Autonomous and Multimodal Vehicle Control Featuring Wireless Interface

Jacob Reyes & Ricardo Rodriguez

April 24th, 2025

Texas A&M University at McAllen - HECM
Multidisciplinary Engineering Department
ESET 369 - Embedded Systems Software

Instructor: Rafael Fox

TABLE OF CONTENTS

Introduction:

Wireless communication is a technological marvel that has revolutionized the way we interact with the world. There are many different methods of wireless communication, with the most popular being Wi-Fi, Bluetooth, cellular, and satellite communication. Through these forms of communication, we have created a global network that allows us to communicate with anyone, from anywhere, across the world. Additionally, we can control devices, homes, and even entire companies remotely.

The purpose of this project is to introduce ourselves to the concept of wireless communication through the form of Wi-Fi to control a small robotic vehicle. We plan to use an external device such as a tablet or smartphone to connect with the CC3200-LAUNCHXL and then have the board interface with the TI MSP-EXP432P4111 microcontroller. The goal is to remotely control the vehicle using a custom-built website, where pressing buttons on a wireless device will move the vehicle according to the user's commands (forward, backward, left, and right). Additionally, we will include a second mode for the robotic vehicle, called "Following Mode." This mode will allow the vehicle to operate autonomously and follow a target at a set distance in a straight line using an ultrasonic sensor. While in “Following Mode,” the vehicle can be commanded to start or stop following by clapping your hands through the implementation of the ADMP504 microphone. One clap will initiate the following procedure, and another will signal the vehicle to halt. We’re also adding the AC-1005G-RPA-LF buzzer to make sure the vehicle is noticeable, helping to avoid any accidents. It’ll play a steady tone while going forward and a set of beeping sounds when reversing mimicking the reverse tones of heavy duty vehicles.

The application of wireless communication through Wi-Fi is endless, with the incorporation of Internet of Things we are able to create entire networks of devices that can all work together wirelessly. By studying and gaining familiarity with Wi-Fi interfacing through this project we are preparing ourselves

for working in industries that rely on wireless technology for data transmission, remote sensor monitoring, and automation.

Description of System Design:

Hardware:

The Hardware used in this project consists of 8 distinct components. These components are listed down below:

- TI MSP-EXP432P4111 [1]- Microcontroller
- HC-SR04 [2]- Ultrasonic Sensor
- L298N H-Bridge [3]- Motor Driver
- 12-V Battery Pack equipped with an On/Off switch
- T Star DSTR - Robot Chassis
- CC3200-LAUNCHXL [4]- Microcontroller
- AC-1005G-RPA-LF [5]- Buzzer
- ADMP504 [6]- Microphone

For this project, we will build upon our final project from ESET 349, meaning we will reuse our previous robotic vehicle chassis. This includes the TI MSP432 microcontroller, HC-SR04 ultrasonic sensor, L298N H-Bridge motor driver, and a 12V battery pack equipped with an on/off switch. Since our previous report discusses these components in greater detail, we will provide a brief summary of them and focus more heavily on the new additions to the vehicle.

The TI MSP432 is a 32-bit microcontroller produced by Texas Instruments, we will be using the MSP-EXP432P4111 (*Figure 1*) version of this controller for our project. Using the IED Keil uVision v5.0 we will program our microcontroller with the C programming language, in order to interface with the ultrasonic sensor and H-bridge motor driver. We will be using almost all pins on the microcontroller for

our project. The most important pins are those that power the microcontroller, send inputs to the motor driver, and read outputs from the ultrasonic sensor and microphone. Pins P4.3 and P4.1 control the starboard-side motors of the vehicle, while pins P4.7 and P4.5 control the port-side motors. Pins P2.4, P2.5, and P3.2 are used to read outputs from the ultrasonic sensor, while pin P3.0 serves as the trigger output from the microcontroller. Finally, pin P3.7 outputs to the buzzer, and pin P6.7 will handle the output from the microphone.

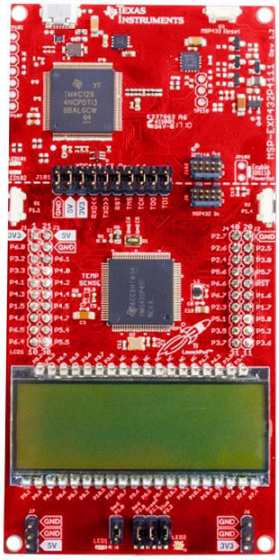

Figure 1: TI MSP-EXP432P4111

The HC-SR04 ultrasonic sensor (Figure 2) emits an ultrasonic sound from its transmitter, and the receiver detects the signal once it bounces off an object. The HC-SR04 has four pins: Vcc, Trigger, Echo, and GND. The Vcc and GND pins provide power and ground, respectively. The microcontroller activates the sensor through the Trigger pin and captures the Echo pulse. The distance to an object is then calculated using the equation: $Distance(M) = (\frac{1}{2} * 343(M/Sec) * \frac{1}{3MHz} * \alpha)$ where α represents the length of the Echo pulse.

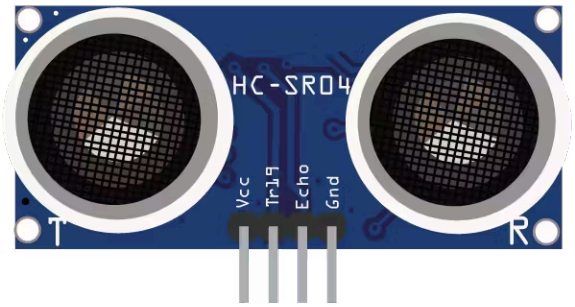

Figure 2: HC-SR04 Ultrasonic Sensor

The L298N H-Bridge Motor Driver (Figure 3) allows us to control the motors using 3.3-volt signals from the microcontroller. The motor driver is powered by a 12-volt supply from the onboard battery pack. The logic pins on the motor driver determine the direction of rotation of the motors. Table 1 shows the motor rotation direction based on the inputs to the logic pins. Inputs 1 and 2 control Output A, while Inputs 3 and 4 control Output B. It is important to note that Output B drives the motors on the right side of the vehicle (starboard), while Output A drives the motors on the left side (port). To control the speed of the motors, we provide a duty cycle to Output A and Output B through the motor driver's enable inputs. Enable A is connected to pin P5.6, and Enable B is connected to pin P5.7.

