James R. Di Re, Abdulrahman Kikia, Claudio Scione

May 31st, 2025

MakerSPACE Internship

# Arduino Radio - Final Report

## **Introduction**

In this report, we will detail the brainstorming, designing, programming and building of our MakerSPACE internship project: an Arduino-powered FM radio. Over the course of the past three months, we have worked to put together a working radio while documenting the entire process to serve as an inspiration and as a guide to anyone else who may want to embark on a similar project. Our goal when choosing this as our project was not only to improve our programming and debugging skills through a hands-on approach, but also to gain an understanding of radio waves, audio processing, and speakers. All this, in addition to enjoying the satisfaction of listening to music and news from local radio stations through a compact device we made from scratch.

## **Project Description**

### **Objective**

The goal of this project was to build a working AM/FM radio, using the Arduino Uno as the microcontroller and the Arduino IDE to write the code. We aimed to not only have it be able to play music through speakers, but also to display information about the frequency and station (RDS) on a digital display. Finally, we wanted to design and build a casing to insert all the hardware into that was practical and portable. While this was the base goal of this project, we hoped to add as many features as possible to the radio within the time frame we had to complete this project. We hoped to add a function to save preset radio stations that you could choose and change as you wished, shortwave compatibility, radio station scanning and even Bluetooth connectivity.

### **Inspiration**

It was James who originally came up with the idea of making a radio using Arduino. He knew initially that he wanted to do something involving audio, since he always enjoyed working with music and had a fascination with how sound waves worked. At first, he looked through many different audio projects, such as synthesizers, MIDI controllers, and distortion-altering guitars, but there was one common problem with all of these projects: nobody in the team knew anything regarding these topics. Eventually, he saw a few videos on YouTube of people using Arduinos to make working radios. A radio project seemed more feasible than other audio projects he had seen, since everyone in the team was familiar with how radios worked. However, the videos used FM-only radio modules, and only detailed the wiring process. We decided it would be interesting to expand on those projects James had seen online to make a radio with more features and a case to enclose it in. After some research on the Arduino forums, Instructables, and other project sites, we saw other examples of radios, each with their own unique construction and design. That made us realize that this was something we could realistically build and design on our own in the months we had to work on this project. Plus, the fact that the radio was practical (we could listen to music and news from it everyday) and intuitive (you can pick up radio signals anywhere) solidified our decision.

### **Design and Planning**

For the design, we wanted to build a radio that is reminiscent of old-school radios. That is, we wanted to use knobs instead of buttons. We also wanted it to be unique with some asymmetry. We shared different sketches of the design and some concept 3D renders to see where our imagination could go. In the end, we couldn’t settle on a specific design until we got all parts working and tested.

Much research was done early on to figure out the exact parts we were going to need. Also, the sheer number of available radio modules to pick from was overwhelming. However, one common theme between the videos we had initially watched was that most of them used the TEA5767 FM-only module, so we decided to get that in case other modules didn’t work the way we wanted them to. It was a logical decision, since if many others had success with it, then we should have success with it too. However, our initial goal was to have an AM/FM radio, with possible support for shortwave too, so we continued looking for more radio modules. After some more research, we were leaning towards the Si4732, which seemed to be very advanced and had many features to it. However, upon digging deeper, we realized that this “radio module” was nothing more than an extremely tiny chip; no breakout board for ease-of-use with Arduino existed. Thus, we had to keep looking.

An easy-to-use AM and FM radio module seemed to be nonexistent, but then James saw a YouTube video of an AM and FM Arduino-powered radio, so then we realized that one must exist (all the videos we had watched previously showcased FM-only radios). This new video we found pointed us in the direction of the TEF6686 module. This had most of the advantages of the Si4732 but had a sort of breakout component that could be used with our Arduino, with the help of pitch changers. However, there were less chances of this module being successful for us, since there were less people online who had used it, there was overall less documentation for it, and the pinout seemed a lot more complicated compared to that of the TEA5767 (we found only a single circuit diagram explaining how the pinout worked). So, we decided that we would try first with the TEA5767, and if we had time, we would try the TEF6686.

