ELEC 291 : Metal Detecting Robot
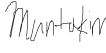
Team A9

April 12th, 2024

Course: ELEC 291/292 Electrical Engineering Design Studio I

Section: L2A

Instructor: Dr. Jesus Calvino-Fraga

Student #	Student Name	%Points	Signature
67286750	Tomaz Zlindra		
21475330	Martin Lieu		
7106221	Muntakim Rahman		
76029933	James Huang		
17542499	Emile Jansen		
46634911	Kyden McCaskill		

Contents

1 Introduction

1.1 Objective

The objective of this project was to develop a controller controlled metal-detecting robot. This was required to be battery operated and controlled using a wireless controller. The controller is used enable smooth of control the robot's motors; determining its direction and speed. It should also notify the user when the robot detects metal, via and LCD display and speaker. We should be able to differentiate between the metals detected by inductance as well as sound.

The ultimate goal of the project is to apply the learnings from the second set of labs in this course, with two families of microcontrollers. The circuitry used is familiar to prior in this course, but is designed to enable firmware detection of inductance with electromagnetism theory.

1.2 System Block Diagrams

In order to meet these system requirements and specifications, we designed the hardware and software as shown below. We will refer to this later in the report.
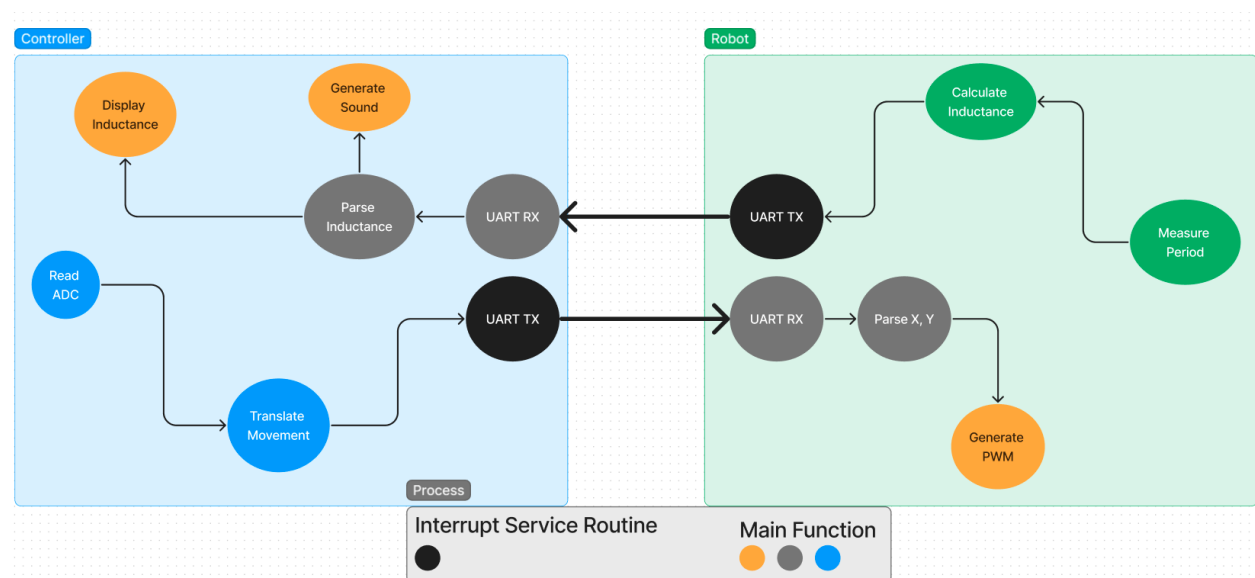

Figure 1: Firmware Block Diagram

2 Investigation

Prior to settling on a final design, we individually and collaboratively scoped out the constraints of our prroject components.

2.1 Idea Generation

We generated ideas for our implementation by discussing the process for meeting each of the individual specifications. This enabled dissecting the problem into smaller and manageable parts, as well as effective task allocation for firmware and hardware development. We discussed how each of these parts would work together to yield a fully functional robot. We prototyped individual components in isolated test environments to confirm details, such as the functionality of the joystick and PWM peripherals, the performance of the movement logic, and the the JDY-40transmission from one microcontroller to the next. Integrating these functional components into a functional product was the final step, allowing us to fine-tune initial designs collaboratively through end-to-end testing.

2.2 Investigation Design

To begin our design, we leveraged our knowledge of Labs 4-6 to understand how sub-circuitry and software modules should be implemented. A few components required further research, such as the Colpitts Oscillator for the metal detector, and the *JDY-40* radio module for data transmission. For our circuitry, we referred to datasheets to ensure our electrical components were powered in the appropriate voltage ranges. We also determined capacitances which would enable a stable steady state for the Colpitts Oscillator and enable us to meet expected inductance values. For the *JDY-40*, we spent some time understanding its limitations and how to properly configure it for communication between the two microcontrollers.

2.3 Data Collection

We collected data through extensive user testing both individually and as a group, akin to regression testing in software development. The key process consisted of using the controller to move the robot and ensuring it met expectations. We debugged the JDY-40 communication with `printf(tx_data);` and `printf(rx_data);` statements, as well as ensuring the system responded to the provided inputs.