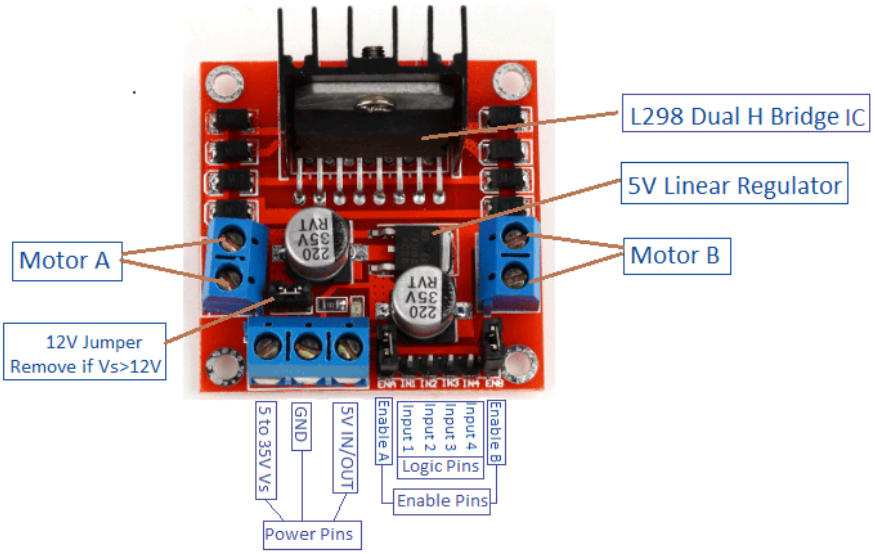

Figure 3: L298N H-Bridge Motor Driver

Table 1: Motor Direction Based on Inputs

Input						Output	
Enable A	Input 1	Input 2	Input 3	Input 4	Enable B	Motor A (1&2)	Motor B (3&4)
	0	0	0	0		0	0
PWM Signal	0	1	0	1	PWM Signal	1(Reverse)	1(Reverse)
PWM Signal	1	0	1	0	PWM Signal	1(Forward)	1(Forward)
	1	1	1	1		0	0
PWM Signal	1	0	0	1	PWM Signal	Rotate Right	
PWM Signal	0	1	1	0	PWM Signal	Rotate Left	

The onboard battery pack (Figure 4) supplies the motor driver with 12 volts to power the four motors of the vehicle. The L298N features a 5-volt regulated output, which will be used to power the MSP432 and other onboard electronics, such as the CC3200, buzzer, and microphone. A On/Off switch is wired in series with the power supply and the motor driver, once the switch is in the off position, there

will be no power to the microcontroller. We installed this switch to conserve battery when the vehicle is not in use, as well as for the convenience it provides when loading new programs onto the microcontroller.

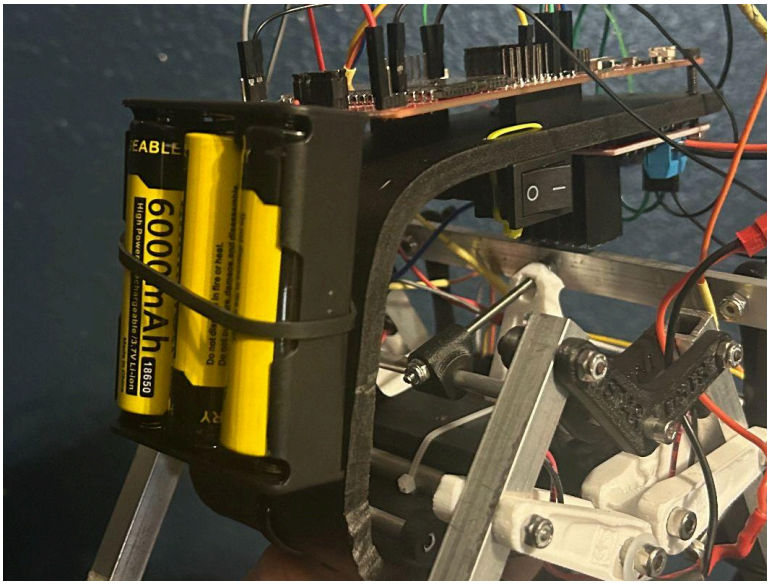

Figure 4: 12-V Battery Pack equipped with an On/Off switch

The T-Star DSTR robot chassis is being repurposed from our project in ESET 349, with the robot's initial condition shown in Figure 5. Several modifications were made to adapt the chassis to our current project. These modifications include attaching an ultrasonic sensor to the front of the vehicle, mounting the CC3200 on top of the MSP432, installing a breadboard, and mounting a microphone and buzzer, along with supporting components such as operational amplifiers, resistors, and capacitors.

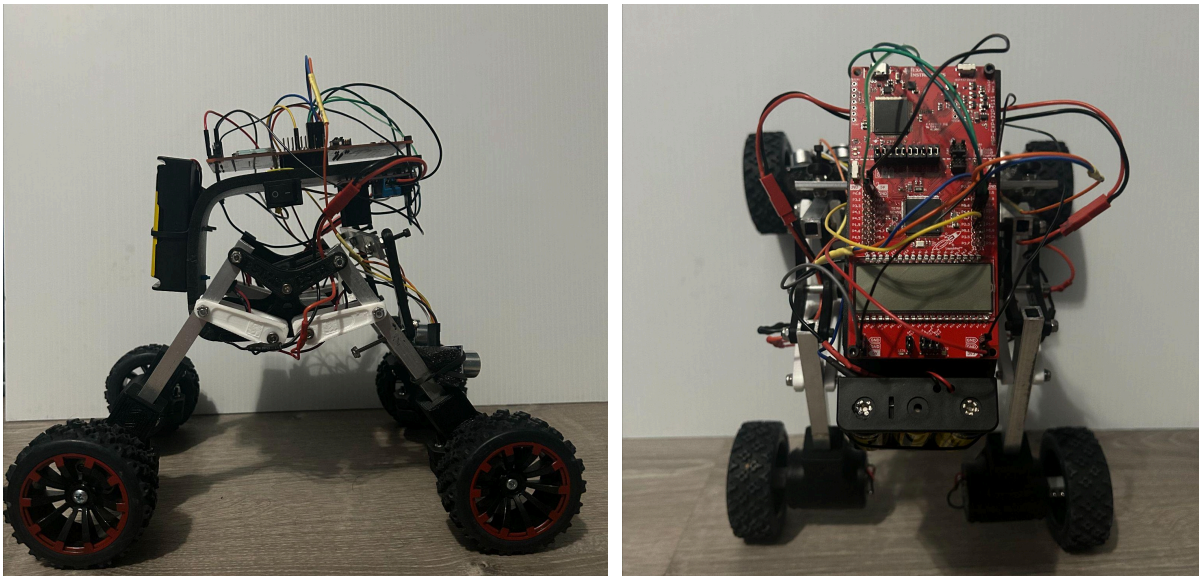

Figure 5: T Star DSTR robot Side & Top Views

The CC3200-LAUNCHXL (Figure 6) is a microcontroller that features built-in Wi-Fi capabilities. This board is essential to our project, as it will mediate the signals from our wireless device to the MSP432. We will be using six pins from the CC3200, which will be connected to six corresponding pins on the MSP432. These connections will allow us to activate the vehicle, switch between its two modes, and control its movements i.e. forward, reverse, rotate right, and rotate left.

