Robotic QR Code Instruction System

Tymofii Maniak, Eduardo Kestler
School of Electrical and Computer Engineering, California State University Northridge

***Abstract*—This project focuses on developing a robotic system capable of scanning QR codes and barcodes to execute embedded instructions, enhancing its adaptability in dynamic environments. The process involves integrating a SparkFun 2D Barcode Scanner Breakout with the robot and programming a development board to translate scanned data into actionable tasks like movement, LCD control, or sound generation. The expected outcome is a robot capable of real-time task reconfiguration through QR and barcode interpretation.**

***Index Terms*—QR Code, Bardcode reader, Cortex-M4F, UART, MSP 432**

I. Introduction

THIS project aims to design and implement a robotic system capable of scanning QR codes and barcodes, and executing tasks based on embedded instructions. The motivation behind this project arises from the need for robots to interact dynamically with human-defined environments. Despite advancements in robotics, a significant challenge persists in enabling robots to seamlessly adapt to new tasks and environments without extensive manual reprogramming. The utilization of QR codes and barcodes as a medium for conveying operational instructions presents a solution to this challenge, allowing for real-time task adaptation and execution.

QR codes and barcodes which are commonplace in retail, manufacturing, and event sectors due to their compact size and efficiency in storing information, serve as an ideal medium for conveying instructions to robots.

The project involves two primary phases: the Integration Phase and the Programming Phase. In the Integration Phase, a camera module, specifically the SparkFun 2D Barcode Scanner Breakout, is integrated with the robot. This setup ensures the robot has an unobstructed view for scanning. The accompanying software is developed to decode data from the camera module and convert it into actionable tasks.

During the Programming Phase, code was created to facilitate the interaction between the camera module and the development board. This board is responsible for translating the information from the scanned codes into specific robot instructions, such as moving a certain distance, changing direction, actuating LCD, or playing melodies through a buzzer.

The expected outcome is a robotic system that can reconfigure its tasks and interact dynamically with its environment, guided by instructions encoded into QR Code.

II. Research

A. QR Code Basics

Quick Response (QR) code is a two-dimensional matrix barcode that is made out of black squares that are positioned within a square grid that is located on white background. The Reed-Solomon algorithmic placement of black squares allows to encode various types of data. The Figure 1.1 shows an example of a QR code where such black square placement encodes a series of ASCII characters that form a phrase ”hello-world.” The exponential growth of QR Code utilization within recent decade can be explained by how robust the technology is and the simplicity of its use. The QR code can be easily read from any angle and the scanner doesn't have to necessarily be aimed in line with the code in order to read it. Moreover, QR codes boast a robust error correction margin, allowing them to function effectively even when approximately 30% of the code is damaged or unreadable. Another reason for the widespread use of the QR technology is the amount of data that can be encoded within the QR Code. The latest standard of the QR Code technology can store up to 2953 bytes of data, which translate to the same number of possible ASCII characters. This amount of data an order of magnitude larger than a regular one-dimensional barcode can store. For the purpose of this project, we used QR Codes to encode alphanumeric data in the form of ASCII characters.

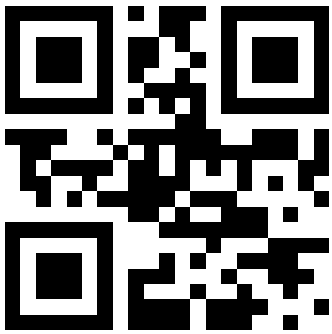

Fig 1.1 QR Code with ”hello-world” message encoded in it

B. 2D Barcode Reader

The decoding of QR Code can be done using specialized hardware. For the purpose of this project, the DE2120 barcode scanner module is the hardware that was used in order to scan and decode QR codes. This module is capable of reading 20 different barcode variations, both 1-D and 2-D. The reading is achieved by using a camera coupled with on-board image processing to identify and decode barcodes. Once the module identifies and reads the QR Code, it sends the decoded data through the TX line by using the UART serial communication protocol. The module has internal memory and thus is highly configurable. By default, the UART baud rate used by module is 115200, however it can be configured to any desired range.

