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 #: A7

Student Number	Student Name	% Points	Signature
	Adrian, Clarence		
	Gao, Jian		
	Lee, Matthew		
	Luo, Edward		
	Windsor-Lewis, JoJo		
	Yu, Jacky		

Date of Submission: April 4, 2019

Table of Contents

1.0 Introduction

This project report provides documentation on the architecture and design of an autonomous coin picking robot prototype by students of Electrical Engineering at the University of British Columbia (UBC). Entailed is a detailed account of our design process, including idea generation, data collection, problem specification, solution assessment, and full illustrations of our design and how we applied appropriate engineering knowledge to do so. This document is written with the criteria of being addressed towards a reasonably expert reader in the field of engineering such as a project manager.

1.1 Design Objective

The objective of the project entails the software and hardware design of an autonomous coin-picking robot. Design stages include the building, programming, and testing of the prototype, along with the implementation of extra functionalities, with an allocated work time of a few weeks. Below are the further detailed requirements for the project:

- Understand the use of PWM (pulse-width modulation) to control servo motors
- Successfully build a magnet using concepts in electromagnetism
- Build a peak detector that can sensitively detect external currents
- Constructing a microcontroller board (soldering, designing, and testing)
- Creation and maintenance of makefiles to compile, link, and load code onto hardware

The list above is a summarized requirement list obtained from the course-provided lab manual which covers each requirement in more detail (see Appendix A1).

1.2 Specifications

1.2.1 Full Parts List

A detailed parts list can be viewed in in the appendices (see Appendix A2).

1.2.2 Specifications

Microcontroller: STM32F051 ARM Cortex-M

Specific diagrams on how we built the microcontroller can be found in Appendix B1, and the considerations of why we chose the STM32 in Section 3.6

1.2.3 Software Specifications and Functionality

Furthermore, we implemented the following basic features for the robot:

1. Fully Battery-Powered Robot Movement (forward, backward, left, and right)
    a. Individual and dual wheel movement (CW/CCW) and speed control
    b. Full rotational movement (left and right turns)
2. Metal Detection
    a. Accurately detect all 5 various Canadian coins and, with minimal alteration, detect other metals (keys, etc)
3. Perimeter wire detection
    a. Ability to accurately detect wire connected to an AC source (~ 5V AC)
4. Electromagnet activation control
    a. Able to be activated with simple “on” and “off” functions
5. Coin servo control
    a. Complete set of instructions optimally timed to pick up coins in a safely manner with minimal risk of dropping/collision
6. Optional Bluetooth control module

2.0 Investigation

2.1 Idea Generation

We started our design process by identifying the stakeholders for this project and their needs. We then formed our design requirements through this process, namely building a robot which moved quickly and smoothly and used the highest frequency possible to transmit data. Once we had our requirements, we began generating ideas which satisfied them.

2.2 Investigation Design

A range of methods including online research and in-lab testing was used to gather data throughout the entire design process. Prior to the start of the project, we conducted in-depth online research on the components we decided for the project, particularly on the STM32 Microcontroller. Furthermore, data sheets obtained from official websites were consulted for each component to gain further understanding and to ensure the the components are used appropriately. From this information, we formulated a detailed design plan to conduct for the following weeks. A detailed list of the apparatus available to us can be seen in Section 1.2 of the project document.

2.3 Data Collection

As mentioned in Section 2.2, data was mainly collected in two ways: reading datasheets and collecting data using the tools and equipment available in the lab. The subsections below describe the techniques we used to collect data through the different equipment.

2.3.1 Multimeter

The lab oscilloscope was frequently used to visualize the waveform generated in our circuits, which let us know whether our circuits or components actually worked. For example, we used the oscilloscope to check whether the frequency of the square wave generated by the microcontroller was altered when it detects a coin in its vicinity, and if the inductor detected a voltage increase when approaching the perimeter wire.

2.3.2 Lab Oscilloscope

The lab oscilloscope was frequently used to visualize the waveform generated in our circuits, which let us know whether our circuits or components were functional. For example, we used the oscilloscope to check whether the frequency of the square wave generated by the microcontroller was altered when it detects a coin in its vicinity, and if the inductor detected a voltage increase when approaching the perimeter wire.

2.3.3 Power Supply

Although the robot has to be fully powered by batteries, during the design period, we used the XPH 35-4T Triple DC Power Supply available in the lab to provide power to our robot to prevent depleting the power from our battery source. Otherwise, the microcontroller was powered by serial port connections when linking code from our local computers.

2.3.4 Putty

Another method of obtaining data was by directly printing values received/produced by the microcontroller to PuTTy when a serial port connection can be established between the microcontroller and a local computer. This allowed us to print any of the values we needed to observe, as well as flags whenever there was outcome that needed to be investigated.

2.4 Data Synthesis

Values for the different components of the robot were obtained through the methods above, and synthesized in real-time to ensure that the voltages, currents, and power received/produced can be observed and tweaked to the standards required by the robot's components.

In order to ensure that all the individual components were working, several different test codes were used to test each component, and their values were taken down before compiling. Some sample tests can be seen in Appendix F3, and results in Appendix E. Otherwise, direct observation of the robot's physical movements and functions were conducted.

2.5 Analysis of Results

We relied on direct testing of the robot's functions to appraise the validity of the data values we took (when the device observably runs its instructions with success 100% of the time). Furthermore, we continually conducted tests and experimented to find the most optimal values (i.e. threshold frequency and ADC voltage), which also provided further validation. Particularly, the threshold frequency and voltage for both the coin and wire detectors were continually iterated to finally find what was most optimal. More detailed testing data can be referred to in Section 3.8 of the document or in the appendices in Appendix E.

When taking values using the lab equipment, we also tried to ensure their correctness by using different equipment on different lab benches and comparing them with each other. This was quite essential as some of the lab equipment were seldom faulty or had some margin of error in them.