Designing a case to house all the parts also proved to be a challenge. The setup out of the case functioned well, but was fragile in the initial phases. The speaker wires kept disconnecting from the amplifier with the slightest movement, thus causing the speakers to stop playing, even after tightening the screws to hold the wires in place. (To fix this, a thicker wire was soldered onto the thinner speaker wire.) There were also pieces that would have to be mounted to the front of the case, so that the user could interact with the radio as intended. This was particularly problematic for the speakers, since they don’t have screw holes. Also, their wires could interfere with the screen if the positioning wasn’t optimal. However, the screen, antenna, and rotary encoder have screw holes, so mounting them wouldn’t be an issue. We made a prototype using cardboard to see how the pieces would go together. With the prototype, we attempted to make measurements for where each of the components would go, and then Abdulrahman incorporated these measurements in the modelling software he was using by adding holes to the box design where necessary.

Initially the idea was to have the six faces of the box snap into each other through connectors that were meant to hold them by pressure. However, as the faces would break with high pressure, we had to change the design so that five faces would be permanently attached (one only would be detachable).

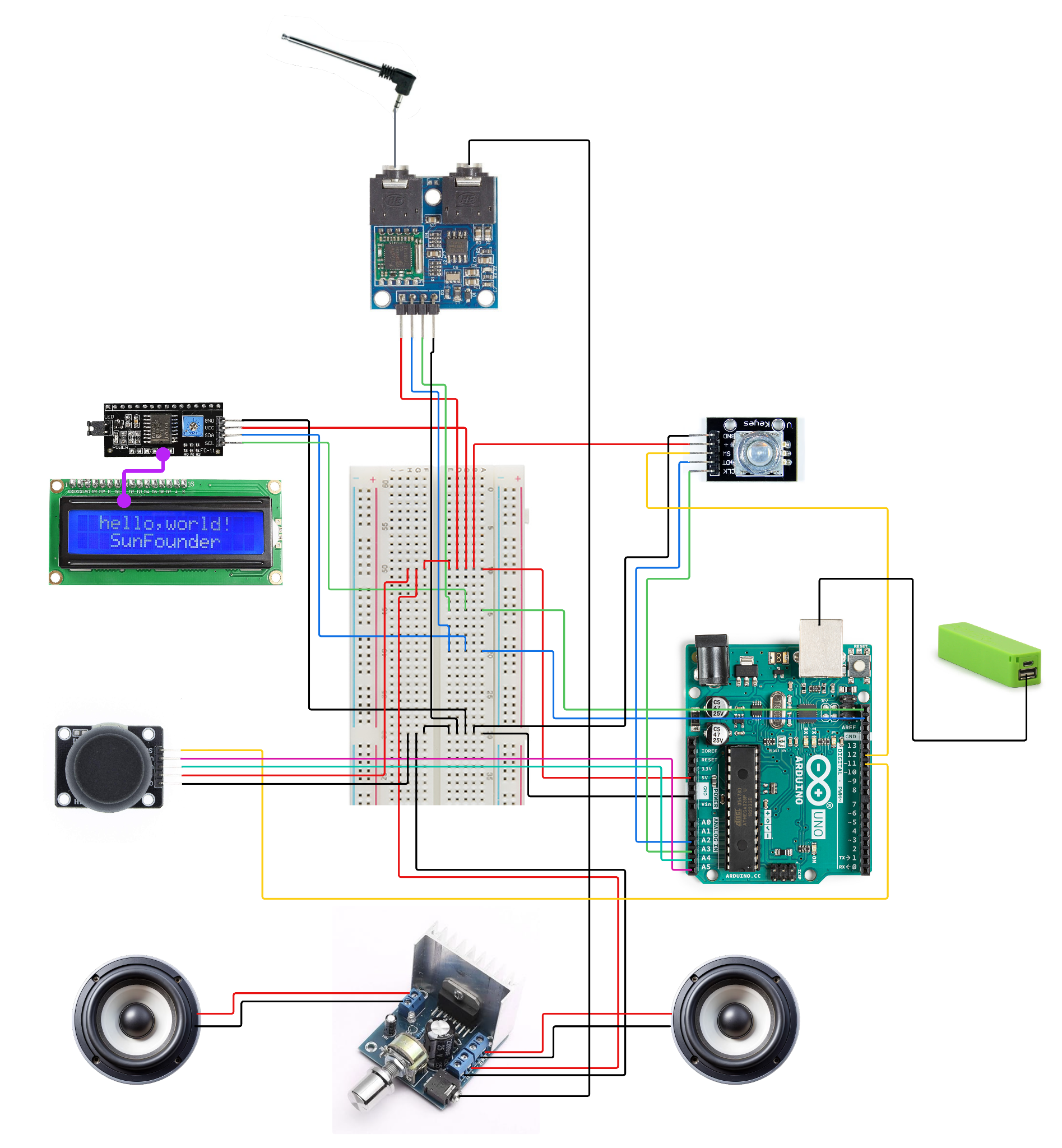
As for our power source, we planned out different options. We thought of using a 9-V battery, but the connector that the battery uses is very stiff and doesn’t come easily out of its terminals, so removing the battery wasn’t a feasible option to cut the circuit. After trying different options, we ended up using a power bank as our power source, as it can be easily removed and charged when needed.