The configuration is done by feeding certain instructions into the module. This can be done in two different ways: Scanning QR Code with embedded instruction or sending instruction though UART RX pin. SparkFun's 2D Barcode Scanner shown in Fig. 1.2 uses DE2120 and simplifies its integration by providing access to Micro-USB and full I/O pin-out. The I/O pin-out was used for the purpose of this project by soldering breakaway headers into I/O holes and connecting microcontroller pins directly to module.

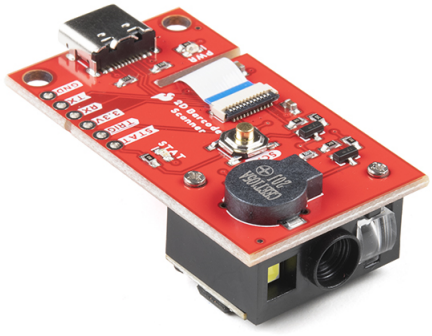

Fig 1.2 SparkFun 2D Barcode Scanner Breakout

C. Texas Instruments Robot Platform (TI-RSLK MAX)

TI-RSLK MAX shown in Fig 1.3 is a robotic platform developed by Texas Instruments that uses MSP432 microcontroller to interact will all its peripherals. The platform provides access to gearmotors and their encoders, bumper switches, QTRX sensor array, and other add-ons that can easily attached or removed. Such platform was used for the purpose of this project with SparkFun 2D Barcode Scanner attached in front. The bumper switches were removed in order to free space for barcode scanner fastening.

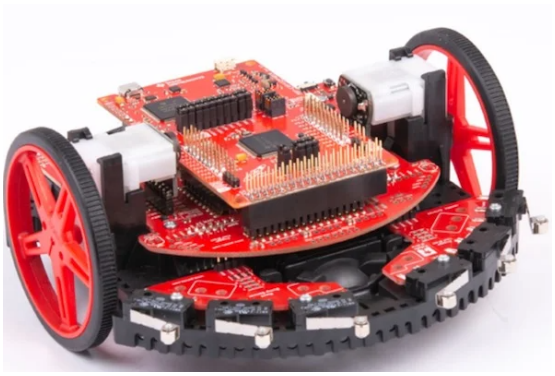

Fig 1.3 TI-RSLK MAX

D. MSP 432 Launchpad

MSP432 is a family of mixed-signal microcontrollers that are powered by 32-Bit ARM Cortex-M4F CPUs. The CPU frequency in such microcontrollers goes up to 48MHz, which is sufficient for needs of this project. Specific configuration of the MSP432 used in the TI-RSLK MAX robot platform is MSP432P401R. The microcontroller is equipped with numerous peripherals such as eUSCI serial communication interface that provided an accessible interfacing between embedded QR instruction and actuation of outside devices such as motors, buzzers, and LEDs.

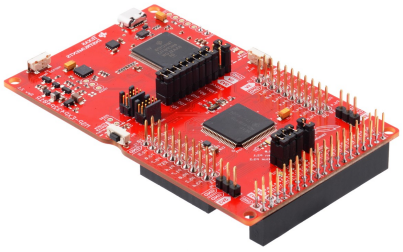

Fig 1.4 MSP432P401R Microcontroller (MCU)

E. Graphic LCD

Nokia 5110 (Fig. 1.5) is a basic graphic LCD display with 84x48 pixel resolution. The display is mounted on a PCB with SPI interface pin-out available for use. The display has internal 84x48 Display Data RAM (DDRAM) that is used in order to store pixel values of a desired image. Such display was used as a part of the instruction system as one of the possible external devices that would respond to QR Code instructions. By encoding the image inside the QR Code, we were able to decode it using barcode reader and used microcontroller's SPI drivers in order to upload the image into internal DDRAM of the display.

