University of British Columbia
Electrical and Computer Engineering
ELEC291/ELEC292 Winter 2019
Instructor: Dr. Jesus Calvino-Fraga
Section 201

Project 2 – Coin Picking Robot

Group #: B9

Student #	Student Name	Points	Grade	Signature
50399807	Can Kerem Gurel	**100**		
67412874	Edward Ma	**100**		
20301727	Atahan Akar	**100**		
29536463	Ege Berk Akkaya	**100**		
91544643	Maggie Jessen	**100**		
36350437	Erik Elbrecht	**100**		

Date of submission: April 4, 2019

Table of Contents

1. Introduction

Design Objective

The project's goal is to design and create a Coin Picking Robot that satisfies the specifications provided in the project outline document [1]. The overall purpose of this project is to learn how to work with unfamiliar, and non-ideal, components and gain experience working with datasheets and user manuals standard in the engineering workplace.

Project Specifications:

a) Hardware Information

1. *Movement Mechanics:* The coin picking robot can move forward & backwards, and can turn both clockwise and counter-clockwise using motors that control the left / right wheels.
2. *Metal Detection:* An inductor responds to changes in it's magnetic field due to nearby coins by changing the oscillation frequency of its RLC circuit: a colpitts oscillator.
3. *Perimeter Detection*: A tank circuit has voltage induced in it when hovering over a changing magnetic field generated by a wire with a square wave signal traveling through it.
4. *Picking Mechanism:* Picking mechanism consists of an electromagnet and two servo motors.

b) Software Information

1. *Programming Language:* The software of the Coin Picking Robot is programmed in C.
2. *Metal Detection:* GPIO digital input is used for the metal detector.
3. *Perimeter Detection:* ADC is used in order to detect the perimeter.
4. *Controlling the Motors:* GPIO digital output is used for the motors and controlled by a 30 kHz timer interrupt.

c) General Functionality

The coin-picking robot autonomously drives forward and scans for coins, then picks up the coins if the coins are within the range. If the perimeter is detected, it turns around.

Behaviour:

- On startup, the robot calibrates its resting parameters for the metal and perimeter detectors
- Upon detecting a coin it rolled over, the robot moves backwards enough for the servo arm to pick up the coin by rotating down and sweeping in front of itself with the electromagnet activated. The coin is held by the electromagnet until the arm maneuvers to the collection bucket, where it deactivates the magnet and drops the coin inside.
- Upon reaching a perimeter, the robot turns at a fixed angle clockwise and continues moving forward and scanning.
- After 20 coins are detected, the robot stops.

User Input and Maintenance:

- The user can turn on or off the robot using a flip switch on the side of the robot.
- A reset button on the breadboard can be used to reset the robot back to startup stage.
- The robot is powered with four AA batteries and one 9V battery.

