists the hardware required for the construction of the system and the software tasks divided into parts.

Investigation Design

The design process began with organizing the hardware and identifying its place on the breadboard. Figure 5 shows the system placement planning on the first and second breadboards to ensure smooth construction of the robot. The parts being MOSFETS, microcontroller, op amps, and optocouplers. The software required to run different parts of the hardware was determined and specified tasks were coded to run the robot as intended.

Data Collection

The data collected contains quantitative values of elements, examples of these elements include resistors, capacitors, inductors, and voltage sources. Frequency measurements for the metal detector were also collected. The angle of movement of the servo motors was recorded and the movement required for the robot was determined using trial and error.

Data regarding the wiring of the systems was collected. The robot requires 7 digital inputs, 1 digital output, and 2 analog inputs. We collected information on how to connect the microcontroller system to optocouplers and the optocouplers to the H-bridges, servos, and electromagnet.

The data collected for component values and system were obtained through the ELEC291_Project_2_2022 posted on canvas and the March 18th ELEC 291 lecture.

Data Synthesis

We integrated the quantitative data we collected into our design planning. We checked our lab kits to see if we have the necessary materials. Then, we assembled the components to construct the coin picking robot. We also took the provided software information and grouped it

accordingly with our divided software tasks as in Table 1. For instance, the provided frequency equation is needed for our metal detector program.

Function	Software to achieve the function
Coin detection	Measure frequency. Equation given by: f = 1/(2*pi*sqrt(LC))
Moving the motor	Sending signal to the H-bridge. Each H-bridge requires two signals.
Perimeter detection	Voltage detection. Determine a threshold voltage using trial and error.
Servo actuation for the picker arm	Use the PWM interrupts to change the angle of rotation for the arms.

Table1: Functions identified for the robot to work.

Analysis of Results

Through data synthesis of all the provided circuitry data and using our breadboard simulator, we constructed the elements in our breadboard. We checked the validity of each system using a multimeter and measuring the voltage and comparing it with the expected result. For instance, in Figure 6, we checked the validity of our perimeter detector by connecting our system to a voltmeter. When a charged wire is brought close to the inductor, we were able to measure a significant change in voltage. This can be seen in Figures 7 (a) and (b). Thus, we analyzed these results and concluded that our perimeter detector was working.

3. Design

Use of Process

Our design process was to regularly attend lab hours. This allowed us to use proper lab equipment and work as a team. During the week of the deadline, our group met in the lab everyday to continue using lab equipment.

An example of a complex problem we solved was the replacement of our microcontroller. The original design of the robot included the use of the SAMD20 microcontroller to run our coin picking robot. However, while testing for short circuits using the power supply, we passed too much current through the circuit which broke the SAMD20 along with the components connected to it towards the left half of the circuit. Figure 8 (b) shows us replacing the left half of our circuit along with the microcontroller we were using. We then decided to use the PIC32 and the entire left side of the circuit was rewired to get the system up and running again, as seen in Figure 8 (a).

Need and Constraint Identification

Our team brainstormed several stakeholder needs for this coin-picking robot. We viewed this robot from a consumer's perspective. Since consumers have less technical knowledge than engineer designers, our coin-picking robot needed visual clarity. This means that users can easily identify how to turn the robot on and off, and the wiring should be clean enough as to not overwhelm the user. Another major aspect is functionality. The coin-picking robot needed to be

able to gently pick up and place down coins. Thus, we inferred that stakeholders had two major needs: visual appeal and functionality.

Problem Specification

Our team inferred that visual appeal and functionality were the two main constraints. Based on this, we listed and categorized additional several design requirements. Table 2 shows the design requirements, such as wiring length and structural clarity.

Visual Appeal	Functionality
● The wire lengths should be minimalized ● The second breadboard should look nice when stationed above the chassis ● The on/off button must be clear and accessible	● Make sure both breadboards are stable ● Robot slows picks up and drops down coins ● Electromagnet shouldn't scrape the bottom but still be able to detect coins

Table 2: Table of additional design requirements

Solution Generation

In order to fit two breadboards into our robot chassis, our team generated two solutions for our robot's structure. As seen in Figure 9 and 10, we thought of either wedging the smaller breadboard in the space between the long breadboard and chassis, or adding a plastic frame to hold the smaller breadboard. These two solutions were our options for the structural integrity and visual appeal of our robot.

Solution Evaluation

In order to evaluate which structure was better, we conducted two systematic evaluations. These evaluations were stability and visual appeal. We tested the stability by shaking the robot chassis and lightly applying force around the second breadboard. For visual appeal, we reviewed both systems as a team and wrote our observations as seen on Table 3.

	Stability	Aesthetic
Wedged breadboard	Shake: - Sometimes gets loose Light force: - Sometimes gets loose	- Unique - Unclear hardware view
Plastic frame	Shake: - Steady and firm Light force: - Stays in place	- Unique - Both breadboard looks nicely aligned - Clarity with hardware

Table 3: stability and aesthetic testing through team discussion

We concluded the plastic holder to be far more stable while carrying the small breadboard. Furthermore, our team agreed that the plastic holder was more aesthetic and provided a double-decker look to our robot. Thus, based on our testing and decision making, we decided our plastic holder to be the final structural design. Figure 11 shows our final design.

Safety/Professionalism

Throughout the project our group made sure to follow the safety guidelines for the lab. An example of this was when we were soldering the wires for the robot. Since we have little experience with soldering, we made sure to take extra precautions to ensure the safety of our

group and our fellow classmates. We wore safety goggles and gloves and made sure to turn the fan on so that no one inhaled fumes from the soldering iron. Additionally, to increase the efficiency of our work, we coordinated the schedule beforehand so everyone was able to contribute equally to the project. This helped us complete the project before the deadline and add extra features as well.

Detailed Design

Hardware

Robot Chassis

The robot chassis was built using the precut and folded aluminum from the project 2 kit. 2 solarbotics motors were installed in the center of the chassis along with two wheels. A steel ball roller was screwed on at the front of the chassis to serve as a front wheel and keep the robot balanced. Additionally, at the front of the chassis, three ferrite core inductors were used for perimeter detection and coin detection.

Electro magnet/coin picker

The electro-magnet was built by wrapping the thin copper wire around the metal screw until the resistance measured was within a range of 5-9 ohms. This leaves only a few meters of the copper wire to be connected to the controller. As seen in Figure 12, the remaining copper wire was wrapped into a solenoid so as to be connected to the microcontroller and reduce noises.

Arm rotator

As seen in Figure 12, the coin picker arm consists of 2 servos to swivel and rotate horizontally and vertically to pick up and drop the coin into the basket in the back of the robot. The arm was constructed using the precut aluminum from the project 2 kit and attached to the robot chassis. The electromagnet was then bolted to the arm rotator.