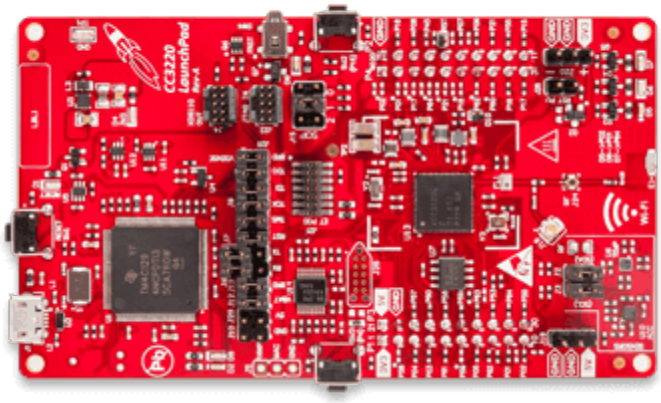

Figure 6: CC3200-LAUNCHXL Microcontroller

The AC-1005G-RPA-LF buzzer (Figure 7) will be used to alert others of the vehicle's presence. A steady tone will be played while the vehicle is moving forward, and an alternating tone will be played while it is moving in reverse. To drive the buzzer, we are using the AD8541[7] operational amplifier, as the MSP432 microcontroller does not supply sufficient current to operate the buzzer directly. Figure 8 is the diagram for the buzzer circuit.

We are able to produce different tones by varying the frequency of the square wave signal generated by the MSP432. Increasing the frequency raises the pitch of the buzzer, while lowering the frequency decreases it. This method is used to generate distinct tones. To generate these varying frequencies, we utilize the special I/O functionality of Timer A1 to output a PWM signal to the operational amplifier. The output frequency is determined using the equation $CCR[0] = (\frac{3Mhz}{Fc}) - 1$ where Fc is the desired frequency. By assigning the target frequency to the variable Fc, we can output that frequency from the MSP432 at a 50% duty cycle.

Figure 7: AC-1005G-RPA-LF Buzzer

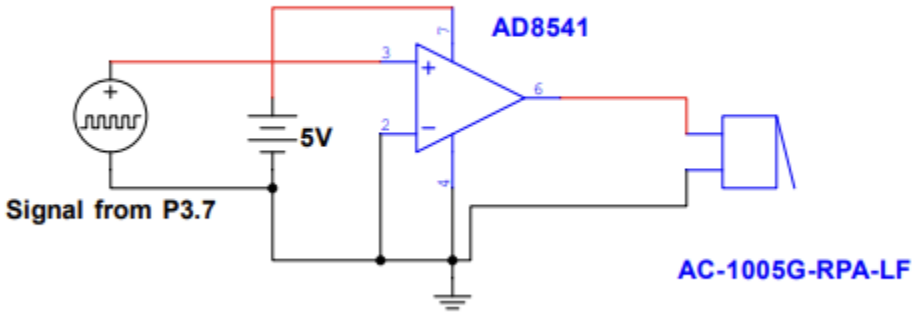

Figure 8: AC-1005G-RPA-LF Buzzer Circuit

The ADMP504 (Figure 9) is a compact microphone that produces a low-level analog output in response to sound. For our project we are using it to detect the sound of a hand clap. When a clap is detected, the microphone sends a signal to the MSP432, which is used to start or stop the vehicle. This microphone requires other components to function properly. A 0.1µF ceramic capacitor is placed into the same row as the power supply pin, this is used to eliminate any noise coming from the power supply. A 1µF electrolytic capacitor is placed in row with the output of the microphone and is used to eliminate any noise from the microphone. Figure 10 shows the circuit diagram for the clap detection circuit. Initially, we attempted to use an operational amplifier to amplify the output voltage; however, the circuit did not function as intended. Due to the very low output voltage of the microphone, we decided to use the AD8561 [8] comparator op-amp along with a very low reference voltage to produce a higher output voltage that the MSP432 could reliably detect.

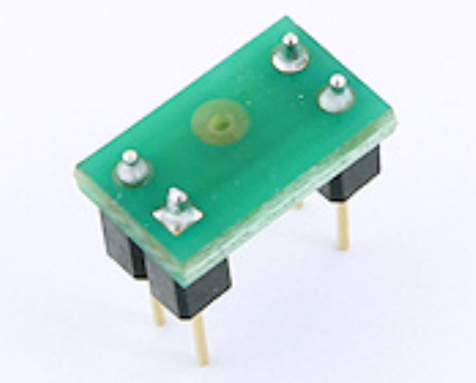

Figure 9: ADMP504 Microphone

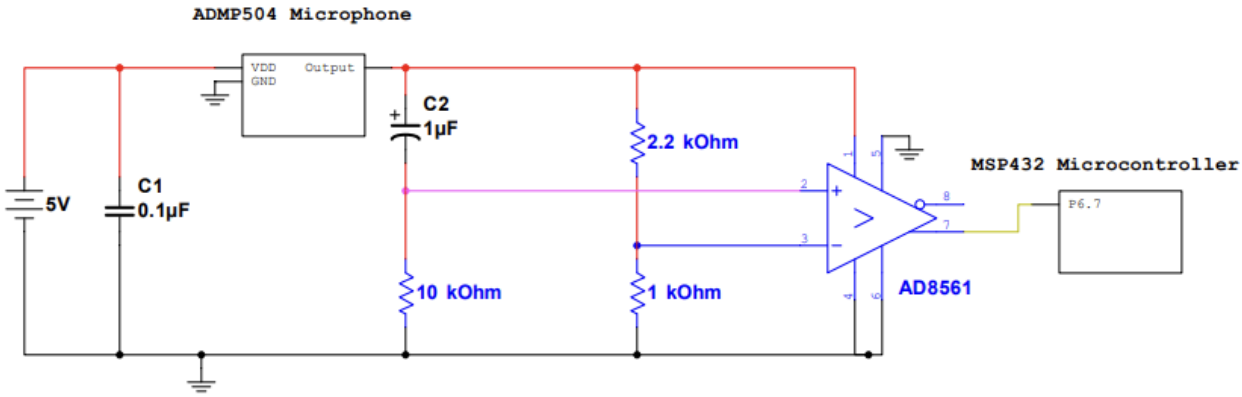

Figure 10 : ADMP504 Microphone Circuit

Software:

The software we utilized for this project consisted of two major IEDs with both of them running in C programming language. We utilized the Energia IED to program successfully the CC3200 Wi-Fi Integrated board, while we used the Keil uVision 5 to program the MSP432P4111. In this section of the lab report we will be talking about some key components of the software/ programming created for our project the "Autonomous and Multimodal Vehicle Control" to be equally successful in its interaction with WiFi control.

