Republic of Iraq
Ministry of Higher Education
and Scientific Research
University of Misan
College of Engineering
Department of Electrical Engineering

Internet World Radio

A graduation project submitted to the **Department of Electrical Engineering**, in partial fulfillment for the requirements for the award of the degree of Bachelor of **Electrical Engineering**

By

Sajjad Hisham Noori

Hussein Khaled Wahwed

Noor Ali Jasem

Ali Abdulzahra

Ahmed Ali Aboud

SUPERVISED BY: Assist Lec. Yasir Ali Khaled

Maysan, Iraq

2025

بسم الله الرحمن الرحيم

((وَلَقَدْ آتَيْنَا دَاوُودَ وَسُلَيْمَانَ عِلْمًا ۖ وَقَالَا الْحَمْدُ لِلَّهِ الَّذِي فَضَّلَنَا عَلَىٰ كَثِيرٍ مِّنْ عِبَادِهِ الْمُؤْمِنِينَ))النمل اية(١٥)

صدق الله العلي العظيم

SUPERVISOR CERTIFICATION

I certify that Sajjad Hisham Noori, Hussein Khaled Wahwed, Noor Ali Jasem, Ali Abdulzahra, and Ahmed Ali Aboud prepared the project titled “Internet World Radio” under my supervision at the General Electrical Engineering Branch, Electrical Engineering Department, University of Misan. This project was partially fulfilled to meet the requirements for the degree of Bachelor of Science in Electrical Engineering.

Supervisor : Assist Lec. Yasir Ali Khaled

Signature:

Scientific Degree:

Date:

DEDICATION

To All parents and to our families who made this accomplishment possible.

ACKNOWLEDGEMENT

*Praise be to **ALLAH** who enabled us to complete this work under his benediction.*

*Cordial thanks and deepest gratitude to our supervisor (**Mr. Yasir Ali Khaled**) for being assistance and for the continuous encouragement and suggestions from him which helped us to successfully complete this project.*

*In addition, we would like to express my thanks to the head of the Electrical Engineering Department (**Dr. Mohammed Khalaf Jumaah**) for his continuous support and encouragement.*

We would like to express our thanks to our University for their encouragements and anyone helped us.

Finally, sincere gratitude and appreciation and love to our families for their encouragements and supports and for all things they have given us.

ABSTRACT

This research presents the design and implementation of an interactive internet radio system that combines the classic look of vintage radios with modern digital streaming technologies. The project transforms the traditional radio experience by integrating a Raspberry Pi 4, rotary encoders, an LCD screen, and a web interface based on Leaflet.js, allowing users to explore and stream global radio stations through an interactive map. A JSON-based database of stations provides efficient data management, while physical controls offer a nostalgic and tactile experience.

The system overcomes common problems of traditional AM/FM radios, such as signal interference and geographical limits, by providing high-quality internet streaming and easy navigation through locations on the map. The software, which includes HTML, JavaScript, and Python scripts, handles station data, user interactions, and live playback. The hardware integration ensures smooth communication between user inputs and dynamic visual feedback.

Through extensive testing, the system's reliability, responsiveness, and ease of use were confirmed. Challenges such as station overlap, broken links, and electrical interference were addressed with technical solutions like real-time validation, zoom limitations, and improved grounding. This project shows the potential of combining older technologies with modern systems to create new and useful devices.

Table of Figures

TABLE OF CONTENTS

Chapter One

Introduction

1.1 Introduction

Technological Development in Radio

In recent decades, technological advancement has reshaped every aspect of human life, profoundly influencing how individuals communicate, access information, and engage with global culture. Among the most transformative technologies in the realm of communication is radio, which has evolved from rudimentary wireless signaling into a multifaceted medium delivering news, entertainment, and educational content to audiences worldwide.

Initially developed in the late 19th century as a method of transmitting electromagnetic signals, radio became a cornerstone of mass communication throughout the 20th century. Its role expanded from military and maritime applications to public broadcasting, fostering social connectivity and public awareness on an unprecedented scale (Britannica, 2023).

The trajectory of radio's development mirrors broader global trends in digitalization. The transition from analog to digital broadcasting has significantly enhanced audio quality, transmission reliability, and user interactivity. Furthermore, the rise of internet-based radio has dismantled geographic boundaries, enabling users to access a global network of stations in real time (Techopedia, 2022).

This evolution demonstrates radio's adaptability and continued relevance. Even in an era dominated by on-demand media and smart devices, radio persists as a dynamic platform—now augmented with digital interfaces, streaming protocols, and user-personalized services—cementing its role as both a cultural artifact and a forward-looking communication technology (IEEE Spectrum, 2021).

1.1.1 Radio: From Invention to Global Spread

The origins of radio trace back to the foundational work of physicist Heinrich Hertz, who, in the late 19th century, confirmed the existence of electromagnetic waves—an essential prerequisite for wireless communication. This scientific milestone was followed by the pioneering efforts of Guglielmo Marconi, who successfully developed and demonstrated the first practical wireless telegraphy system in the 1890s. Marconi's work laid the technological foundation for the modern radio by enabling long-distance signal transmission without the need for physical connectors (Hong, 2001; IEEE History Center, 2023).

By the early 20th century, radio technologies had evolved beyond scientific experimentation and military use to become tools of mass communication. The widespread adoption of vacuum tube amplification enabled stronger and clearer audio signals, accelerating the commercial viability of radio broadcasting. The 1920s and 1930s marked the beginning of public radio as a household staple, with stations delivering real-time news, music, and cultural programming to growing audiences around the world (Sterling & Kittross, 2002).

Radio quickly became a unifying medium, fostering national identity, informing the public, and offering entertainment on an unprecedented scale. In wartime, it served as a critical communication tool; in peacetime, it helped shape societal values and promote cultural cohesion. As broadcasting regulations were standardized and infrastructures expanded, radio emerged as a globally accessible platform, influencing politics, education, and public discourse throughout the 20th century (Briggs & Burke, 2009).

1.1.2 Different Eras of Radio

The evolution of radio as a technology and cultural medium can be classified into three distinct eras—each characterized by significant advancements in electronic engineering and shifts in user experience: the Vacuum Tube Era, the Transistor Era, and the Digital Radio Era.

- **Vacuum Tube Era (Early 20th Century)**

The first radios utilized vacuum tubes (thermionic valves) to amplify and detect audio signals. These devices, while bulky and power-hungry, enabled long-distance signal transmission and were primarily employed in military and commercial contexts. By the 1920s, vacuum tube radios were introduced into the consumer market, albeit at high cost and limited portability. Nevertheless, they marked the beginning of mass media broadcasting and laid the foundation for modern public radio infrastructure (Fagen, 1975).

- **Transistor Era (1950s–1970s)**

The invention of the transistor in 1947 and its rapid adoption in consumer electronics during the 1950s revolutionized radio technology. Transistor-based radios were smaller, cheaper, more energy-efficient, and highly portable—characteristics that made them immensely popular among consumers. The widespread use of these radios expanded access to information and entertainment, especially in automobiles and remote locations, further embedding radio into the fabric of daily life (Riordan & Hoddeson, 1997).

- **Digital Radio Era (Late 20th Century – Present)**

The emergence of digital signal processing in the late 20th century ushered in a new era of broadcasting. Digital Audio Broadcasting (DAB), introduced in the 1990s, offered higher fidelity, reduced noise and interference, and the ability to transmit additional data such as station metadata, song titles, and traffic updates. Furthermore, digital platforms facilitated the integration of radio with emerging technologies, enabling interactive features and on-demand services (WorldDAB, 2023). This transformation redefined the way audiences engage with audio content and broadened the scope of radio beyond linear programming.

1.1.3 Modern Technology and Radio Development

The ongoing advancement of digital technologies has significantly reshaped the radio landscape, transitioning it from traditional broadcast infrastructure to a globally connected, user-driven ecosystem. One of the most profound changes has been the emergence of internet radio, which enables listeners to stream audio content from virtually any location with internet access. This innovation effectively removes the geographic limitations imposed by terrestrial broadcasting, allowing users to explore thousands of stations worldwide in real time (Techopedia, 2022).

Unlike analog radio, which is constrained by frequency allocation and atmospheric interference, internet radio offers high-fidelity audio and the capacity to integrate advanced metadata, such as track information, station descriptions, and interactive features. Platforms like TuneIn, iHeartRadio, and Spotify Radio exemplify how radio has merged with streaming technology to offer personalized content recommendations and on-demand playback

(Gannes, 2011).