Qualitative testing involved trying to smoothly maneuver the robot in the desired paths (i.e. straight line, square, figure-8, etc.) as well as testing the ability to indicate detected metal objects (i.e. via LCD display and speaker).

To quantitatively measure performance, the lab equipment (i.e. oscilloscope, multimeter) was used to measure the signals and ensure they were within the expected range. This was especially useful in timing our Interrupt Service Routines (ISRs) and ensuring that the PWM signals were generated correctly.

2.4 Data Synthesis

During testing, we emphasized having peer group members evaluate the performance of individual modules (i.e. movement controls, inductance detection, JDY-40 communication) to ensure that the system was functioning as expected. The developer of the given module participated in QA testing as well, but needed feedback from others to ensure we were able to address bugs without bias. This was especially useful in evaluating *JDY-40* communication for the movement controls and inductance detection.

We also developed a *Python* script to access the microcontroller *COM* ports and log both the movement and inductance data shared between the microcontrollers. This enabled us to observe both response time and accuracy betweeen transmitted and received data streams.
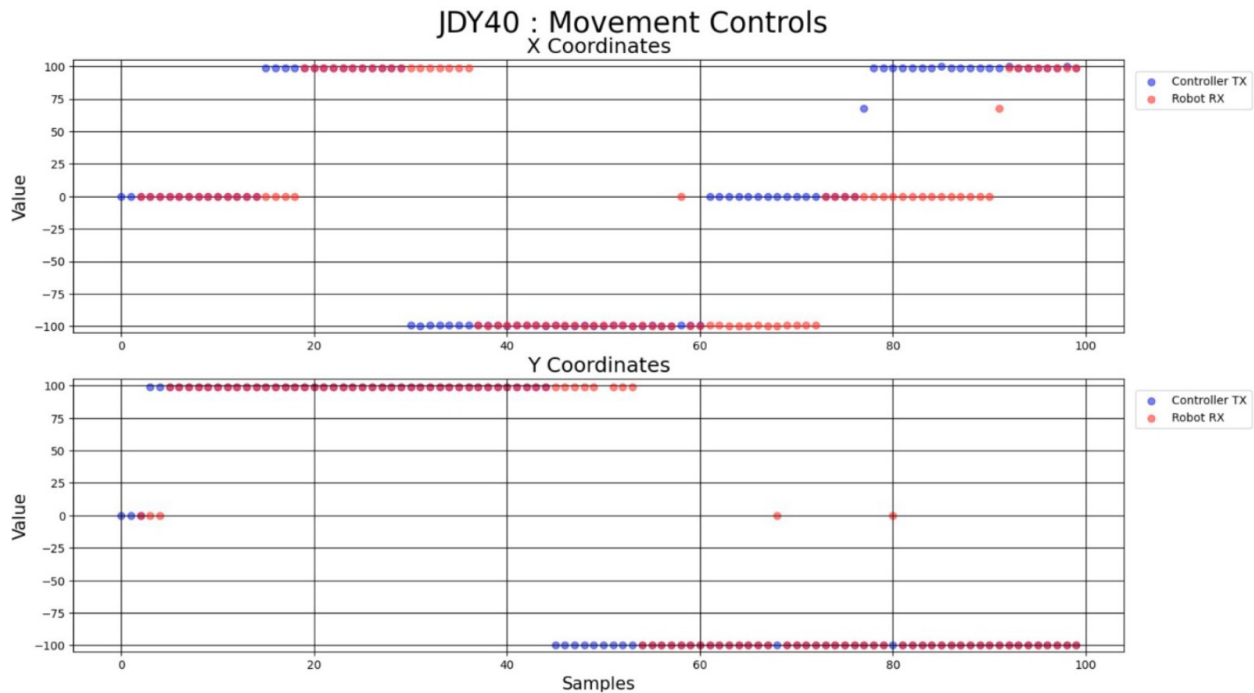

Figure 2: Movement Controls : Data Logs

2.5 Analysis of Results

From the graphs above, we note that the latency is displayed as a number of samples. This was due to the fact that we had many samples per second and wanted to visualize the microcontroller responsiveness to the
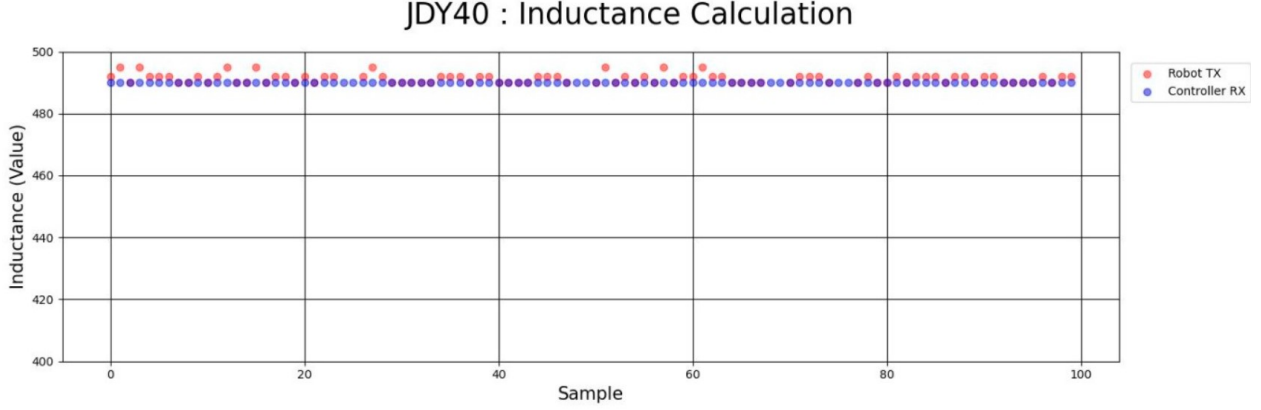
Figure 3: Inductance Detection : Data Logs