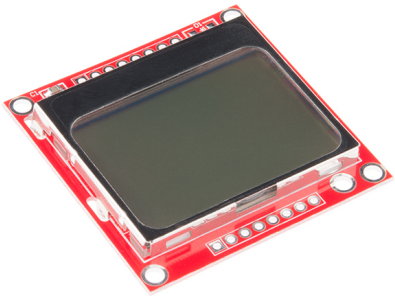

Fig 1.5 Graphic LCD 84x48 - Nokia 5110

F. Piezzo Buzzer

A piezo buzzer, as shown in Figure 1.6, is a small electronic component that generates sound through the mechanical vibrations of a piezoelectric crystal. When an electrical voltage is applied to a piezoelectric crystal, it can vibrate to produce audible sound waves. Piezo buzzers are known to produce high-pitched tones, generating sounds from simple beeps to

more complex tones depending on the voltage and frequency applied. For the purpose of this project, the piezzo buzzer will be connected to microcontroller and used to respond to sound generation instructions, producing the melody encoded in QR Code.

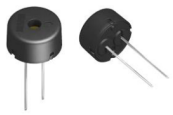

Fig 1.6 Piezzo Buzzer

G. System as a whole

End-to-end data flow of the final assembled system meant the ability to feed QR Code into the Barcode scanner and consequent recognition of the instruction embedded into the QR code by MSP432 microcontroller, resulting in the actuation of external device such as those introduced before: gearmotors, piezzo buzzer, and LCD display. The Functional Diagram that we used as a reference while integrating the system is shown in Figure 1.7. Fully assembled version of the final system that is capable of producing the desired data flow is shown in the Figure 1.8. The Barcode Scanner is attached to the front of TI-RSLK using a specialized stand, while piezzo buzzer and LCD are placed onto the breadboard that is attached in the back.

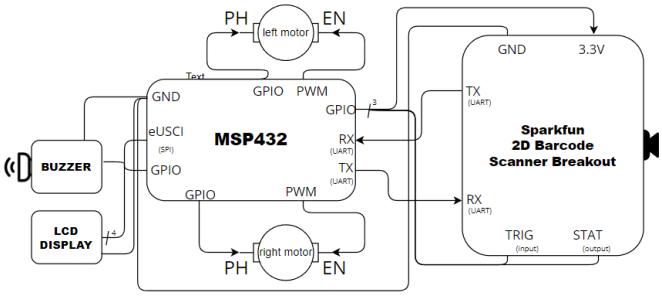

Fig 1.7 Robotic QR Instruction System Functional Diagram

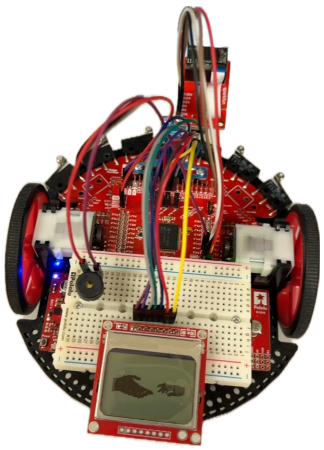

Fig 1.8 Robotic QR Instruction System

III. Methods

A. Integration

The integration of the described modules required the choice of proper pin out, so that the basic function of the robot were not interrupted. The basic functions were defined as the use of PWM to drive both motors. The following Pin out was used to interact with the QR reader module, the piezo buzzer, and the LCD screen.

1) Motor Controller: The pins used to provide the motor controller with the enable, direction and speed signals are as follows:

- Direction Left Motor ↔ MSP432 LaunchPad PWM Pin P5.4
- Direction Right Motor ↔ MSP432 LaunchPad PWM Pin P5.5
- Enable Left Motor ↔ MSP432 LaunchPad PWM Pin P3.6
- Enable Right Motor ↔ MSP432 LaunchPad PWM Pin P3.7
- PWM(Speed) Left Motor ↔ MSP432 LaunchPad PWM Pin P2.6
- PWM(Speed) Motor ↔ MSP432 LaunchPad PWM Pin P2.7

2) QR Module: The pins used to utilize the UART serial communication protocol to received the data obtained from the QR code are P3.3(RX) and P3.2(TX). TX and RX from the QR Module were wired to P3.3 and P3.2 respectively.