CC3200 - Energia

The CC 3200 board has a built-in IED that allows us to see all past examples of how to interface with Wi-Fi. This IED is called Energia. In the Energia ino file we were able to create due to past examples of how to interface correctly with any desired Wi-Fi network. We first configured our ino file to be able to connect to our Wi-Fi network, which will then result in an IP address unique to that specific Wi-Fi network. Therefore, any connection to the CC 3200 board will require the user to be in the same Wi-Fi network as it was configured.

Once we have an IP address due to Wi-Fi connection, we will then declare all the general purpose I/O pins that we will be using. There are six in total, one for the activation pin, one to determine the mode the robot will be in, and the other four pins will determine in which direction the robot will move. These four pins consist of directions such as forward, reverse, right shift, and left shift. The pins configured to the CC3200 board are stated in the sketch diagram of the robot.

The built-in function provided by the " Wi-Fi.h" library allows us to create a website with our desired information by simply writing our IP address in any web browser. This function is called " client.println", this function allows the web browser to determine that the following set of instructions are to be labeled in HTML coding. HTML coding what is not a topic we covered in this course, however we

have experience that allowed us to create this webpage. We created four separate web pages all linked together through a series of buttons. The first web page labeled “Activation”, was to initialize the robot to move, the second web page labeled “Modes” allowed us to decide on which of the two modes the user wanted. Each of the modes has its own unique web page, in the controlling mode we have six separate buttons, four for directional control, one stop button, and one button to return to the main page. In the next section of the report we will be analyzing some importante HTML coding that allowed us to create the website to interface the CC3200 LaunchXL with the MSP432 board.

HTML - CC3200

To first understand HTML programming, we first must declare the most important rule in HyperText Markup Language, which would be its very strict structure that must always be followed for it to function as it was intended. HTML programming basically dictates how we want the text to be displayed in an easy to understand manner. The following four rules exist for HyperText Markup Language.

Rules:

1. *There must always be an initializing coding line that denotes the document as a HTML website. Ex: “<!DOCTYPE html>”. Declares that this is an HTML document.*
2. *We must initialize two sections of the HTML website to write proper code. The <html> opening case, alongside its ending bracket </html> provides the location for the web server to determine where our data should start. And the second section that works of utmost importance would be*

<body> and its closing case </body>. This code determines the data inside the body such as titles, headers, and paragraphs.

3. *The third rule of HTML programming is that each opening case, for instance <h1> "a heading denotation", must always have an ending case to tell the webpage when the data send is done. Unless we are talking about adding a break on the section of code which is just one singular,
.*
4. *The last rule of HTML would be that if we wish to program a button, we must always give that specific button, <button>, a special class name to only denote that specific button. For instance, <button onclick ="location.href='TURN_OFF'"> Button 1</button>. This creates a button called "Button 1" which will then send that data to the webpage if it is clicked, resulting in "https://webpage_name/TURN_OFF".*

This following example shown below is a simple HTML code example that will represent how we would code with simple HTML coding and how that would translate to our Energia code that generates the same webpage.

```
<!DOCTYPE html>
<html>
    <head>
        <title>MSP CAR - Directinal and Activation</title>
    </head>
    <body align=center>
        <h1 align=center><font color="black">ESET 369 FINAL</font></h1>
        <button onclick="location.href='/Following'"><font color="green">Following</button>
        <button onclick="location.href='/Control'"><font color="red">Control</button><br>
        <button onclick="location.href='/TURN_OFF'"><font color="red">TURN OFF</button>
    </body>
</html>
```

Figure 11: Raw HTML Program for "Mode Choosing" Web page

```
client.println("<meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0\">");
// the content of the HTTP response follows the header:
client.println("<html><head><title>MSP CAR - Directinal and Activation</title></head><body align=center>");
client.println("<h1 align=center><font color=\"red\">ESET 369 FINAL</font></h1>");
client.print("MODE <button onclick=\"location.href='/Following'\"><font color=\"green\">Following</button></font>");//Mode
client.println(" <button onclick=\"location.href='/Control'\"><font color=\"red\">Control</button><br></font>");//Mode
client.print("Turn OFF! <button onclick=\"location.href='/TURN_OFF'\"><font color=\"red\">TURN OFF</button></font>");//TURNED_OFF
```

Figure 12: Energia Programing Code for "Mode Choosing" Web Page

Figure 13: "Mode Choosing" Website

As we can see from the two codes above in *Figure 11* & *Figure 12,* we can see that the code in raw HTML is inadvertently different from our energia code due to the fact that we need to send the HTML code to the web server to create the web page in question. Therefore, to do this, we require all our syntax for HTML to be noticeable to the web server with the use of "\" when there would be a quotation mark needed to identify that attribute. For instance, *<button onclick ="location.href='TURN_OFF'">* would just be *<button onclick=\"location.href='/TURN_OFF'\">* in Energia's built- in Library function.

Each time we click a button we send that specific data from the button to the buffer that will then distinguish between which button was pushed / activated from its location reference. For instance, if the user pushed the following mode button, it would result in our output pin going high and the yellow light emitting diode on the CC3200 board being on. The web page generated by the CC 3200 board is always looking for the user after the user has clicked a button. For instance, in *Figure 14*, we can see that we decide what course of action to take after the buffer is deciphered. For the example below, we turn off the Red LED, while turning the Power Pin (Pin P15 in CC3200) active, so later we can transmit this high to the MSP432 microcontroller.

```
if (endsWith(buffer, "GET /TURN_OFF")) {
  digitalWrite(RED_LED, LOW);                // TURN OFF THE ROBOT
  Serial.print("\nOFF!\n");
  digitalWrite(Power, LOW);
  Page = -1;
}
```

Figure 14: If the "TURN OFF" button is pressed.