The result of all this planning is a radio that is fairly portable, and is comparable to other radios of its kind in size.

## **Implementation**

### **Hardware Setup**

* Arduino Uno microcontroller
* Rotary encoder with button - for frequency tuning
* LCD display that is I2C compatible - for displaying radio data
* TEA5767 FM radio module with antenna - for receiving radio signals
* TDA7297 amplifier - for amplifying the audio signal coming from the radio module and for allowing volume control
* Male-to-male headphone wire - for sending the audio signal from the radio module to the amplifier
* Speakers
* Joystick - for choosing between different preset stations (up, down, left, right = 4 presets)
* Power source: power bank OR 9V battery OR power from a computer
* Male-to-male jumper wires
* Male-to-female jumper wires
* Breadboard



### **Software Development**

Process

Firstly, James already had C++ knowledge from a tutorial series he had watched on YouTube a few years ago by The Cherno, so because of that, it was a lot easier for him to write code for the project compared to someone just starting out. However, this was still his first time putting that knowledge into real practice. James recommended to the other group members that they watch some videos from the tutorial series too.

The features in the code were at first programmed in the Arduino IDE, and were added gradually (for example, first the code for the rotary encoder and LCD display were added, then the code for the radio module, then the code for the joystick, then the code for preset-saving, etc.). The code was always kept in a maintainable condition that was simple to expand upon. This was a top priority, as James knew from past projects that untidy code gets exponentially hard to maintain. Eventually, even though the code was kept organized, it was still overwhelming to keep all the functionality in one file. Thus, it was separated into six separate files: the main cpp file, a main header file, a class for managing radio state, a class for handling input, a class for handling the LCD display, and a class for handling the radio module. Later on in development, this proved to be a great choice, as each of these files grew in size. Looking back, it would not have been feasible to keep everything in a single file.

However, separating the code into different files didn’t come without its own problems. For one thing, there were many bugs that got introduced as a result, and all of them were isolated to being issues regarding C++’s copying of objects in unexpected areas. As a result, in fixing these bugs, the team grew more familiar with C++’s copying conventions, and became more aware of when it was appropriate to use pointers (\*) or references (&).

Eventually, the entirety of the code was moved to Visual Studio Code. This provided many advantages over the Arduino IDE. For one thing, VS Code allowed for easy integration with version control software such as Git and GitHub. Also, it underlined compilation errors in real-time, a feature which the Arduino IDE lacks entirely. There were also refactoring tools and an easy way to view the project structure. The extension used that introduced Arduino compatibility in VS Code was PlatformIO. A GitHub repository for the project was created.