3) Piezzo Buzzer Module: The pins used to drive the Piezzo Buzzer module using PWM are:

- Piezzo VCC ↔ MSP432 LaunchPad PWM (3.3V)
- Piezzo GND ↔ MSP432 LaunchPad GND

4) LCD Module: The pins used to drive the LCD module are:

- Nokia 5110 LCD VCC ↔ MSP432 LaunchPad VCC (3.3V)
- Nokia 5110 LCD GND ↔ MSP432 LaunchPad GND
- Nokia 5110 LCD SCE ↔ MSP432 LaunchPad Pin P9.4 (SCE, Chip Enable)
- Nokia 5110 LCD RST ↔ MSP432 LaunchPad Pin P9.3 (Reset)
- Nokia 5110 LCD D/C ↔ MSP432 LaunchPad Pin P9.6 (D/C, Data/Command)
- Nokia 5110 LCD MOSI ↔ MSP432 LaunchPad Pin P9.7 (MOSI)
- Nokia 5110 LCD SCLK ↔ MSP432 LaunchPad Pin P9.5 (SCLK)
- Nokia 5110 LCD LED ↔ Unconnected

B. QR Instruction Encoding

1) Instruction Encoding Method: The most important part of the project was to properly standardize instructions so that we can easily encode them into QR Code and consequently use barcode reader and microcontroller to interpret the QR Code. Since Barcode reader used UART protocol in order to send serial data to the microcontroller character by character, we've decided to interpret instruction as a simple C array of

ASCII characters that was expected to fit inside the global serial data buffer defined within the software. The first two characters in such array define instruction's type and action, while the rest of the array store the instruction data. There are two types of instructions: pre-defined and encoded. Pre-defined instruction is stored inside the microcontroller's ROM, while encoded instruction is stored inside the serial data buffer once instruction is read from QR code. The idea behind these two types of instructions is to be able to both store some commonly used instructions inside the system's memory and also be able to feed unique instructions into the system as needed. The system was programmed to support instructions that perform three different actions: perform movement, play sound, show image. Each action was respectively related to actuation of the following external devices: gearmotors, piezzo buzzer, LCD display. In order to encode instructions into the QR Code, a free online Barcode Generator software created by TEC-IT was used. Figure 1.9 demonstrates how such software converts string of ASCII characters (instruction) into QR Code.

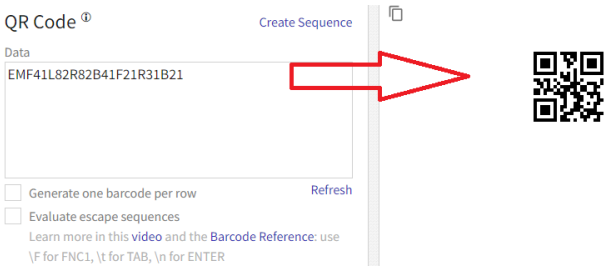

Fig 1.9 Robotic QR Instruction System

2) Encoding Movement Instructions: Feeding QR Code with movement instruction into the system supposed to actuate gearmotors using Pulse Width Modulation (PWM) technique and thus make the whole system move. In order to tell the microcontroller that it has to actuate gearmotors, the second character encoded into the QR Code was chosen to be 'M' for "Movement." The first character was supposed to determine whether the movement sequence is encoded into QR Code or pre-defined within the flash memory of the microcontroller. Character 'E' would indicate the encoded movement, while character 'P' would indicate the pre-defined movement. If the first character is 'P', the system should then expect a single digit ASCII character as third character that would indicate which pre-defined movement command should be executed. On the other hand, if first character is 'E', the system should expect a series of movement sub instructions. A single movement sub instruction consists out of three characters:

1) Direction forward, back, left, right [F,B,L,R]
2) Speed 0-9 controlled by PWM signal [0-9]
3) Duration 1-9 seconds [1-9]

For example, the instruction [EMF55L32], once decoded by microcontroller, should actuate gearmotors such that they move forward at speed 5 for 5 seconds, and then turn left with speed 3 for 2 seconds.