H bridge with optocoupler

Each H bridge was built using a combination 2 P-mosfet and 2 N-mosfet along with 2 LTV- 846 optocoupler. Two 10k ohms resistors are connected between the P-mosfet and the optocoupler. And finally, the source voltage was connected to a 1k ohm resistor as pictured in Figure 13.

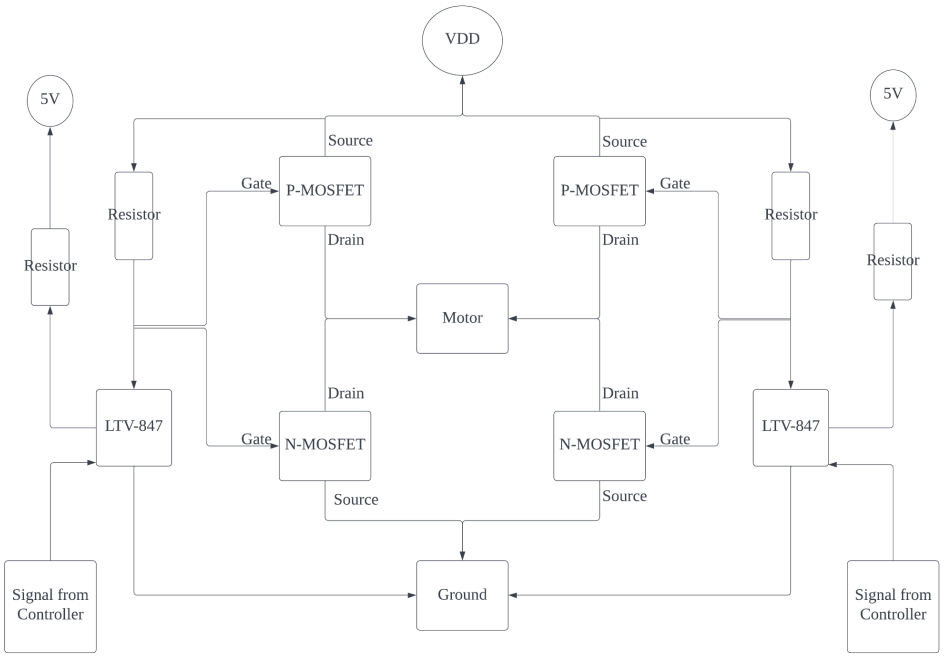

Figure 13 : Block diagram of the H-bridge-Octocoupler diagram.

Perimeter detector

The perimeter consists of 3 parts; The tank circuit, amplifier and the peak detector. The tank circuit was one of the three inductors included in the project kit. The peak detector was built using a 1k ohm resistor connected to ground, a 33k ohm resistor between the output and the inverting terminal. In addition, 10 nF and 100 nF capacitors were used in the circuit. A schottky diode is joined between the amplifier and the peak detector. Together, the whole perimeter detector was built on a smaller breadboard and connected to the PIC32 microcontroller.

Perimeter:

The perimeter was made of a long piece of wire into an area of 0.5 m^2 and connected to a function generator that generates a16khz sinewave. The perimeter was secured using tape.

Software

Our software block diagram remained the same as our first plotting from Figure 3. We split our tasks to the metal detector, coin picking servos, wheel movement, and perimeter detector.

Coin Picking Servos:

To pick up a coin, we had to program the rotation of two servo motors. For each servo, we changed the angle in increments of 10. This allows the arm to gently pick up coins and ensures it stays connected to the electromagnet.

Figure 14 shows the pickCoin function. First, the electromagnet is turned on. Then, ISR_pwm2 is given a new angle in increments of 10 with delays. This was done as moving the servo arm with no delay was too fast, and coins were flung off the electromagnet. This code also sweeps the arm horizontally for nearby coins that were not exactly in front of the metal detector.

Following the sweep, the ISR_pwm1 raises the arm up and turns it over the box as seen in Figure 15. Then, the electromagnet is turned off to drop the coin in the box. Afterwards, the arm is rotated to its original position with additional delay.

Wheel Movement

For the movement of our robot, we created functions that send signals to the H-bridges. Depending on the pins, the signal moves the wheels either moves forwards, backwards, to the right, to the left, or stops. Figure 16 shows the different signals applied to "LATAbits.LATA0" and "LATAbits.LATA1" to change the wheel directions.

Perimeter Detector

Figure 17 shows our perimeter detector code. To detect a perimeter, we created a variable called "thresholdV". The threshold value was set through trial and error, and if either the voltages 'v1' or 'v2' detected by the inductors surpass this threshold, the robot moves backward and orients itself to the right. The angle to the right is randomized depending on the value of 'adcval1'.

Metal detector

To detect a metal, we created a variable called “thresholdP”. The value of thresholdP is the regular frequency detected from the table plus one hundred. Figure 18 shows that if the threshold is crossed, the program waits 100ms and checks again if the threshold is crossed to account for noise. If so, the program calls on the pickCoin function to start picking up the coin. Lastly, we measure the baseline frequency again and set a new value to thresholdP.

Celebratory Song and Dance (extra functionality)

To add extra functionality to our robot, we made our robot play the Super Mario theme song in sync with a blinking LED, and then it dances after picking up 20 coins. Figure 19 shows the initialization of a second timer and its interrupts. Timer 2 was used to control the pitch of the tune by changing number multiplied by 256 in “PR2 = SYSCLK/(256*...) - 1” as seen in Figure 20. After 20 coins are picked up, Figure 21 shows that the song function is called and the wheels are turned left and right to do a celebratory dance.

This extra functionality was added after we filmed a video of our robot picking 20 coins. However, our robot should be able to continuously pick up 20 coins then do the celebratory song after adding code for it.

Solution assessment

Coin detector

For the coin detector, we decided to measure the frequency calculated from the ADC and set a threshold for detection. Shown in Figure 22, we compared the voltage read from the multimeter when connected to the coin detector against the voltage read from the PIC

microcontroller with PUTTY. The comparison provided us a threshold range within 3 degrees of error to be implemented in our function. We were able to get the coin detector to detect all 20 coins we placed inside the perimeter.

Arm Rotator

For the arm servos used in the arm rotator, we had to reduce the speed of the servo after it picked up the coin because the default speed was too fast and caused the coin to be thrown off the magnet when rotating. To determine the optimal rotation increment and delay, we tried several different combinations. As seen in Table 4, our tests concluded that a 10 degree increment of rotation and 200 ms delay was the perfect combination for our arm.

Rotation Increments (°)	Delay (ms)	Notes
15	100	The arm moves too fast and the coin falls off.
5	400	The arm moves too slow and takes too long to deposit the coin in the container
10	200	Optimal speed for the arm movement. Coin is secured on the electromagnet and deposited in the container in a timely fashion

Table 4: Testing arm rotation table

Electromagnet

Overall, the electromagnet was the most simple to implement and fit the criteria of having a resistance in the range of 5-7 ohms. If it is either over or under the range, we just have to increase or decrease the winding of the solenoid. The biggest challenge we faced is that the electromagnet would pick up the coin but drop it halfway. We found that the connection between

the connector and the wire of the solenoid was loose and we solved that by wrapping the wire more and soldering the connection.