Moreover, the integration of radio services into smart devices—including smartphones, tablets, smart speakers (e.g., Amazon Echo, Google Nest), and vehicle infotainment systems—has increased both accessibility and user engagement. These smart systems often employ voice control, cloud connectivity, and AI-driven personalization, thereby transforming radio from a passive listening experience into a dynamic, user-centered interaction

(Statista, 2023).

This convergence of internet connectivity and smart technology illustrates how radio continues to adapt and remain relevant in a competitive digital media environment. Rather than becoming obsolete, radio has evolved into a hybrid communication platform that bridges the nostalgia of traditional audio broadcasting with the interactivity and convenience of modern digital ecosystems.

1.1.4 Preserving Cultural Heritage Through Technology

In the digital age, rapid technological advancement poses both challenges and opportunities for the preservation of cultural heritage. Traditional communication mediums, such as analog radio, risk obsolescence due to the proliferation of internet-based and on-demand platforms. However, through strategic integration and innovation, these historical technologies can be revitalized and reimagined to serve contemporary purposes while retaining their cultural value (UNESCO, 2023).

Radio, as a medium, has long been a vessel for the transmission of cultural narratives, local music, indigenous languages, and region-specific knowledge. With the decline in analog broadcasting, much of this intangible heritage is at risk of being lost. Internet radio offers a means to mitigate this loss by enabling the digitization and global distribution of culturally significant audio content. Moreover, it facilitates access to minority voices and underrepresented communities that are often excluded from mainstream media platforms (Barrowclough & Kozul-Wright, 2018).

Projects that combine legacy radio hardware with modern digital technologies—such as streaming protocols, interactive interfaces, and geographic mapping—offer a compelling approach to cultural preservation. These hybrid systems not only extend the functionality of old devices but also engage new generations of users in discovering, valuing, and participating in cultural expression through a familiar yet enhanced medium.

Thus, the convergence of innovation and preservation becomes a pathway for sustaining heritage in the digital era. Rather than discarding obsolete technologies, recontextualizing them within current digital infrastructures contributes to both technological sustainability and cultural continuity.

1.2 The Internet Radio Project and Its Objectives

The Internet Radio Project represents an innovative response to the evolving nature of audio media in the digital era. Rooted in the objective of modernizing traditional radio experiences, the project integrates classical hardware aesthetics—such as rotary knobs and vintage enclosures—with advanced digital functionalities to deliver a hybrid listening platform. This approach not only preserves the nostalgic value of radio as a cultural artifact but also enhances user experience through interactive and customizable features enabled by internet connectivity.

At its core, the project envisions a globally accessible radio system powered by a compact computing platform (e.g., Raspberry Pi) and complemented by intuitive physical controls, a visual interface, and a wide range of international content. By bridging analog user interaction with digital broadcasting infrastructure, the Internet Radio Project fosters both functional innovation and cultural inclusivity.

Project Objectives:

- Global Content Accessibility:

Provide users with seamless access to thousands of radio stations worldwide, allowing for enriched entertainment and exposure to diverse musical, linguistic, and cultural content.

- Interface Hybridization:

Merge traditional control mechanisms (e.g., rotary encoders) with modern interface technologies to offer a familiar yet enhanced user experience.

- Cultural Exchange Promotion:

Encourage intercultural understanding by enabling users to explore regional programming from different parts of the world, thus broadening their global perspective.

- Interactive Navigation:

Incorporate geographic visualization (such as an interactive world map) to simplify station selection and provide spatial context for audio content.

By achieving these objectives, the project seeks to reimagine radio as an educational, cultural, and technologically relevant medium for a global audience in the 21st century.

1.3 Project Idea

The World Radio Project seeks to bring radio into the digital age by reimagining the traditional radio experience through internet connectivity. The key components of the project include:

- Raspberry Pi 4: The primary computing platform that drives the device's processing capabilities.

- Rotary Encoders: Allowing users to easily navigate through stations and adjust volume, giving the experience of traditional manual controls.

- Interactive Screen: Displays station names and additional information, enhancing the user's experience and providing real-time feedback.

- PAM8406 Audio Amplifier: Provides high-quality sound output, ensuring an immersive audio experience.

- Dedicated 5V Power Supply: Ensures stable and reliable operation for all components.

By combining these elements, the project merges the tactile feel of physical interfaces with the flexibility of modern digital technology, offering an intuitive and engaging radio experience that is both nostalgic and innovative.

1.4 Advantages of the World Radio Project Compared to Classical Radio

The World Radio Project offers several key advantages over traditional radio, which is typically limited by geographical constraints and signal quality. Below is a comparison highlighting these differences:

Radio type	Classic radio	Internet world radio
Content Access	Limited to local stations	Global access to thousands of stations
Cultural Discovery	Limited exposure to regional content	Access to diverse cultures and languages
Geographical Reach	Restricted by signal range	Accessible worldwide via the internet
Sound Quality	Prone to interference and static	High-quality digital audio
Station Selection	Manual tuning required	Interactive map selection
Feature	Traditional Radio	World Radio Project

Additional Features:

- Interactive Global Map: The user can select radio stations based on geographical locations rather than frequency tuning, which enhances user engagement and simplifies navigation.

- Remote Communication and Information Access: In areas where traditional radio signals are weak, this project offers a reliable way to receive important updates, such as news, weather forecasts, and emergency information.

- Cultural Exploration: By allowing users to listen to stations from various countries, the project promotes global cultural awareness, helping users learn about different traditions, music, and languages.

Why Does This Project Matters ?

The World Radio Project combines the value of old radio designs with the power of modern digital technology. Instead of using traditional radio signals like AM or FM, it works through the internet to access thousands of radio stations from around the world. This allows people to listen to international content without limits of distance or borders. By adding digital features to vintage radios, the project keeps the classic feel of old devices while making them useful in today's digital world. It also helps preserve cultural identity and creates new ways for people to learn, communicate, and connect globally.

Chapter Two

Components of the Project

2.1 Components of the Project

The "World Radio" project relies on a combination of hardware and software components, each carefully selected to ensure functionality, reliability, and usability. Below is a detailed explanation of each component:

1. Raspberry Pi 4 (Model B, 8GB):

○ Processor and Memory: The Raspberry Pi 4 is powered by a 1.5GHz quad-core Broadcom BCM2711 processor, delivering strong performance that enables complex tasks like streaming Internet radio stations without lag. The 8GB of LPDDR4 RAM ensures smooth multitasking, improving overall performance, especially when browsing radio stations or interacting with the interactive map.

○ Storage and Connectivity: The Raspberry Pi 4 uses a microSD card to store the operating system and software, offering flexibility for upgrades or replacements. It supports internet connectivity via Wi-Fi 802.11ac and Bluetooth 5, allowing seamless access to online radio stations. If the wireless network is unstable, the Ethernet port provides a stable wired connection.

○ Display and Graphics: The Raspberry Pi 4 features dual HDMI ports supporting 4K displays, offering flexibility for showing an interactive map displaying radio stations worldwide. The VideoCore VI graphics processor enables smooth graphics playback, perfect for dynamic updates on the map while interacting with stations.

○ GPIO Control: The Raspberry Pi has 4 GPIO pins to connect devices such as rotary encoders, buttons, or sensors. This allows direct interaction with the map, controlling station playback, volume adjustments, and navigation, offering a flexible user experience.

○ Power and Applications: The Raspberry Pi 4 requires a 5V/3A USB-C charger for stable performance. With low power consumption, the Raspberry Pi 4 ensures long-term operation without power concerns. It supports operating systems like Raspbian and Ubuntu, making it ideal for the Global Internet Radio project, where developers can customize the system to their needs.

○ Programming: Raspberry Pi programming is primarily in Python, utilizing libraries like RPi.GPIO to operate devices connected to GPIO pins. Players such as VLC are used to stream radio stations. The Raspberry Pi OS is set up, with necessary libraries installed, while HTML/CSS/JavaScript web interface.