3.0 Design

3.1 Use of Process

We applied knowledge of general design processes obtained from previous lectures and past courses to design our circuit. From the lab manuals provided, we identified our needs and stakeholders, and clearly established the purpose and requirements of the project. Afterwards, we constructed a detailed design plan and started construction of the body, circuits, and program with continued testing at appropriate intervals.

3.2 Need and Constraint Identification

Our main stakeholders are Dr. Jesús Calviño-Fraga, the course instructor, the TAs, and all of us who will be using the robot for our project.

As per the requirements mentioned in Section 1.1 of this document, we needed to ensure that the robot is capable of detecting all current Canadian coins and can successfully pick them up using a magnet while operating within a wire boundary. Data on the detailed constraints of each component were obtained from online datasheets and the provided lab manuals. Furthermore, the main constraint was our time span of just a few weeks to fully complete the robot and its report.

3.3 Problem Specification

In order to exceed the requirements set for us by the lab manuals and resources, we needed to make the robot as reliable and specified a few additional design requirements to focus on during the construction of our robot, including:

- Ability to detect coins with 99.9% accuracy, and not accidentally pick up empty space/other random objects that do not represent the coins.

- Have adjustable speed and complete flexibility in its movement
- Be visually pleasant in circuitry and display clear wire connectivity

3.4 Solution Generation

Once we had a clear idea of the problem specifications, we used brainstorming sessions to create a range of possible solutions. The best contenders were modelled, which allowed for direct comparison between each of the ideas. We focused on designs which were built from modular subsystems, as we felt this would allow team members to work independently and support more efficient collaboration. We then used iterative design processes to revise our design into our final prototype and move forward into the next stage of the project.

Bluetooth Module

Being fully able to operate by itself as a requirement, we decided to add a bit of user interfacing by adding a Bluetooth module (HC-05) that adds further functionality to the robot that is subject to user input. The Bluetooth module adds remote operation, such as go forward, go back, turn left, turn right, and auto mode activation

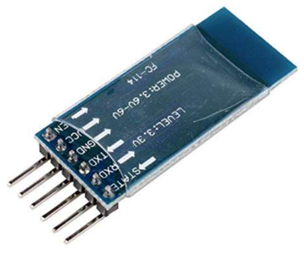


Figure 1: Bluetooth Module

Extra Hardware and Software Features

Furthermore, in addition to our base design, we also added a few extra less-observable features with the purpose of adding functionality or overcoming the problems we encountered, including:

- Stopping and Going. As an attempt to increase accuracy in detecting the coins, we made the robot momentarily 'pause' in its spot when it detects a frequency change in its metal detection. Before completely deducing that a coin has been detected, further values are to be taken as it 'pauses'. Only after a specific degree of certainty will the robot attempt to pick up the object in front of it.
- Indicators. To add to the visual effect, we added blinking LED lights as indicators to show when the robot is turning and the direction in which it is turning.
- Weight Detection (to identify and classify coins)
- PuTTy Analysis. Custom-printed lines of data are being constantly feeded into the serial port connection (if connected) for increased ease with analysis and debugging when needed. Otherwise, there are no visual additions to the robot itself.

3.5 Solution Evaluation

Our robot meets the needs of the stakeholders using motors and electromagnet to pick up all Canadian coins. It is fully battery powered and runs smoothly. It can distinguish the magnetic field of the coins and the perimeter wire, and then perform the tasks corresponding to those two scenarios. The bluetooth module allows us to have another creative choice of controlling the robot. Not only can it be autonomously operated, but we can also give it instructions to operate in ways that we want. Also, the LEDs and the speaker monitor the robot which inform us of which task the robot is doing.

3.6 Detailed Design

Having been given a number of microcontrollers to choose from, our team decided to utilize the STM32F051 based on its reputation as a cost-effective microcontroller intended for 8- and 16-bit applications where 32-bit performances are required [1]. Additionally, the STM32 uses a GCC development environment for ARM, which we are assumptively more familiar with, having worked with ARM products quite frequently before.

3.6.1 Hardware Design

The body of the robot was put together using pre-folded components, assembled and put together using basic screws and bolts.

3.6.2 STM32

In order to have a fully functional programmable STM32, the microcontroller was soldered onto a LQFP32 to DIP32 adapter board. Then, the two 16-pin header connectors are soldered on as well. The full pin connections can be seen in the appendices (Appendix B1).

3.6.3 PWM-Controlled Servo Motors

Attaching the servo motors to the folded aluminum body of the robot were one of the first steps in building the robot. They were each tightly screwed into place. Having done this at a very early stage, we were able to test the motor and wheels by using PWM pulses on an earlier lab. Throughout its design, some iterations needed to be made to adjust the position of the servo motors, particularly the coin-picker. Otherwise, the structural integrity of the servo was kept as close to our initial design (see Appendix E1).

H-Bridge Optocoupler

The four electric servos used to control the wheels of the robot were operated by PWM pulses produced by the STM32 microcontroller. In order to allow the motor to move in opposite directions and at the speed and time desired, an H-Bridge was built using P and N-type MOSFETs that are activated by pulses sent by the STM32 through light sources, triggering photosensitive BJTs (Bipolar Junction Transistors). This process is handled by the LTV-847 Optocoupler in a manner that allows complete control over the motor (see Appendix B2).

Timer Initialization

In order to generate the PWM signals passed on to the motors, timers were initialized. A total of two timers were used (TIM1 and TIM3) to operate the two sets of motors in the robot. Initializing them involved a lot of trial and error altering header files and connecting libraries through the use of makefiles. The blinking LED program was used to run and test the timers. (see Appendix F3). Timer interrupts for both TIM1 and TIM3 needed to be activated through individual bit activations in the NVIC (Nested Vector Interrupt Controller) registers of the STM32 Microcontroller. Both timers were initialized with modes for downcounting, peripheral clock, and global interrupts enabled through setting appropriate bits of the STM32's registers.