Figure 1.0: Microprocessor Pin Connection Diagram

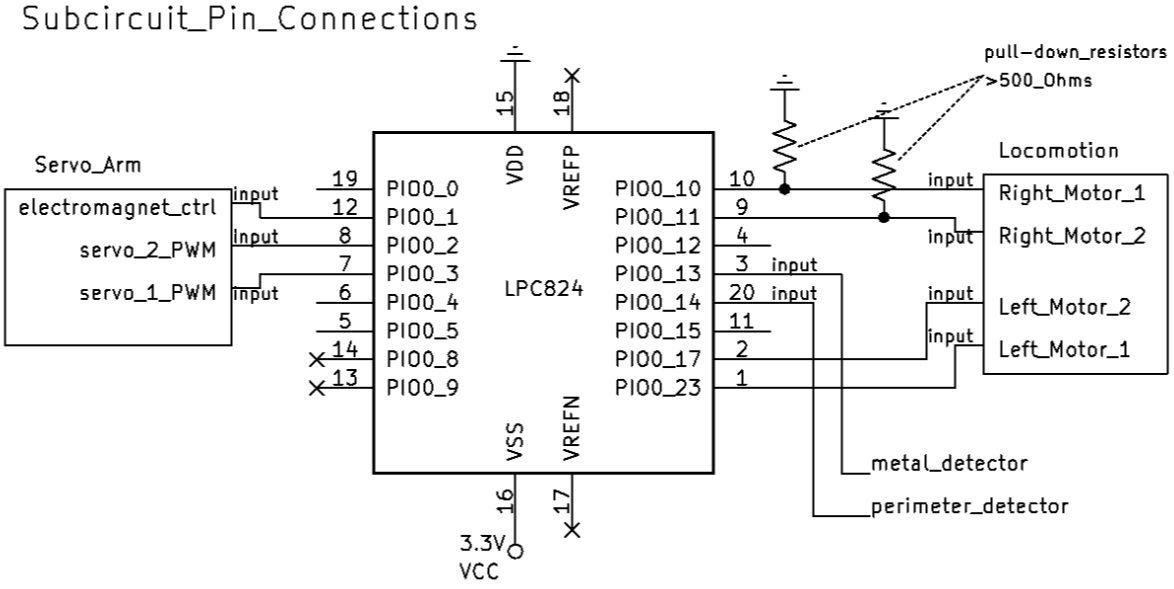


Figure 1.1: Motor H-Bridge Circuit

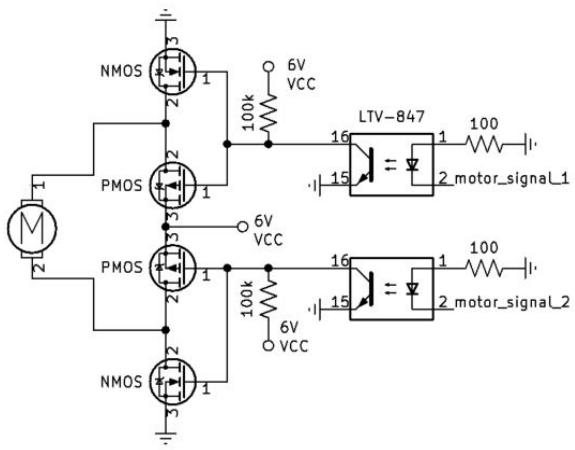


Figure 1.2: Servo Arm Circuit Diagram

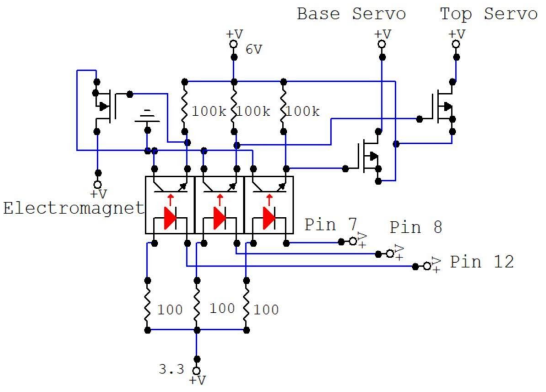


Figure 1.3: Colpitts Oscillator Metal Detector

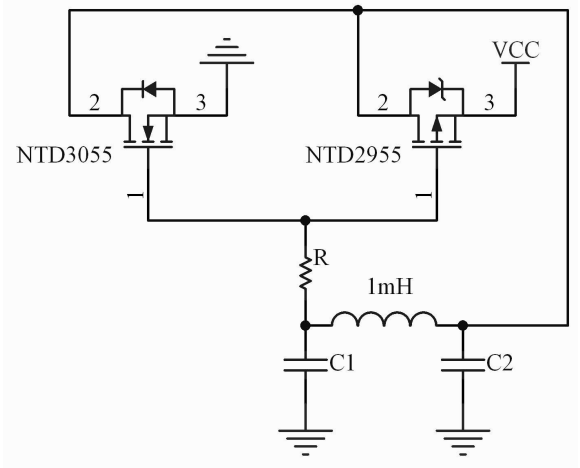


Figure 1.4: Perimeter Detector Circuit

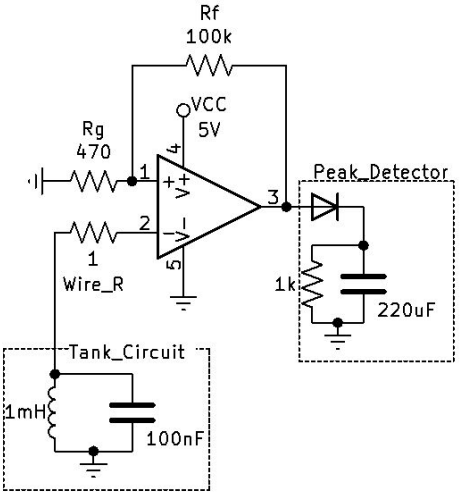


Figure 1.5: 555 Timer Frequency Generator

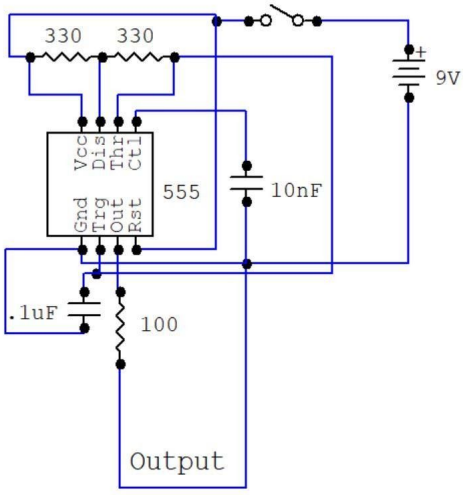


Figure 1.6: Kinetic Shock Detector Circuit

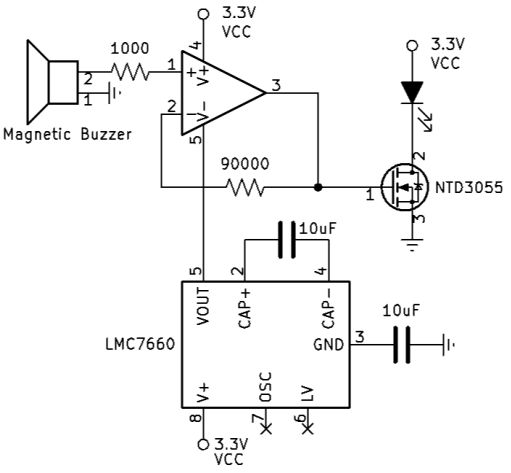
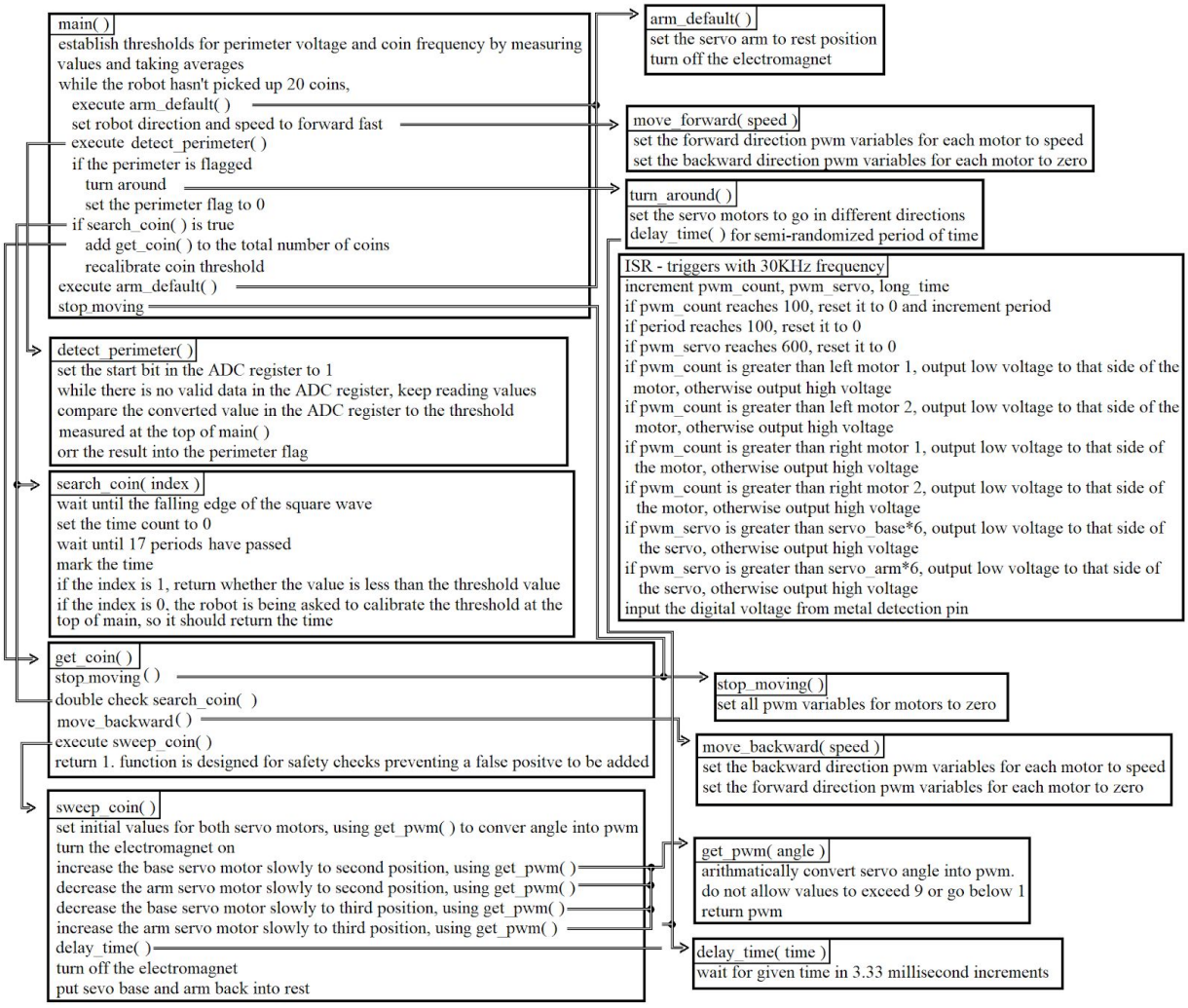