This program will be provided on the reference sheet alongside a file submission in conjunction with this report.

MSP432 - Keil uVision 5

The Keil uVision 5 has been the most frequent IED we have been working on through the past semester. Therefore, we have the most experience with programming the MSP432 board. In the programming for the Keil uVision 5, we generate two C files and one C header file. One C file works to store all our functions used for the computer project, meanwhile the other C file is the actual file we are running on our MSP 432 board. The C header file works to connect these two separate C files together to work in conjunction with one another. Thus making our code much more organized and easier to maintain.

C Header File - "Final_Project.h"

In the following few paragraphs we will examine the key components of this C Header File and why it is insurmountable in importance to keeping a cleaner and more concise program code.

The C Header File allowed us to connect two different C files by calling this very same header file on both of the files. Effectively creating a very short .c file that only runs the most critical part of our code to work, while the other .c file has all the function's information running in the "background". In this header file, we identified 16 functions alongside one integer variable that would be shared between these two files. Of these 16 functions created, we created six functions to initialize the specific pins required for each specific action of the multimodal car, while we created one timer function, and the remaining functions are used to generalise the action of a our multimodal car; this functions involve actions such as setting motor speed, calculating distance from ultrasonic sensor.

```
#ifndef Final_Project
#define Final Project

// Motor Functions
void Motor_Init(void);
void motors(float, float, int, int);

//Ultrasonic Sensor
void Sonic_Init(void);
void sensor_activation(void);
float conv(int x,int y);
void state(float sensor_dis);

//WiFi Functions
void WiFi_init(void);
void WiFi_Control(void);

//Microphone Functions
void mic_init(void);

//Following Mode
void Following(void);

//Buzzer Functions

void buzzer_init(void);
void map(void);
void foreward_sound (void);
void reverse_sound(void);
void stationary(void);

//Timer Delay Functions
void delayMs(int n);

//Innitializes Everything at once
void all_init(void);

//Variables needed in both Files
extern int Clap;

#endif
```

Figure 15: C Header File - ESET369_Final.

Secondary C File - "ESET369_Functions.c"

The previous C header file basically works as a pointer to our C main file to locate the actual location of the functions utilized which is this C secondary file. This C file will hold the actual coding of the pointer functions. However, we will not explore each and every function generated due to the vast amount of programming code. We will focus on the two main functions that determine the actual modes

the Multimodal Vehicle Control will have. These two main functions are known as the “Following” and the “WiFi_Control” functions.

Following():

```
void Following(void){
                //Activate Sensor
                sensor_activation();
                TIMER_A0 ->CTL |= 0x0024;
                while(Global_Variable == 0){
                }
                Global_Variable = 1;
                delayMs(200);
                return;
        }
```

Figure 16: “Following()” Function

As we can see in *Figure 16*, when we call the “Following ( )” function, the first course of action would be the activation of the Ultrasonic sensor with a 10 microsecond pulse that is being done with the “sensor_activation()” function. The next course of action will be to initialize the timer to properly capture the time it takes for the ultrasonic sound wave to bounce back from the nearest object in a 30-degree arc from its front. The next step of the code would be to get into an infinite while loop. The code will not be able to leave this while loop until the interrupt GPIO of the falling edge of the capture mode of the pin is captured and thus will cause us to go into our Interrupt Service Routine shown in *Figure 17*.

```
void PORT3_IRQHandler(void){
                SR = TIMER_A0->CCR[1];
                SF = TIMER_A0->CCR[2];
                myA = conv(SR, SF);


                if (myA <= 0){
                        myA = 0.35;
                        state(myA);
                }
                else{
                state(myA);
                delayMs(60);
                Global_Variable = 1;
                }

                P3->IFG &= ~0x04;

}
```

Figure 17: "PORT3_IRQHandler" Interrupt Service Routine

Once the hardware detects the falling edge of the echo pin of the ultrasonic sensor, it will first store the rising edge and falling edge of the "Timer A's Capture Mode". We will then input those two integer values into the "conv(rising edge, falling edge)" function, which takes the two inputs and gathers the difference between them to put them into the following equation as the alpha.

Equation for Ultrasonic Sensor Distance:

$$Distance(M) \ = \ (\frac{1}{2} * \ 343(M/Sec) \ * \frac{1}{3MHz} * \ \alpha)$$

This result from the "conv()" function will result in the distance between the ultrasonic sensor and that of the nearest object. Afterwards, we will get the distance generated by the "conv()" function to put it into our "state(distance)" function, which basically functions as a state machine which will determine if the object falls between three categories. These three categories are seen in its completeness in *Figure 18.* If the distance is further than 1.1 meters then the robot should not move, if it's between 1.1 to 0.5 meters then the robot will follow the object and will follow forward. However, if the distance between the object and the sensor is less than 0.25 meters then the robot will reverse. If the object finds itself in between 0.5 to 0.25 meters then the robot should stop.

```
void state(float sensor_dis){          //This function decides how the robot should move
        float far, mid_1, small;                        // with the input of the sensor disctance.
        far = 1.1;
        mid_1 =0.5;
        small = 0.25;

        if (sensor_dis >= far){ //If person out of 1.1 meter range
                motors(0.0, 0.0, 1, 1);                 //Robot will not move
                return;
        }
        else if(( sensor_dis > mid_1) && (sensor_dis < far)){
                motors(0.3, 0.3, 1, 1);         //Person in between 1.1>x>0.5 meter range
                //Robot will accelerate with a 30% PWM
                return;
        }
        else if(( sensor_dis <= mid_1) && (sensor_dis > small)){
                motors(0.0, 0.0, 1, 1);                 //If Person between 0.25 < x <= 0.5 meter range
                return;
        }
        else if(sensor_dis <= small){   //If person closer than 0.25 meters
                motors(0.3, 0.3, 0, 0);                 //Robot will reverse back until in 0.25 range.
                return;
        }
        return;
}
```

Figure 18: State Machine Function

As we can see in the Figure above, each state will set our motors with a predetermined duty cycle of 30% in both forward and reverse configurations. And to finalize our interrupt, we will then declare the Global Variable to be a value of 1 after a 60 mille second delay and after clearing the interrupt flag. This changing of the Global variable will allow us to leave our while loop found in our original "Following" function, thus ending the whole iteration of the function to end with only one action done by our robot. This mode will only work if it is located in a while loop.