change in information. From figure 2, we see that the direction and speed in transmitted accurately from the *STM32* to the *EFM8*. In figure 3, we see that the inductance values are also transmitted accurately between the two microcontrollers. We motified both X and Y coordinates to test possible combinations and this was received in a reasonable number of samples.

The inductance exhibited similar behavior at the end of the development process, but we underwent many iterations on our firmware implementation to ensure that the period was measured accurately and we were able to calculate an inductance which could be accurately used to determine magnetic thresholds. Note that it was important to be able to bucketize inductance ranges (i.e. $400 - 500 \, mH$, $500 - 600 \, mH$, etc) to determine the type of metal detected.

3 Design

As we settled on preliminary designs of our subsystems, we required approval from team members to ensure requirements and constraints were addressed in our thought processes.

3.1 Use of Process

We followed the Work Breakdown Structure (*WBS*) principle during development. A well planned design of individual elements ensured that we could develop and test each module independently. This allowed us to identify and address issues in the early stages of development, and to ensure that we can swiftly and effectively integrate building blocks into a final product.

Another process we emphasized was keeping track of software revisions in a well managed *Git* repository. We used *Visual Studio Code : Live Share* to collaborate on the same codebase and to ensure that we were able to debug and test the code in real-time.

This was especially useful during dedicated group work sessions where we were adding parallel features to a *development* branch. We pushed this branch to the remote repository and merged it with the *main* branch after ensuring it works exactly as intended on the microcontrollers. Tracking firmware revisions with source control also enabled quick iterations on software modules, which helped us meet project timelines. Requirement functionality was prioritized in the integration process, including the movement controls, inductance detection, and JDY-40 communication. Our additional features (i.e. lock/unlock, speed limiter, turn limiter) were added after the core functionality was confirmed to be working as expected.

3.2 Need and Constraint Identification

.

During firmware development, we identified the following constraints from the documents provided by Dr. Jesus Calvino-Fraga[10] as well as from our own comprehension :

- The controller and robot must be driven by two microcontrollers from different families (e.g. *ARM Cortex* and *8051*)

- The robot must be able to detect metal using an inductor built from magnet wire and a coil frame.

- The robot must be powered by AA batteries to drive wheels and 9V for remaining functions.

- The speed and direction of the robot must be smoothly controlled with a joystick.

- The controller and the robot must wirelessly transmit data to one another.

- The metal detected by the robot should be indicated to the user via visual and auditory feedback.

- The transmission of data must be reliable and accurate.

3.3 Problem Specification

We encountered several related and recurring issues throughout the project, including inaccurate period measurements of the Colpitts Oscillator, unreliable JDY-40 communications between the microcontrollers, and the overall code integration between modules.

JDY Communications A key module which required extensive development time was the *JDY-40* communication between the *STM32* and *EFM8* microcontrollers. The bugs caused by this were the main blocker

7

in integrating our firmware, as the communication was not reliable and often failed to transmit data between the two microcontrollers.

In our design, we decided to time the transmission of data to ensure transmission occurred at a consistent rate (i.e. via ISR). This was to avoid the usage of synchronous delays, which block the execution of other code. We timed the ISR transmission to strike a balance, ensuring it wasn't excessively slow or fast. Otherwise, we risked losing data due to transmitting too quickly or rendered it obsolete due to slow data transmission. We also had to ensure that the data was sent in a format that could be easily and accurately parsed by the receiving microcontroller.

We also had to ensure that the data was sent in a format that could be easily parsed by the receiving microcontroller. This was especially important in translating the X and Y coordinates from the joystick to the robot microcontroller.

A major obstacle was finding that *Git* commits with a functional *JDY-40* module would suddenly behave erratically (i.e. transmission delays, inaccurate data to be received). We believe this was partially due to software bugs, but also due to the high number of *JDY-40s* in the lab environment during the latter part of the project.

Period Measurement Although a seemingly trivial task, measuring a period which altered by less than 1ms was difficult to measure accurately. This was due to the implementation with *EFM8* Timer 0, which monitored the number of clock cycles for a period signal during n periods.