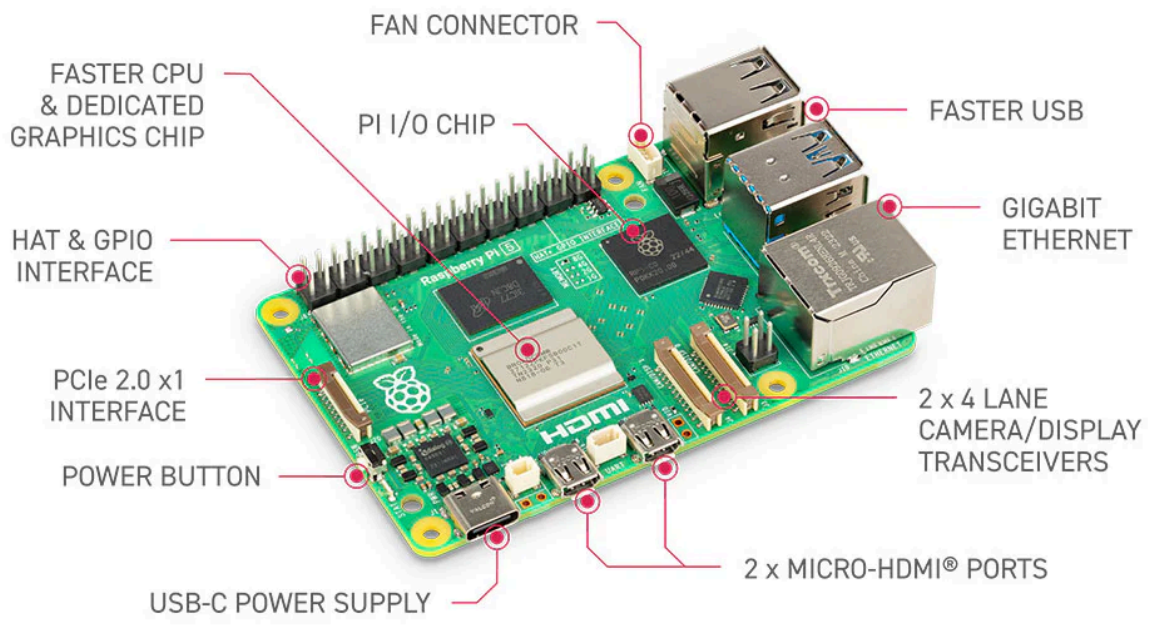

Fig 2.1

2. Micro SD Card (64GB): The microSD card serves as the storage medium for the Raspberry Pi, containing the operating system, software libraries, and project-specific applications. The 64GB capacity ensures enough space for system files, cached data, and potential updates. A high-speed card is used to ensure faster data access and quicker boot times.

3. 12-inch Display Screen: The 12-inch screen provides a user-friendly interface, displaying the interactive global map and relevant information about selected radio stations. The display is crucial for enhancing user experience and ensuring smooth navigation.

○ Resolution: Full HD (1920x1080 pixels) provides excellent clarity for displaying fine details such as a detailed map. Lower resolutions, like 1280x800 pixels, still provide a decent display but with less clarity.

○ Interface: the disply connects via HDMI, providing high-quality video and audio output for accurate, clear images.

4. Three Rotary Encoders: Rotary encoders detect rotational movement and direction, providing intuitive control over various functions like cursor movement and zoom adjustments. Each encoder connects to the Raspberry Pi through GPIO pins and uses software libraries to process signals, ensuring smooth and precise user interaction.

○ Technical Specifications:

■ Operating Voltage: 5V

■ Pulses per 360° Rotation: 20

■ Output: 2-bit gray code

■ Mechanical Angle: 360° continuous rotation

■ Built-in push button switch for additional functionality

○ Functionality: The rotary encoders generate pulses that indicate the direction and degree of rotation. By monitoring these pulses, the Raspberry Pi updates the cursor position or zoom level accordingly. The built-in push button toggles between different modes, such as coarse and fine adjustments.

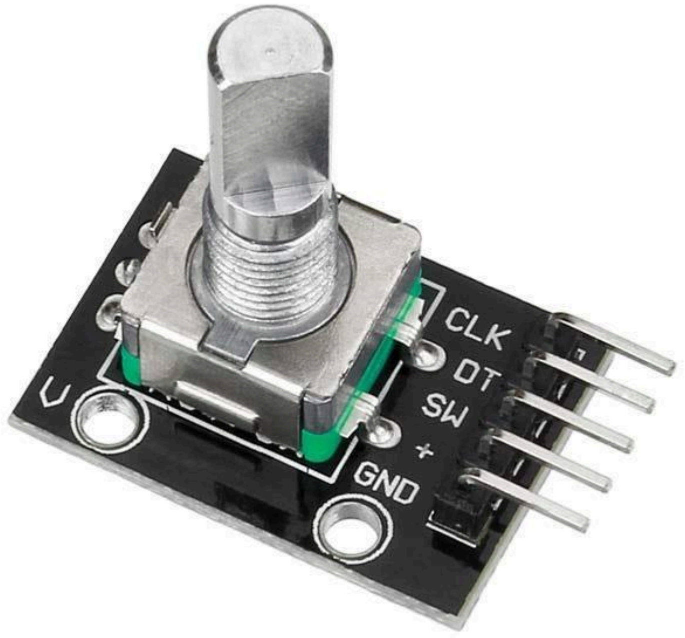

Fig 2.2

5. HDMI Cable: The HDMI cable connects the Raspberry Pi to the display, ensuring high-quality video output and a clear graphical interface.

6. Male and female Jumper Wires: These wires are used to connect electronic components, such as rotary encoders, to the Raspberry Pi directly or through the breadboard , providing reliable electrical connections while maintaining modular design flexibility.

7. Power Supply for Raspberry Pi: A dedicated power supply (5V Dc) ensures that the Raspberry Pi receives the correct voltage and current for stable operation.

8. Mouse and Keyboard (Temporary): A mouse and keyboard are used during the initial setup and programming of the

Raspberry Pi but are not required once the project is fully operational.

9. Power Supply for Display Screen: The display requires a separate power source (5V Dc) . A compatible power supply ensures consistent operation and optimal brightness for the user interface.

10. AUX Cable: The AUX cable connects the Raspberry Pi to the amplifier, enabling high-quality audio output for the radio stations, ensuring clear and uninterrupted broadcasts.

11. 5V Power Supply for the Amplifier: The PAM8406 audio amplifier is powered by a dedicated 5V power supply, connected via two wires (positive and negative) to ensure stable and reliable operation.

12. External Speakers and Amplifier: The project uses a PAM8406 audio amplifier with a volume control rotary controller that includes an on/off switch, along with 6-watt stereo speakers. This setup provides clear, loud, and adjustable sound for a better listening experience. The PAM8406 is a small and efficient amplifier that works with 2.5V to 5.5V power and gives up to 5 watts per channel. It can run in two modes: Class-D for lower heat and better power saving, and Class-AB for higher sound quality. It also has a built-in system to reduce noise and make the sound cleaner. This makes it a great choice for Raspberry Pi projects like the interactive radio.

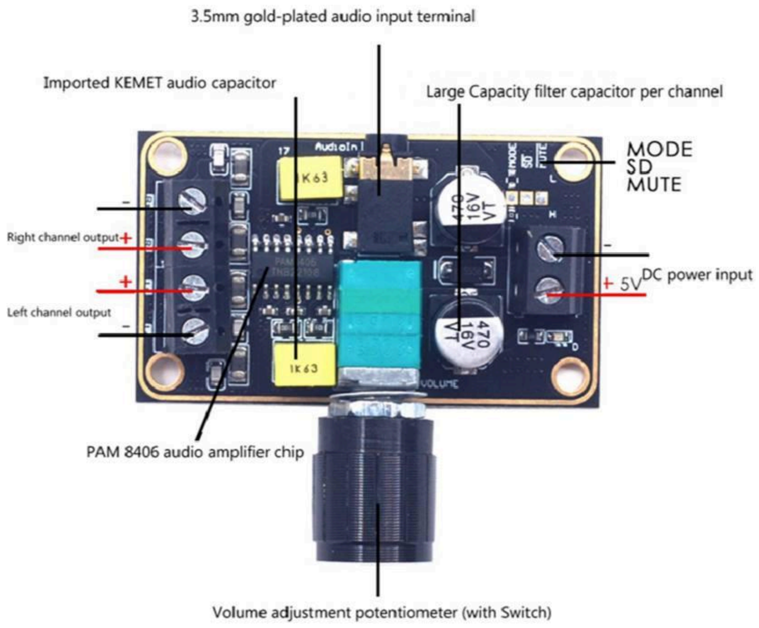
3.5mm gold-plated audio input terminal
Imported KEMET audio capacitor
Large Capacity filter capacitor per channel
MODE
SD
MUTE
Right channel output
Left channel output
5V DC power input
PAM 8406 audio amplifier chip
Volume adjustment potentiometer (with Switch)

Fig 2.3

Fig 2.4

2.2 Circuit Design

- The Raspberry Pi is connected to the screen via an HDMI cable.