Once everything was in VS Code, all the old iterations of code we had which were not under version control were pushed to the GitHub repository in a separate folder. This was in an effort to document everything we could. However, all new code added to the project was properly pushed to GitHub in the proper fashion, with frequent commits.

New functionality for the radio kept getting added until just a few days before the MakerSPACE presentation on May 30th. Once the code was entirely done, James looked through all of it one last time to see if any little details could be improved upon, then committed and pushed the changes. This was the code iteration that was used in the in-class demo.

Libraries

As for libraries, a couple of them were used. First, there was hd44780 by Bill Perry, an LCD library using I2C communication, which was exactly what we needed, since the LCD screen we had purchased supported I2C communication through a little board on the back. Also, this library was supposedly more up-to-date and compatible with more LCD models than other libraries, even those that have been around for a long time. Since this library was meant to be easy-to-use for those that were already familiar with other, more-established LCD libraries, many of the function names matched those from other libraries. Some useful functions include setCursor(int, int), print(“”), and clear().

The other library that was used was RotaryEncoder by Matthias Hertel. This library saved us a lot of headache, since it removed the problem of trying to find a good way to read the quickly changing voltages from the data and clock pins on our rotary encoder to determine the direction of turning. Also, all code that I had seen for this task was flawed in some way. Thus, this library did a lot of the heavy lifting for us. However, one minor issue with it was that input needed to be polled very frequently in order to catch all turns done on the rotary encoder. This meant that we had to ensure that our loop() function was kept short, and was short to execute. Sometimes, during debug sessions, where we had Serial.println() calls scattered throughout the code, the rotary encoder would become completely non-functional. Nonetheless, the library did get the job done. Some useful functions include tick() and getDirection().

Also, although not really a library, a snippet of code from Kevin Darrah’s YouTube video was used, which communicated desired frequency values to the radio module. This code was pasted directly into our code, and was left mostly as-is. Kevin Darrah mentions in his video that the code is from a TEA5767 library by Simon Monk.

An important reference that helped us frequently was the official Arduino documentation. For instance, everything we needed to know about EEPROM was included in the documentation, and was presented in a clear, concise manner. In addition, as mentioned earlier, the YouTube tutorial series by The Cherno was occasionally consulted throughout development.

### **Sensor Documentation**

* TEA5767 Radio Module

The radio module picks up FM radio signals. Signal reception is improved when an antenna is attached. To communicate with the radio module, which uses I2C, we need to use the Arduino Wire library.

The following is the code provided by Kevin Darrah in his YouTube video (mentioned earlier) to communicate with the radio module to change stations.

void setFrequency(float frequency)

{

// The calculation that needs to be performed before sending the frequency is outlined

// in the datasheet for the TEA5767

unsigned int frequencyB = 4 \* (frequency \* 1000000 + 225000) / 32768;

// Getting the high byte of the modified frequency

byte frequencyH = frequencyB >> 8;

// Getting the low byte of the modified frequency

byte frequencyL = frequencyB & 0XFF;

Wire.beginTransmission(0x60); // Begin communication with the radio

Wire.write(frequencyH); // Writing high byte

Wire.write(frequencyL); // Writing low byte

// The following three bytes that are written correspond to a set of parameters being sent to the radio module. Generally, each bit indicates a setting for the radio module to follow. The exact details of each bit in each byte are found on the datasheet.

Wire.write(0xB0);

Wire.write(0x12);

Wire.write(0x00);

Wire.endTransmission(); // End communication

delay(100);

}

* TDA7297 Amplifier

The amplifier detects sound signals (from the antenna) and transmits them into the speakers. It also has a potentiometer that controls the power given to the speaker (the volume coming out of them).

* Rotary Encoder

The rotary encoder helps with navigating radio frequencies. Each step in the rotation gives the Arduino a signal to either increase or decrease the frequency picked from the radio module by 0.2 MHz, which is the standard for radio stations in Montreal. The rotary encoder works by taking into account the voltage patterns over time across the data wire and the clock wire. With this information, we can deduce which way the rotary encoder is being turned. However, the way to deduce this is slightly complicated, so a rotary encoder library is used.