WiFi_Control():

This main function code is rather simplistic in comparison to the previous function due to its logical implementation. We will be able to see this implementation as seen in the *Figure Below.*

```
void WiFi_Control(void){
        while(((P3 -> IN & 0x20)== 0)){
                //Foreward if Foreward Button is pressed
                if(((P5 -> IN & 0x01)!= 0)){
                        motors(0.9,0.9,1,1);
                }

                //Reverse if reverse Button is pressed
                else if(((P5 -> IN & 0x02)!= 0)){
                        motors(0.9,0.9,0,0);
                }

                //Left Shifting if left shifting Button is pressed
                else if(((P5 -> IN & 0x04)!= 0)){
                        motors(0.99,0.99,0,1);
                }
                //Right Shifting if right shifting Button is pressed
                else if(((P1 -> IN & 0x80)!= 0)){
                        motors(0.99,0.99,1,0);
                }
                //If all are off, when the STOP button is pressed, the robot will not move
                if(((P1 -> IN & 0x80)== 0)&&((P5 -> IN & 0x04)== 0)&&((P5 -> IN & 0x02)== 0)&& ((P5 -> IN & 0x01)== 0))
                        motors(0,0,1,1);
        }
}
```

Figure 19: WiFi_Control Function

As we can notice in *Figure 19,* for the control mode to operate to its full capability, we must first configure all the pins connected from the CC3200 board directly to the MSP432 in its correct configuration. This is because every button pressed in this mode will result in a high output (CC3200) to be seen in the input of the MSP432 board. Therefore, each pin is checked continuously and will only follow the desired direction if the user has pressed the button in the webpage. The primary function seen

in this main function is the “motors(float %, float %, int a , int b)”. This function will allow the user to dictate the duty cycle for the Pulse Width Modulation in a float value to the left and the right motors as the first and second input respectively. Meanwhile, the third and fourth input (*int a, int b*)work as a directional range for left and right motors respectively. Meaning, a 1 will indicate that side of the motors to go forward, while a 0 will result in a reversing motion. This can be shown in the following Figure down below*, Figure 20.*

```
void motors(float left, float right, int Left_Dir, int Right_Dir ){  //Function that moves motors
        int n,v;
        TIMER_A2 ->CCR[1] = 0x0000;                             //Load PWM in Register for TimerA
        TIMER_A2 ->CCR[2] = 0x0000;
        n = 60000 * left;
        v = 60000 * right;
        if ((Right_Dir == 0) && (Left_Dir == 0)){               //Reverse
                P4 ->OUT &= ~0xAA;
                P4 ->OUT |= 0x22;
                //reverse_sound();
        }else if ((Right_Dir == 1 )&&(Left_Dir == 0)){ //Turn Left
                P4 ->OUT &= ~0xAA;
                P4 ->OUT |= 0x28;
        }else if((Right_Dir == 0 )&&(Left_Dir == 1)){   //Turn Right
                P4 ->OUT &= ~0xAA;
                P4 ->OUT |= 0x82;
        }else{                                                                                  //both straight forward
                P4 ->OUT &= ~0xAA;
                P4 ->OUT |= 0x88;
        //      foreward_sound();
        }
        TIMER_A2 ->CCR[1] = n;                          //Load PWM in Register for TimerA
        TIMER_A2 ->CCR[2] = v;


        return;
}
```

Figure 20: Motors Function

Primary C File - “ESET369_Main.c”

In this C file, we will be including the main function which is the actual code that will be running through the MSP432 board while calling all the functions from the secondary C file. Therefore, our code below shown in *Figure 21* is the actual code that is being runned at all times.

```
int main(void){

        all_init(); //Initializes every sub init for every function used.

        while(1){
                if(Clap == 1){
                        while(((P3 -> IN & 0x20)!= 0)){ //If in Following Mode by checking P3.5 is High
                                Following();
                        }
                        while(((P3 -> IN & 0x20)== 0)){ //If in WiFi control mode by checkingP3.5 is Low - Default
                                WiFi_Control();
                        }
                }
                else if(Clap ==0){
                        //stationary();
                        //Stay Here if Clap is not Active
                }
        }
        return 0;
}
```

Figure 21: Main Function

Our code above, first initializes all the pins that will be used for each action through the use of all_init(). After that, we will instruct the MSP432 to be running in a continuous loop that will be checking if the “Clap” of the microphone was being detected. If it is activated by a loud sound, “Clapping”, then the program will check if the mode has been selected through the web site created by the CC3200. The WiFi control mode is always on by default, while the Following mode will only be active if Pin 3.5 is high, which is the equivalent of pressing the “Following” button in the web site. In comparison, if we don't activate the system with a “Clap”, then the robot will remain in stagnation until otherwise activated.

All this will conclude with the major components created for the software of the Multimodal Vehicle Control to work completely to its objective.

Test Plan and Results:

The first task we focused on was installing the CC3200 onto the robot chassis that would allow easy access to the microcontroller's pins. Initially, we considered mounting the CC3200 on the underside of the chassis, but the pins were not as accessible and the wires were not long enough to reach the pins. We then decided to mount the CC3200 on top of the MSP432. However, to prevent any accidental shorts, we added insulation between the two microcontrollers. We used a thick piece of foam found in the lab for insulation and secured the CC3200 to the chassis using multiple zip ties. We then connected six wires from the CC3200 to the MSP432 as shown in our main circuit diagram (Figure 23) and powered the board using the 5 volt output from the H-bridge.

Once the CC3200 was installed, we turned our attention to installing the microphone circuit (Figure 10) and the buzzer circuit (Figure 8) onto the robot chassis. We condensed both circuits to fit on a single breadboard that will be installed onto the chassis. This required significant preparation: we measured, cut, and stripped wires to minimize the size of the circuits. We then inserted the breadboard in between some of the linkages of the chassis, and secured the breadboard using multiple zip ties on either side. We then connected the 5 Volt output from the H-bridge to power the two circuits, and grounded the circuit to the ground input on the H-bridge.

After installing the CC3200 as well as the microphone and buzzer circuit we focused cleaning up the connections and making sure all wires were securely connected to the pins of the microcontrollers. We then attached some quick connectors to the main power supply wires, which allowed ease of swapping battery packs. We also experimented with using cordless tool batteries which provided 20 volts with 1.5 Amp hours. This greatly improved the speed of the vehicle however we ran into issues with this power source. When powering the H-bridge motor driver with more than 12 volts, the 5 volt output regulator on the H-bridge does not work properly. Since we were relying on the 5 volt output from the H-bridge to power our two microcontrollers, microphone circuit, and buzzer circuit we had to use an additional 5 volt

power supply along with the 20 volt tool battery. This made the circuit on the vehicle even more difficult and complicated to implement, in the end we decided to return to the 12 volt battery pack.