Perimeter Detector

Since we have many components to fit onto the bread board and we are limited in size to place on the robot. We built the perimeter detector on a smaller breadboard that would make it easier to debug and to be placed on top of the robot in a double deckered design. Shown in Table 5, we tested and determined the threshold voltage by bringing a wire connected to the function generator and observing the jump in PUTTY.

Threshold Voltage (V)	Observation
0.400	Sometimes turned too early when approaching the wire
0.500	Consistently turned perfectly
0.600	Sometimes it does not detect the wire or turns very close to the wire.

Table 5: Testing perimeter threshold table

4. Live-Long Learning

For this project, some parts of the robot required soldering. To complete the robot our team learned how to solder. In addition, a course that we took in the past that helped us with this project was CPSC 259. This course helped us understand the base code and adapt it to include the functions required.

Conclusion

The design and functionality of this project was to create a robot that could move around and pick up all the coins in a predefined region. The robot uses an arm that has an electromagnet attached to one end which is used to pick up coins.. The coins are then dropped into a container which is built onto the frame of the robot. Once the coin has been picked up and deposited into the container the robot then looks for another coin to pick up. Throughout the process of completing the robot, the most challenging obstacle we faced was connecting all of the small portions to make a completely functioning robot.

References

[1] Calvino-Fraga, J, “Lab 3: 555 timer and Capacitance meter”, UBC, January, 2022

[2] Calvino-Fraga, J, “Lab 2: Timer, Interrupts and Pushbuttons”, UBC,January 2022

Bibliography

- Calvino-Fraga, J., Project 1: Reaction Game with Capacitive Sensors Lecture, Department of Electrical and Computer Engineering, UBC, 2022
- Calvino-Fraga, J., 555 timer and Capacitance Meter Lecture, Department of Electrical and Computer Engineering, UBC, 2022
- Calvino-Fraga, J., Timers, Interrupts, and Push Buttons Lecture, Department of Electrical and Computer Engineering, UBC, 2022
- Steiner, C., 8051/8052 Instruction Set, Hobby Projects, 2020.
- AT89LP51RB2 DataSheet, Atmel Corporation, 2011.

Appendices

Figures

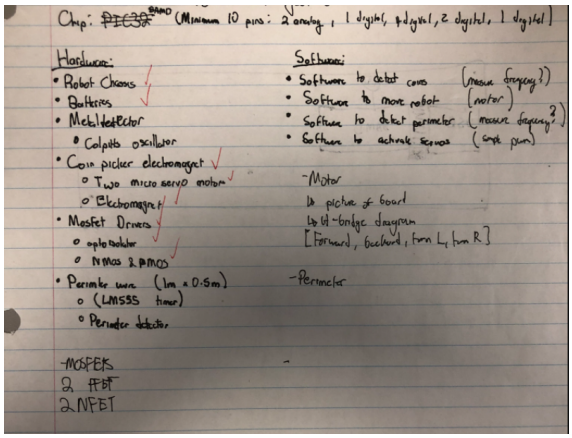

Figure 4: Categorization of software and hardware parts

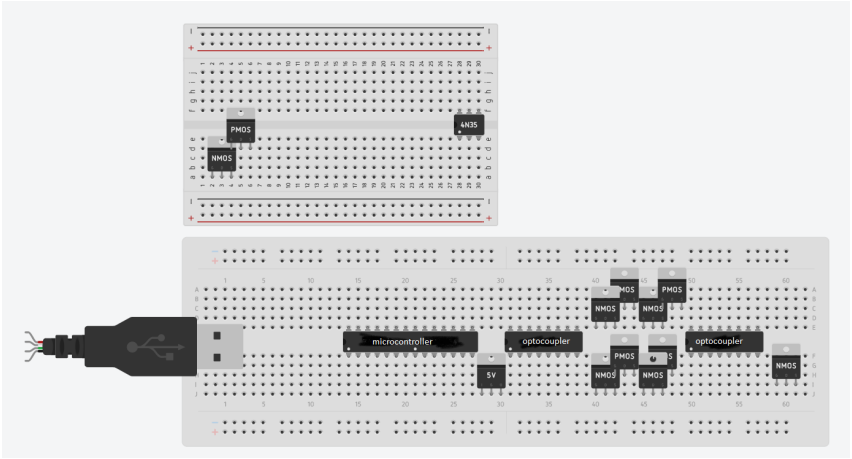

Figure 5: Plotting the location of chips, i.e op-amps, microcontrollers, and optocouplers.

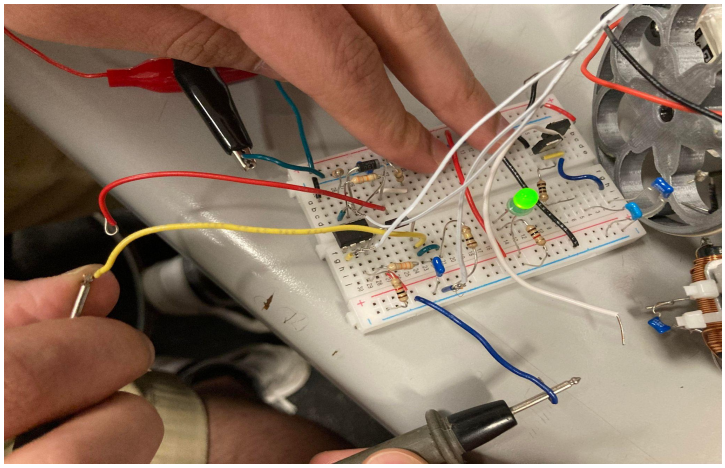

Figure 7 (a): Connecting the perimeter detector to a voltmeter

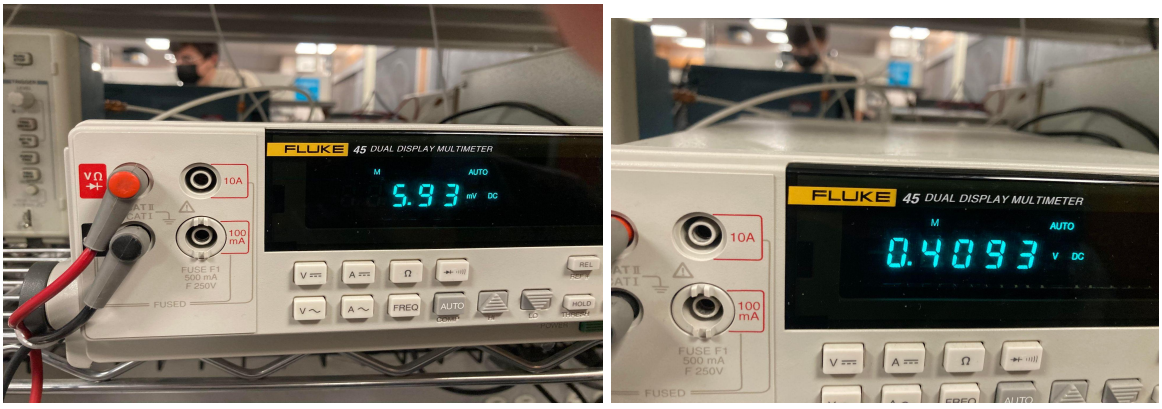

Figure 7 (a) (b): Initial voltage reading jumping when a 16kHz current carrying wire is brought near the inductor