Code to initialize a RotaryEncoder object with the RotaryEncoder library:

// PIN\_A and PIN\_B refer to the clock and data lines of the rotary encoder. If the rotary encoder behaviour is opposite to what you expect, just switch these pins in the constructor.

RotaryEncoder encoder(PIN\_A, PIN\_B, RotaryEncoder::LatchMode::FOUR3);

Code to poll rotary encoder input (should be called every time loop() is run):

encoder.tick();

Code to get the direction that the rotary encoder has turned:

// Returns 0 if not turning, 1 if clockwise, -1 if counterclockwise

encoder.getDirection();

* Joystick

The joystick has two potentiometers for dual axis rotation. It can detect the rotation of the stick in a continuous manner (but only four with precision) and in our case is the optimal way to select and tune in on presets. Input read via the analog pins.

Code for reading in the x and y values of the joystick, which range from 0 to 1023:

int rawX = analogRead(PIN\_X);

int rawY = analogRead(PIN\_Y);

To convert this value to one that is between -1 and 1, which is a lot easier to work with:

float x = (rawX / 1023.0f) \* 2 - 1;

float y = (rawY / 1023.0f) \* 2 - 1;

## **Results**

At the end of the project, we had a functional radio that was capable of tuning to FM radio stations for extended periods of time, which made it practical for everyday use. In addition, the audio quality was not bad at all. The rotary encoder for changing between stations worked without any problems. The LCD display also showed relevant information, such as band, frequency, and preset number without any problems; all of the code written for it worked well. We did manage to get preset-tuning and preset-saving to work in the last week or two leading up to the final MakerSPACE presentation, although there were some problems involving it (explained a bit later). Also, throughout the development process, up until the end, the code was still well organized, and didn’t get too untidy.

However, there were some persisting issues that lasted until the end of the project. For one thing, every time the LCD screen updated, interference could be heard through the speakers (a static sound). This was not really noticeable when the screen contents changed periodically, but whenever the contents changed frequently (such as when text was scrolling), it would become really noticeable. We were never able to fix this. In addition, there was a similar type of interference issue with the joystick. Whenever moving the joystick (especially in the direction defined as preset 2 in our code), a loud buzzing noise could be heard, and would continue to play until the joystick was let go.

In addition to that, the joystick for some reason interfered significantly with any I2C communication, and we never figured out why. We tried including capacitors in our circuit to regulate the problem, but they didn’t have any effect on the issue. Since the joystick would interfere with I2C, sometimes missing or strange characters would appear on the LCD screen or the code would freeze. We knew the code froze because we sprinkled our code with Serial.println() calls to isolate the issue, and at one point, it would stop printing at a Serial.println() call that was right before any Wire-related Arduino function. To confirm that it was the joystick causing the problem (and not some other component), we changed the code to switch radio stations every one second instead of changing them via the rotary encoder, and commented out all code involving the joystick. However, we still kept the joystick connected. Under these conditions, whenever the radio station would stop switching, it would always happen when the joystick was being moved around.

One last problem with the radio was that the volume knob can’t be turned all the way, or very bad distortion would occur.

As for the box, many compromises after printing had to be made. According to Claudio (who printed part of the radio box at his house and whose brother’s friend printed another part of the radio box at his house), most of the measurements that we had made for fitting everything in the box were wrong. This meant that he and his brother had to come up with creative solutions for getting everything in the box. For instance, our radio’s handle holes (holes at the top to easily hold the radio) turned out to be the holes through which the antenna and Arduino ports were accessible, as the real holes we made for them were not positioned properly. Also, the breadboard didn’t fit lying flat on the radio’s floor, which was the initial plan, so it had to be placed sideways against the back. Also, the little shelves that were made to place the speakers on were also not positioned properly, and so they had to be broken off. Also, although our initial plans were to screw all the components in, we ended up hot gluing most of them instead to save on time. To save on more time, the print job was done at twice the intended speed so that the box would be ready on presentation day. This resulted in a battered-up box. However, despite all these drawbacks, the radio was still entirely functional.