Wheel Control Motor Logic

To operate the two GM4 clear servos, timer 1 (TIM1) was initialized with interrupts configured at 1kHz and assuming a system clock of 8000000L. TIM1 was used to constantly check whether the left and right wheels were in clockwise/counterclockwise operation mode, and volatile flag values (i.e. "left_wheel_dir", "right_wheel_dir", "left_wheel", "right_wheel") were introduced into the code to indicate whether the PWM pulse needed to be sent to either of the wheels. Additionally, variables indicating the speed of rotation of either of the wheels were declared and used as a count value limitation that adjusts the width of each pulse in the PWM. Time delays that also depended on TIM1 were used to adjust the time that the PWM signals needed to be sent to the individual motors. A number of functions (“turn_on”, “turn_on”, “left_wheel_on”, “left_wheel_speed_change”, “turnLeft”) were placed in the main body of the code to make the code more intuitive and to ease the operation of the robot during the latter stages of its design.

Coin Picker Motor Logic

Originally, operation of the two MG90S 450 LKY61 servo motors were done using the same STM32 timer (TIM1) as the one we used for the GM4 servos. This introduced numerous problems, as we tried to introduce different counters to compensate for the need for different frequency values in operating the two different type of servo motors. Attempting to operate two servos with one timer seemed to be too much load for the timer itself to run through before its next interrupt. Multiple tests and trials led us to the conclusion that we needed to initialize a second timer with the sole function of operating the LKY61 servos. Thus, we brought about a second timer (TIM3), configured with an interrupt frequency of 100kHz and a separate delay function (“delay_ms”). A separate function that initializes the starting position of the two servos was also created to increase convenience in testing and also as a safety feature in the robot’s activation.

3.6.4 Metal Detector

Colpitts Oscillator

$$f_0 = \frac{1}{2\pi\sqrt{L\frac{C_1 C_2}{C_1+C_2}}}.$$

The metal detector utilized concepts in electromagnetism by using a frequency oscillator that is affected by external magnetic material. The full circuit diagram for the oscillator can be seen in the appendices (see Appendix B3). Essentially, the Colpitts Oscillator is a frequency generator made up of a 'NOT' CMOS inverter, an inductor, and capacitors that are used to adjust the frequency. Calculations are done through the use of the formula above.

Period Detector

Following an increase in the frequency of the generated wave as a result of an external magnetic material, the period detection function receives this change as an indication that a coin might be detected. Experimentally, it was found that the frequency detected by the microcontroller often fluctuates as the robot moves and enters different terrain due to the varying external fields around it; a small change in the surrounding field easily caused false detection. Therefore, several precautions had to be put into place to prevent this.

As soon as the robot detects one positive signal, the robot is made to momentarily pause to take another set of readings to confirm whether the first signal was accurate. In order to do this, an exact 5 extra readings, separated by 0.01s intervals, are taken. If at least 4 out of the 5 extra readings confirm a positive detection, the robot will attempt to pick up a coin. Otherwise, the robot continues moving until another event interruption.

It was decided that 5 extra readings provided optimal results as it produced the highest rate of successful outcomes. Through repeated experimentation, two threshold frequencies were found that statistically produced the best results. This data is provided in table 1 below.

	First Positive Detection	**Subsequent Positive Detection(s)**
Threshold Frequency	64145	64650
Number of Values to Take	1	5
Required Positive Values	1 (100%)	4 (80%)

Table 1 - Coin Detection Optimal Detection Data

3.6.5 Wire Detector

Peak Detector

Two peak detectors were fitted onto the circuit (see Appendix B4) to detect the peak voltage through the inductor placed at the bottom of the robot. Through concepts in basic electromagnetism, the peak detectors will work as wire detectors as they will vary rapidly when coming into close proximity with an external AC current. With this in mind, the rapid fluctuations need to be filtered, before being amplified and sent to the microcontroller as data.

ADC Converter

The signals passed on by the peak detectors were received through the STM32's internal ADC (analog-to-digital converter). Pins 14 and 15, as well as channel 9 of the ADC, were initialized to work in analog mode to support the input from the peak detectors. Readings from the ADC were done as frequent as possible and the 12-bits received are converted, in real-time, to voltage values for easier comparison. Threshold values were found and set to be about by experimentation. Repeated testing and experimentation yielded a final value of 1.5V to be the appropriate threshold. Following wire detection, the robot's commands are to reverse for a distance before rotating at an angle of about 160 degrees.

3.6.6 Compiled Program

The many unsuccessful attempts to link and initialize each of the component's specific functions through makefiles led the team to decide on trying to fit the bulk of the important functions into one *C* file extension type. A majority of the functions used to operate the device were included in the main file, "*main.c*", as they heavily relied on each other and the timers initialized there. The remaining functions were declared and defined in subsidiary *C* files referred to in "*main.c*".

As a result, the robot is programmed to start moving as soon as the code has been loaded or after it has been reset by the Boot Loader. As soon as it starts moving, frequency values from the Colpitts Oscillator (coin detection) and voltage values from the peak detector (perimeter detector) are constantly feeded into logical *if* statements in an infinite while loop. Events that occur when an *if* criteria is met are handled on a firstly basis. As soon as this happens, a set of instructions in the handler functions are executed and the program continues where it left off. A complete block diagram of the software logic, as well as hardware setup, can be viewed in the appendices (see Appendices C2 and C3, respectively).

3.6.7 Bluetooth Module

To control our robot wirelessly, an HC-05 Bluetooth module was purchased online. In terms of wiring, we connected the VCC to the 3.3V rail powered by STM32, GND to the common ground, RXD to PA2, and TXD to PA3. Receiving strings from our Android/IOS terminal, we managed to control the global variables to go forward, make turns, and pick up coins.

3.7 Professional and Safety

To create a professional environment the team decided to follow Agile Scrum methodology. This allowed us to create a self-organizing, cross-functional team structure where we could progress the project in sprints focused around individual features. We also ensured that all code was fully commented and documented as it was created, to create traceability within the software.

To ensure that a high level of personal safety was upheld, we decided to first review and then uphold all the guidelines of working within the UBC laboratory environment, as we had learnt in the Safety Orientation for Electrical and Computer Engineering course.