Figure 1.7: Diagram of Software Structure



2. Investigation

2.1 Idea Generation

While referring to the project outline [1] and the slideset [2], we generated several prototypes by wiring subcircuits on separate breadboards, rearranging the components such that they would fit into the breadboard completely. During the project, we generated several bonus features and attempted to save space on our main breadboard for their hardware components. Some of our initial ideas were to make our robot remote controlled by using the HC-06 Bluetooth module, to count and display the number of coins picked up on a LED display, and to make the robot perform a victory dance after picking up all the coins. Reviewing the available space in the breadboard, the challenges of the ideas and the time we had before the project deadline, we were able to implement a basic kinetic-shock detector (see section 3.7.6).

2.2 Design Investigation

Each hardware component and software program were tested and measured separately before combining them piece-by-piece in order to isolate potential errors in the design.

The main breadboard was tested with linear DC power supplies to eliminate oscillations (due to switch-mode power supplies) affecting the Colpitts oscillator (see section 3.7.4). Before connecting to the microprocessor, the small servo motors were tested using a DC power supply and a function generator to act as the PWM signal controlling the angle.

Each function in the software was tested individually within their own main program, function-by-function.

When testing the hardware with software uploaded onto the microprocessor, the oscilloscope was used to measure signals coming into and out of the microprocessor to ensure expected behaviour.

2.3 Data Collection

Robot behaviour was based off of the example video provided by Dr. Jesus Calvino Fraga [8]. From there, we experimented each part of the hardware to figure out the correct values of components. We did many frequency, voltage and current measurements, using both the oscilloscope and multimeter. Threshold values from metal and perimeter detection circuitry were acquired and used to build the software. This allowed for early requests to be made on behalf of software programming that the metal detector frequency be decreased, despite the fact that this would drastically decrease the change in frequency when a coin was applied to the circuitry.

2.4 Data Synthesis

Since the motors were the first software component to be fully implemented, they were used to determine the input voltages for both the perimeter and metal detection by setting the speed of the motors to the input voltage. In the absence of breakpoints this was the most practical solution, and simply required use of the oscilloscope. Since the robot was often handled separately to implement/repair its hardware, the code was primarily tested on a spare microcontroller. Notably, this method was used to determine that the digital GPIO input for the perimeter detection could not be used and subsequently to test the ADC channel that replaced it. The oscilloscope was often used to debug the hardware. Our algorithm to detect thresholds for perimeter and coin detection involved measuring these values when the robot was neither near a coin nor the perimeter and then increased or decreased these values as

appropriate by a set amount to establish a final threshold. These set values had to be determined experimentally.

2.5 Analysis of Results

The most decisive conclusion our tests concluded was that GPIO would be sufficient for the motors, servo motors, electromagnet, and coin detection, but not for perimeter detection. For perimeter detection, we used ADC. Our tests for detection of coins demonstrated that as a precautionary measure, we needed to institute a double check. The perimeter detection reliably detected the perimeter with our threshold. Practical experiments determined that the servo arm could pick up a coin anywhere where the coin could be sensed. However, there was some difficulty where the coin was not reliably dropped into the bucket. To combat this, we inserted longer time delays into the sweep_coin( ) algorithm an resolved the problem.

3. Design

3.1 Use of Process

For testing code and hardware, we used another breadboard set up with a second microprocessor chip, separate from the breadboard with the full circuit. Other breadboards were used to setup and test smaller circuits separately before adding them to the main board. When we had unexpected issues with our main breadboard we used our test breadboards to see if the problems were due to hardware. If not, we would go on to fix the issues with the software. We maintained this use of process throughout the project and it hastened the bug-fixing process.

3.2 Need and Constraint Identification

We identified our stakeholders as students in electrical engineering programs, instructors for ELEC 291, and electronics hobbyists. In addition to the project requirements listed in the project document [1], constraints included not being allowed to the EFM8 microcontroller that we have used for the previous labs, not being able to use a programming language other than C, and a two week timespan to work before the project deadline.

3.3 Problem Specification

Our group analyzed and evaluated project requirements and compared them with stakeholder needs to specify the problems. One significant challenge was to fit all the required components in the main breadboard, maintaining an organized structure. For an additional feature, we wanted to confirm if a coin was successfully collected by using a buzzer.

3.4 Solution Generation

In addition to the project requirements and constraints, we decided to use an additional, half-sized breadboard as we did not have enough space on the main breadboard to fit all the components. The additional breadboard was mounted on the back of the robot, maintaining the compact functionality. As well, we had a variety of microcontrollers to choose from [Appendix I]. The microcontrollers varied in clock speed, CPU architecture, and size. We had narrowed the processors to use down to the PIC32 and the LPC824.