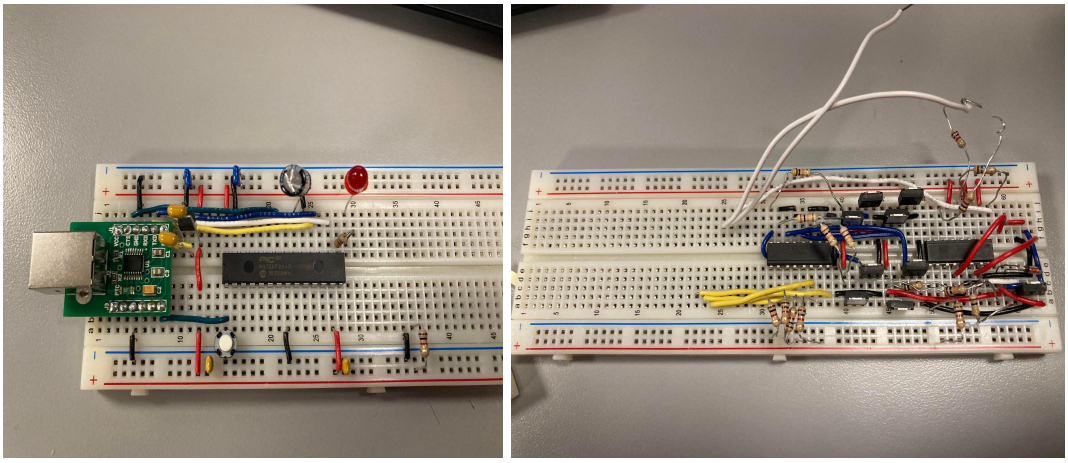

Figure 8 (a) (b): The PIC32 microcontroller system (left) had to replace the left side of our main breadboard (right).

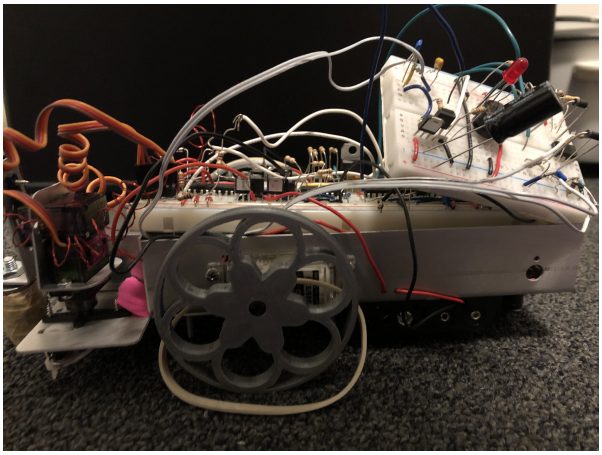

Figure 9: Smaller breadboard wedged in a small space

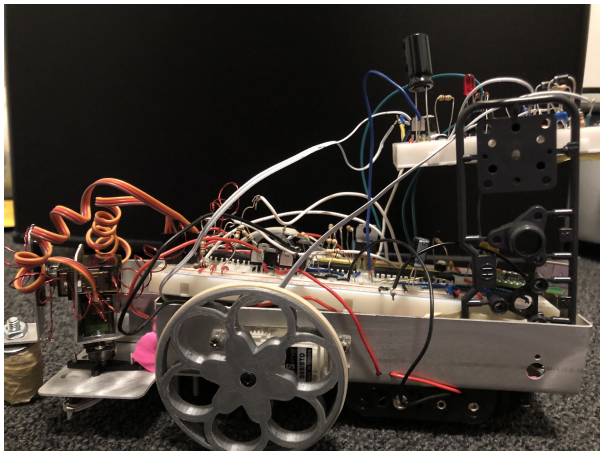

Figure 10: smaller breadboard propped up by a plastic frame

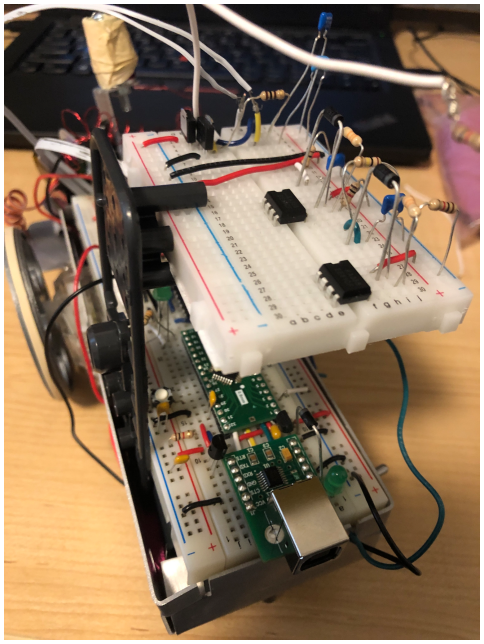

Figure 11: Final design for the plastic holder

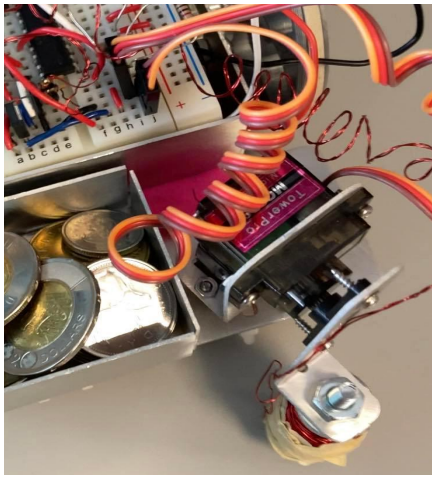

Figure 12: An electromagnet connected to two servo motors for horizontal and vertical rotation

```
void pickCoin (void){
    LATBbits.LATB4 = 1;
    waitms(1000);

    //sweep
    ISR_pwm2=160;
    waitms(200);
    ISR_pwm2=170;
    waitms(200);
    ISR_pwm2=180;
    waitms(200);
    ISR_pwm2=190;
    waitms(200);
    ISR_pwm2=200;
    waitms(200);
    ISR_pwm2=210;
    waitms(200);
    ISR_pwm2=220;
    waitms(200);
    ISR_pwm2=210;
    waitms(200);
    ISR_pwm2=200;
    waitms(200);
    ISR_pwm2=190;
    waitms(200);
    ISR_pwm2=180;
    waitms(200);
    ISR_pwm2=170;
    waitms(200);
    ISR_pwm2=160;
    waitms(200);
    ISR_pwm2=150;
    waitms(200);
    ISR_pwm2=140;
    waitms(200);
    ISR_pwm2=130;
    waitms(200);
    ISR_pwm2=120;
    waitms(200);
    ISR_pwm2=130;
    waitms(200);
    ISR_pwm2=140;
    waitms(200);
    ISR_pwm2=150;
    waitms(200);
    ISR_pwm2=160;
    waitms(200);
```