By nature, this needed to be an uninterrupted calculation such that the period could be accurately measured. We found that the period was often inaccurate due to the ISRs we implemented for parallel functionality (i.e. JDY-40 communication, PWM generation).

Integration Integrating all the code together was a major obstacle, partially due to the issue mentioned above. We tested our firmware modules and observed functionality individually, but when we combined them, we found that the code would often malfunction.

One example of this was when we added the *JDY-40* program to the speaker code on the *STM32*. We had configured Timer 2 in PWM mode via Channel 1, which required a pin in conflict with the *UART2* peripheral. Another key example was when we incorporated the *JDY-40* code with the PWM code on the *EFM8*. Timer 3 was being used for a *JDY-40* delay, but we also needed to use Timer 3 for the PWM generation. This required us to reconfigure the PWM to use Timer 5 ISR to ensure that they were not in conflict with one another.

3.4   Solution Generation

We found that the best way to develop our solution was through the same four-step process as when developing the Oven Reflow Controller in Project 1. This involved :

1. Discussing the individual requirements and assigning development to team members.

2. Developing sub modules in a test environment.

3. Integrating the sub modules into a single, cohesive product.

4. Exhaustively test and debug functionality until requirements are met.

To address recurring and ongoing issues, we relied on the *WBS* principle to isolate them from the rest of our system. This was critical in investgating the firmware issues related to the *JDY-40* radio and to understand whether the transmission rate was the culprit in unreliable and inaccurate data. We also identified the issue with period measurement could be solved by thwarting ISR execution during the period calculation. Once we established functionality in these test environments, we were able to hone in on the root cause of the issues and develop solutions to address them. This was most evident in our reconfiguration of timers to deal with conflicts in between modules (i.e. speaker and *JDY-40* on the *STM32*, *JDY-40* and PWM on the *EFM8*).

When redesigning our *JDY-40* communication, we started from the test file with limited functionality and added features incrementally to find the cause behind the erratic behavior. We initially had both *UART* transmission and reception in the same ISR for both microcontrollers, but we observed that receiving data asynchronously caused our microcontroller to crash. We soon realized this functionality had to be synchronous and a part of the main program. Since the response to an input is a synchronous process, we should update it synchronously as well. However, we appreciated being able to control the timing of transmission to ensure that the data was sent at a consistent rate. Due to this, we continued to use ISRs for transmission, and these were the contributing factor in transmission rate.

Tackling the problems we discovered in a collaborative manner allowed us to address them comprehensively and perform extensive testing during the development process. We also found that ownership over individual subcomponents allowed us to effectively identify the problems in the section above. Having a group member more familiar with a specific implementation was critical in swiftly addressing development challenges.

3.5 Solution Evaluation

To evaluate our final implementation, we emphasized end-to-end testing. This involved adding modules incrementally and seeing how they interact with one another to perform intended functions. Stress testing the controls was a key part of this, as we wanted to ensure that the robot could move the way the user intends. Since the user could be someone unfamiliar with the implementation, we included group members working on parallel functionality to test a given implementation. If the robot met a qualitative checklist of response time, accuracy, and reliability (as determined by the user), we considered the implementation successful. This was especially important once we had quantiatively tracked the output of signals with the lab equipment and with `printf();` statements. Once the modules functioned by themselves in a test environment, and functioned together as observed with available tools, we relied on user testing as our final evaluation metric.

3.6 Safety/Professionalism

To ensure our product was safe, we developed many safety algorithms in the program. Before the robot can be controlled, the user must enter a 4 digit passcode to "unlock" the remote controller. This was paired with a lock feature which would disable the remote in the need of an emergency or other purpose. In tandem with this, we had a power switch to both conserve battery life and to ensure that the robot could not be controlled in this state. In addition, we implemented a speed limiter and turn limiter on the remote controller for the movement of the robot. If the user wishes to operate at a lower speed without the risk of too much speed, a simple click would enable this.

Furthermore, all this information was displayed on an LCD to provide visual feedback of the controller settings. To ensure professionalism, we maintained a clean and organized codebase, and managed development updates with good source control practices. We also maintained a professional attitude in our communication and collaboration during the project.

3.7 Detailed Design

3.7.1 Hardware Design

The robot's EFM8 circuit and the controller's STM32 circuit were intricate systems, featuring numerous connections. However, these were not complex in design.
As seen in figure **??**, the relevant circuits include the voltage dividers, joystick (ADC), JDY-40, and LCD. Conversely, in the figure **??**, the important circuits were the voltage dividers, JDY-40, metal detector and optocoupler/H-bridge circuits.

Voltage Dividers

We had a total of 4 step down voltage dividers for this project. Since the controller and the robot needed to be battery powered, we were required to use an LM7805 to drop the voltage of a standard 9V alkaline battery to 5V to power circuits such as the EFM8, LCD, speakers and the metal detector circuit. For stabilizing capacitors we used 0.22uF and 0.1uF capacitors as shown in the LM7805 datasheet. Subsequently, this 5V voltage was also dropped down to a voltage of 3v3 to power the ADC, JDY-40s and the STM32. We used 2 1uF capacitors to stabilize the signal.