- Rotary Encoder Connections: The three rotary encoders are connected directly to the Raspberry Pi using female-to-female jumper wires for the DT (Data) and CLK (Clock) pins. The + (VCC) and GND pins are connected to the Raspberry Pi through a breadboard, using male-to-female jumper wires to common power and ground points, ensuring a stable and organized wiring setup.

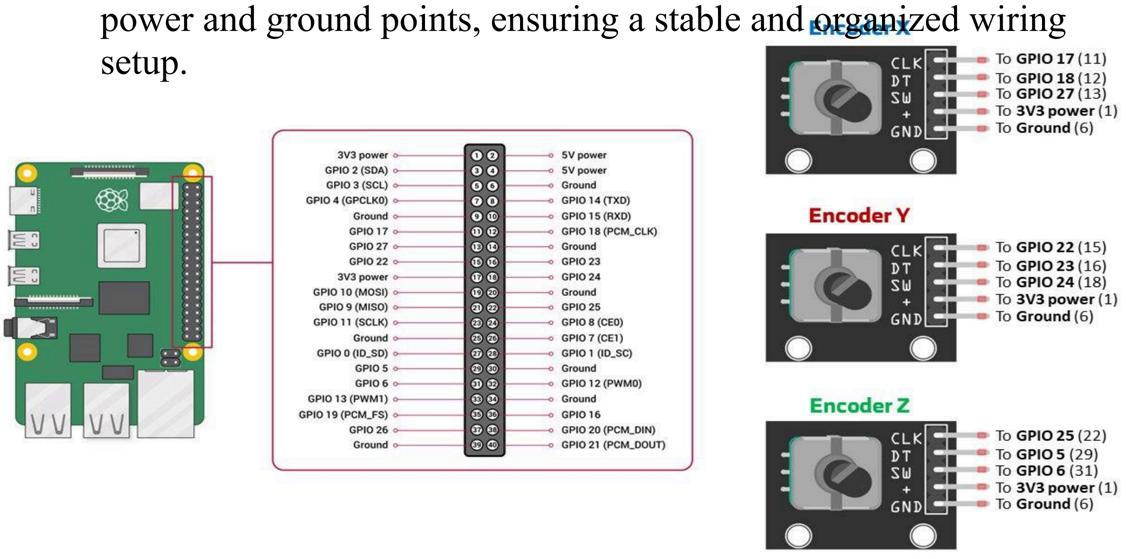

Fig 2.5

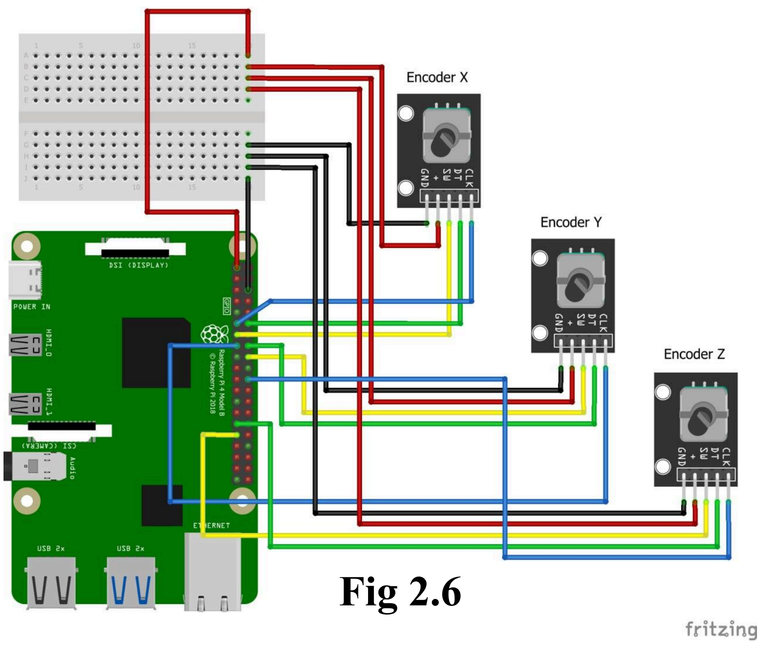

Fig 2.6

- Audio System and Connectivity: The PAM8406 audio amplifier chip connects to the Raspberry Pi via the 3.5mm audio jack. The amplifier outputs sound to the 6-watt speakers through two wires (positive and negative), ensuring a stable and clear audio transmission.

- Power Supply and Connectivity: The Raspberry Pi, screen, and amplifier each have separate power supplies. The Raspberry Pi and the screen are powered by individual 5V chargers. The PAM8406 audio amplifier is also powered by a 5V charger, connected via two wires (positive and negative) to ensure stable operation.

2.3 Integration of Components

Each component in the "World Radio" project is carefully integrated to create a cohesive and functional system. The Raspberry Pi coordinates input from the rotary encoders, manages internet connections, and streams radio stations. The display screen serves as the visual interface, while the amplifier delivers high-quality audio. The connections between these components are optimized for reliability and ease of use. This modular approach allows for future enhancements or modifications, ensuring the project can adapt to new requirements or technological advancements.

Chapter Three

Implementation and Design

3.1 The Role of the Software Side in the Project

The software side of the project serves as the foundation, enabling interaction between various components to accomplish the overall system goals. In this interactive radio project, the software components manage the interface through which users can explore and stream radio stations worldwide. The main function of the software is to facilitate dynamic interaction with the map interface, provide seamless data management, and manage the backend processes to ensure real-time updates of the stations and streaming audio.

The system is composed of three key components: the JSON file, the HTML interface, and the Python script. The JSON file holds the station data, while the HTML interface is responsible for displaying the map and stations on it. The Python script controls the rotary encoders for user interaction, adjusting the cursor position and map zoom levels, with the additional feature of automatically starting or stopping the audio stream based on the cursor's location. By working in tandem, these elements create an intuitive, seamless user experience that enhances the ability to browse and listen to radio stations interactively.

Additionally, the project incorporates an innovative feature where radio station streams are triggered based on the cursor's location on the map. As the cursor enters a station’s coverage area (represented as a boundary on the map), the audio stream for that station begins. When the cursor exits the area, the stream automatically stops. This functionality is managed using three rotary encoders: one for the X-axis, another for the Y-axis, and a third for zoom control, providing users with full navigational control over the interactive map and the audio streams.

3.2 Key Components of the Code

The codebase for this project consists of several key files and scripts, each serving a specific function within the system. These components include:

1. stations.json (JSON file containing the station data).

2. radiomap.html (HTML file for rendering the map and integrating JavaScript logic).

3. rotary_encoder.py (Python script for controlling cursor movement with rotary encoders).

Each of these components is described below, detailing their functionality and contributions to the overall project.

3.2.1 JSON File Code (stations.json)

The stations.json file serves as the central data repository for the interactive internet radio system, storing structured information about various radio stations across the globe. Each entry in this JSON file corresponds to a particular station and contains metadata such as geographic coordinates (latitude and longitude), streaming URLs, and the associated city name.
The use of JSON (JavaScript Object Notation) is justified by its lightweight nature, ease of parsing, and human-readable structure, making it highly suitable for real-time web applications.

Example of the stations.json format:

```
{
  "Beirut - Lebanon": {
    "coords": {
      "n": 33.8938,
      "e": 35.5018
    },
    "urls": [
      {
        "name": "Radio Liban",
        "url": "http://radioliban.ice.infomaniak.ch/radioliban-128.mp3"
      }
    ]
  }
}
```

3.2.2 Interface and Processing Code (radiomap.html)

The radiomap.html file is a key part of the project, acting as a link between the radio station database (stations.json) and the rotary encoder input (rotary_encoder.py). It is responsible for fetching, processing, and displaying data, ensuring a smooth and dynamic user experience. The file is built using HTML, JavaScript, and the Leaflet.js library to create an interactive, map-based interface with real-time updates.

The process within the file consists of three main stages:

1. Fetching:
The system retrieves the necessary data from an external JSON file (stations.json), which contains the details of the radio stations. The fetch() function in JavaScript is used to get the data asynchronously, meaning it does not affect the user interface, ensuring high performance and fast data retrieval.