Figure 14: pickCoin function. It first sweeps horizontally to pick up a nearby coin.

```
    //pwm1:90

    ISR_pwm1=250;
    waitms(200);
    ISR_pwm1=240;
    waitms(200);
    ISR_pwm1=230;
    waitms(200);
    ISR_pwm1=220;
    waitms(200);
    ISR_pwm1=210;
    waitms(200);
    ISR_pwm1=200;
    waitms(200);
    ISR_pwm1=190;
    waitms(200);
    ISR_pwm1=180;
    waitms(200);
    ISR_pwm1=170;
    waitms(200);
    ISR_pwm1=160;
    waitms(200);
    ISR_pwm1=150;
    waitms(200);
    ISR_pwm1=140;
    waitms(100);
    ISR_pwm1=130;
    waitms(200);
    ISR_pwm1=120;
    waitms(200);
    ISR_pwm1=110;
    waitms(200);
    ISR_pwm1=100;
    waitms(200);
    ISR_pwm1=90;
    waitms(200);
    //pwm2:80
    ISR_pwm2=150;
    waitms(200);
    ISR_pwm2=140;
    waitms(200);
    ISR_pwm2=130;
    waitms(200);
    ISR_pwm2=120;
    waitms(200);
    ISR_pwm2=110;
    waitms(200);
    ISR_pwm2=100;
    waitms(200);
    ISR_pwm2=90;
    waitms(200);
    ISR_pwm2=80;
    waitms(200);


    LATBbits.LATB4 = 0;
    waitms(1000);


    ISR_pwm2=155;
    waitms(1000);
    ISR_pwm1=260;
    waitms(1000);
}
```

Figure 15: Raising the arm and moving it above the coin box.

```
void goFowards(void){

    //H-bridge 1
    LATAbits.LATA0 = 1;
    LATAbits.LATA1 = 0;

    //H-bridge 2
    LATBbits.LATB0 = 1;
    LATBbits.LATB1 = 0;

}
```

```
void goBackwards(void){

    //H-bridge 1
    LATAbits.LATA0 = 0;
    LATAbits.LATA1 = 1;

    //H-bridge 2
    LATBbits.LATB0 = 0;
    LATBbits.LATB1 = 1;

}
```

```
void goRight(void){

    //H-bridge 1
    LATAbits.LATA0 = 1;
    LATAbits.LATA1 = 0;

    //H-bridge 2
    LATBbits.LATB0 = 0;
    LATBbits.LATB1 = 1;

}
```

```
void stop(void){

    //H-bridge 1
    LATAbits.LATA0 = 0;
    LATAbits.LATA1 = 0;

    //H-bridge 2
    LATBbits.LATB0 = 0;
    LATBbits.LATB1 = 0;

}
```

```
392  void goLeft(void){
393
394      //H-bridge 1
395      LATAbits.LATA0 = 0;
396      LATAbits.LATA1 = 1;
397
398      //H-bridge 2
399      LATBbits.LATB0 = 1;
400      LATBbits.LATB1 = 0;
401
402  }
403
```

Figure 16: Functions for different wheel movements based on the signal sent to the H-bridges

```
//perimeter code
if (v1 > thresholdV || v2 > thresholdV){
    if (adcval1%2==0) {

        remainder = 1;

   }
   else {

        remainder = 0;

    }

    goBackwards();
    waitms(500);

    if (remainder) {
        goRight();
        waitms(400);
   }
    else {
        goRight();
        waitms(600);

    }

   goFowards();
   //waitms(1000);
}
```

Fig 17: Perimeter detector code

```
count=GetPeriod(100);
if(count>0)
{
    f=((SYSCLK/2L)*100L)/count;
    uart_puts("f=");
    PrintNumber(f, 10, 7);
    uart_puts("Hz, count=");
    PrintNumber(count, 10, 6);
    uart_puts("          \r");
}
else
{
    uart_puts("NO SIGNAL                    \r");
}

if(f > thresholdP) {
    waitms(100);
    count=GetPeriod(100);
    f=((SYSCLK/2L)*100L)/count;
    if(f > thresholdP){

        goBackwards();
        waitms(250);
        stop();
        pickCoin();
        //increment counter;
        counter++;
        //measure baseline freq
        waitms(500);
        count=GetPeriod(100);
        f=((SYSCLK/2L)*100L)/count;
        thresholdP = f + 120;
    }

}
```

Figure 18: Metal Detector Code

```
66  void __ISR(_TIMER_2_VECTOR, IPL5SOFT) Timer2_Handler(void)
67  {
68      if(LATAbits.LATA4 == 1)
69          LATAbits.LATA4 = 0;
70      else
71          LATAbits.LATA4 = 1;
72      IFS0bits.T2IF=0; // Clear timer 1 interrupt flag, bit 4 of IFS0
73
74
75  }
76
77  void SetupTimer2 (void)
78  {
79
80      __builtin_disable_interrupts();
81      PR2 = SYSCLK/(256*100) -1;
82      TMR2 = 0;
83      T2CONbits.TCKPS = 3; // 3=1:256 prescale value, 2=1:64 prescale v
84      T2CONbits.TCS = 0; // Clock source
85      T2CONbits.ON = 0;
86      IPC2bits.T2IP = 5;
87      IPC2bits.T2IS = 0;
88      IFS0bits.T2IF = 0;
89      IEC0bits.T2IE = 1;
90
91      INTCONbits.MVEC = 1; //Int multi-vector
92      __builtin_enable_interrupts();
93  }
94
```

Figure 19: Initialization of timer 2 and its interrupts

```
415  void song(void){
416          PR2 = SYSCLK/(256*110) -1;
417      T2CONbits.ON = 1;
418      waitms(150);
419      T2CONbits.ON = 0;
420      waitms(100);
421      PR2 = SYSCLK/(256*110) -1;
422      T2CONbits.ON = 1;
423      waitms(150);
424      T2CONbits.ON = 0;
425      waitms(200);
426      PR2 = SYSCLK/(256*110) -1;
427      T2CONbits.ON = 1;
428      waitms(400);
429      T2CONbits.ON = 0;
430
431      PR2 = SYSCLK/(256*87) -1;
432      T2CONbits.ON = 1;
433      waitms(200);
434      T2CONbits.ON = 0;
435
436      PR2 = SYSCLK/(256*110) -1;
437      T2CONbits.ON = 1;
438      waitms(500);
439      T2CONbits.ON = 0;
440      waitms(100);
441      PR2 = SYSCLK/(256*130) -1;
442      T2CONbits.ON = 1;
443      waitms(1100);
444      T2CONbits.ON = 0;
445      waitms(100);
446
447      PR2 = SYSCLK/(256*65) -1;
448      T2CONbits.ON = 1;
449      waitms(1100);
450      T2CONbits.ON = 0;
451      waitms(100);
452  }
```