The metal detector uses a colpitts oscillator, as recommended by the slideset [2]. However, to maintain compactness, we could not use more than two capacitors, and experimentation found that some pairs such as 1nF and 10nF produced a frequency that was too fast for our timer interrupt to consistently measure accurately, requiring us to experiment with further pairs. To detect if a coin was dropped and what kind it was, we considered measuring the

different frequency change response of each coin drop, using a magnetic buzzer as an input, effectively acting as a high-pass filtered microphone.

The peak detector circuit was iterated upon multiple times in order to make it more compact and more effective. It was using a dual power-supply configuration with two peak detector circuits before further revisions made it more compact.

With regards to coding, there was debate about when to use digital logic versus ADC as triggering inputs. Although digital logic was more convenient as more pins could support external interrupts, we found some hardware, such as the perimeter detector, could not effectively send distinct high and low signals. The ADC was more accurate, but was more difficult to work with due to the limited amount of pins that could support ADC readings.

3.5 Solution Evaluation

- We chose the LPC824 microcontroller system out of the 5 microcontroller options because we found these advantages:
    - It has atomic, 32 bit variables which avoids complexity with interrupts, unlike the PIC32.
    - It is a small, 10 pin wide microcontroller, allowing extra space for other components.
    - It has a fast clock frequency which allowed us to run the motors faster with the microprocessor still keeping up.
- In the metal detector circuit we decided to use 10 nF for capacitor 1, 100 nF for capacitor 2, and 100 Ohms for the resistor to produce the steepest slope and approach an idealized square wave. This allowed us to observe the maximum frequency change.

3.6 Safety/Professionalism

Throughout the project, we always followed the safety requirements for working in a lab and made sure we took the appropriate precautions before testing our hardware. Each team member spent approximately equal amounts of time working on this project and had mutual respect for each other. We were organized and cooperated on project work, making good use of the time we had for it. All members followed the Code of Ethics guidelines, and at no point during the project did we plagiarize another team's work.

3.7 Detailed Design

Hardware Setup

3.7.1 Full Circuit Description

The circuits for this project are broken up into four main sections with three independent voltage sources. The first and second sections of the circuit control either a motor, a servo, or the electromagnet and is powered by the independent 6V AA battery power supply which is isolated from the precision hardware circuitry through an optocoupler. The third section is a colpitts oscillator with variable frequency depending on whether or not any metallic objects are near the inductor, powered by a 5V CMOS logic inverter. The fourth section is a combined circuit on the main breadboard (tank circuit, amplifier, and peak detector) using the 5V power supply and outside the perimeter (555 timer frequency generator) using a 9V supply to establish a perimeter circuit.

3.7.2 Locomotion

The robot moves using a pair of 7.25cm diameter wheels and a roller-ball to reduce drag. The wheels are driven by a pair of Solarbotics GM8 motors, individually controlled with their own MOSFET H-bridges (see Figure 1.1). The H-bridges are powered by 6V with a 3.3V signal that is amplified and isolated from the microcontroller through an optocoupler. The optocoupler uses 100Ω resistors on the input LEDs to achieve easy saturation, and 100kΩ

pullup resistors on the emitter, greatly reducing the current to ensure the output signals are correspondingly amplified to match the 6V high. The electric isolation serves to prevent any noise the motors generate on the 6V supply from interfering with the precision circuits on the 5V and 3.3V supplies.

3.7.3 Electromagnet Servo Arm

The arm uses an electromagnet and micro servos to pick up coins.

Operation:

When a coin is detected by the metal detector, the servo motor moves the arm from neutral position (base forward and electromagnet up) down to a lowered position, to get close to the coin. The electromagnet is then activated through GPIO_B11 (Physical Pin 12) to attract and collect the coin, and the arm then sweeps the area in front of the metal detector. Then the arm moves to the coin collecting area to deactivate the electromagnet and drop the coin. Servo motor finally turns the arm back to the neutral position.

3.7.4 Metal Detector

The metal detector is a colpitts oscillator with the inductor attached to the bottom of the robot at the front.

Operation:

Metal affects the magnetic field around the inductor, proportional to distance. The change in the inductors magnetic field affects it's inductance and thus changes the frequency of the colpitts oscillator. The change in frequency is measured by the microcontroller sampling the signal at the clock frequency of 330Mhz.

The colpitts oscillator uses a 10nF and a 100nF capacitor, which was chosen to establish an approximately 60kHz oscillation frequency. This was done to ensure the processor could measure small changes in frequency, as frequencies greater than 150 kHz, the next most

convenient frequency we could implement with only a pair of capacitors, were not as consistent. See section 3.7 for software details.

Other Notes:

The colpitts The BO230XS blaster port, used to power and upload code to the microcontroller, has built-in power regulation to protect connected laptops from being overloaded. However, it caused voltage oscillations that affected the frequency of the Colpitts oscillator, causing the frequency to shrink and grow by 2.0kHz, an unacceptable variation as the minimum frequency change we needed to detect was 0.5 kHz for detecting a dime.

3.7.5 Perimeter Detector

The perimeter detector circuit consists of two perpendicular tank circuits, a 1 mH inductor in parallel with a 100nF capacitor. The output of each tank circuit is independently amplified using a non-inverting op-amp with a gain of about 200 times intended to reach saturation of 5V when the tank circuit is directly above the perimeter wire. The amplified signals are then combined and fed through a peak detector circuit, a diode in series with the parallel of a 1KΩ resistor and a 220μF capacitor, to generate a DC signal that can be interpreted by the microcontroller ADC on GPIO pin 14 (physical pin 20). The other part of the perimeter circuit consists of a 555 timer in astable mode set to operate at 14.3KHz. This was fairly close to the resonance frequency of the tank circuit which was around 15KHz. For the specific layout of these circuits, please refer to the circuit diagrams (see figures 1.4 and 1.5).

3.7.6 Kinetic Shock Detector

The kinetic shock detector uses a CEM-1203 magnetic buzzer as a sensor to detect coin dropped in the collection bucket. The buzzer is attached flat against the bucket, then detects the hard shocks of a coin landing in the metal bucket. the output of the buzzer is about 250mV high when a dime is dropped, so the signal is amplified to saturation. Amplification is done using a TL081 op-amp provided with

+3V/-3V power supply using an LMC7660 chip. The final implementation of the circuit lights up an LED using an N-MOSFET controlled by the saturated output of the OP-AMP. time limitations prevented us from further connecting the op-amp to the board to use as an input signal for confirming coin collection.