2. Processing:
After the data is fetched, it is validated. The system checks that the audio links are working correctly. It also maps each station's location based on geographical coordinates. Additionally, the map's theme is automatically adjusted between day and night mode depending on Iraq's local time, providing an engaging and relevant user experience.

3. Rendering:
The validated data is then displayed using the Leaflet.js library. The map shows the stations according to their locations, with the nearest station appearing in the center of the map. A popup appears when users interact with a station, displaying detailed information about the selected radio station.

This smooth integration of fetching, processing, and rendering ensures users can interact with the system easily and intuitively, providing them with an engaging and dynamic experience while exploring available radio stations based on their location.

1. Data Fetching

The first step in the interface's workflow is retrieving station data from an external JSON file (stations.json). This file contains critical information such as:

- Station Name
- Streaming URL
- Latitude and Longitude (to position the station on the map)
- Coverage Radius (defining the interactive area for each station)

The JavaScript fetch() function asynchronously loads this data to ensure that the map remains responsive and does not freeze during data retrieval.

```
fetch("stations.json")
  .then(response => response.json()) // Convert response to JSON
  .then(data => {
    stations = data; // Store station data globally
    initializeMap(); // Call function to render stations
  })
  .catch(error => console.error("Error loading stations:", error));
```

How It Works:

- The fetch() function sends an HTTP request to load stations.json.
- The data is converted into a JavaScript object (response.json()).
- The parsed data is stored in the stations variable for later use.
- If fetching fails, an error message is displayed in the console.

By handling data asynchronously, the webpage remains fast and smooth.

2. Data Processing

Once the data is fetched, the system validates the information to ensure that:
- The audio stream links are functional.
- Each station has valid geographic coordinates.
- The map adjusts its theme dynamically.

❖ Audio Stream Validation

To avoid broken links, the system checks if each station's streaming URL is accessible before displaying it on the map.

```
function checkStream(url) {

  return fetch(url, { method: "HEAD" })

    .then(response => response.ok)

    .catch(() => false);

}
```

❖ Day/Night Mode Adaptation

To enhance usability, the map automatically switches between day mode and night mode based on Iraq’s local time.

```
function setMapTheme() {

  let hour = new Date().getHours();

  let isNight = hour < 6 || hour > 18; // Night mode from 6 PM to 6 AM

  let mapStyle = isNight ? "dark-vintage" : "light-vintage";


L.tileLayer(`https://api.mapbox.com/styles/v1/{mapStyle}/tiles/256/{z}/{x}/{y}`
, {

    attribution: "© OpenStreetMap contributors"

  }).addTo(map);

}
```

- Between 6 PM and 6 AM, the map switches to dark mode for better night visibility.

- During the daytime, it returns to a bright theme for clarity.

This feature improves readability and provides a visually appealing experience.

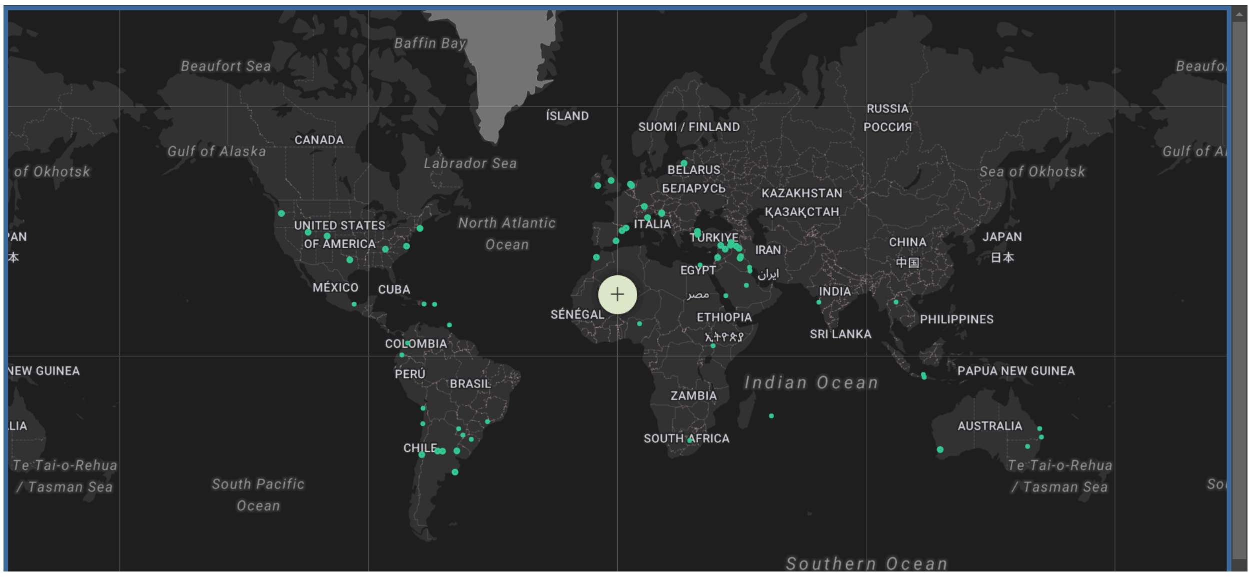

Fig 3.1

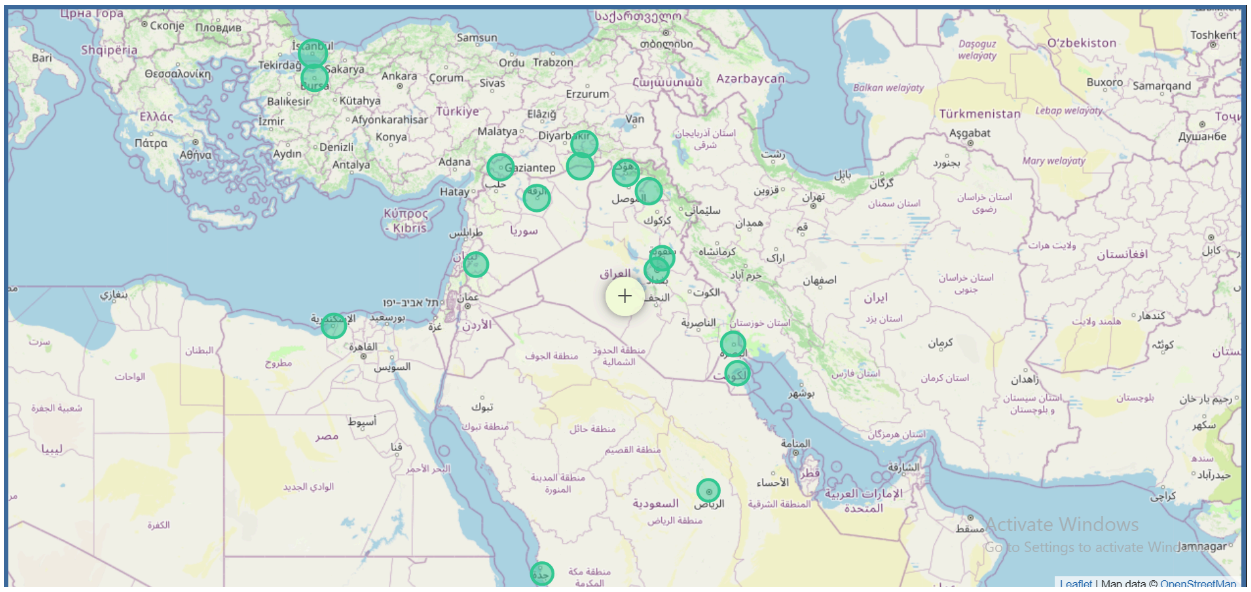

Fig 3.2

3. Data Rendering (Map Visualization)

The final step is converting the processed station data into a visually interactive map using Leaflet.js.

❖ Displaying Stations on the Map

```
function displayStations() {

  stations.forEach(station => {

    let marker = L.marker([station.latitude, station.longitude]).addTo(map);
    marker.bindPopup(`
      <b>${station.name}</b><br>
      <audio controls>
        <source src="${station.stream_url}" type="audio/mpeg">
      </audio>
    `);
  });
}
```

How It Works:

- Each station's latitude and longitude are used to position markers on the map.
- Clicking a marker opens a popup displaying the station name and an embedded audio player.
- This allows users to listen to the station directly from the map.