Figure 20: Mario song program using timer 2

```
    song();
    while(1){
        goRight();
        waitms(1000);
        goLeft();
        waitms(500);
        goRight();
        waitms(2000);
        goLeft();
        waitms(1000);
        goRight();
        waitms(500);
        goLeft();
        waitms(2000);
    }
}
```

Figure 21: The song and dance code

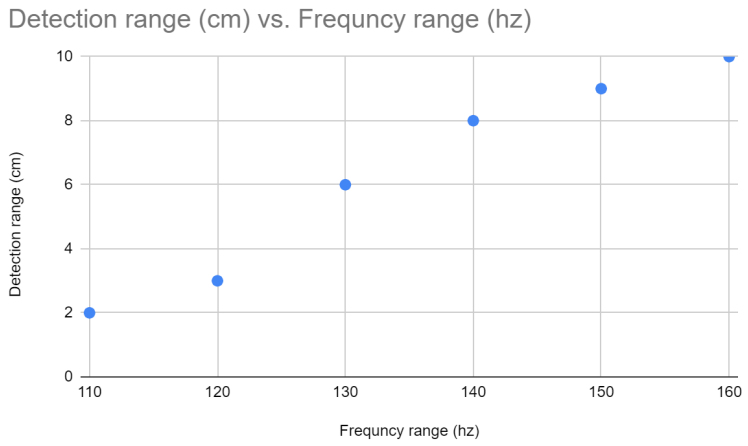

Figure 22: comparison of frequency range and detection range

Source Code:

```
#include <XC.h>
#include <sys/attribs.h>
#include <stdio.h>
#include <stdlib.h>

// Configuration Bits (somehow XC32 takes care of this)
#pragma config FNOSC = FRCPLL       // Internal Fast RC oscillator (8 MHz) w/ PLL
#pragma config FPLLIDIV = DIV_2     // Divide FRC before PLL (now 4 MHz)
#pragma config FPLLMUL = MUL_20     // PLL Multiply (now 80 MHz)
#pragma config FPLLODIV = DIV_2     // Divide After PLL (now 40 MHz)
#pragma config FWDTEN = OFF         // Watchdog Timer Disabled
#pragma config FPBDIV = DIV_1       // PBCLK = SYCLK
#pragma config FSOSCEN = OFF        // Turn off secondary oscillator on A4 and B4

// Defines
#define SYSCLK 40000000L
#define FREQ 100000L // We need the ISR for timer 1 every 10 us
#define Baud2BRG(desired_baud)( (SYSCLK / (16*desired_baud))-1)


volatile int ISR_pwm1=250, ISR_pwm2=155, ISR_cnt=0;

// The Interrupt Service Routine for timer 1 is used to generate one or more standard
// hobby servo signals.  The servo signal has a fixed period of 20ms and a pulse width
// between 0.6ms and 2.4ms.
void __ISR(_TIMER_1_VECTOR, IPL5SOFT) Timer1_Handler(void)
{

	IFS0CLR=_IFS0_T1IF_MASK; // Clear timer 1 interrupt flag, bit 4 of IFS0

	ISR_cnt++;
	if(ISR_cnt==ISR_pwm1)
	{
		LATAbits.LATA2 = 0;
	}
	if(ISR_cnt==ISR_pwm2)
```

```
        {
                LATAbits.LATA3 = 0;
        }
        if(ISR_cnt>=2000)
        {
                ISR_cnt=0; // 2000 * 10us=20ms
                LATAbits.LATA2 = 1;
                LATAbits.LATA3 = 1;
        }
}

void SetupTimer1 (void)
{
        // Explanation here: https://www.youtube.com/watch?v=bu6TTZHnMPY
        __builtin_disable_interrupts();
        PR1 =(SYSCLK/FREQ)-1; // since SYSCLK/FREQ = PS*(PR1+1)
        TMR1 = 0;
        T1CONbits.TCKPS = 0; // 3=1:256 prescale value, 2=1:64 prescale value, 1=1:8 prescale
value, 0=1:1 prescale value
        T1CONbits.TCS = 0; // Clock source
        T1CONbits.ON = 1;
        IPC1bits.T1IP = 5;
        IPC1bits.T1IS = 0;
        IFS0bits.T1IF = 0;
        IEC0bits.T1IE = 1;

        INTCONbits.MVEC = 1; //Int multi-vector
        __builtin_enable_interrupts();
}

void __ISR(_TIMER_2_VECTOR, IPL5SOFT) Timer2_Handler(void)
{
        if(LATAbits.LATA4 == 1)
                LATAbits.LATA4 = 0;
        else
                LATAbits.LATA4 = 1;
        IFS0bits.T2IF=0; // Clear timer 1 interrupt flag, bit 4 of IFS0


}

void SetupTimer2 (void)
{

        __builtin_disable_interrupts();
        PR2 = SYSCLK/(256*100) -1;
        TMR2 = 0;
        T2CONbits.TCKPS = 3; // 3=1:256 prescale value, 2=1:64 prescale value, 1=1:8 prescale
value, 0=1:1 prescale value
        T2CONbits.TCS = 0; // Clock source
        T2CONbits.ON = 0;
        IPC2bits.T2IP = 5;
        IPC2bits.T2IS = 0;
        IFS0bits.T2IF = 0;
        IEC0bits.T2IE = 1;

        INTCONbits.MVEC = 1; //Int multi-vector
        __builtin_enable_interrupts();
}

// Use the core timer to wait for 1 ms.
void wait_1ms(void)
```

```
{
    unsigned int ui;
    _CP0_SET_COUNT(0); // resets the core timer count

    // get the core timer count
    while ( _CP0_GET_COUNT() < (SYSCLK/(2*1000)) );
}

void waitms(int len)
{
       while(len--) wait_1ms();
}

#define PIN_PERIOD (PORTB&(1<<5))

// GetPeriod() seems to work fine for frequencies between 200Hz and 700kHz.
long int GetPeriod (int n)
{
       int i;
       unsigned int saved_TCNT1a, saved_TCNT1b;

    _CP0_SET_COUNT(0); // resets the core timer count
       while (PIN_PERIOD!=0) // Wait for square wave to be 0
       {
              if(_CP0_GET_COUNT() > (SYSCLK/4)) return 0;
       }

    _CP0_SET_COUNT(0); // resets the core timer count
       while (PIN_PERIOD==0) // Wait for square wave to be 1
       {
              if(_CP0_GET_COUNT() > (SYSCLK/4)) return 0;
       }

    _CP0_SET_COUNT(0); // resets the core timer count
       for(i=0; i<n; i++) // Measure the time of 'n' periods
       {
              while (PIN_PERIOD!=0) // Wait for square wave to be 0
              {
                     if(_CP0_GET_COUNT() > (SYSCLK/4)) return 0;
              }
              while (PIN_PERIOD==0) // Wait for square wave to be 1
              {
                     if(_CP0_GET_COUNT() > (SYSCLK/4)) return 0;
              }
       }

       return  _CP0_GET_COUNT();
}

void UART2Configure(int baud_rate)
{
    // Peripheral Pin Select
    U2RXRbits.U2RXR = 4;    //SET RX to RB8
    RPB9Rbits.RPB9R = 2;    //SET RB9 to TX

    U2MODE = 0;         // disable autobaud, TX and RX enabled only, 8N1, idle=HIGH
    U2STA = 0x1400;     // enable TX and RX
    U2BRG = Baud2BRG(baud_rate); // U2BRG = (FPb / (16*baud)) - 1