Software Setup

3.7.7 Main Program

The program initially auto-calibrates the thresh_coin and thresh_perim variables by collecting 3 measurements of each perimeter and coin detection voltage and taking the average of them. These variables are used to find a threshold value to detect when the robot is near the perimeter or a coin. These thresholds are saved in global variables to be more easily accessible in subfunctions. As such, the robot cannot be situated near either a coin or the perimeter at the start or these threshold values will be miscalculated.

The main while loop is run through while there are fewer than 20 coins. In this loop, it sets the servo arm to rest position and motors to move forward fast. Next, it calls the detect_perimeter() and if this function sets the perimeter flag, the robot turns around and the perimeter flag is cleared. If the search_coin() functions returns 1, then get_coin() function is executed and its return value (1 or 0) is added to the total count of coins and the coin detection is recalibrated.

3.7.8 detect_perimeter() and ADC

A hardware solution to achieve the necessary threshold for a digital GPIO input here was not found to be possible. As such, ADC was necessary here. Sample code was used to copy ADC_Calibration() and as a basis for InitADC() and detect_perimeter. Pin 20 (ADC 2) was initialised as ADC both in ConfigPins() and InitADC(). In detect_perimeter(), a value is

converted from the ADC and compared to the perimeter threshold. Because a false negative for perimeter detection is disastrous, the new value for the perimeter flag is orred with the previous value; therefore the only way to clear the perimeter flag is to turn around.

3.7.9 search_coin()

The function search_coin() waits until the voltage read from the square wave is low, then starts by setting the long_time variable to 0 and records the time passed between 17 periods, and stores that time in the long_time variable. If the index variable received from the function call is equal to 0, then this means that the robot has just started and it is autocalibrating the coin detection mechanism by getting the time value to set as a threshold. If it is not equal to 0, this means that the auto-calibration is done and that the program is in the while loop, therefore it compares the time read to the threshold time, and if the time is lower than the threshold time, it returns 1, indicating a coin is detected in range.

3.7.10 get_coin()

The function get_coin() handles the broad algorithm of the coin pickup. It stops the robot from moving forward, double checks that there is a coin because a false positive here is disastrous and executes sweep_coin(). If the double check fails, the function abruptly aborts and returns 0. If there is no unexpected hitches, the program will return 1. The return value is added to the overall coin count in the main code.

3.7.11 sweep_coin()

The function sweep_coin() handles the movements of the servo arm to pick up a coin. The robot stops moving by calling the stop_moving() function while it is picking up a coin. In order to pick up a coin, the electromagnet is turned on and then the servo_arm and the servo_base angles change accordingly with the angle received from the get_pwm() function.

When the coin is picked up, it is put to the robot's bag by turning off the electromagnet. After a coin is picked, servo_arm and servo_base return back to their rest positions.

3.7.12 Interrupt Service Routine

A state configurable timer (SCT) was used as our interrupt service routine. Any variable that is used in the ISR is defined as volatile and long and complex lines of code are not used. The ISR is mainly used to keep track of the time passed by using the period, pwm_count and pwm_servo variables. pwm_count is reset to 0 after it reaches 100 and increments period by one, while pwm_servo is reset to 0 after it reaches 600. The reason for this is that pwm_count creates a 300 Hz square wave whereas pwm_servo creates one at 50 Hz. The other use of the ISR is to set the left and right motors, servos and voltage_coin.

3.8 Solution Assessment

It was established that for there to be a successful solution, it was essential for there to be no false negatives for perimeter detection and no false positives for coin detection. The sensitivity of each detection mechanism was adjusted and ultimately, the design succeeded in that these two parameters were met. To guarantee this outcome, we attempted a series of tests for the robot approaching the perimeter from all angles while measuring the voltage increase with the oscilloscope and also for running over a variety of coins. The search_coin( ) function determines whether the period of the metal detection circuitry had decreased by a sufficient amount from the rest state. This amount was changed experimentally, with the results in Table 3.8.1. There was also concern that the motors would create enough noise to inhibit reliable readings of these measurements; however we were able to amplify these signals well above any noise level and they could be relied upon for accuracy.

Table 3.8.1: Final results of tests conducted for coin detection

threshold value minus rest state value	coins able to be detected
0	constant false positives
1	detect toonie when it's too far away to pick up
2	cannot detect dime unless it’s centered (this was deemed acceptable)
3	cannot detect dime
4	can only detect larger coins

4. Lifelong Learning

In this project, we had to work with a new microcontroller, since the use of the EFM8 microcontroller was restricted in the project outline document [1]. None of us were familiar with the LPC824 microcontroller before, but the sample codes provided on canvas [5], the user manual [6] and the datasheet [7] improved our understandings. Moreover, by searching through these resources, we learned how to research more effectively within the time provided. As we had to adapt to many different coding languages and different programming tools throughout CPEN 211, CPSC 259 and the previous project and labs of ELEC 291, the transition to LPC824 was manageable. Understanding the previous lab material, specifically lab 4 [3] which provides a guideline for how to setup the ISR most effectively and lab 6 [4] which is on motor control, significantly helped us complete this project. Furthermore, we were more organized and time efficient as a team compared to the last project, which points out the teamwork and collaboration skills we obtained during this course. One important catch we made from this project was that some pins do not have programmable

pull-up/pull-down resistors (these are usually called “true open drain pins”). In our case, physical pins 9 and 10 that are connected to our right motors did not have programmable pull-up/pull-down resistors and using their GPIO’s the same way we did with other pins did not seem to work. To tackle this problem, external pull-up/pull-down resistors were added to the breadboard to configure these pins properly.

5. Conclusions

Overall, our final design was able to collect an assortment of 20 coins ranging from a nickel to a toonie in value randomly distributed within a 1.5m^2 area. It operates autonomously by detecting the perimeter wire and turning around, and by detecting metal objects it rolls over and picking them up. We encountered several problems during the design and testing phase of our robot mainly linked to hardware restrictions. We were greatly limited by the amount of space available on our breadboard which prevented us from being able to implement some bonus features, and we were additionally required to add another half breadboard to hold the H-Bridge circuit for motor control. Further hardware issues arose from noise generated by the switching mode power supply that showed up in both the amplified perimeter detector signal and changes in the colpitts oscillator frequency. This was a source of much confusion for us because we had assumed that the supply was stable and DC, so it took us a long time to find out that the source of the noise was the power supply we were using. Because our final design was battery powered, this issue was solved on its own. Our team spent about 44 hours per person working on this project.