## **Comparing with Initial Objectives**

Compared to the initial objectives, there were some features that were not achieved for the final product. For instance, the final radio did not support AM or shortwave bands, since we did not have time to try out the TEF6686 radio module, which supported these features. There was also no radio station scanning, RDS, or Bluetooth connectivity. Since there wasn’t enough time to design a PCB for the components, we had to stick to the breadboard, which took a lot of space inside the chassis. Besides these lacking features though, everything else we had planned on was included.

## **Discussion and Conclusion**

Overall, the project was a lot of fun to work on, and we learnt a lot throughout the semester, such as how to use Git and GitHub, libraries, VS Code, C functions like sprintf(), and EEPROM. Also, we learnt how to package components into a box. By the end, we were really excited that we had created our very own radio.

Most of the successes occurred early on in the project, when we were testing the radio features before packaging everything in a box. Adding more code and testing it went mostly smoothly. I think this was because (like mentioned earlier), James already had prior C++ knowledge before starting the project, and Abdulrahman had also been learning Python on his own for some time. In addition, Claudio learnt some programming in the engineering physics class, so all of us had the knowledge necessary to think in a logical, problem-solving manner.

Many of the challenges occurred much later in the project, mostly in the last week leading up to the presentation, which was the week when the final box was designed, printed, and altered. None of us had printed 3D models made to house various components that needed to interact with each other, so we guessed that this part was going to be challenging. Our guess was correct, as many last-minute changes and compromises had to be made due to the bad measurements we had taken. However, while running into mistakes due to measurements is normal, as many of the placements were eyeballed, what really affected us the most was the limited time to print the enclosure and adjust the measurements. There was basically no time to print multiple versions and get the optimal radio enclosure to house all the parts in a neat way, which even the most skilled mechanical engineers would do. So for a first version that was printed in half the required time, it still managed to hold all of the components, and even had a unique look.

Once again, at the beginning of the Winter 2025 semester, we had set an objective for ourselves to build a portable AM and FM radio that included features such as preset saving, scanning, RDS, and Bluetooth compatibility. The idea to build a radio came from James wanting to do a project involving audio and from watching many YouTube videos of others building Arduino radios. Then, throughout the course of the semester, we gradually obtained more and more hardware to continue expanding upon our radio, adding features as we went. As more features were added, the codebase was split into separate files to maintain control, organization, and scalability. Libraries also facilitated the development process. By the end of the project, we had built a fully functional Arduino radio. Although it did not include every feature that we had planned to implement, we were still really proud of what we created. Not only that, but we also gained much knowledge and skill that can be applied for future projects.

Some possible improvements for the project include those already mentioned that did not make the final build (i.e. AM, shortwave, scanning, RDS, Bluetooth compatibility). In addition, it could be interesting to experiment with different box designs, such as one made out of wood. Moreover, we are curious as to how we would go about making a settings menu that supported changing audio settings, such as bass, treble, etc. We’re not sure as to how we could get that to work, since it seems like achieving this feature would mean routing the audio signal through the Arduino, which doesn’t seem feasible since we don’t think the Arduino can process audio signals fast enough for it to still be perceived as good sound.

## **References**

**Link to the repository:**

<https://github.com/james-dire-1/arduino-radio-project>

**Libraries:**

[1] Bill Perry, “hd44780: Extensible hd44780 LCD library,” *GitHub*, 2016. https://github.com/duinoWitchery/hd44780/tree/master (accessed Jun. 01, 2025).

[2] Matthias Hertel, “RotaryEncoder: RotaryEncoder Arduino Library,” *GitHub*, Jul. 18, 2022. https://github.com/mathertel/RotaryEncoder/tree/master (accessed Jun. 01, 2025).