    U2MODESET = 0x8000;     // enable UART2
}
```

```
void uart_puts(char * s)
{
	while(*s)
	{
		putchar(*s);
		s++;
	}
}

char HexDigit[]="0123456789ABCDEF";
void PrintNumber(long int val, int Base, int digits)
{
	int j;
	#define NBITS 32
	char buff[NBITS+1];
	buff[NBITS]=0;

	j=NBITS-1;
	while ( (val>0) | (digits>0) )
	{
		buff[j--]=HexDigit[val%Base];
		val/=Base;
		if(digits!=0) digits--;
	}
	uart_puts(&buff[j+1]);
}

// Good information about ADC in PIC32 found here:
// http://umassamherstm5.org/tech-tutorials/pic32-tutorials/pic32mx220-tutorials/adc
void ADCConf(void)
{
    AD1CON1CLR = 0x8000;    // disable ADC before configuration
    AD1CON1 = 0x00E0;       // internal counter ends sampling and starts conversion
(auto-convert), manual sample
    AD1CON2 = 0;            // AD1CON2<15:13> set voltage reference to pins AVSS/AVDD
    AD1CON3 = 0x0f01;       // TAD = 4*TPB, acquisition time = 15*TAD
    AD1CON1SET=0x8000;      // Enable ADC
}

int ADCRead(char analogPIN)
{
    AD1CHS = analogPIN << 16;    // AD1CHS<16:19> controls which analog pin goes to the ADC

    AD1CON1bits.SAMP = 1;        // Begin sampling
    while(AD1CON1bits.SAMP);     // wait until acquisition is done
    while(!AD1CON1bits.DONE);    // wait until conversion done

    return ADC1BUF0;             // result stored in ADC1BUF0
}

void ConfigurePins(void)
{
    // Configure pins as analog inputs
    ANSELBbits.ANSB2 = 1;   // set RB2 (AN4, pin 6 of DIP28) as analog pin
    TRISBbits.TRISB2 = 1;   // set RB2 as an input
    ANSELBbits.ANSB3 = 1;   // set RB3 (AN5, pin 7 of DIP28) as analog pin
    TRISBbits.TRISB3 = 1;   // set RB3 as an input

       // Configure digital input pin to measure signal period
       ANSELB &= ~(1<<5); // Set RB5 as a digital I/O (pin 14 of DIP28)
    TRISB |= (1<<5);   // configure pin RB5 as input
    CNPUB |= (1<<5);   // Enable pull-up resistor for RB5
```

```
    // Configure output pins
    //H-bridge 1
       TRISAbits.TRISA0 = 0; // pin  2 of DIP28
       TRISAbits.TRISA1 = 0; // pin  3 of DIP28
       //H-bridge 2
       TRISBbits.TRISB0 = 0; // pin  4 of DIP28
       TRISBbits.TRISB1 = 0; // pin  5 of DIP28
       //servo
       TRISAbits.TRISA2 = 0; // pin  9 of DIP28
       TRISAbits.TRISA3 = 0; // pin 10 of DIP28
       //electromagnet
       TRISBbits.TRISB4 = 0; // pin 11 of DIP28
       //speaker
       TRISAbits.TRISA4 = 0; // pin 12 of DIP28




       INTCONbits.MVEC = 1;
}

void pickCoin (void){
       LATBbits.LATB4 = 1;
       waitms(1000);
       LATAbits.LATA4 =1;
       //sweep
       ISR_pwm2=160;
       waitms(200);
       ISR_pwm2=170;
       waitms(200);
       ISR_pwm2=180;
       waitms(200);
       ISR_pwm2=190;
       waitms(200);
       ISR_pwm2=200;
       waitms(200);
       ISR_pwm2=210;
       waitms(200);
       ISR_pwm2=220;
       waitms(200);
       ISR_pwm2=210;
       waitms(200);
       ISR_pwm2=200;
       waitms(200);
       ISR_pwm2=190;
       waitms(200);
       ISR_pwm2=180;
       waitms(200);
       ISR_pwm2=170;
       waitms(200);
       ISR_pwm2=160;
       waitms(200);
       ISR_pwm2=150;
       waitms(200);
       ISR_pwm2=140;
       waitms(200);
       ISR_pwm2=130;
       waitms(200);
       ISR_pwm2=120;
       waitms(200);
       ISR_pwm2=130;
```

```
waitms(200);
ISR_pwm2=140;
waitms(200);
ISR_pwm2=150;
waitms(200);
ISR_pwm2=160;
waitms(200);


//pwm1:90

ISR_pwm1=250;
waitms(200);
ISR_pwm1=240;
waitms(200);
ISR_pwm1=230;
waitms(200);
ISR_pwm1=220;
waitms(200);
ISR_pwm1=210;
waitms(200);
ISR_pwm1=200;
waitms(200);
ISR_pwm1=190;
waitms(200);
ISR_pwm1=180;
waitms(200);
ISR_pwm1=170;
waitms(200);
ISR_pwm1=160;
waitms(200);
ISR_pwm1=150;
waitms(200);
ISR_pwm1=140;
waitms(100);
ISR_pwm1=130;
waitms(200);
ISR_pwm1=120;
waitms(200);
ISR_pwm1=110;
waitms(200);
ISR_pwm1=100;
waitms(200);
ISR_pwm1=90;
waitms(200);

//pwm2:80
ISR_pwm2=150;
waitms(200);
ISR_pwm2=140;
waitms(200);
ISR_pwm2=130;
waitms(200);
ISR_pwm2=120;
waitms(200);
ISR_pwm2=110;
waitms(200);
ISR_pwm2=100;
waitms(200);
ISR_pwm2=90;
waitms(200);
ISR_pwm2=80;
```

```
        waitms(200);


        LATBbits.LATB4 = 0;
        waitms(1000);
                LATAbits.LATA4 =0;

        ISR_pwm2=155;
        waitms(1000);
        ISR_pwm1=260;
        waitms(1000);
}

void goFowards(void){

        //H-bridge 1
        LATAbits.LATA0 = 1;
        LATAbits.LATA1 = 0;

        //H-bridge 2
        LATBbits.LATB0 = 1;
        LATBbits.LATB1 = 0;

}

void goBackwards(void){

        //H-bridge 1
        LATAbits.LATA0 = 0;
        LATAbits.LATA1 = 1;

        //H-bridge 2
        LATBbits.LATB0 = 0;
        LATBbits.LATB1 = 1;

}

void goRight(void){

        //H-bridge 1
        LATAbits.LATA0 = 1;
        LATAbits.LATA1 = 0;

        //H-bridge 2
        LATBbits.LATB0 = 0;
        LATBbits.LATB1 = 1;

}
void goLeft(void){

        //H-bridge 1
        LATAbits.LATA0 = 0;
        LATAbits.LATA1 = 1;

        //H-bridge 2
        LATBbits.LATB0 = 1;
        LATBbits.LATB1 = 0;

}

void stop(void){
```

```
        //H-bridge 1
        LATAbits.LATA0 = 0;