3.8 Solution Assessment

Our robot was tested extensively to ensure it met design requirements. First of all, we used one coin to see if it could detect and pick up the coin. Once it was successful, we put another four kinds of coins to ensure the electromagnet worked for all Canadian coins. Afterwards, we used the function generator to feed a current to a wire and used that to test the perimeter wire detector. Both coin and wire detectors passed the tests and worked perfectly. After implementing the basic functionalities, we added the extra features - Bluetooth module, LED, and speaker. We performed a general test to collaboratively run through all features with the first round being operated autonomously and the second run being Bluetooth controlled.

4.0 Life-Long Learning

In this project, our team referred to many technical concepts and skills we had gained in the prerequisite courses such as ELEC 201- Circuit Analysis and APSC 160 - Introduction to Computation in Engineering Design. ELEC 201 made us more familiar with circuits, and provided us with basic knowledge of electrical components while APSC 160 introduced us with C programming skills, which are used throughout the project.

In terms of current courses, ELEC 221 - Signals and Systems and ELEC 211 - Electromagnetism are extremely helpful as they helped us better understand the signals we needed to generate and the magnetic fields being implemented on our coin and perimeter wire detectors.

The skills we have developed over the project will be useful not only for the continuation of our education but for our future careers as electrical engineers.

5.0 Conclusion

In summary, our robot operates in the following steps. It departs with a normal square wave. When the coin approaches, the magnetic field generated by the coin reduces the period of the square wave. In that case, the robot stops and moves backward a little bit. Then the motor for controlling the electromagnet starts to move in certain directions, and current is fed into the electromagnet. It attracts the coin later, and the coin is dropped into the container as current is cut off. This ends one picking-up procedure. When the robot detects the perimeter wire boundary ahead, it stops, steps backward, turns right, and moves in another direction. It continues until all coins have been picked up.

As for extra functionalities, we implement a Bluetooth module which allows us to remotely control the robot via an Android/IOS application. We also add a speaker and LEDs to indicate coin picking process after the robot detects the coin.

Over this project, we encountered challenges with our circuit design and software implementation. One such problem was that the square wave could not be generated properly. Later, we found it was because of the structure of the circuit resulting that no current could pass the capacitors at the beginning. We solved this problem by changing our circuit structure which then provides current to that part. Another problem was that the frequency was not stable enough which made it difficult for the robot to identify coins. We solved this issue by introducing an intricate algorithm that has multiple detection processes before validating that the object in front is really a coin.

6.0 References

[1] STMicroelectronics, "Microcontrollers & Microprocessors", retrieved from https://www.st.com/en/microcontrollers-microprocessors/stm32f0x1.html

7.0 Bibliography

Calvino-Fraga, J. (2017). The STM32F051 Microcontroller System [Electrical and Biomedical Engineering Design Studio ELEC291/ELEC292]. University of British Columbia, Vancouver.

Components 101. (2018, March 10). HC-05 - Bluetooth Module. Retrieved April 4, 2019, from https://components101.com/wireless/hc-05-bluetooth-module

STMicroelectronics. (2017). RM0091 Reference manual. RM0091 Reference Manual, Rev 9, 1-1004. docID:018940

STMicroelectronics. (2017). STM32F051x4 STM32F051x6 STM32F051x8. STM32F051x4 STM32F051x6 STM32F051x8, Rev 7, 1-122. docID:022265

8.0 Appendices

Appendix A – Project Data

<u>Appendix A1 – Lab Requirements</u>

The project **<u>must</u>** include the following components and/or functionality:

1. **A microcontroller system that is not based in the 8051 microcontrollers:** The microcontroller used to control the robot must not be an 8051 derivative. Parts to assemble a microcontroller system based on some popular microcontrollers are provided with the project 2 kit. They are listed in the table below[1].

Maker	Family	Microcontroller
Texas Instrument	MSP430	MSP430G2553
ST Microelectronics	ARM	STM32F051
Microchip	PIC32	MX130F064B
Atmel/Microchip	AVR	ATMega328P[2]

   You may use any other (non-8051) microcontroller you choose with the condition that you must also obtain the tools and software required to perform development with it.

2. **Battery Operated**: The robot must be battery operated. Batteries are neither provided in any of the kits nor will be provided to you at any moment because they are very expensive. You need to buy your own batteries. Both an AA battery holder and a 9V battery clip are included in the project kit.
3. **Metal Detection**: Coins must be detected using a metal detector. To assemble the metal detector you can use one of the inductors provided in the project 2 kit as part of a Collpitts oscillator. Your robot must be able to detect all the current Canadian coins in circulation (0.05$, 0.1$, 0.25$, 1$, and 2$).
4. **Coin picker mechanism and electromagnet:** A coin picker mechanism consisting of two micro servo motors and an electromagnet is provided with the project 2 kit. Your coin picker must be capable of picking any of the current Canadian coins in circulation (0.05$, 0.1$, 0.25$, 1$, and 2$; provided that they can be attracted by a magnet) and carefully deposit them in a container carried by the robot.
5. **Robot construction:** You can use any material you find available for the chassis of the robot (paper, cardboard, wood, plastic, metal, etc.). You can also use the materials and

[1] The ARM based LPC824 from NXP is also supported but not included in your project 2 kit. If you are interested in using the LPC824, contact your course instructor. A limited quantity of LPC824microcontrollers is available from previous years.

[2] This is the same processor used in the Arduino board. Arduino boards, programs, or anything to do with the Arduino environment are not acceptable for this project. If you choose to use this processor it must be programmed in plain 'C' using either GCC or any other available C compiler.

tools available in the MCLD workshop (staff may charge you for expensive material). If you plan to use the workshop, you need to complete the workshop safety training first. Also remember that you are required to wear safety glasses and safety shoes while in the workshop.