ADC Joystick

The PS2 joystick was a basic dual ADC device to control the robot. This also did not require any complex designs as it is just two potentiometers in the x and y directions. This device was connected to 3V3 with the VRX and VRY pins connected to two ADC pins on the STM32 to measure ADC values. We also ended up using the built-in switch as a seperate button connected to a GPIO pin on the microcontroller.

JDY-40 Circuit

The JDY-40 (5) was used on both controller and the robot for wireless communication. The wiring was very straightforward, it was powered with 3V3 and the TX and RX pins are connected to the RX and TX pins, respectively. The SET pin was connected to a basic GPIO output on the microcontroller, used during the pairing process for AT commands (set to GND whenever an AT command needed to be sent).

Metal Detector Circuit

To detect metal, we used Colpitts Oscillator (4) with a discrete CMOS Inverter. This would allow us to measure a frequency related to the inductance of the inductor by using the formula $f = \frac{1}{2\pi\sqrt{LC_T}}$, where $C_T = \frac{C_1 C_2}{C_1 + C_2}$. Since we wanted to use lower capacitances to allow for a larger Δf, we ended up using $C_1 = $ 1nF and $C_2 = $ 10nF. This allowed for a stable steady state and were also within the recommended range proposed by Dr. Jesus Calvino-Fraga.

Optocoupler and H-Bridges

In order to convert our voltage PWM signal into a movable operation, we needed the h-bridge and optocouplers for each motor. The H-bridges would switch the polarity of the applied voltage, enabling for forward or backwards motion depending on the strength of the applied signal. Furthermore, the contrasting high voltage operations of the motor and low voltage operations from the EMF8 is an electrical concern that we addressed by implementing the optocouplers. The optocouplers electrically isolate the two circuits, prevent-

ing voltage spikes and limiting noise, improving compatibility and efficiency.

Additionally, we used adequate resistors values to satisfy the 50% current transfer ratio (CTR) and other conditions to achieve proper functionality.

3.7.2 Firmware Design

This project required a strong firmware foundation to ensure the robot behaved as intended. This entailed developing C programs in close coordination for the *STM32* and *EFM8* systems. The fundamental modules include the joystick input standardization, movement logic/PWM control, JDY-40 communications, speakers, and period calculations (metal detector).

Joystick Input Standardization

The raw ADC values from the joystick were not sent directly to the robot microcontroller. First, we subtracted the values by the initial ADC values they had initially (around $\frac{2^{12}-1}{2}$), to ensure when the joystick is in the middle, the value was 0.

```
1  void standardize_directions(float* x_value, float* y_value) {
2      *x_value = -1 * (*x_value - X_MIDPOINT);
3      *y_value = -1 * (*y_value - Y_MIDPOINT);
4  }
```

Listing 1: Joystick Calibration

This was paired with some more logic to convert this number into a percentage from 1-100% for the x and y directions. As shown in figure 2, the value was divided by the total in the direction it was pointed to get a percent. This was paired with a minimum active threshold of 5%, due to the joystick value fluctuating slightly when it was not in motion.

```
1  if (y_value > MINIMUM_PERCENT_ACTIVE * (4095 - Y_MIDPOINT) || y_value < - MINIMUM_PERCENT_ACTIVE * Y_MIDPOINT) {
2      if (y_value >= 0) standardized = (int)(100 * y_value / (4095 - Y_MIDPOINT));
3      else if (y_value < 0) standardized = (int)(100 * y_value / (Y_MIDPOINT));
4  }
```

Listing 2: PWM Percentage Conversion

Movement Logic/PWM Control

Once the values for X and Y are sent via the JDY-40 (PWM percentages), they are inputted into the function in the listing below (3). This logic would establish independence between the x and y values, where y would control the strength of the PWM, and x would control the percentage difference between the two wheels.

```
1  void PWM_manager(float x_value, float y_value)
2  {
3      if (x_value >= 0) // RIGHT TURN
4      {
5          left_wheel = abs(y_value);
6          right_wheel = (100 - abs(x_value)) * abs(y_value) / 100;
7      }
8      else if (x_value < 0) // LEFT TURN
```

```
 9      {
10          left_wheel = (100 - abs(x_value)) * abs(y_value) / 100;
11          right_wheel = abs(y_value);
12      }
13  }
```

Listing 3: EFM8 : PWM Pulse Count Calculation

The outputted values would be counters from 0-100, which would be used for counters for the PWM duty cycles for each wheel. We ran Timer 5 at a frequency of 10kHz to allow for very precise duty cycles. The exact code for the Timer 5 ISR is in the Appendix (10) for more details.

JDY-40 Communications

In order to share data between microcontrollers, we had to ensure our firmware operated deterministically and reliably. This required ensuring that the data received by each microcontroller was accurate to the transmitted data. It also required ensuring that this did not slow down the remaining functionality of the system, especially for the robot *EFM8* microcontroller.

