runlinc Project 15 AI6: Simple AI Voice Chatbot (E32W Version)

Contents

Introduction

Problem

How can we use microchips to turn on objects by asking an AI to do it by speaking to it? What if we're too hot and want to be cooled down?

Background

By learning E32W, you can learn to program microchips to tell them what to do. We can also make it better by letting the microchips interact with us in a more direct way, which is to talk with us directly. We can chat with it to let it commence some commands. There are two ways of language input: reading and listening. Previously, we've already made a chatbot with simple functions that can interact with humans by reading what humans type into a chat box. Can we make another way of input happen, which is to let the microchip listen to what we say and give an appropriate response?

Ideas

Look at the E32W controller board and the control webpage. Can you see any inputs, i.e. something that we can touch or change to tell the microchip something? What about an output, i.e. something the microchip can change to tell us something? What kind of inputs and outputs are generally on an alarm system? What outputs can we give our AI to use?

Plan

So, we want the runlinc AI to be able to talk to us and others. First, we are going to need a way to talk to our AI. We'll also need to give our AI information so it can talk and give responses. To do these, we'll have to create an input through our webpage, which will allow our AI to talk to us. We will build the code for voice recognition, voice output and some simple reactions for some simple voice inputs.

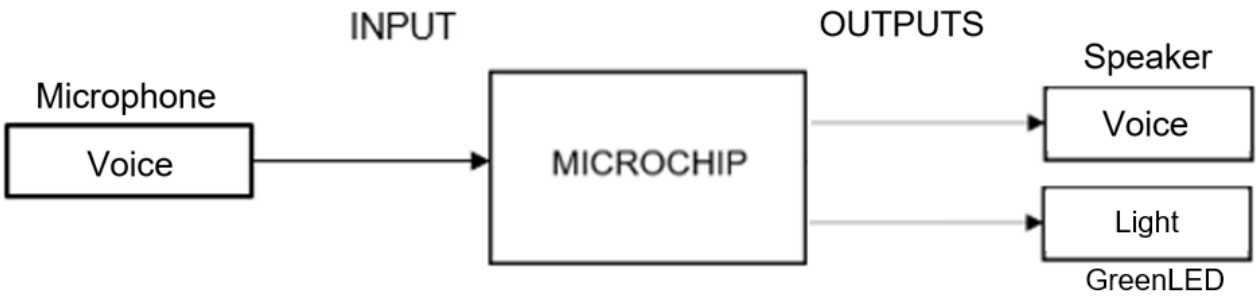

Figure 1: Block diagram of Microchip inputs and outputs

runlinc Background

runlinc is a web page inside a Wi-Fi chip. The programming is done inside the browsers compare to programming inside a chip. The runlinc web page inside the Wi-Fi chip will command the microchips to do sensing, control, data logging Internet of Things (IoT). It can predict and command. It can also use the webpage to gain access to input on our PCs like camera and microphone.

Part A: Design the Circuit on runlinc

Before proceeding, ensure you can access the runlinc IDE via a secure HTTPS connection. You can do this by making sure the E32W controller is connected to the network and that PageKite has been set up correctly and by using the PageKite webpage. For more details, refer to the Appendix A section of the Setup Guide for STEMSEL E32W V1.0. Ensure you have a functional microphone and a functional speaker or headphones connected to your PC.

The following works need to be done on the PageKite page.

Note: Refer to runlinc Wi-Fi Setup Guide document to connect to runlinc

Use the left side of the runlinc web page to construct an input/output (I/O).

For port D5 set it as DIGITAL_OUT (used as a red pin of LED – no name needed).

For port D18 name it lamp and set it as DIGITAL_OUT.

For port D19 set it as DIGITAL_OUT (used as negative pin of LED – no name needed).

In our circuit design, we will be using a 3-pin LED module. We happen to have these in our kits, so these can be used on our circuit design, as per the plan.