❖ Cursor Interaction and Auto-Playback

The system also highlights the nearest station based on the cursor's position.

```
function updateNearestStation() {
  let closestStation = null;
  let minDistance = Infinity;

  stations.forEach(station => {
    let distance = getDistance(cursorLat, cursorLon, station.latitude,
station.longitude);
    if (distance < station.radius && distance < minDistance) {
      minDistance = distance;

      closestStation = station;

    }

  });


  if (closestStation) {

    document.getElementById("station-info").innerHTML = `Now Playing:
${closestStation.name}`;

    playStation(closestStation.stream_url);

  } else {

    stopPlayback();

  }

}
```

● The system calculates the cursor's distance from each station.

● If the cursor enters a station's coverage area, it automatically starts playing the stream.

● If the cursor exits all coverage areas, playback stops.

The radiomap.html file seamlessly integrates data retrieval, processing, and rendering to create an intuitive user experience. By combining efficient data handling, dynamic theme adaptation, and interactive map rendering, the system ensures that users can explore and listen to online radio stations effortlessly.

3.2.3 Encoder Code (rotary_encoder.py)

The rotary_encoder.py script is responsible for controlling the cursor movement on the interactive map. It utilizes three rotary encoders to allow precise navigation in four directions (left, right, up, and down), along with zooming in and out. The movement is based on the rotation direction of each encoder, and the cursor updates dynamically on the map.

Rotary Encoder Functionality:

Each rotary encoder controls a specific aspect of movement:

1. Encoder 1 (Horizontal Movement - Left/Right)
Rotating clockwise moves the cursor to the right (increases longitude).
Rotating counterclockwise moves the cursor to the left (decreases longitude).

2. Encoder 2 (Vertical Movement - Up/Down)
Rotating clockwise moves the cursor upward (increases latitude).
Rotating counterclockwise moves the cursor downward (decreases latitude).

3. Encoder 3 (Zoom Control - Zoom In/Out)
Rotating clockwise zooms in (increases map scale).
Rotating counterclockwise zooms out (decreases map scale).

1. Library Dependencies

```
import RPi.GPIO as GPIO

import uinput

import pyautogui

import time
```

The script begins with importing four essential libraries:

- RPi.GPIO: Provides access to the Raspberry Pi GPIO pins, allowing the script to interface with the physical encoders through digital input readings.

- uinput: A Linux subsystem that enables user-level programs to emulate input events (e.g., keyboard, mouse, joystick). This is critical for injecting cursor movement and scrolling events into the graphical user interface.

- pyautogui: A high-level library used to simulate GUI interactions. Here, it is used to perform a mouse click that focuses the map window, ensuring it is ready to receive navigation input.

- time: A standard Python library utilized to introduce timed delays, thus controlling the input rate and preventing excessive event triggering.

2. Encoder Pin Configuration

```
ENCODERS = {
   'x': {'clk': 17, 'dt': 18},
   'y': {'clk': 22, 'dt': 23},
   'zoom': {'clk': 25, 'dt': 5}
}
```

This dictionary defines the mapping between logical encoder functions and physical GPIO pins on the Raspberry Pi. Each encoder provides two output signals—CLK and DT—which together form a quadrature encoding system. The axes are assigned as follows:

- X-axis Encoder: Controls horizontal cursor movement using GPIO pins 17 (clk) and 18 (dt).

- Y-axis Encoder: Controls vertical cursor movement using GPIO pins 22 (clk) and 23 (dt).

- Zoom Encoder: Controls zoom-in and zoom-out behavior using GPIO pins 25 (clk) and 5 (dt).

This modular configuration allows scalable and flexible integration of multiple encoders.

3. GPIO Initialization

```
GPIO.setmode(GPIO.BCM)

for axis, pins in ENCODERS.items():

   GPIO.setup(pins['clk'], GPIO.IN, pull_up_down=GPIO.PUD_UP)

   GPIO.setup(pins['dt'], GPIO.IN, pull_up_down=GPIO.PUD_UP)
```

The script initializes the GPIO pins using the BCM (Broadcom SoC) numbering scheme. Each encoder pin is configured as a digital input with an internal pull-up resistor (GPIO.PUD_UP) to maintain a defined voltage level and prevent floating states when the encoder is idle.

4. UInput Device Initialization

```
device = uinput.Device([

   uinput.KEY_RIGHT,

   uinput.KEY_LEFT,

   uinput.KEY_UP,

   uinput.KEY_DOWN,

   uinput.REL_WHEEL,

])
```

The script sets up a virtual input device capable of emitting synthetic keyboard and mouse scroll events. These events correspond to:

- Arrow key presses (KEY_LEFT, KEY_RIGHT, KEY_UP, KEY_DOWN)

- Mouse scroll (REL_WHEEL), which is used to simulate zooming

The uinput.Device instance provides an interface to emit these actions dynamically in response to encoder rotation.

5. Encoder Reading Function

```
def read_encoder(axis):
    clk = GPIO.input(ENCODERS[axis]['clk'])
    dt = GPIO.input(ENCODERS[axis]['dt'])
    if clk != dt:
        return 1 if clk > dt else -1
    return 0
```

This function performs real-time quadrature decoding of the encoder signals. By comparing the state of the clk and dt pins:

- If clk > dt, the encoder is rotated in the forward direction (clockwise).
- If clk < dt, the encoder is rotated in the reverse direction (counterclockwise).
- If both signals are equal, no movement is detected, and the function returns zero.

This method allows determination of both the presence and direction of rotation with minimal computational overhead.

6. Control Parameters

```
STEP = 1
ZOOM_STEP = 1
```

These constants define the granularity of movement:

- STEP: Determines how many virtual keypresses to emit per encoder tick for directional movement.
- ZOOM_STEP: Defines the number of scroll steps to issue per tick for zooming.

These values can be tuned to modify sensitivity.

7. Focus Function for GUI Activation

```
def focus_on_map():

    print("Focusing on the map...")

    pyautogui.moveTo(500, 500)

    pyautogui.click()
```

Before initiating movement control, this function ensures that the graphical map interface has input focus. It programmatically moves the mouse to a predefined coordinate and simulates a left click. This is essential in full-screen kiosk applications to avoid input loss due to window focus issues.

8. Main Event Loop

```
try:

    focus_on_map()

    print("Rotary encoder controlling arrow keys and zoom...")

    while True:

        moved = False

        for axis in ENCODERS:

            movement = read_encoder(axis)

            if movement != 0:

                moved = True
```

The main loop initializes the system by calling focus_on_map() once. It then enters an infinite polling loop, constantly checking each encoder for movement. Upon detecting a non-zero movement, the script proceeds to determine the appropriate response based on the encoder axis.

9. Event Handling per Axis

Horizontal Movement (x axis):

```
if axis == 'x':
    for _ in range(abs(movement) * STEP):
        if movement > 0:
            device.emit(uinput.KEY_RIGHT, 1)
            device.emit(uinput.KEY_RIGHT, 0)
        elif movement < 0:
            device.emit(uinput.KEY_LEFT, 1)
            device.emit(uinput.KEY_LEFT, 0)
```

Vertical Movement (y axis):

```
elif axis == 'y':
    for _ in range(abs(movement) * STEP):
        if movement > 0:
            device.emit(uinput.KEY_UP, 1)
            device.emit(uinput.KEY_UP, 0)
        elif movement < 0:
            device.emit(uinput.KEY_DOWN, 1)
            device.emit(uinput.KEY_DOWN, 0)
```

Zooming (zoom axis):

```
elif axis == 'zoom':
    device.emit(uinput.REL_WHEEL, movement * ZOOM_STEP)
```

Each movement is translated into one or more keypress or scroll events. The emit method issues a "key down" followed by a "key up" event, which simulates a real keyboard press. The scroll axis uses REL_WHEEL to simulate mouse wheel rotation.

10. Console Logging and Input Throttling

```
print(f"Axis: {axis}, Movement: {movement}")

time.sleep(0.01)
```

Logging output is printed to the console for debugging and observation. A 10-millisecond delay is introduced to limit the polling rate, ensuring responsive yet manageable system performance and reducing CPU usage.

11. Graceful Termination and Cleanup

```
except KeyboardInterrupt:

   print("Exiting...")

finally:

   GPIO.cleanup()
```

To ensure clean termination upon receiving a keyboard interrupt (e.g., Ctrl+C), the script enters a cleanup routine that resets all GPIO pin states. This prevents potential conflicts with future GPIO-based scripts or hardware malfunctions due to uncleared configurations.