6. **C programming:** The code for this project must be completed using the C programming language.
7. **MOSFET drivers:** To drive and control the motors of your robot you must use MOSFETs (Metal Oxide Semiconductor Field Effect Transistors). Both NMOS and PMOS transistors are included in the course parts kit. If you find the motors are too noisy (electronically noisy, that is) you may want to consider isolating the microcontroller from the motors using optoisolators to control the MOSFETs. The LTV847 optoisolator is included in the parts kit. Obviously, the robot should be able to move forward and backward as well as turn left or right.
8. **Perimeter Wire Generator and Detection:** Your robot must be able to detect a perimeter wire carrying an AC current and remain inside such perimeter. To generate the field you can use either the function generator in the laboratories or build your own perimeter AC source using an oscillator such as the LM555 timer. The minimum perimeter area should be 0.5 m2. The recommended perimeter surface consists of a rectangle 1 m x 0.5 m as it easily fits on the laboratory benches.
9. **Basic Robot Task:** The robot should individually pick 20 coins (four of each type) placed randomly in the area defined by the perimeter wire, without completely leaving the perimeter at any time3. After the robot finishes picking the 20 coins, it stops.

[3] Parts of the robot may temporarily be outside the defined perimeter. For example, if the robot is turning and one of the wheels comes out of the perimeter it is still acceptable.

Appendix A2 – Detailed Parts List

Robot Hardware

- Solarbotics GM4 Clear Servo (2)
- MG90S Metal Geared Micro Servo Motor 9G 450 LKY61 (2)
- Tamiya 70144 Ball Caster
- 1.5V AA Batteries (4), including battery clip/holder
- Electromagnet
- Aluminum Robot Chassis (folded as body of the robot)

Circuit Hardware for the STM32 Microcontroller Assembly

- STM32F051 ARM Cortex-M0 microcontroller
- BC1148CT-ND 0.1 uF Ceramic Capacitor (2)
- BC1157CT-ND 1 uF Ceramic Capacitor (2)
- 270QBK-ND 270Ω Resistor (2)
- 330QBK-ND 330Ω Resistor (2)
- 67-1102-ND Red 5mm LED
- 67-1108-ND Green 5mm LED
- MCP17003302E 3.3 Voltage Regulator
- BO230XS USB adapter
- LQFP32 to DIP32 adapter board
- A26509-16-ND 16-pin Header Connector
- Push Button Switch

Additional Hardware

- LTV-847 Optoisolator (1)
- LM358 Op Amp
- XPH 35-4T Triple DC Power Supply
- HC-05 Bluetooth Module

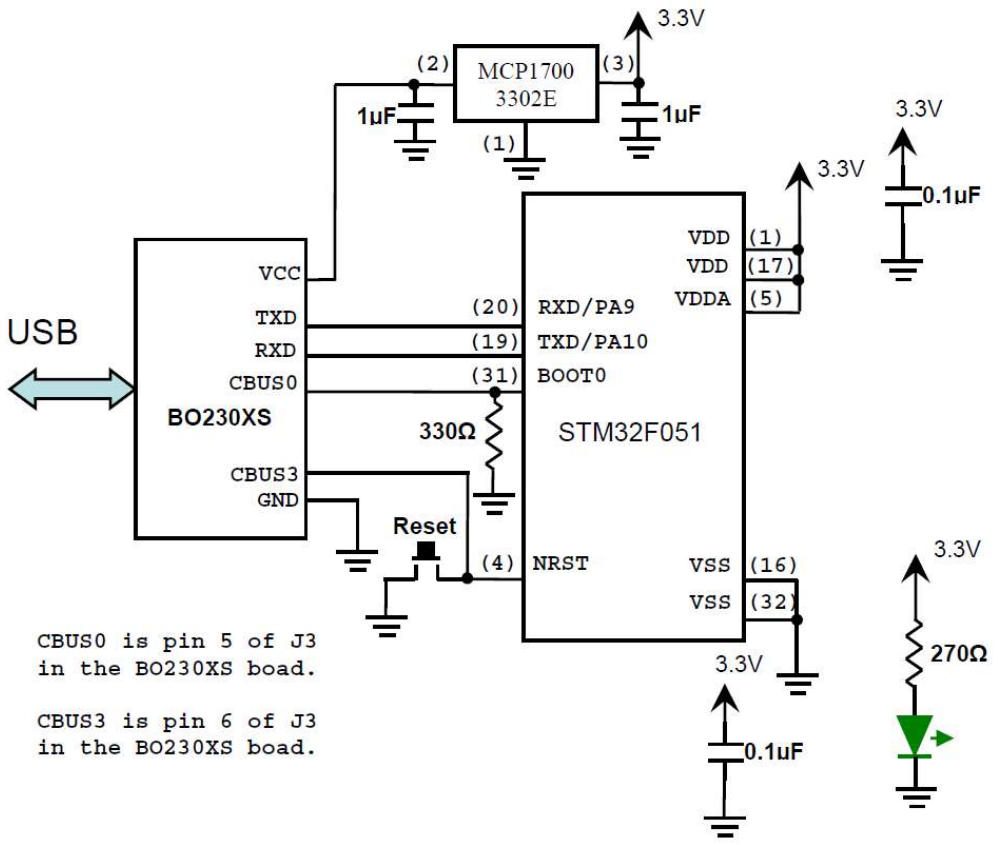
Software Programs

The following programs were used to compile and link code onto hardware:

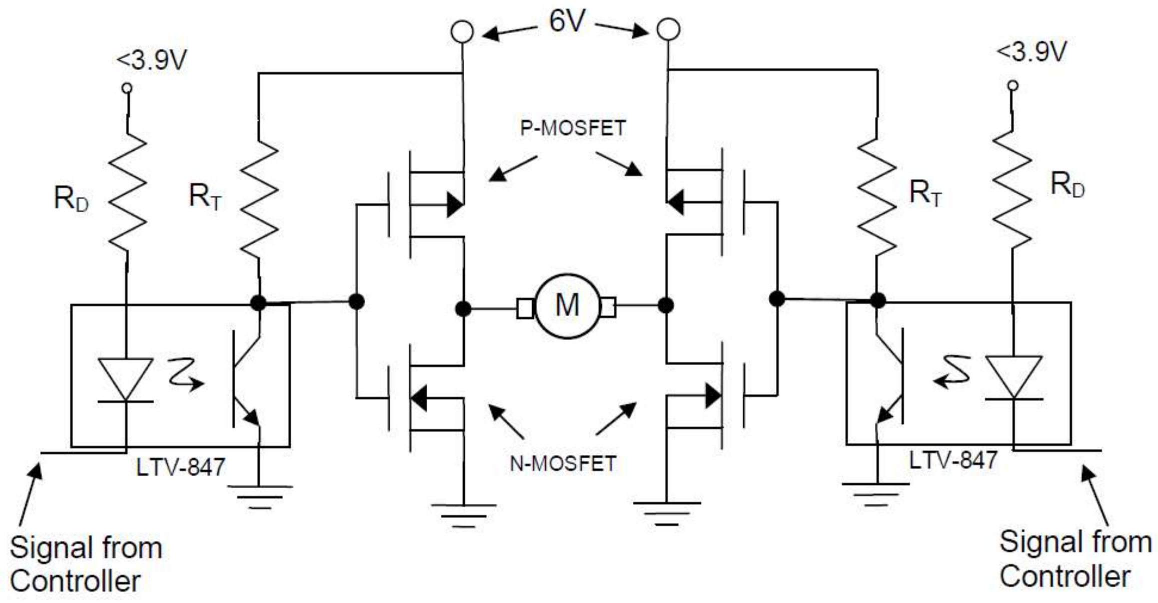
- CrossIDE v2.2
- GNU Make v4.2
- GNU ARM Embedded Toolchain
- PuTTy v0.71 (64-bit)

Appendix B – Circuit Diagrams

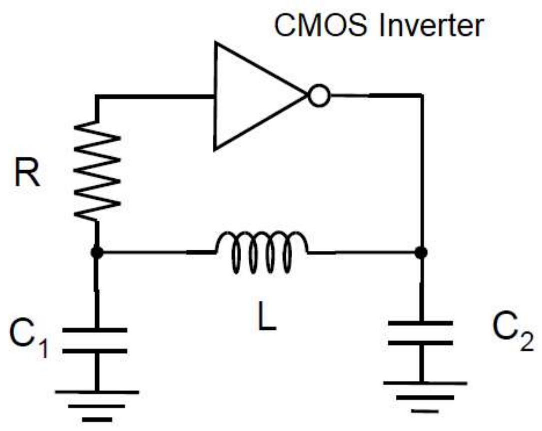
Appendix B1 – Microcontroller System Schematic