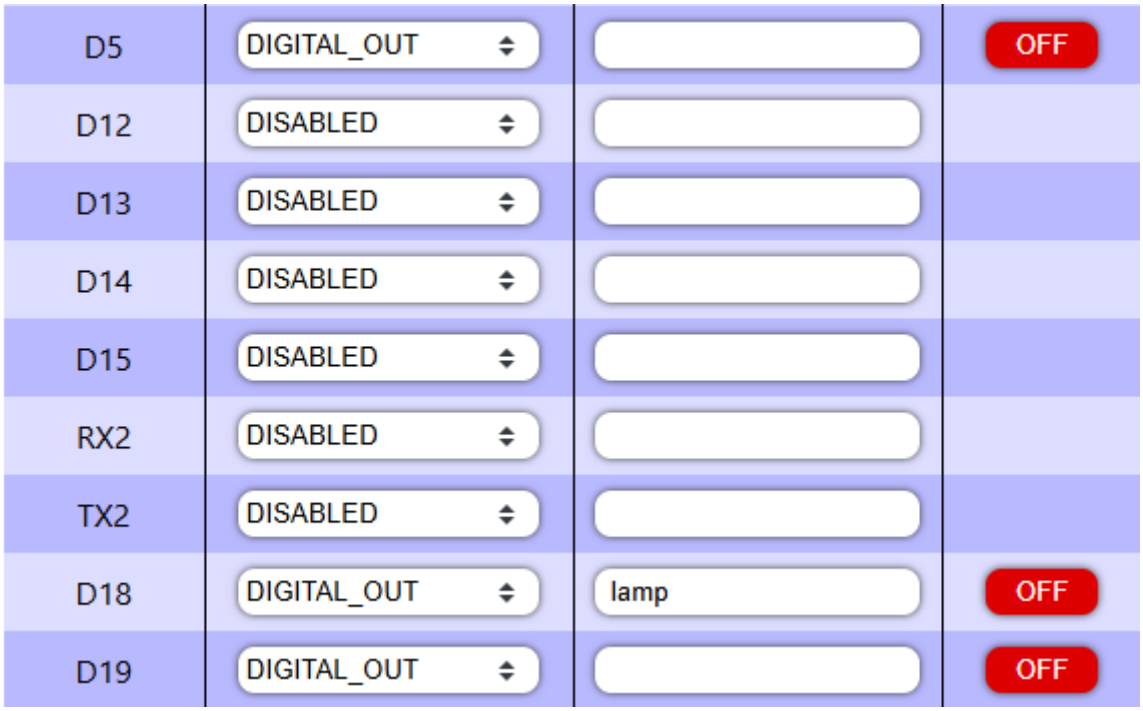

Figure 3: I/O configurations connections

Part B: Build the Circuit

Use the STEMSEL E32 board to connect the hardware. For this project we are using both the left and right I/O ports, with **negative port (-ve)** on the outer side, **positive port (+ve)** on the middle and **signal port (s)** on the inner side (as shown below).

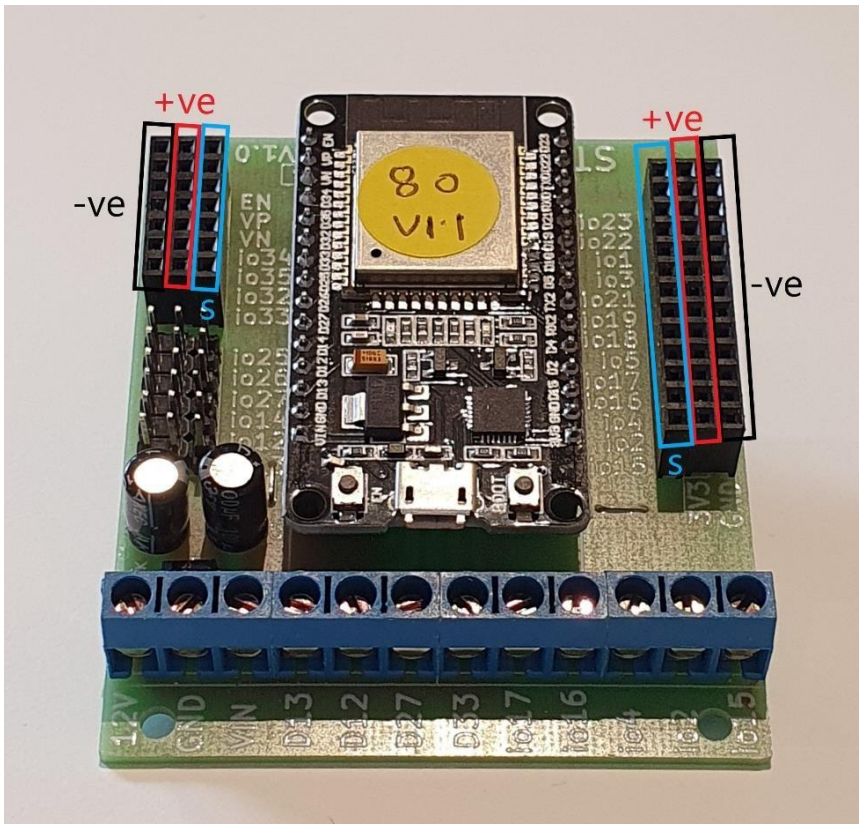

Figure 4: Negative, Positive and Signal port on the E32 board

There is only one I/O part we are using for this project, a 3-pin LED light, its respective pins are shown in the figure below.