Chapter Four
Results and Discussion

4.1 Challenges and Solutions

During the development of the interactive radio project, several technical and usability challenges were encountered. Below is a detailed discussion of each issue and its corresponding solution.

1. Unreliable Station Links

Problem: Many radio stations had broken or invalid URLs, leading to playback failures when users tried to stream them.

Solution: A real-time URL validation mechanism was implemented using Python. Before adding a station to the interface, a request is sent to verify the availability of the streaming link. The following script was used:

```
import requests

def validate_url(url):
  try:
    response = requests.get(url, timeout=5)
    return response.status_code == 200
  except requests.RequestException:
    return False
```

This approach ensures that only functional stations appear on the map.

2. Overlapping Station Ranges

Problem: Some stations had coverage areas that overlapped, making it difficult for users to select the intended station.

Solution: A priority-based selection system was implemented. When multiple stations were within the selection radius, the station closest to the map center was automatically chosen.

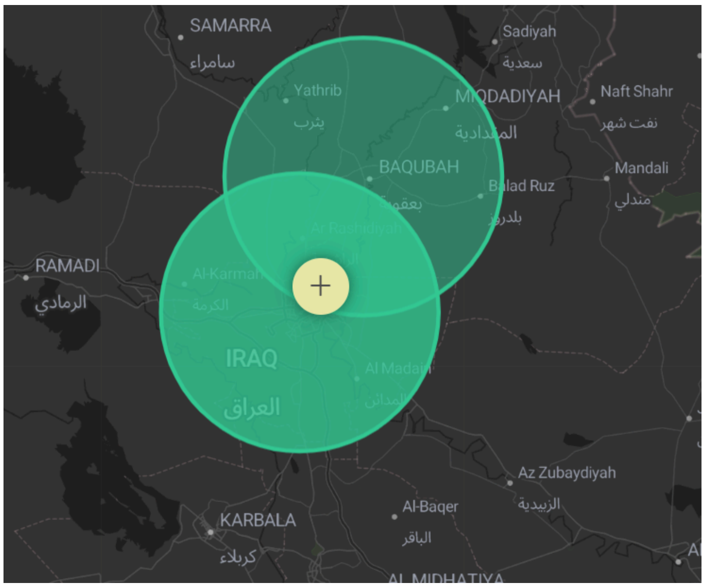

Fig 4.1

3. Map Boundary and Repetition Issues

Problem: Users could navigate the map beyond intended boundaries, causing distortions and loss of reference points.

Solution: Map boundaries were restricted using the Leaflet.js library:

```
var mapBounds = [[-85, -180], [85, 180]];
map.setMaxBounds(mapBounds);
```

This prevents users from scrolling indefinitely beyond defined geographic limits.

4. Unrestricted Zoom Levels

Problem: Excessive zooming out or in made it difficult to interact with station markers effectively.

Solution: A zoom constraint was set within reasonable limits:

```
map.setMinZoom(3);
map.setMaxZoom(12);
```

This ensures an optimal user experience.

5. Slow Link Validation Process

Problem: Checking all station links at once caused performance issues and slow page loading times.

Solution: A batch processing approach was introduced where URLs were validated asynchronously in smaller groups.

from threading import Thread

```
def check_links_in_batches(stations):
   for batch in split_into_batches(stations, batch_size=10):
      threads = [Thread(target=validate_url, args=(station['url'],)) for station
in batch]
      for thread in threads:
         thread.start()
      for thread in threads:
         thread.join()
```

This significantly improved response time.

6. Lack of Clear Station Activation Indicators

Problem: Users had no way of knowing which station was currently active.

Solution: Active stations were highlighted on the map with a color change using CSS and JavaScript.

```
function highlightActiveStation(stationId) {
   document.getElementById(stationId).style.backgroundColor = "#FFD700";
}
```

7. Insufficient User Feedback for Station Presence

Problem: There was no visual cue to indicate when a station was within the selection range.

Solution: A glowing effect was added to stations when they came into range.

```
.station-marker.in-range {
   box-shadow: 0px 0px 10px 5px rgba(255, 215, 0, 0.8);
}
```

8. Missing Stations from Specific Regions

Problem: Certain regions, especially the Middle East and Iraq, had few available stations.

Solution: Additional stations were manually integrated by collecting and verifying new station URLs.

9. Error Handling for Station Data Loading

Problem: When station data failed to load, users were not informed of the issue.

Solution: A structured error reporting system was added:

```
fetch("stations.json")
   .then(response => response.json())
   .catch(error => alert("Failed to load station data. Please try again."));
```

This ensures users receive clear feedback when an error occurs.

These solutions collectively enhanced system reliability, efficiency, and user-friendliness, ensuring a smoother experience.

10. Grounding Issue Affecting Display

Problem: The screen flickered and failed to function properly whenever the amplifier was powered on. This issue was caused by improper grounding, leading to electrical noise and instability in the system.

Solution: A direct ground connection was established between one of the Raspberry Pi's ground pins and a common ground point in the circuit. This resolved the interference issue, stabilizing the display performance.

4.2 Results : Setting Up and Running the Interactive Radio System on Raspberry Pi

This part tells you how we got the interactive radio system working well on the Raspberry Pi. It also explains how we made it start automatically when you turn on the radio (the Raspberry Pi).

4.2.1 Setting Up the Lighttpd Web Server:

We used the Lighttpd web server to show the project's website. This website lets people use the map and choose radio stations. Here's what we did:

Updated the System and Installed Lighttpd: To make sure everything was up-to-date and to put the server on the Raspberry Pi correctly, we used these commands:

```
  sudo apt update
sudo apt install lighttpd -y
```

Started and Enabled the Lighttpd Service: To make the server run and start by itself when the Raspberry Pi turns on, we used these commands:

```
  sudo systemctl start lighttpd
sudo systemctl enable lighttpd
```

Checked the Server Status: To see if the server was working right, we used this command:

```
  systemctl status lighttpd
```

Put Project Files Here: We put the project's files inside the /var/www/html/ folder so the website could see them.

4.2.2 Enabling the uinput Kernel Module:

To let the system understand signals from the Rotary Encoder (the thing you turn to change stations), we needed to turn on something called the uinput kernel module. We did this:

Loaded the Module: We used this command to load it:

```
sudo modprobe uinput
```

Checked if it Was On: To make sure it was working, we used this command:

```
lsmod | grep uinput
```

4.2.3 Running the Rotary Encoder Python Script:

We made a small program in Python to read signals from the Rotary Encoder and tell the system to change radio stations. We ran it with this command:

```
sudo python3 /home/radio/rotary_encoder.py
```

This program runs in the background and listens to the Rotary Encoder. When you turn it, the program tells the system to pick a different station.

4.2.4 Launching the Interactive Map:

To show a picture of the radio stations and let people click on them, we opened the map website in the Chromium browser using this command:

```
chromium-browser http://localhost/radiomap.html
```

When it opens, you should see dots for the radio stations on the map. You can try turning the Rotary Encoder to see if it changes the selected station.

4.2.5 Making the System Start Automatically:

To make everything easy to use, we set up the Raspberry Pi so that the important parts start by themselves when you turn it on.

1. Setting Up the Raspberry Pi: We used a program called VNC Viewer to control the Raspberry Pi from another computer. The Raspberry Pi's name was radio.local, the username was radio, and the password was 1234. We updated the system and installed the things we needed, like the Chromium browser and some parts of Python. We also made sure the uinput thing was ready.

```
sudo apt update && sudo apt upgrade -y
```

2. Making Chromium Start Automatically: To make the Chromium browser open by itself in full screen when the Raspberry Pi starts, we made a special file called chromium_browser.service and put it in the /etc/systemd/system/ folder. This file told the system to run Chromium in a special way (kiosk mode) and open a specific website:

```
[Unit]
Description=Autostart Chromium in Kiosk Mode
After=graphical.target
[Service]
User=pi
ExecStart=/usr/bin/chromium-browser --noerrdialogs --kiosk
http://localhost/map/index.html --incognito
--disable-translate --no-first-run
Restart=on-failure
[Install]
WantedBy=graphical.target
```

Then we used these commands to make it work:

```
sudo systemctl daemon-reload
sudo systemctl enable chromium_browser.service
sudo systemctl start chromium_browser.service
```

3. Making the Rotary Encoder Program Start Automatically: We also made a special file to run the Python program for the Rotary Encoder (rotary_encoder.py) when the Raspberry Pi starts. This file was called rotary_encoder.service and we told it to run as the main user (root) so it could talk to the hardware.

```
  [Unit]
Description=Rotary Encoder Input Script
After=network.target

[Service]
User=root
WorkingDirectory=/home/radio/
ExecStart=/usr/bin/python3 rotary_encoder.py
Restart=on-failure

[Install]
WantedBy=multi-user.target
```

We used these commands to make it work:

```
  sudo systemctl daemon-reload
sudo systemctl enable rotary_encoder.service
sudo systemctl start rotary_encoder.service
```

Checking if Everything Started: We used these commands to see if both of our special programs were running correctly after the Raspberry Pi restarted:

```
  sudo systemctl status chromium_browser.service
sudo systemctl status rotary_encoder.service
```

We also used another tool called journalctl to look at logs if something went wrong.