Figure 1.10, shows the output of each pin that is used to control the motors when a left instruction is interpreted. P5.5 and P5.4 are the signals that provide the controller with the direction of each motor. Low moves forward the motor and high moves backwards the motor. P2.7 and P2.6 are the PWM signals that are fed to the controller. They control the velocity of the motors P3.7 and P3.6 are the signals that enable each motor for the movement.

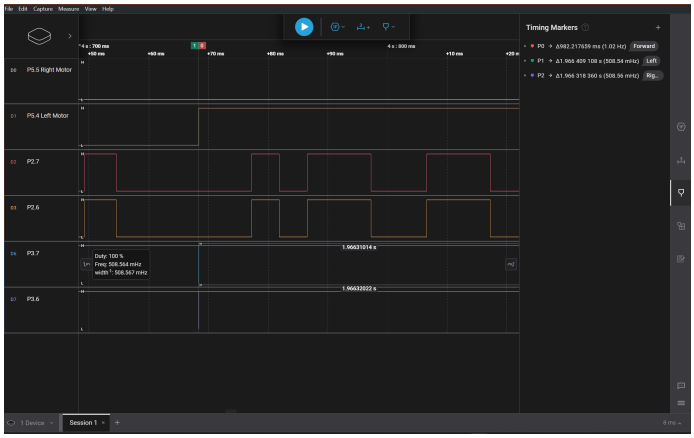

Fig 1.10 Signal after the instruction is processed.

3) Encoding Sound Instructions: Feeding QR Code with sound instruction supposed to actuate piezzo buzzer using square wave signals with frequencies of regular note frequencies and thus produce the desired sound encoded in instruction. Frequencies for the the 4th octave set of notes are stored in ROM. In this way, we can encode the a song using the standard char symbols for notes, together with their respective duration. For this specific project, a 4 by 4 time signature was assumed. Also, for duration only the 4 basic duration were used. The first character 'E' would indicate this is an encoded instruction, while character 'S' would indicate this is a song that will make use of the piezzo.

1) Do,Re,Mi,Fa,Sol,La,Si [C,D,E,F,G,A,B]
2) Duration (1 quarter tone, 2 half tone, 3 quarters tone, 4 whole tone)(Assuming a 4 by 4 time signature) [0-4]
3) Signals the end of the encoded song. [X]

For example, the instruction ESC1C2C3C4 would be interpreted as Do 1 quarter tone, Do 2 half tone, Do 3 quarters tone, Do 4 whole tone.

4) Encoding Image Instructions: Feeding QR Code with image instruction supposed to actuate 84x48 LCD display using the SPI protocol and transfer the Image data from the QR Code into the LCD's DDRAM. In order to tell the microcontroller that it has to actuate eUSCI driver and send Image data to the LCD, the second character encoded into the QR Code was chosen to be 'I' for "Image." The first character was supposed to determine whether the image data is encoded into QR Code or pre-defined within the flash memory of the microcontroller. Character 'E' would indicate the encoded image, while character 'P' would indicate the pre-defined image. If the first character is 'P', the system should then expect a single digit ASCII character as third character that would indicate which pre-defined image should be sent to LCD. On the other hand, if first character is 'E', the system

should expect a series of characters that encode an image. An encoded image is an array of 504 chars (bytes) that define Bitmap for the 84x48 = 4032 pixels of the LCD. A single char defines whether each of the 8 pixels should be either on or off. Thus, an image array maps every char to 504 segments of 8 pixels of the LCD display. In order to encode an image we had to use a specialized software called LCD Assistant. Such a software takes a Bitmap image, converts it to C array of chars, and saves the array into a text file.