References

[1] Jesus Calvino-Fraga, Project 2 – Coin Picking Robot, 2006-2019 (PDF)

https://canvas.ubc.ca/courses/19445/files/4093209?module_item_id=1015179

[2] Jesus Calvino-Fraga, Project 2 – Coin Picking Robot, March 25, 2019 (Lecture slides)

https://canvas.ubc.ca/courses/19445/files/4268304?module_item_id=1082245

[3] Jesus Calvino-Fraga, "Lab 4: Capacitance Meter and Photo Electric Heart Rate Monitor 2019 (slides): ELEC 291-292 201 Electrical Engineering Design Studio I." [Online]. Available:

https://canvas.ubc.ca/courses/19445/files/2154803?module_item_id=1008410

[4] Jesus Calvino-Fraga, "Lab 6: Motor Control 2019 (slides): ELEC 291-292 201 Electrical Engineering Design Studio I." [Online]. Available: https://canvas.ubc.ca/courses/19445/files/2154757?module_item_id=1014577

[5] Jesus Calvino-Fraga, "LPC824.zip: ELEC 291-292 201 Electrical Engineering Design Studio I." [Online]. Available:

https://canvas.ubc.ca/courses/19445/files/3786732?module_item_id=1014827.

[Accessed: 30-March-2019].

[6] NXP Semiconductors, "LPC 82x User manual", 2016.

Available: http://ds.arm.com/media/resources/db/chip/nxp/lpc82x/UM10800.pdf [Accessed: 30-March-2019]

[7] NXP Semiconductors, "LPC 82x Product datasheet", 2018.

Available: https://www.nxp.com/docs/en/data-sheet/LPC82X.pdf [Accessed: 30-March-2019]

[8]J. Calvino-Fraga,

“http://courses.ece.ubc.ca/281/2017/ELEC291_292_Project2_Coin_Picking_Robot_2019.”

Vancouver, Mar-2019. Available:

http://courses.ece.ubc.ca/281/2017/ELEC291_292_Project2_Coin_Picking_Robot_2019.mp4

Bibliography

“Single Supply Op Amp Design.” [Online]. Available: http://www.swarthmore.edu/NatSci/echeeve1/Ref/SingleSupply/SingleSupply.html. [Accessed: 05-Apr-2019].

George, “Power Supply Design: Switch-Mode vs. Linear,” *Simply Smarter Circuitry Blog*, 23-Oct-2014.