As seen in figure 1, we decided to periodically transmit data to the *JDY-40* radio. This was done asynchronously via ISRs. Receiving and parsing the data was done in the main function however, as the subsequent instructions depended on a newly received value. This appeared as a function call to the following function on the *STM32*. Note that this only updates the global variable RX_BUFF when 2 bytes are received.

```
1  void RX_I(void) {
2    if (ReceivedBytes_2() > 0){
3      egets_2(RX_BUFF, sizeof(RX_BUFF)-1);
4      // printf("RX_BUFF: %s\r\n", RX_BUFF);
5    }
6    return;
7  }
```

Listing 4: STM32 : JDY-40 RX

We prioritized frequent data transmission of movement controls from the joystick and implemented an *STM32* ISR with a software counter to transmit this every 250ms via *UART* communication. This ensured we can smoothly control the speed and direction of the PWM for the robot Optocoupler and H-Bridge circuitry.

```
1  void TIM21_Handler(void) {
2    TIM21->SR &= ~BIT0; // Clear Update Interrupt Flag
3    TX21Count++;
4
5    if (TX21Count > 250) {
6      TX21Count = 0;
7      TX_XY();
8    }
9  }
```

Listing 5: EFM8 : Timer 21 ISR

We were able to provide slightly more leeway for transmission of inductance values, as responsiveness wasn't as time-critical as movement controls. However, we decided to mirror the logic above on the *EFM8* to send the calculated value every 250 *ms* to the *STM32*. This is shown in the Appendix (11).

Data parsing and validation was an additional layer of complicated for the *EFM8* system, as we needed to accurately process the X, Y, and Z values from the *STM32* transmission. This involved checking the length to ensure it was within the expected range of 12-14 characters. Note that we had to transmit a fixed length from the *STM32* to account for this. Following this initial validation, we implemented a custom function to trim the non-numeric characters from the string, and assign the relevant data to the respect variables. This is shown in the Appendix (12).

Parsing the received inductance values on the *STM32* was a straightforward process as we only expected a single value and did not prepend any additional characters to the string. Hence, we were able to use `atoi()` to convert the string to an integer.

Period Measurement and Inductance Calculation

In order to measure period, we used Timer 0 on the *EFM8* to measure the number of clock cycles for a period signal. This was done synchronously, and the period was measured uninterrupted in our final implementation. This involved disabling all ISRs on the EFM8 during this critical code segment, as shown below in 6.

```
1   // Measure the period of a square signal at PERIOD_PIN
2   unsigned long GetPeriod (int n) {
3       unsigned int overflow_count;
4       unsigned char i;
5
6       EA=0; // Disable interrupts
7       TR0=0; // Stop Timer/Counter 0
8       TMOD&=0b_1111_0000; // Set the bits of Timer/Counter 0 to zero
9       TMOD|=0b_0000_0001; // Timer/Counter 0 used as a 16-bit timer
10
11      ... // Wait for the signal to go low
12
13      // Reset the counter
14      TR0=0;
15      TL0=0; TH0=0; TF0=0; overflow_count=0;
16      TR0=1; // Start the timer
17
18      for(i=0; i<n; i++) // Measure the time of 'n' periods {
19          while(PERIOD_PIN!=0) // Count 16-bit timer overflows { ... }
20          while(PERIOD_PIN!=1) // Count 16-bit timer overflows { ... }
21      }
22
23      TR0=0; // Stop timer 0, the 24-bit number [overflow_count-TH0-TL0] has the period in clock cycles!
24      EA=1; // Enable interrupts
25
26      return (overflow_count*65536+TH0*256+TL0);
27  }
28
```

```
29  unsigned long GetFrequency_Hz(void) {
30      long int count = GetPeriod(5);
31      long int frequency = (SYSCLK*5.0)/(count*12);
32
33      // printf("Frequency: %ld Hz\n", frequency);
34
35      if (count>0) return frequency;
36      else return 0;
37  }
```

Listing 6: EFM8 : Frequency Calculation

Disabling the ISRs was a critical part of this process, as we needed to ensure that the period was measured accurately. We reduced the number of periods measured to 5 to prevent the robot from being unresponsive to the user. This was a balance between accuracy and responsiveness, as we needed to ensure that the robot could still move while the inductance was being measured. This was a risk we took to build our final product. The inductance calculation itself involved basic float arithmetic and casting the result to an integer, as shown below in 7.

```
1  int GetInductance_microH(void) {
2      long int freq = GetFrequency_Hz();
3      float CT_mF = GetCapacitance_mF();
4    float I_kH = MILLI_MULTIPLIER/(squared(2*PI*freq) * CT_mF);
5
6      return (int)(MILLI_MULTIPLIER * MILLI_MULTIPLIER * I_kH);
7  }
```

Listing 7: EFM8 : Inductance Calculation

Note that we made a decision to perform calculations from peripheral reads on the transmitting microcontroller, whether that be converting a period measurement to inductance on the *EFM8*, or the joystick values to standardized values on the *STM32*. This was to ensure that the receiving microcontroller could focus on its core intended functionality, and not be bogged down by calculations it should not be responsible for.

It was essential for our robot to function both as a vehicle and to incorporate a metal detecting circuit as a feature of equal importance.