**Inspiration sources:**

[1] mircemk, “Simple TEF6686 Arduino + PC AM-FM Radio Receiver,” *Instructables*, Mar. 31, 2023. https://www.instructables.com/Simple-TEF6686-Arduino-PC-AM-FM-Radio-Receiver/ (accessed Jun. 01, 2025).

[2] pu2clr, “All Band Receiver With SI4732 / SI4735 (FM / RDS, AM and SSB) With Arduino,” *Instructables*, Mar. 26, 2020. https://www.instructables.com/All-Band-Receiver-With-Si4735-FM-RDS-AM-and-SSB-Wi/ (accessed Jun. 01, 2025).

[3] Switch & Lever, “Building a MIDI Controller Using Arduino,” *YouTube*, Dec. 03, 2019. https://www.youtube.com/watch?v=JZ5yPdoPooU (accessed Jun. 01, 2025).

[4] GreatScott!, “Make your own FM Radio - Part 1,” *YouTube*, Mar. 06, 2016. https://www.youtube.com/watch?v=RqyhvlMKt14 (accessed Feb. 13, 2024).

[5] EDISON SCIENCE CORNER, “How To Make FM RADIO using Arduino | TEA5767 | giveaway results,” *YouTube*, Aug. 16, 2024. https://www.youtube.com/watch?v=Xy7k\_ZvcVhU (accessed Jun. 01, 2025).

[6] Kevin Darrah, “Simple & Cheap Arduino FM Radio! - Tutorial + Code,” *YouTube*, Dec. 17, 2016. https://www.youtube.com/watch?v=n1hPj2wfsnA (accessed Jun. 01, 2025).

**Instructional/learning sources:**

[1] How To Mechatronics, “How I2C Communication Works and How To Use It with Arduino,” *YouTube*. Oct. 05, 2015. Accessed: Jun. 01, 2025. [YouTube Video]. Available: <https://www.youtube.com/watch?v=6IAkYpmA1DQ>

[2] DroneBot Workshop, “Using LCD Displays with Arduino,” *YouTube*. Mar. 19, 2018. Accessed: Jun. 01, 2025. [YouTube Video]. Available: <https://www.youtube.com/watch?v=wEbGhYjn4QI>

[3] DroneBot Workshop, “Using Rotary Encoders with Arduino,” *YouTube*, Mar. 23, 2019. https://www.youtube.com/watch?v=V1txmR8GXzE (accessed Jun. 01, 2025).

[4] amandaghassaei, “Powering Arduino With a Battery,” *Instructables*. https://www.instructables.com/Powering-Arduino-with-a-Battery/ (accessed Jun. 01, 2025).

[5] Abstract programmer, “How to program Arduino with VSCode,” *YouTube*, Aug. 13, 2022. https://www.youtube.com/watch?v=gQ2lsSuXvVU (accessed Jun. 01, 2025).

[6] “A guide to EEPROM,” *Arduino Docs*, Jun. 19, 2024. https://docs.arduino.cc/learn/programming/eeprom-guide/ (accessed Jun. 01, 2025).

[7] The Cherno, “C++ Series,” *YouTube*. https://www.youtube.com/playlist?list=PLlrATfBNZ98dudnM48yfGUldqGD0S4FFb (accessed Jun. 01, 2025).

## **Acknowledgements**

Thank you to Abdulrahman’s high school robotics team for allowing him to use software such as Onshape regularly, which allowed us to design the radio box ourselves.

Thank you to Claudio’s brother and his friend William for printing our radio, and thank you again to Claudio’s brother for helping come up with solutions to the incorrect measurements we had made.

Thank you to all the instructional/learning sources and libraries listed above. Every one of them helped the radio project come to life.

Thank you to all the inspiration sources also listed above. Without them, we may have not even built a radio.

And a huge thank you to Joel Trudeau and Manuel Zpata for mentoring the MakerSPACE internship, and thank you to Dawson College for covering some of the costs for hardware. The Friday meetings were really the highlight of the school week!

And thank you to everyone who showcased their projects during the final presentation, effectively encouraging and inspiring everyone to continue working on electronics projects, even after the internship has concluded.