Appendices

Appendix I: Microprocessor List

1. LPC824
2. MSP430
3. PIC32
4. STM32F051
5. ATmega328

Appendix II: Software main.c

```
// final program
//some code taken from and coded with reference to sample code
including PrintADCeff2 and Freq_Gen. Makefile framework from
Freq_Gen was used.
/*
PIN ASSIGNMENTS
physical pin    pio assignment  thing                input/output
1          GPIO_B23        MOTOR_L1             OUTPUT
2          GPIO_B17        MOTOR_L2             OUTPUT
9          GPIO_B10        MOTOR_R1             OUTPUT
10         GPIO_B11        MOTOR_R2             OUTPUT
```

```
12          GPIO_B1                EMAG                    OUTPUT
3           GPIO_B13         COIN_DETECT                   INPUT
7           GPIO_B3                SERVO_1                       OUTPUT
8           GPIO_B2                SERVO_2                       OUTPUT
13          GPIO_B9/ADC_2    PERIMETER_DETECT1     INPUT
*/

#include "lpc824.h"
#include "serial.h"

#define SYSTEM_CLK 30000000L
#define DEFAULT_F 15000L

#define COINS_TOTAL 20

#define COIN_DETECT GPIO_B13    //coin detect - 3
#define MOTOR_L1 GPIO_B23  //motor 1 left - 1
#define MOTOR_L2 GPIO_B17  //motor 2 left - 2
#define MOTOR_R1 GPIO_B10  //motor 1 right - 9
#define MOTOR_R2 GPIO_B11  //motor 2 right - 10
#define SERVO_1 GPIO_B3    //servo 1  - 7
#define SERVO_2 GPIO_B2    //servo 2  - 8
#define EMAG GPIO_B1 //electromagnet - 12

#define FAST 90 //speed
#define NORMAL 50    //speed

#define TIME1_CONST 100

//positions in the servo sweep
#define SERVO1_REST 180
#define SERVO2_REST 0
#define SERVO1_A 50
#define SERVO2_A 180
#define SERVO1_B 170
#define SERVO2_B 10
#define SERVO1_C 22
#define SERVO2_C 50

/*global variables and volatile variables*/
//motors:
//left motor:
volatile int moL_1 = 0 , moL_2 = 0;
//right motor:
volatile int moR_1 = 0 , moR_2 = 0;
//servos:
```

```
volatile int servo_base = 0 , servo_arm = 0;
//flags. perimeter is set in the interrupt, coin is set in
search_coin() function
volatile int perimeter_flag = 0;
//time related stuff:
volatile int  pwm_count = 0 , period = 0, pwm_servo = 0,
long_time=0;
//thresholds
volatile int thresh_coin, thresh_perim;
volatile int voltage_coin = 0;

// Configure the pins as outputs
void ConfigPins(void)
{
     //enable switch matrix clock (page 41, 42)
     SYSCON_SYSAHBCLKCTRL |= BIT7;

     //turn off stupid extra features (page 87)
     SWM_PINENABLE0 |= BIT4 | BIT5 | BIT11 | BIT12 | BIT23 | BIT6
| BIT7;
     SWM_PINENABLE0 &=~BIT15; // Enable the ADC function on PIO_14
page 88

     //kill the clock
     SYSCON_SYSAHBCLKCTRL &= ~BIT7;

     GPIO_DIR0 |= BIT23; //motor
     GPIO_DIR0 |= BIT17;  //motor
     GPIO_DIR0 |= BIT11;  //motor
     GPIO_DIR0 |= BIT10;  //motor

     GPIO_DIR0 |= BIT2;   //servo
     GPIO_DIR0 |= BIT3;   //servo

     GPIO_DIR0 &= ~(BIT13);     //coin detection

     GPIO_DIR0 |= BIT1;   //emag
}


/* Start ADC calibration */
void ADC_Calibration(void)
{
     unsigned int saved_ADC_CTRL;
```

```
	// Follow the instructions from the user manual (21.3.4
Hardware self-calibration)

	//To calibrate the ADC follow these steps:

	//1. Save the current contents of the ADC CTRL register if
different from default.
	saved_ADC_CTRL=ADC_CTRL;
	// 2. In a single write to the ADC CTRL register, do the
following to start the
	//    calibration:
	//    - Set the calibration mode bit CALMODE.
	//    - Write a divider value to the CLKDIV bit field that
divides the system
	//      clock to yield an ADC clock of about 500 kHz.
	//    - Clear the LPWR bit.
	ADC_CTRL = BIT30 | ((300/5)-1); // BIT30=CALMODE,
BIT10=LPWRMODE, BIT7:0=CLKDIV
//    ADC_CTRL &= ~BIT10;
	// 3. Poll the CALMODE bit until it is cleared.
	while(ADC_CTRL&BIT30);
	// Before launching a new A/D conversion, restore the
contents of the CTRL
	// register or use the default values.
	ADC_CTRL=saved_ADC_CTRL;
}

void InitADC(void)
{
	// Will use Pin 1 (TSSOP-20 package) or PIO_23 for ADC.
	// This corresponds to ADC Channel 3.  Also connect the
	// VREFN pin (pin 17 of TSSOP-20) to GND, and VREFP the
	// pin (pin 17 of TSSOP-20) to VDD (3.3V).

	SYSCON_PDRUNCFG &= ~BIT4; // Power up the ADC
	SYSCON_SYSAHBCLKCTRL |= BIT24;// Start the ADC Clocks
	ADC_Calibration();
	ADC_SEQA_CTRL &= ~BIT31; // Ensure SEQA_ENA is disabled
before making changes

	ADC_CTRL =1;// Set the ADC Clock divisor
	ADC_SEQA_CTRL |= BIT2 ; // Select Channel 2  page 88
	ADC_SEQA_CTRL |= BIT31 + BIT18; // Set SEQA and Trigger
polarity bits

	//disable pull up / pull down resistors
```

```
      PIO0_13 |= BIT3;
      PIO0_14 |= BIT3;
}


void InitTimer(void)
{
      SCTIMER_CTRL |= BIT2; // halt SCTimer

    // Assign a pin to the timer.
    // Assign GPIO_14 to SCT_OUT0_O
      SWM_PINASSIGN7 &= 0x00ffffff;
      SWM_PINASSIGN7 |= (14 << 24);

      SYSCON_SYSAHBCLKCTRL |= BIT8; // Turn on SCTimer
      SYSCON_PRESETCTRL |=  BIT8; // Clear the reset SCT control

      SCTIMER_CONFIG |= BIT0; // Unified 32 bit counter
      SCTIMER_MATCH0 = SYSTEM_CLK/(DEFAULT_F*2L); // Set delay
period
      SCTIMER_MATCHREL0 = SYSTEM_CLK/(DEFAULT_F*2L);
      SCTIMER_EV0_STATE = BIT0;  // Event 0 pushes us into state 0
      // Event 0 configuration:
      // Event occurs on match of MATCH0, new state is 1
      SCTIMER_EV0_CTRL =  BIT12 + BIT14 + BIT15;
      // State 1 configuration
      SCTIMER_EV1_STATE = BIT1;  // Event 1 pushes us into state 1
      // Event 1 configuration
      // Event occurs on MATCH0, new state is 0
      SCTIMER_EV1_CTRL =  BIT12 + BIT14;
      // OUT0 is set by event 0
      SCTIMER_OUT0_SET = BIT0;
      // OUT1 is cleared by event 1
      SCTIMER_OUT0_CLR = BIT1;
      // Processing events 0 and 1
      SCTIMER_LIMIT_L = BIT0 + BIT1;
      // Remove halt on SCTimer
      SCTIMER_CTRL &= ~BIT2;

      SCTIMER_EVEN = 0x01; //Interrupt on event 0
      NVIC_ISER0|=BIT9; // Enable SCT interrupts in NVIC
}

void Reload_SCTIMER (unsigned long Dly)
{
      SCTIMER_CTRL |= BIT2; // halt SCTimer
```

```
	SCTIMER_MATCH0 = Dly; // Set delay period
	SCTIMER_MATCHREL0 = Dly;
	SCTIMER_COUNT = 0;
	SCTIMER_CTRL &= ~BIT2;     // Remove halt on SCTimer
}

//ISR
void STC_IRQ_Handler(void)
{
	SCTIMER_EVFLAG = 0x01; // Clear interrupt flag
	pwm_count++;
	pwm_servo++;
	long_time++;
	//motors
	if(pwm_count>=100){
		pwm_count=0;
		period++;
	}
	//delay_time
	if(period>=TIME1_CONST){