        LATAbits.LATA1 = 0;

        //H-bridge 2
        LATBbits.LATB0 = 0;
        LATBbits.LATB1 = 0;

}
void song(void){
               PR2 = SYSCLK/(256*110) -1;
        T2CONbits.ON = 1;
        waitms(150);
        T2CONbits.ON = 0;
        waitms(100);
        PR2 = SYSCLK/(256*110) -1;
        T2CONbits.ON = 1;
        waitms(150);
        T2CONbits.ON = 0;
        waitms(200);
        PR2 = SYSCLK/(256*110) -1;
        T2CONbits.ON = 1;
        waitms(400);
        T2CONbits.ON = 0;

        PR2 = SYSCLK/(256*87) -1;
        T2CONbits.ON = 1;
        waitms(200);
        T2CONbits.ON = 0;

        PR2 = SYSCLK/(256*110) -1;
        T2CONbits.ON = 1;
        waitms(500);
        T2CONbits.ON = 0;
        waitms(100);
        PR2 = SYSCLK/(256*130) -1;
        T2CONbits.ON = 1;
        waitms(1100);
        T2CONbits.ON = 0;
        waitms(100);

        PR2 = SYSCLK/(256*65) -1;
        T2CONbits.ON = 1;
        waitms(1100);
        T2CONbits.ON = 0;
        waitms(100);
}

// In order to keep this as nimble as possible, avoid
// using floating point or printf() on any of its forms!
void main(void)
{
        volatile unsigned long t=0;
    int adcval1, adcval2;
    //perimeter threshold voltage
    int thresholdV = 500;
    int remainder;
    //threshold period for metal detector;
    int thresholdP;
    long int v1, v2;
        unsigned long int count, f;
        unsigned char LED_toggle=0;
```

```
        int counter = 0;

        CFGCON = 0;

    UART2Configure(115200);  // Configure UART2 for a baud rate of 115200
    ConfigurePins();
    SetupTimer1();
    SetupTimer2();
       stop();
    ADCConf(); // Configure ADC
   /*
    waitms(500); // Give PuTTY time to start
       uart_puts("\x1b[2J\x1b[1;1H"); // Clear screen using ANSI escape sequence.
       uart_puts("\r\nPIC32 multi I/O example.\r\n");
       uart_puts("Measures the voltage at channels 4 and 5 (pins 6 and 7 of DIP28
package)\r\n");
       uart_puts("Measures period on RB5 (pin 14 of DIP28 package)\r\n");
       uart_puts("Toggles RA0, RA1, RB0, RB1, RA2 (pins 2, 3, 4, 5, 9, of DIP28
package)\r\n");
       uart_puts("Generates Servo PWM signals at RA3, RB4 (pins 10, 11 of DIP28
package)\r\n\r\n");
       */
       LATBbits.LATB4 = 0;


       //measure baseline freq
       waitms(500);
       count=GetPeriod(100);
       f=((SYSCLK/2L)*100L)/count;
       thresholdP = f + 100;

       while( counter < 1)
       {

              //gofowards
              goFowards();
              //pickCoin();
       adcval1 = ADCRead(4); // note that we call pin AN4 (RB2) by it's analog number
              uart_puts("ADC[4]=0x");
              PrintNumber(adcval1, 16, 3);
              uart_puts(", V=");
              v1=(adcval1*3290L)/1023L; // 3.290 is VDD

              PrintNumber(v1/1000, 10, 1);
              uart_puts(".");
              PrintNumber(v1%1000, 10, 3);
              uart_puts("V ");

              adcval2=ADCRead(5);
              uart_puts("ADC[5]=0x");
              PrintNumber(adcval2, 16, 3);
              uart_puts(", V=");
              v2=(adcval2*3290L)/1023L; // 3.290 is VDD
              PrintNumber(v2/1000, 10, 1);
              uart_puts(".");
              PrintNumber(v2%1000, 10, 3);
              uart_puts("V ");

              //perimeter code
              if (v1 > thresholdV || v2 > thresholdV){
            if (adcval1%2==0) {
```

```
        remainder = 1;

    }
    else {

        remainder = 0;

     }

     goBackwards();
     waitms(500);

     if (remainder) {
        goRight();
         waitms(400);
    }
     else {
        goRight();
         waitms(600);

     }

    goFowards();
    //waitms(1000);
}


       count=GetPeriod(100);
       if(count>0)
       {
              f=((SYSCLK/2L)*100L)/count;
              uart_puts("f=");
              PrintNumber(f, 10, 7);
              uart_puts("Hz, count=");
              PrintNumber(count, 10, 6);
              uart_puts("          \r");
       }
       else
       {
              uart_puts("NO SIGNAL                     \r");
       }

       if(f > thresholdP) {
              waitms(100);
              count=GetPeriod(100);
              f=((SYSCLK/2L)*100L)/count;
              if(f > thresholdP){

                     goBackwards();
                     waitms(250);
                     stop();
                     pickCoin();
                     //increment counter;
                     counter++;
                     //measure baseline freq
                     waitms(500);
                     count=GetPeriod(100);
                     f=((SYSCLK/2L)*100L)/count;
                     thresholdP = f + 120;
              }

       }
```

```
        }

        song();
        while(1){
                goRight();
                waitms(1000);
                goLeft();
                waitms(500);
                goRight();
                waitms(2000);
                goLeft();
                waitms(1000);
                goRight();
                waitms(500);
                goLeft();
                waitms(2000);
        }
}
```