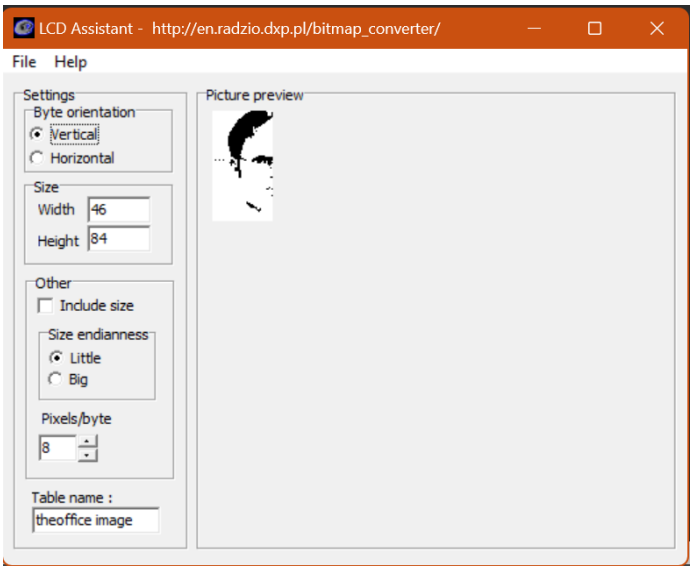

Fig 1.11 LCD Assistant program.

After the program is run, we obtain a text file that contains a C array with the 8 pixel values stored in char values.

```
//-----------------------------------------------------------------------------
// File generated by LCD Assistant
// http://en.radzio.dxp.pl/bitmap_converter/
//-----------------------------------------------------------------------------

const unsigned char theoffice image [] = {
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x80, 0xC0, 0xC0, 0xE0, 0xF0, 0xF8, 0xFC, 0xFC, 0xFE, 0xFE, 0xFF, 0xFF, 0xFF,
0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFE, 0x70, 0x70, 0x30, 0x1E, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0xE0, 0xF0, 0xFC,
0xFE, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x0F, 0x3F, 0x9F,
0x1F, 0x0F, 0x0F, 0x07, 0x03, 0x03, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xF8, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x1F, 0x1F, 0x0F, 0x0F, 0x1F, 0x07, 0x06, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x07, 0x7F, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
0x3F, 0x03, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x20, 0x00, 0x00, 0x20, 0x00, 0x00,
0x20, 0x00, 0x00, 0x00, 0x00, 0x80, 0xE7, 0xE7, 0xCF, 0x1F, 0x1F, 0xFF, 0xFF, 0x1D, 0x10, 0x10,
0x20, 0x20, 0x00, 0x00, 0x00, 0x00, 0x60, 0x70, 0x30, 0x38, 0xF8, 0xFC, 0xFC, 0x7C, 0x3C, 0xFC,
0xF8, 0x38, 0xF8, 0xF8, 0xB8, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x03, 0x00, 0x00, 0x39, 0x38, 0x3F, 0xFF, 0xE0, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x01, 0x03, 0x09, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x07, 0x07, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x04, 0x04, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x08, 0x08, 0x04, 0x00, 0x08,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x60, 0xC0, 0xC0, 0x80, 0x80, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x03, 0x03, 0x03, 0x02, 0x06,
0x08, 0x18, 0x0C, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00,
};
```

Fig 1.12 LCD Assistan output text file.

In order to use the QR encoding, each hex value had to be encoded as a character and send as such. This meant that we are using two char values to represent one char value from the image array. To send the data, the array values had to be modified to remove spaces, commas and hex prefix.