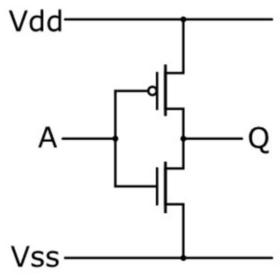


Appendix B2 – Optocoupler Diagram



Appendix B3 – Colpitts Oscillator Diagram

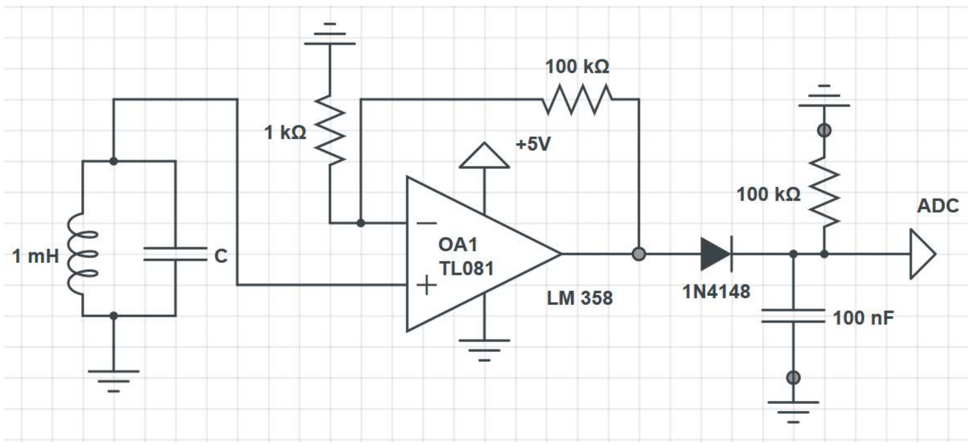




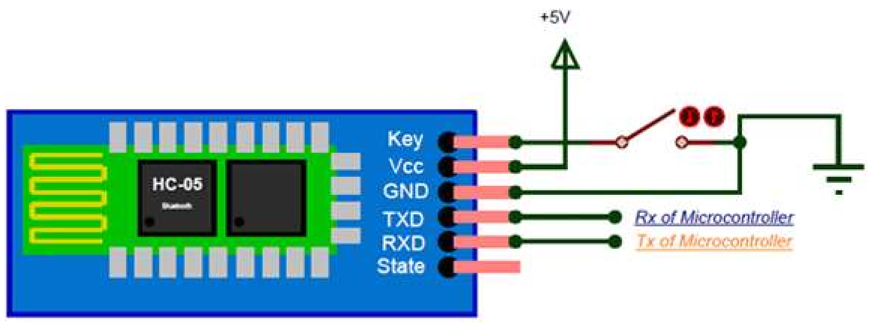
The Colpitts Oscillator

The CMOS Inverter

Appendix B4 – Peak Detector Diagram



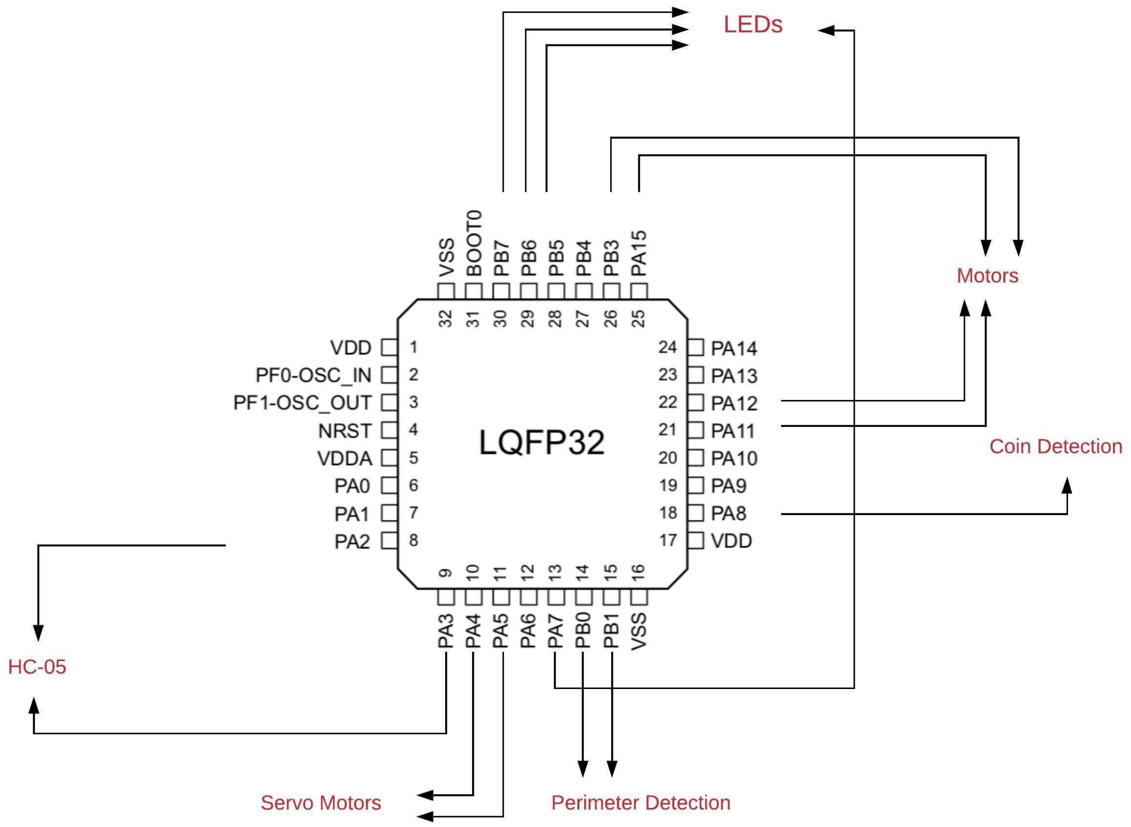
Appendix B5 – Bluetooth Module Diagram



Taken from: https://components101.com/wireless/hc-05-bluetooth-module

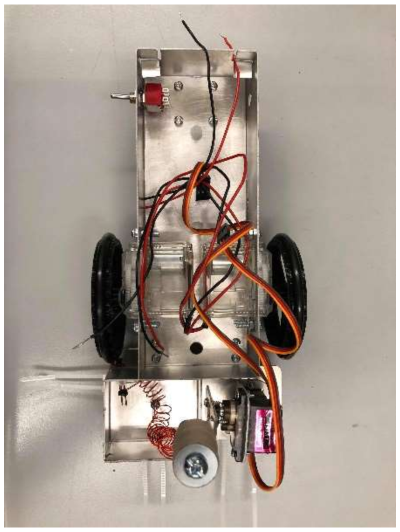
Appendix C – Block Diagrams

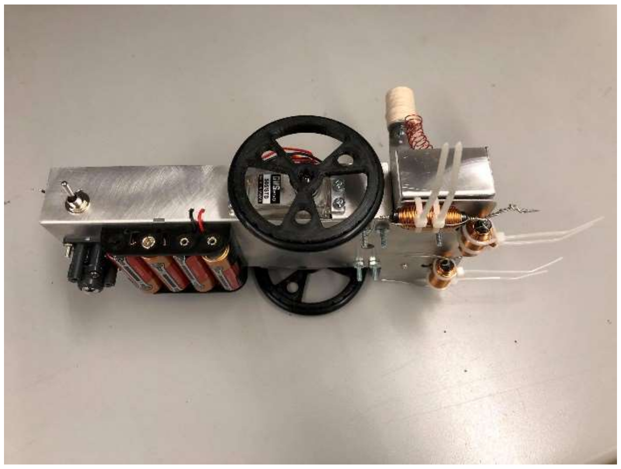
Appendix C1 – Hardware Block Diagram



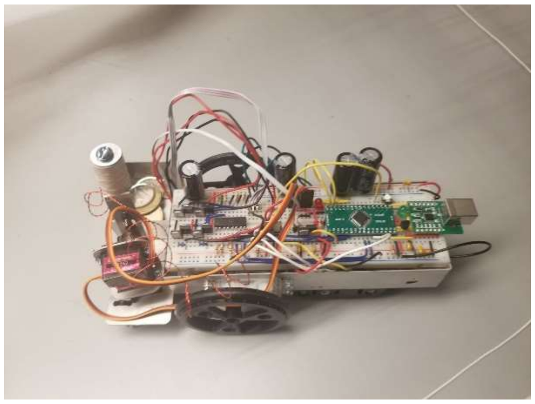
Appendix D – Design Documentation

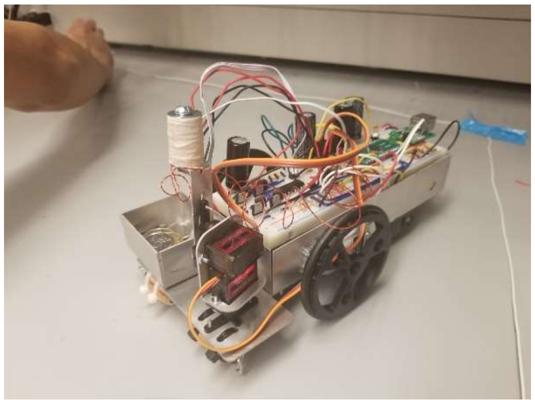
Appendix D1 - Robot Initial Body





Appendix D2 – Final Completed Device





Appendix E – Test Results

Appendix E1 – Wire Testing Results

Trial 1

Voltage	Number of Trials	Number of Detects	Percentage Success
1.0V	10	10	**100%**
1.5V	10	10	**100%**
2.0V	10	10	**100%**
2.5V	10	10	**100%**
3.5V	10	7	**70%**
5V	10	None	**0%**