Speaker

We implemented the speaker on the remote using timer 2 of the *STM32*. We used a base frequency of 7040Hz, as this allowed us to output the perfect "A" note (sample chart shown here: 6), whereas other frequencies have decimal points (dividing the frequency by 2^n). In the code shown in 8, the new_ratio value served as a counter in the Timer 2 ISR to control the PWM speed for the speaker. The change in frequency was tied to the inductance value measured by the *EFM8*.

```
1  float SetSpeakerFreq(float inductance_mH, float current_ratio) {
2      float new_ratio, new_freq;
3
4      // Inductance Thresholds for Speaker Frequency
```

```
5        if (inductance_mH <= 435.0) new_ratio = 1;
6        else if (inductance_mH < 445.0) new_ratio = 2;
7        else if (inductance_mH < 455.0) new_ratio = 4;
8        else if (inductance_mH < 465.0) new_ratio = 8;
9        else new_ratio = 16;
10
11       new_freq = TICK_FREQ_TIM2 / new_ratio;
12
13       return new_ratio;
14   }
```

Listing 8: STM32 : Speaker Thresholds

3.8   Solution Assessment

We assessed our design through a set of tests. Our final quantitative affirmation was observing the number of *UART RX* samples it took for the robot to respond to the joystick input, as well as the number of *UART RX* samples it took for the controller to receive the inductance value from the robot. This is shown in the figures 2 and 3. This accompanied our qualitative testing, where we validated this with a human user to ensure that the robot was responsive to the joystick input and provided both visual and auditory feedback when metal was detected.

In our logging of the data, we noticed that data between transmitting and receiving microcontrollers were separated by a few samples (which spanned a few milliseconds). This was a latency which was deemed acceptable as in both cases, it met human expectation of response time. To further confirm this, we saved the log time of the data with a *Python* script as shown below.

```
1   ... # Import Packages, Open COM Ports
2
3   s = 0
4   while (robot_port.isOpen()) and (controller_port.isOpen()) and (s < 100):
5       print(f'[{dt.datetime.now()}] Reading data...')
6
7       robot_data = robot_port.readline().decode('utf-8').strip('\r\n')
8       controller_data = controller_port.readline().decode('utf-8').strip('\r\n')
9
10      ... # Parse and Process Data
11
12      new_df = pd.DataFrame(
13          {
14              'Datetime': [dt.datetime.now()],
15              'Robot X': [robot_x],
16              'Robot Y': [robot_y],
17              'Controller X': [controller_x],
18              'Controller Y': [controller_y]
19      })
20
21      if log_df.empty:
22          log_df = new_df
23      else:
24          log_df = pd.concat([log_df, new_df], ignore_index=True)
25
26      s += 1
27
```

```
28    log_df.to_csv('Coordinate_Logs.csv', index=False)
```

<div align="center">Listing 9: Python : Data Logging</div>

This generated the *csv* file with the following data in table 1.

Datetime	Robot X	Robot Y	Controller X	Controller Y
2024-04-09 15:34:58.248238	-100.0	-100.0	-99.0	-100.0
2024-04-09 15:34:58.537355	-100.0	-100.0	-99.0	-100.0
2024-04-09 15:34:58.831054	-99.0	-100.0	0.0	-100.0
2024-04-09 15:34:59.115151	-99.0	-100.0	0.0	-100.0
2024-04-09 15:34:59.418781	-100.0	-100.0	0.0	-100.0
...	...	...	...	...
2024-04-09 15:35:09.029631	99.0	-100.0	99.0	-100.0
2024-04-09 15:35:09.330765	99.0	-100.0	99.0	-100.0
2024-04-09 15:35:09.625331	99.0	-100.0	100.0	-100.0
2024-04-09 15:35:09.914547	99.0	-100.0	99.0	-100.0

<div align="center">Table 1: Selective log data for Robot and Controller Positions</div>

4   Life Long Learning

In this 3-week project, we learned what were perhaps some of the most important lessons in our undergraduate engineering degree so far. This project was a culmination of the knowledge we had gained in our previous courses, and it was a test of our ability to apply that knowledge in a practical setting. There was intricate circuitry which required us to apply our academic foundation in circuits and electromagnetism, as well as firmware development. This latter experience was best explored in this course, as we firsthand experienced the obstacles and constraints in programming single-purpose embedded systems. Software statements, including but not limited to `printf();` were not an option to ensure our robot behaved deterministically. Once we program it and power it via the battery, it must work as intended.

This helped us appreciate the well thought out design of the embedded electronics in everyday appliances, as well as feats of engineering (e.g. Apollo 11 Moon Landing). Another skill this taught us, which is coordinating and communicating challenges in a team of engineers. In the professional environment, an effective team is probably the leading contributing factor to the success of a project, and on a larger scale, the success of a company. This project was a microcosm of that and we were exposed to the key attributes of a valuable team member. This includes effective and respectful communication, a willingness to learn and adapt, and a strong work ethic. We also learned the importance of project management as we needed to develop individual components of the project in parallel, and then integrate them by the deadline. Altogether, this course provided a step into the professional world of engineering, and we are grateful for the experience.