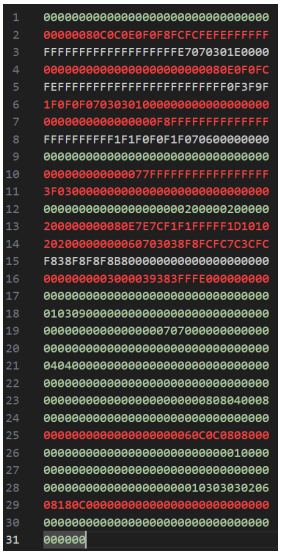

Fig 1.13 Cleaned image data.

Finally, this cleaned data can be added to the first two characters of the instruction and it can be encoded in a QR code.

Fig 1.14 Encoding of image instruction into a QR code.

C. Programming and instruction execution

1) eUSCI Serial Communication Driver (UART): In order to communicate with the SparkFun QR Code Reader, we had to use the microcontroller's eUSCI module. For the purpose of the project we used internal eUSCI A2 module that used pin 3.2 for RX and pin 3.3 for TX. The module was setup to be work in Asynchronous mode in order to support UART communication. By setting proper registers, we defined the UART baud rate to be 115200 (rate used by Barcode reader) and data frame to have no parity, one stop, and eight data bits. Most importantly, we had to prompt the eUSCI to expect the data in LSB-first format in order to properly receive the data from the QR Code Reader.

2) Reading and storing QR Data: In order to read the data from eUSCI buffer and store it into global array buffer called 'bufferString', the function EUSCI_A2_UART_InString() was used. This function took both array of characters and size of the array as two arguments. Inside, the function was reading

characters from the buffer and storing them into the array one-by-one until it saw the terminating character that would indicate the ending of the instruction. Default terminating character produced by the SparkFun Barcode Reader was the return carriage character (0x0D), which is the character we expected to see in the end of the instruction. The Figure 1.12 below shows reading of a logic analyzer that was connected to the TX pin of the barcode reader. The logic analyzer shows a movement instruction right after it was read by the barcode. Note that the last character produced by the barcode reader is the return carriage termination character '\r'. Such an instruction, once read by QR Code Reader, was expected to be stored inside the 'bufferString' array of characters.

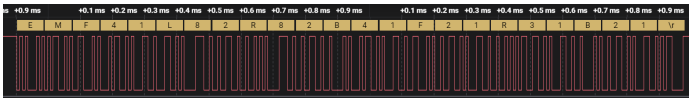

Fig 1.15 Movement instruction read by logic analyzer

3) QR Data handling: Once, the instruction is stored inside the global array of characters called 'bufferString', it is ready to be processed. In order to process the instruction and determine what it is supposed to do, we created the function called QR_Data_Handler() that took both 'bufferString' and it's size as arguments. Using if/else statements, the function first determines what type of instruction that is stored inside the 'bufferString' and whether the instruction is encoded or pre-defined. If instruction is pre-defined, the function reads the third character in order to determine which of the pre-defined instructions to execute. Otherwise, if instruction is encoded, the function starts parsing the rest of the 'bufferString' in order to produce the desired result of the encoded instruction. For every type of instruction the parsing differed since the character encoding was different. For example, for movement instruction we interpreted the characters 'F','B','L', and 'R' as regular C character. This way for movement instruction the character 'R' was interpreted as decimal 82, according to ASCII table. On the other hand, for display instruction we had to interpret the characters as 8-bit Hex values. This way the series of characters 'DD' had to be interpreted as 0xDD (decimal 221), and not as two standalone ASCII Ds (13).