		period = 0;
	}
	//search_coin
	if(long_time>=10000000000000){
		long_time = 0;
	}
	//servo
	if (pwm_servo>=600){
		pwm_servo = 0;
	}
	//left motor - 300Hz
	MOTOR_L1 = (pwm_count>moL_1)?0:1;
	MOTOR_L2 = (pwm_count>moL_2)?0:1;
	//right motor - 300Hz
	MOTOR_R1 = (pwm_count>moR_1)?0:1;
	MOTOR_R2 = (pwm_count>moR_2)?0:1;
	//servos - *6 because they must operate and 50Hz
	SERVO_1 = (pwm_servo>servo_base*6)?1:0;
	SERVO_2 = (pwm_servo>servo_arm*6)?1:0;
	//coin voltage
	voltage_coin = COIN_DETECT;
}

//use adc to determine perimeter flag
void detect_perimeter (void){
```

```
	ADC_SEQA_CTRL |= BIT26; // Start a conversion:. page 330.
bit26 is always start
	//from first pin (pin 20 - PIO14)
	while( ( (ADC_SEQA_GDAT) & BIT31)==0); // Wait for data
valid. table 281 page 330
	perimeter_flag = ( ( (ADC_SEQA_GDAT >> 4) & 0xfff)    >
thresh_perim    )? 1: 0; //perimeter_flag;

	return;
}

//delays for a probably short amount of time - need to test
further
//set time0 to number <100 and it will set a delay for that long
in deciseconds(?)
void delay_time(int time0){
	period = 0;
	while (period != time0*TIME1_CONST/100){
		//time wasting loop
	}
	return;
}

void move_forward(int speed){
	//left motor
	moL_1=speed;
	moL_2=0;
	//right motor
	moR_1=speed;
	moR_2=0;
	return;
}
void move_backward(int speed){
	//left motor
	moL_1=0;
	moL_2=speed;
	//right motor
	moR_1=0;
	moR_2=speed;
	return;
}
void stop_moving(void){
	//left motor
    moL_1=0;
    moL_2=0;
	//right motor
```

```
    moR_1=0;
    moR_2=0;
    return;
}

void turn_around(void){
     //turn wheels different directions
     //left motor:
     moL_1=FAST;
     moL_2=0;
     //right motor:
     moR_1=0;
     moR_2=FAST;
     delay_time(50); //time spent turning is slightly randomised
     delay_time(50);
     return;
}

int search_coin(int index){

     int time, i;
     while(voltage_coin==0);    //in case function starts halfway
through v=low
     while(voltage_coin > 0);   //wait until v is no longer high
     //we're now at falling edge of signal -> record time
     long_time=0;
     for(i=0;i<17;i++){    //17 was experimentally found to be
sufficiently large to get a reliable reading
          while(voltage_coin==0);    //wait until voltage is no
longer low
          while(voltage_coin>0);//record time again
     }
     time = long_time;     //record time taken
     if(index != 0){ //compare to threshold
          if(time < thresh_coin)  //if there's a coin (it might
have to be <= not >= need to check if its period or frequency that
increases)
               return 1;
          else //if no coin
               return 0;
     }
     else{ //establish threshold
          return time;
     }
}
```

```
//for servo motors - get pwm out of angle
int get_pwm(int angle){
      int pwm;
      pwm= ((double)angle/18.0 + 2.75);
      if (pwm < 1){
            pwm= 1;
      }
      if (pwm > 9){
            pwm= 9;
      }
      return pwm;
}

//arm will move to pick up coin
void sweep_coin(void){
      int angle;
      stop_moving();
      servo_base = get_pwm(SERVO1_REST);
      //    delay_time(5);
      servo_arm = get_pwm(SERVO2_REST);
      //move servo motor
      servo_base = get_pwm(SERVO1_A);
      delay_time(90);
      servo_arm = get_pwm(SERVO2_A);
      delay_time(90);
      EMAG = 1;
      delay_time(90);

      for (angle=SERVO1_A; angle<SERVO1_B ; angle++){
            servo_base=get_pwm(angle);
            delay_time(1);
      }
      for (angle=SERVO2_A; angle>SERVO2_B ; angle--){
            servo_arm = get_pwm(angle);
            delay_time(1);
      }

      for (angle=SERVO1_B; angle>SERVO1_C; angle--){
            servo_base = get_pwm(angle);
            delay_time(1);
      }
      for (angle=SERVO2_B; angle<SERVO2_C ; angle++){
            servo_arm = get_pwm(angle);
            delay_time(2);
      }
      //in position for drop
```

```
	delay_time(90);
	EMAG = 0;
	delay_time(90);
	servo_base = get_pwm(SERVO1_REST);
	servo_arm = get_pwm(SERVO2_REST);
	return;
}

int get_coin(void){
	stop_moving();
	//double check that there is actually a coin
	delay_time(90);
	if ( !search_coin(1) ){
		return 0;
	}

	//retreat more for set time period to go from coin detection
to coin pickup spot
	move_backward(NORMAL);
	delay_time(90); //this will need to be changed experimentally
	stop_moving();
	//perform the coin pickup
	sweep_coin();
	//return coin_check;
	return 1;
}

void arm_default(void){
	servo_arm = get_pwm(SERVO2_REST);
	servo_base = get_pwm(SERVO1_REST);
	EMAG = 0;
	return;
}

void main(void)
{
	ConfigPins();
	InitTimer();
	initUART(115200);
	InitADC();
	enable_interrupts();

	int coins = 0;
	stop_moving();
	arm_default();
	delay_time(90);
```

```
	//calibrate thresholds
	thresh_coin = search_coin(0);
	ADC_SEQA_CTRL |= BIT26; // Start a conversion:. page 330.
bit26 is always start
	while( ( (ADC_SEQA_GDAT) & BIT31)==0); // Wait for data
valid. table 281 page 330
	thresh_perim += ((ADC_SEQA_GDAT) >> 4) & 0xfff; //try setting
it equal to perim1 if things go wrong
	delay_time(90);
	thresh_coin += search_coin(0);
	ADC_SEQA_CTRL |= BIT26; // Start a conversion:. page 330.
bit26 is always start
	while( ( (ADC_SEQA_GDAT) & BIT31)==0); // Wait for data
valid. table 281 page 330
	thresh_perim += (ADC_SEQA_GDAT >> 4) & 0xfff; //try setting
it equal to perim1 if things go wrong
	delay_time(90);
	thresh_coin += search_coin(0);
	thresh_coin /= 3;
	ADC_SEQA_CTRL |= BIT26; // Start a conversion:. page 330.
bit26 is always start
	while( ( (ADC_SEQA_GDAT) & BIT31)==0); // Wait for data
valid. table 281 page 330
	thresh_perim += (ADC_SEQA_GDAT >> 4) & 0xfff; //try setting
it equal to perim1 if things go wrong
	thresh_perim /= 3;
	thresh_coin -=2;
	thresh_perim +=200;
	perimeter_flag = 0;

	while(coins < COINS_TOTAL) {
		arm_default();
		move_forward(FAST);
		detect_perimeter();
		if(perimeter_flag){
			turn_around();
			perimeter_flag = 0;
		}
		if(search_coin(1)) {
			coins += get_coin();
			//recalibrate coin detection
			thresh_coin = search_coin(0);
			delay_time(90);
			thresh_coin += search_coin(0);
			delay_time(90);
```

```
                thresh_coin += search_coin(0);
                thresh_coin /= 3;
                thresh_coin -=2;
            }
        }
        stop_moving();
        while (1){
            arm_default();
        }
}
```

Appendix III: Hardware Final Design Pictures