5   Conclusion

In this project, we were able to develop a fully functional metal detecting robot, which can be smoothly controlled with a joystick. This was built through integration of electronic circuitry and firmware, designed to meet a predetermined set of requirements. We were able to operate within identified constraints and develop a solution which also took into account features which improved the user experience. This was a challenging project, but it was also a rewarding one. We were able to apply our academic knowledge in a practical setting, and this will be invaluable as we proceed to our third year in undergraduate electrical engineering.

6 References

7 Appendix
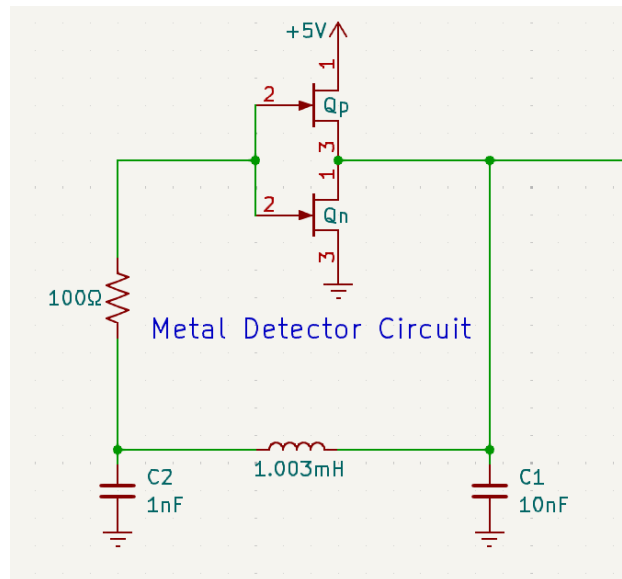
Figure 4: Colpitts Oscillator with Discrete CMOS Inverter
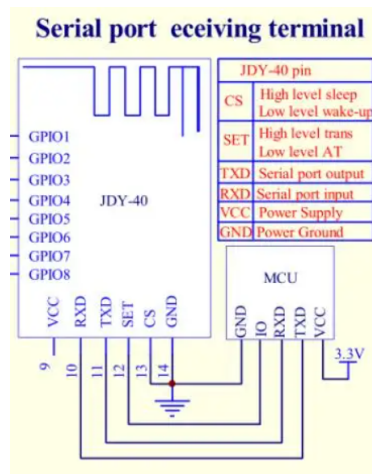
Figure 5: Example JDY-40 connections with a Microcontroller

```
1   if (count > 100)
2      {
3          count = 0;
4      }
5      if (PWM_percent_y >= 0)
6      {
7          LEFT_MOTOR_LHS = (count > left_wheel ) ? 0:1;
8          RIGHT_MOTOR_LHS = (count > new_right_wheel) ? 0:1;
9      }
10     else
11     {
12         LEFT_MOTOR_LHS = (count > left_wheel) ? 1:0;
13         RIGHT_MOTOR_LHS = (count > new_right_wheel) ? 1:0;
14     }
```

Figure 6: Musical Notes and their Frequencies

```
15
16      count++;
```

Listing 10: EFM8 : Timer 5 PWM Generation

```
1     void Timer4_ISR (void) interrupt INTERRUPT_TIMER4 {
2         int current_TR0 = TR0, current_TR5 = TR5;
3         TR0 = 0, TR5 = 0;
4
5         SFRPAGE=0x10;
6         TF4H = 0; // Clear Timer4 interrupt flag
7         TXcount++;
8
9         if(TXcount >= 250){
10            TXcount=0;
11            flag == 0;
12        }
13
14        if (current_TR0 == 1) TR0 = 1;
15        if (current_TR5 == 1) TR5 = 1;
16    }
```

Listing 11: EFM8 : Timer 4 ISR

```
1  void RX_XY() {
2     if(RXU1()) {
3       getstr1(RXbuff);
4       clearUART1Buffer();
5       length = strlen(RXbuff);
6        // printf("UNPARSED RX: %s\r\n", RXbuff);
7       if(length >= 12 && length <= 14){
8         Trim(RXbuff, &commands[0],&commands[1], &commands[2]);
9       }
10  }
11 }
12
13 void Trim(char *str, int *xin, int *yin, int *zyn) {
14    int i, j;
15    char * strStart = str;
16    char *x = malloc(5 * sizeof(char));
17    char *y = malloc(5 * sizeof(char));
18    char *z = malloc(2 * sizeof(char));
19
20    for (i = 0, j = 0; str[i] != '\0'; i++) {
21        if (isdigit(str[i]) || str[i] == '-') {
22            str[j++] = str[i];
23        }
24    }
25    str[j] = '\0';  // Null-terminate the resulting string
26    splitString(str, x, y, z);
27
28    //printf("%p \n", y);
29    //printf("%p \n", x);
```

```
30
31     *xin = stringToInt(x);
32     *yin = stringToInt(y);
33     *zyn = stringToInt(z);
34
35
36     free(x);
37     free(y);
38     free(z);
39 }
```

Listing 12: EFM8 : JDY-40 RX