4) Executing movement instruction: The code begins by checking the data in dataBuffer for an 'E', indicating an encoded command, followed by an 'M', which specifies that the following instructions are related to movement. The command sequence follows a structure of 'E' for Encoded, 'M' for Movement, and then a series of instructions each consisting of a direction ('F', 'B', 'L', 'R' for Forward, Backward, Left, Right), a speed (ranging from 0 to 9), and a duration (from 1 to 9 seconds). The code processes these instructions in a loop, parsing the direction, speed, and duration for each movement command. The direction is parsed by checking the character at index 2*i + i - 1 in dataBuffer. The index calculation accounts for the initial 'EM' and then steps through each set of instructions (3 characters per instruction: direction, speed, duration). The speed is obtained from the character immediately following the direction, at index 3*i. This character is converted from an ASCII character to an integer and typically multiplied to match the motor control specifications. The duration is located two characters after the direction, at index 4 + (i - 1) * 3. Like the speed, this value is converted from ASCII to an integer and then multiplied to convert it into milliseconds for the delay function. Depending on the direction indicated, corresponding functions like MotorForward, MotorBackward, MotorLeft, or MotorRight are called with speed parameters derived from the data buffer and scaled appropriately. After executing each movement, the system stop the motors, allowing the robot to prepare for the next instruction in the buffer.

5) Executing sound instruction: The code executes the sound instruction by iterating through bufferString, checking each even-indexed position for a note character ('C' to 'B') and its subsequent duration indicator at the odd-indexed position. Based on the character found, it plays the corresponding musical note using predefined frequencies in the PlayNote function. The duration for each note is determined by the value following it, which is multiplied by 50 to calculate the number of iterations for playing the note. After playing each note, there is a 50 ms delay before proceeding to the next note. The process continues until the character 'X' is encountered, which indicates the end of the encoded song.

6) Executing image instruction: The code processes the dataBuffer, by iterating through its elements. In each iteration, the code reads two characters from dataBuffer, interprets them as a hexadecimal number, and converts them into a single byte. This conversion is achieved by checking if each character is a digit or a letter, translating it into a numerical value accordingly. The first character of each pair is multiplied by 16 to shift it to the high nibble of a byte, and the second character is added to this value as the low nibble. This combined byte is then stored in the 'Image' array. The process essentially converts a hexadecimal representation of image data in dataBuffer into a binary format in the 'Image' array, for further image display. Each pair of hexadecimal characters in dataBuffer is thus converted into a single byte, with the entire loop effectively processing 504 pairs of characters to fill the 'Image' array. The image is then displayed onto the LCD Display using Nokia5110_DrawFullImage() with 'Image' array as an argument.

IV. Results

The following is a list of Youtube videos that show the demos for the project and the GitHub link for the project as follows:

1) https://youtu.be/naU8icNVj24 Maze solving (movement instruction demo)
2) https://youtu.be/-CH57oR08O0 Sound instruction demo
3) https://youtu.be/JETsPS_X3QY Sound and image encoded. Displaying image to LCD
4) https://youtube.com/shorts/JH6pmHVL630?feature=share Reading image and displaying to LCD
5) https://github.com/csun-ece-robotics/ece595rl-fall-2023-final-project-group-11.git GitHub link

V. Conclusion

In conclusion, this project successfully demonstrates the innovative integration of QR code and barcode scanning technology with a robotic platform, resulting in a adaptable and dynamic system. The utilization of the SparkFun 2D Barcode Scanner Breakout and the MSP 432 microcontroller, in conjunction with other key components like the graphic LCD and piezo buzzer, has enabled the creation of a robot capable of interpreting and executing a variety of tasks encoded in QR codes and barcodes. This includes movements, sound generation, and displaying images, which are translated from the scanned data in real-time.

The project's significance lies in the way robots can interact with their environment, making them more responsive and versatile without the need for extensive manual reprogramming. By using common and easily accessible QR and barcode technology, the system opens up new possibilities for robotic applications in various fields such as retail, manufacturing, and event management.

Moreover, The ability of the robot to dynamically reconfigure its tasks based on simple, encoded instructions offers a glimpse into the future of autonomous and adaptable robotic systems. This project not only achieves its initial objectives but also sets a foundation for future innovations in robotic technology.