After completing the hardware components (Figure 22), we began developing the software to interface with the vehicle over Wi-Fi. We started by researching further into the CC3200 microcontroller, focusing on its Wi-Fi capabilities and the available functions within its libraries. Using the Energia IDE, we programmed the CC3200 and referenced example code to establish wireless communication with devices such as laptops, tablets, and smartphones on the same network. Once we successfully connected to the CC3200, we updated our existing code for the MSP432 using the Keil IDE to handle multiple inputs from the CC3200. Additionally, we created a simple HTML-based website to serve as the graphical user interface (GUI).

Once the simple HTML website was created, it was a simple implementation of software due to already having the following mode implemented in past versions of the vehicle. We created the software as stated in the *Software* section of the report. The main change between this software and the previous iteration of the code would only be its organization of its functions readability due to library implementation.

For the testing phase of our project for the software, we implemented the serial communication known as “UART” to determine data, distance, as well as where in our code the robot was stuck or non operational. This serial communication allowed us to visually understand what was wrong with our code, however, due to the nature of the robot being autonomous and containing limited wired connection to a computer in its complete operation, we decided to remove all the software from the MSP432 pertaining to serial communication. This resulted in a more organized and readable code in the secondary C File as well as the C Header File.

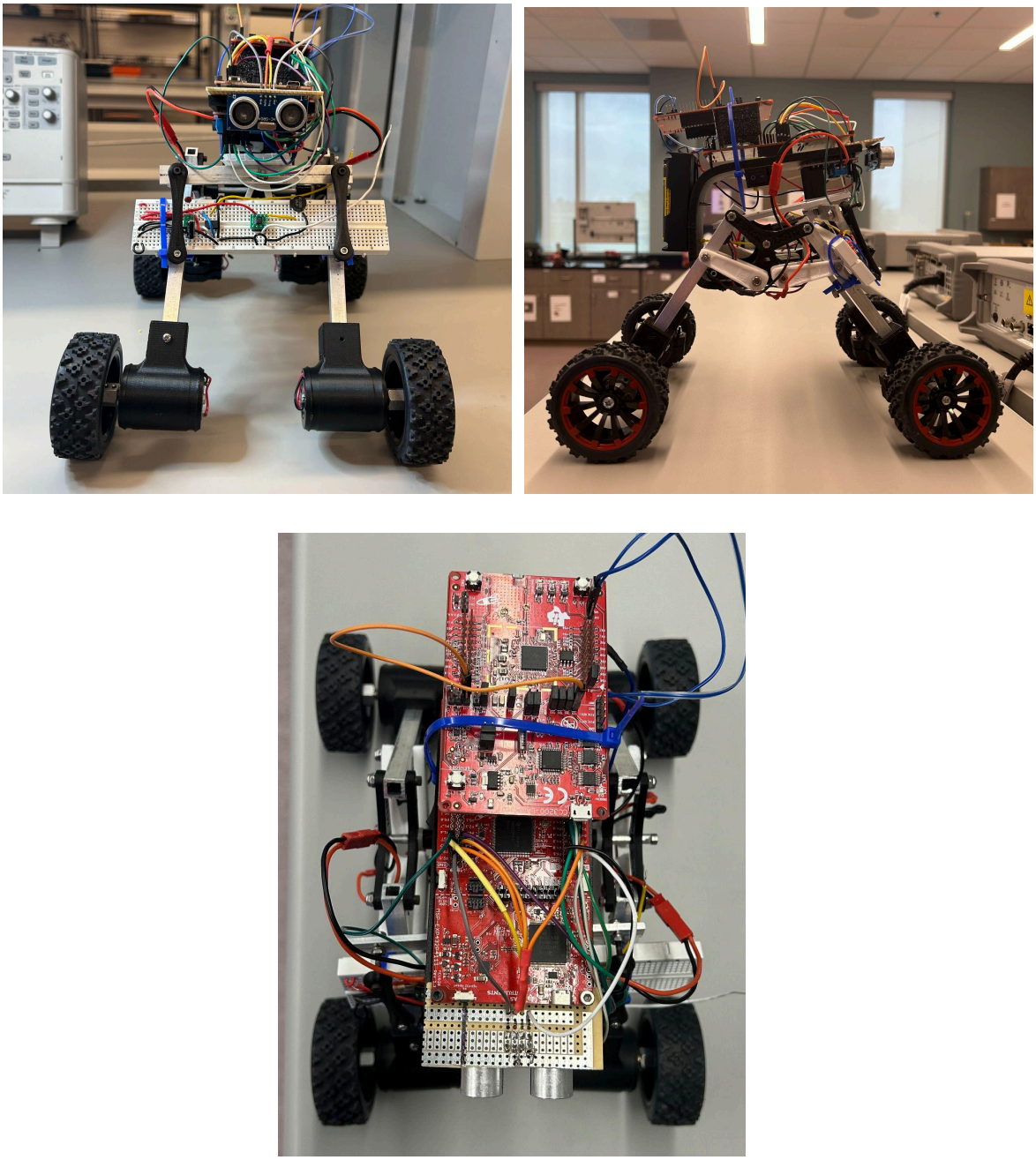

Figure 22: Completed Vehicle, Front View (Left), Side View (Right), and Top View (Bottom)

Description of System Operation:

The robotic vehicle has two modes of operation, each of which is selected through a remote device connected wirelessly to the CC3200. The CC3200 then interfaces with the MSP432 to configure the robot for either Following Mode or Remote Control Mode.

The first mode is Following Mode. In this mode, the vehicle awaits a clap from the user to begin following; a second clap will stop the vehicle. Once the vehicle detects the first clap, it uses the ultrasonic sensor to follow any object within a set distance. If an object gets too close, the vehicle will reverse while playing a reversing tone on the buzzer. If the sensor detects an object within the designated following range, the vehicle will move forward toward the object while playing a steady tone on the buzzer.

The second mode is Remote Control Mode. In this mode, the vehicle is remotely controlled by a wireless device that interfaces with the CC3200. This is accomplished by connecting both the wireless device and the CC3200 to the same Wi-Fi network and accessing a custom-made website on the wireless device. The website features four controls: Forward, Reverse, Rotate Left, and Rotate Right. When a button is pressed, the CC3200 sends a corresponding signal to the MSP432, which in turn sends commands to the motor driver, moving the vehicle in the selected direction.