Trial 2

Voltage	Number of Trials	Number of Detects in a Minute	Percentage Success
1.5V	10	10	**100%**
2.5V	10	10	**100%**
3.0V	10	9	**90%**

Appendix E2 – Coin Testing Results

Iteration 1 – Single Check

Threshold Frequency	Number of Coins	Number of Detects	Percentage Success
60,000	5	No detects	**0%**
59,000	5	3	**60%**
58,000	5	2	**40%**
57,000	5	Constantly Detects	**0%**

Iteration 2 – Multiple Checks

Threshold Frequency	Threshold Frequency 2	Number of Coins	Number of Detects	Percentage Success
59,000	61,000	8	No detects	**0%**
59,000	60,900	8	10	**N/A**
59,000	60,800	8	11	**N/A**
58,000	61,000	8	Too many detects	**0%**
58,000	60,750	8	Too many detects	**0%**

59,500	60,500	8	Too many detects	**0%**
59,500	60,450	8	Too many detects	**0%**
59,700	60,600	8	7	**87.5%**

Iteration 3 – Multiple Checks with Flags

Threshold Frequency	**Threshold Frequency 2**	**Number of Coins**	**Number of Detects**	**Percentage Success**
59,000	61,000	8	10	**N/A**
59,000	60,900	8	8	**100%**
59,000	60,800	8	10	**N/A**
58,000	61,000	8	11	**N/A**
58,000	60,750	8	Too many detects	**0%**
59,500	60,500	8	Too many detects	**0%**
59,500	60,450	8	Too many detects	**0%**
59,700	60,600	8	5	**62.5%**

Appendix F – Code (Links to Github)

The source code is hosted on a gist link, which can be viewed at this link.
https://gist.github.com/ad2969/5bfeb28b544e42fdca1819e39f5af2b8

Barcode Link to Gist
(Source Code)

Appendix F1 – Timer Initialization

```
RCC_AHBENR |= 0x00020000;               // peripheral clock enable for port A
RCC_AHBENR |= 0x00040000;               // peripheral clock enable for port B
GPIOA_MODER |= BIT8;                    // Make pin PA4 output (pin 10)
GPIOA_MODER |= BIT10;                   // Make pin PA5 output (pin 11)
// Information here: http://hertaville.com/stm32f0-gpio-tutorial-part-1.html
GPIOA_MODER &= ~(BIT16 | BIT17);        // Make pin PA8 input
GPIOA_MODER |= BIT22;                   // Make pin PA11 output (pin 21)
GPIOA_MODER |= BIT24;                   // Make pin PA12 output (pin 22)
GPIOA_MODER |= BIT30;                   // Make pin 25 output
GPIOB_MODER |= BIT6;                    // Make pin 26 output
GPIOB_MODER |= BIT8;                    // Make pin 27 output

// Activate pull up for pin PA8:
GPIOA_PUPDR |= BIT16;
GPIOA_PUPDR &= ~(BIT17);

// Set up timer 1 (first timer)
RCC_APB2ENR |= BIT11; // turn on clock for timer1
TIM1_ARR = SYSCLK/TICK_FREQ;
ISER |= BIT13;                  // enable timer interrupts in the NVIC
TIM1_CR1 |= BIT4;               // Downcounting
TIM1_CR1 |= BIT0;               // enable counting
TIM1_DIER |= BIT0;              // enable update event (reload event) interrupt

// Set up output port bit for blinking LED
RCC_AHBENR |= 0x00020000;               // peripheral clock enable for port A
// GPIOA_MODER |= 0x00000001;           // Make pin PA0 output (pin 6)

// Set up timer 3 (second timer)
RCC_APB1ENR |= BIT1;                    // TIM 3 timer clock enable
TIM3_ARR = SYSCLKTIM3/DEF_F;
ISER |= BIT16;                          // enable timer interrupts in the NVIC
TIM3_CR1 |= BIT4;                       //downcounting
TIM3_CR1 |= BIT0;                       // enable timer 3
TIM3_DIER |= BIT0;                      // enable interrupt
enable_interrupts();
```

Appendix F2 – ADC Initialization

```
void initADC(void)
{
        RCC_AHBENR |= BIT18;            // Turn on GPIOB
        RCC_APB2ENR |= BIT9;            // Turn on ADC
        GPIOB_MODER |= (BIT2+BIT3);     // Select analog mode for PB1 (pin 15 of LQFP32 package)
        GPIOB_MODER |= (BIT0+BIT1);     // Select analog mode for PB0 (pin 14 of LQFP32 package)
        ADC_CR |= BIT31;                // Begin ADCCalibration
        while ((ADC_CR & BIT31));       // Wait for calibration complete
        ADC_SMPR=7;                     // Long sampling time for more stable measurements
        ADC_CHSELR |= BIT9;             // Select Channel 9
        ADC_CCR |= BIT22;               // Enable the reference voltage
        ADC_CR |= BIT0;                 // Enable the ADC
}
```

Appendix F3 – Test Codes

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "stm32f05xxx.h"
#include "adc.h"

void main(void)
{
	float a;
	int j;
	printf("ADC/SERIAL/CLOCK test.\r\n");
	initADC();

	RCC_AHBENR |= 0x00020000; // peripheral clock enable for port A
	GPIOA_MODER |= 0x00000001; // Make pin PA0 output
	while(1)
	{
		j=readADC();
		a=(j*3.3)/0x1000;
		printf("ADC[9]=0x%04x V=%fV\r", j, a);
		fflush(stdout);
		GPIOA_ODR |= BIT0; // PA0=1
		delay(500000);
		GPIOA_ODR &= ~(BIT0); // PA0=0
		delay(500000);
	}
}
```