4.2.6 Making the Startup Look Better:

To make the experience nicer, we let the normal desktop screen show for a little bit before the website opened. This makes it look smoother. We also changed the background picture to something we liked and hid the icons on the desktop to make it look cleaner. We did this in the Raspberry Pi's settings under "Appearance Settings".

Screen Size and Background Picture Size: To match the real screen size

(24 cm by 16 cm), we picked a background picture that was the right size so it wouldn't look stretched or weird.

In Short:

By doing all these steps carefully, we got the interactive radio system working well on the Raspberry Pi. We also made the system understand the button you turn (Rotary Encoder) and made everything start automatically when you turn on the radio. This makes it easy to use.

Chapter Five

Conclusion and Future Work

5.1 Conclusion

This research successfully developed an interactive internet radio system that integrates Raspberry Pi technology with a vintage radio framework. The primary objective of this project was to modernize traditional radio devices by enabling them to stream online radio stations, thereby bridging the gap between nostalgia and modern digital advancements. Through the integration of hardware and software components, the system allows users to explore and interact with global radio stations in an intuitive and engaging manner.

The implementation process involved key components, including a Raspberry Pi as the central processing unit, a rotary encoder for navigation, an LCD display for visual feedback, and an interactive map-based interface developed using Leaflet.js. This interface enables users to browse and select radio stations based on their geographic locations, enhancing accessibility and user engagement.

One of the significant achievements of this project was the seamless integration of these components into a functional system that retains the aesthetic appeal of a vintage radio while providing the convenience of digital streaming. Several challenges were addressed during development, such as optimizing response time, ensuring stable audio streaming, and efficiently managing data. The successful completion of this project demonstrates the feasibility of transforming an old radio into a smart, interactive device, expanding its functionality beyond conventional FM/AM broadcasting.

5.2 Future Work

The successful implementation of the interactive internet radio system lays a solid foundation for further exploration and innovation in this domain. Despite the system's current functionality and user-friendly design, several potential enhancements can be pursued to significantly expand its capabilities, improve performance, and elevate user experience.

One promising avenue is the integration of advanced voice assistant functionality, enabling users to control the system through natural language commands such as “Play a station from Spain” or “Find jazz music in the United States.” This can be achieved using speech recognition libraries like *SpeechRecognition* in Python, coupled with dynamic playback through *VLC* or *HTML5 audio* interfaces. This feature would not only enhance accessibility but also accommodate users with physical impairments.

In parallel, the development of a cross-platform mobile application using frameworks such as *Flutter* or *React Native* could significantly improve interaction by allowing remote access and control over the system. The mobile interface could support real-time station browsing, favorite management, and live feedback.

Another critical improvement lies in search optimization and content discovery. By implementing a sophisticated search engine, users could filter stations by geographic location, language, genre, or even keywords. Integration with third-party APIs such as *Last.fm* or *Shazam* can enrich the station metadata and offer personalized recommendations based on listening behavior.

Exploring gesture-based control mechanisms using sensors like *Leap Motion* or *ultrasonic sensors* could enable touchless interaction for changing stations or adjusting volume, offering an intuitive and hygienic interface in shared or public spaces.

To support outdoor or remote-area applications, the system could be transformed into a portable, solar-powered device, integrating small photovoltaic panels with rechargeable battery packs. This feature promotes energy sustainability and extends usability in off-grid locations.

In addition, the concept of user-generated radio content could be implemented, allowing individuals to create and stream their own custom stations, thereby fostering a community-based broadcasting model. This initiative could be supported by simple streaming tools and content moderation frameworks.

The integration of local broadcasting technologies such as *LoRa* or *Zigbee* would enable short-range FM-like communication, which could be useful for educational institutions, events, or rural communities lacking internet infrastructure.

Another transformative feature would be the application of real-time audio translation powered by artificial intelligence. Using APIs like *Google Translate* in combination with edge AI hardware such as *NVIDIA Jetson Nano*, users could listen to foreign-language stations with simultaneous translation, enhancing global accessibility.

Moreover, implementing a hybrid FM/AM and internet radio mode would offer redundancy in case of internet outages. An embedded FM/AM receiver would allow the system to function as a traditional radio when connectivity is unavailable, making it more robust in variable network environments.

To enhance energy efficiency, the system could incorporate smart power management algorithms, including automatic standby modes, low-energy processors, and adaptive brightness for the display. These improvements would not only extend battery life but also contribute to the device's environmental sustainability.

Finally, integrating real-time weather and local condition displays based on the geographic metadata of the currently playing station could provide contextual information, enriching the listening experience. Personalized station lists and user-defined filters could further streamline the user interface by allowing quick access to favorite stations and preferred regions.

By addressing these potential enhancements, the proposed system could evolve into a highly sophisticated and adaptive multimedia platform, balancing the charm of vintage design with cutting-edge digital functionality. As emerging technologies continue to mature, this project holds the potential to serve as both a nostalgic artifact and a modern tool for interactive, global radio communication.

References

1. Make Projects. (2020). *Raspberry Pi World Radio Project*. Retrieved from https://makeprojects.com/project/raspberrypi-world-radio-project

2. Hackaday. (2020). *Raspberry Pi World Radio*. Retrieved from https://hackaday.io/project/174631-raspberry-pi-world-radio

3. Britannica. (2023). *Radio technology*. Retrieved from https://www.britannica.com/technology/radio

4. IEEE Spectrum. (2021). *How radio changed everything*. Retrieved from https://spectrum.ieee.org/how-radio-changed-everything

5. Techopedia. (2022). *What is Internet Radio?* Retrieved from https://www.techopedia.com/definition/13655/internet-radio

6. Briggs, A., & Burke, P. (2009). *A social history of the media: From Gutenberg to the Internet* (3rd ed.). Polity Press.

7. Hong, S. (2001). *Wireless: From Marconi's black-box to the Audion*. MIT Press.

8. IEEE History Center. (2023). *Guglielmo Marconi and the development of radio*. Retrieved from https://ethw.org/Guglielmo_Marconi

9. Sterling, C. H., & Kittross, J. M. (2002). *Stay tuned: A history of American broadcasting* (3rd ed.). Routledge.

10. Fagen, M. D. (Ed.). (1975). *A history of engineering and science in the Bell System: The early years (1875–1925)*. Bell Telephone Laboratories.

11. Riordan, M., & Hoddeson, L. (1997). *Crystal fire: The invention of the transistor and the birth of the information age*.

W. W. Norton & Company.

12. WorldDAB. (2023). *Digital radio overview*. Retrieved from https://www.worlddab.org/

13. Gannes, L. (2011). *Why Internet radio is the next big battleground*. GigaOm. Retrieved from https://gigaom.com/2011/05/10/why-internet-radio-is-the-next-big-battleground/

14. Statista. (2023). *Smart speaker usage worldwide*. Retrieved from https://www.statista.com/statistics/792604/worldwide-smart-speaker-user-base/

15. Barrowclough, D., & Kozul-Wright, R. (2018). *Creative economy outlook: Trends in international trade in creative industries*. United Nations Conference on Trade and Development (UNCTAD).

16. UNESCO. (2023). *Digital heritage*. Retrieved from https://en.unesco.org/themes/digital-heritage

17. Raspberry Pi Foundation. (n.d.). *Documentation, specifications and datasheets*. Retrieved from:
- https://www.raspberrypi.com/
- https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf
- https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/
- https://download.kamami.pl/p586119-cm4-datasheet.pdf
- https://www.raspberrypi.com/documentation/
- https://pinout.xyz/
- https://www.handsontec.com/dataspecs/module/Rotary%20Encoder.pdf