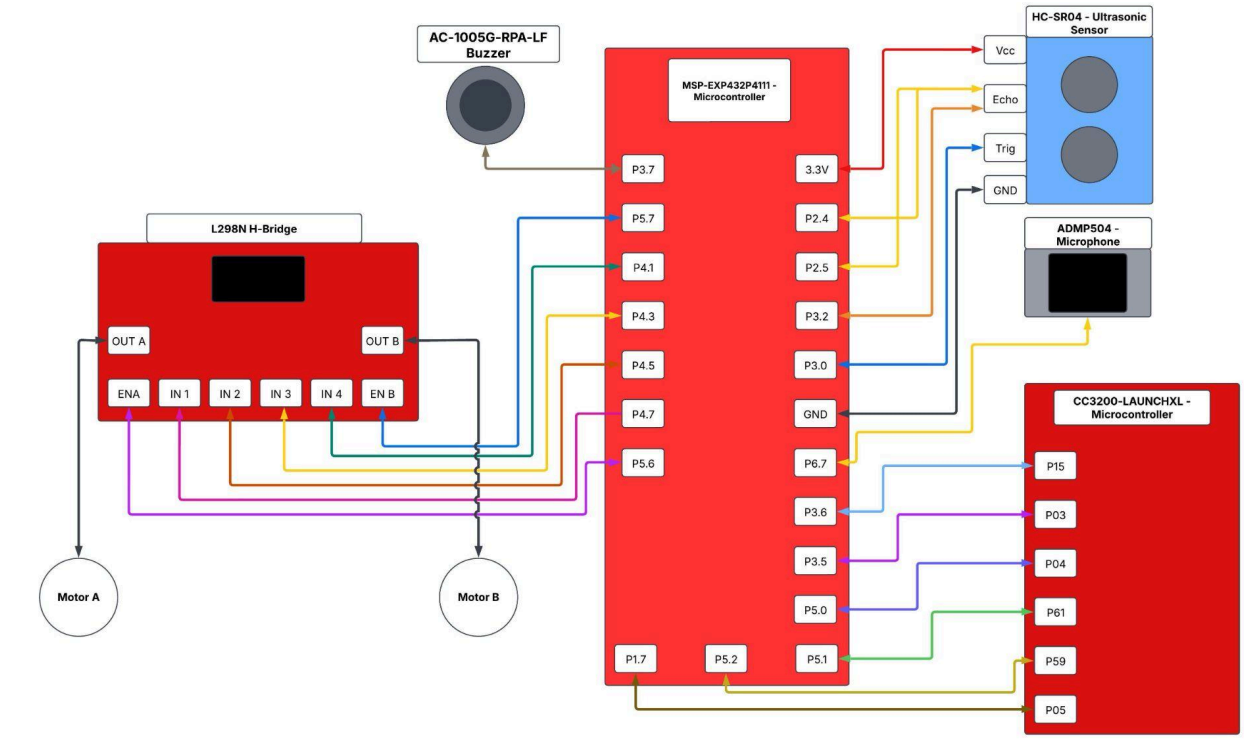

Figure 23: Main Circuit Diagram (All connections to the MSP4322)

Conclusion:

In conclusion, we were able to implement various types of lessons taught to us through the natural track of the spring semester. Lessons such as code readability & organization, how to implement interfacing with sensors, as well as serial communication. However, what makes this final project so special is that we were able to implement a new type of Microcontroller (CC3200 LaunchXL) that then allowed us to interface wireless communication with WiFi. This new way of sending and receiving data allowed us to create a wirelessly controlled vehicle with little knowledge in HTML programming.

However, we still wish to build upon this project with some improvements we saw while testing and completing this project to its completion. These improvements, for future students, would be like changing the power source to a more powerful source so the motors can work better under stress and under its relatively (to its body) heavy weight. Another notable improvement, would be the importance of improving on our web server to be more user friendly and have more appeal than its brutish appearance as of this moment.

All in all, this final project allowed us to both grow in programming embedded systems in C programming language as well as grow in developing circuits to function to our needs. This project allowed us to both learn and develop new ways to interface wirelessly with a system. In conclusion, this was a great experience and final project to make due to vast topics covered in relation to the course material, implementing serial communication, interrupts, sensor interfacing, etc.

References:

[1] “MSP-EXP432P4111 | DigiKey Electronics,” *DigiKey Electronics*, 2025. https://www.digikey.com/en/products/detail/texas-instruments/MSP-EXP432P4111/8347679?s=N4IgTCBcDaILYGcAOAWAzGVBGHIC6AvkA (accessed Apr. 26, 2025).

[2] Handson Technology, “Handson Technology User Guide HC-SR04 Ultrasonic Sensor Module User Guide User Guide: Ultrasonic Sensor V2.0.” Available: https://www.handsontec.com/dataspecs/HC-SR04-Ultrasonic.pdf (accessed Apr. 26, 2025).

[3] Handson Technology, “Handson Technology User Guide L298N Dual H-Bridge Motor Driver.” Available: https://www.handsontec.com/dataspecs/L298N%20Motor%20Driver.pdf (accessed Apr. 26, 2025).

[4] Texas Instruments, “CC3200 SimpleLink™ Wi-Fi ® and Internet-of-Things Solution, a Single-Chip Wireless MCU,” 2013. Accessed: Apr. 26, 2025. [Online]. Available: https://www.ti.com/lit/ds/symlink/cc3200.pdf

[5] Advanced Acoustic Technology Corporation, “AC-1005G-RPA-LF ,” Jun. 2018. https://www.mouser.com/datasheet/2/1005/20200508291bd-2324810.pdf?srsltid=AfmBOooX32kxmExAefZGovc5SA8fRAT7A5bJPZJyh3yrbfz_wWkHmP-x (accessed Apr. 26, 2025).

[6] Analog Devices, “Ultralow Noise Microphone with Bottom Port and Analog Output.” Accessed: Apr. 26, 2025. [Online]. Available: https://www.analog.com/media/en/technical-documentation/obsolete-data-sheets/admp504.pdf

[7] Analog Devices, “AD8541,” *Analog.com*, 2023. https://www.analog.com/en/products/ad8541.html (accessed Apr. 26, 2025).

[8] Analog Devices, “Ultrafast 7 ns Single Supply Comparator .” Accessed: Apr. 26, 2025. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/AD8561.pdf