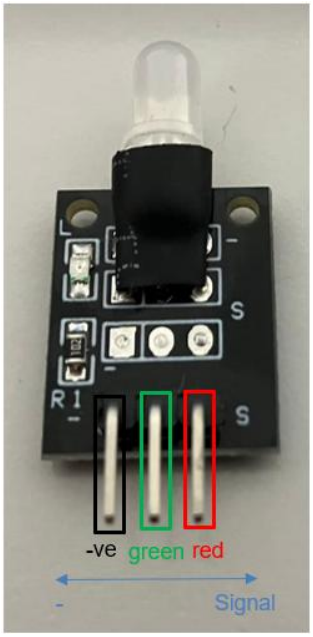

Figure 5: I/O parts with negative and signal pins indicated

Wiring instructions

a.) Plug in the LED to signal ports of io5, io18 and io19 on the E32W board with the “S” pin on the light goes into io5 (we did not configure port D5 on the webpage for this connection because for this 2 color LED light, the right pin which is connected to io5 is for lighting up the red color, and we don’t need it for this project).

b.) Make sure the (-ve) pin is on the GND (outer) side of the I/O port.

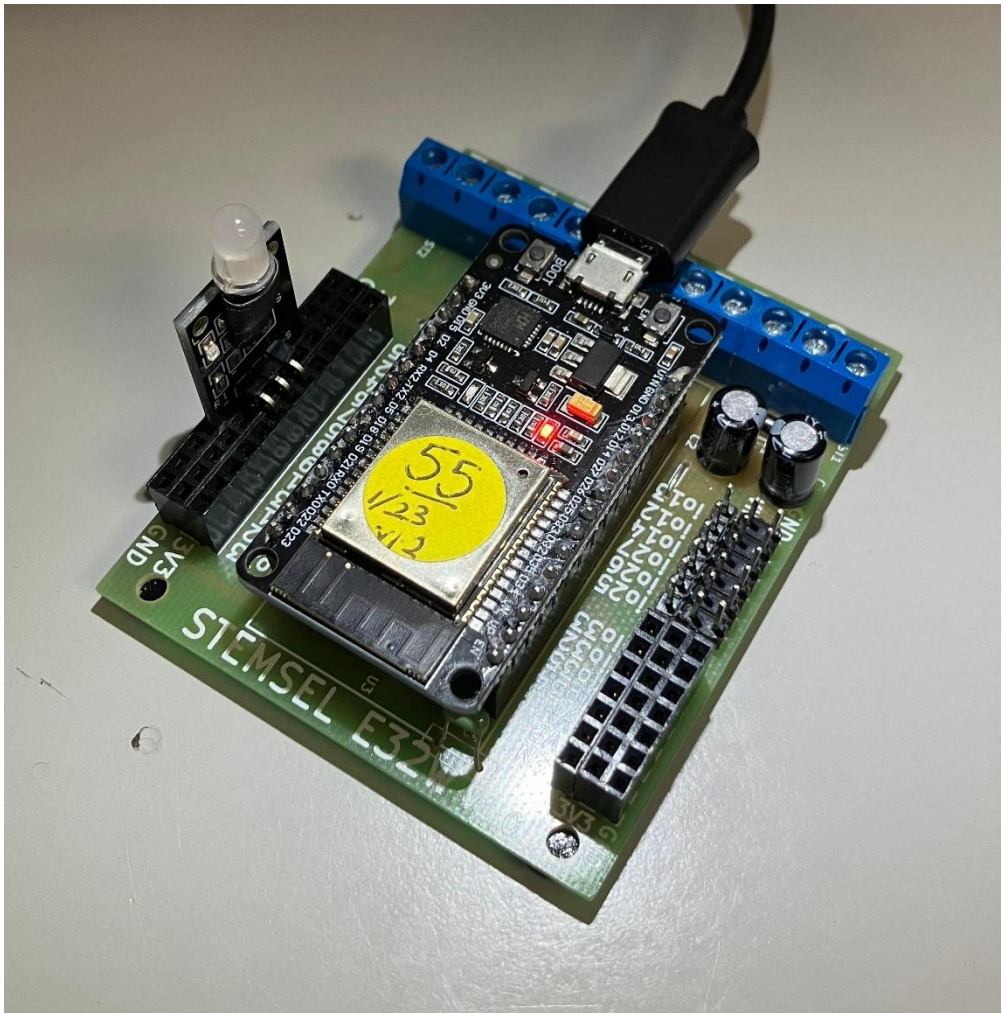

Figure 6: Circuit board connection with I/O parts (side view)

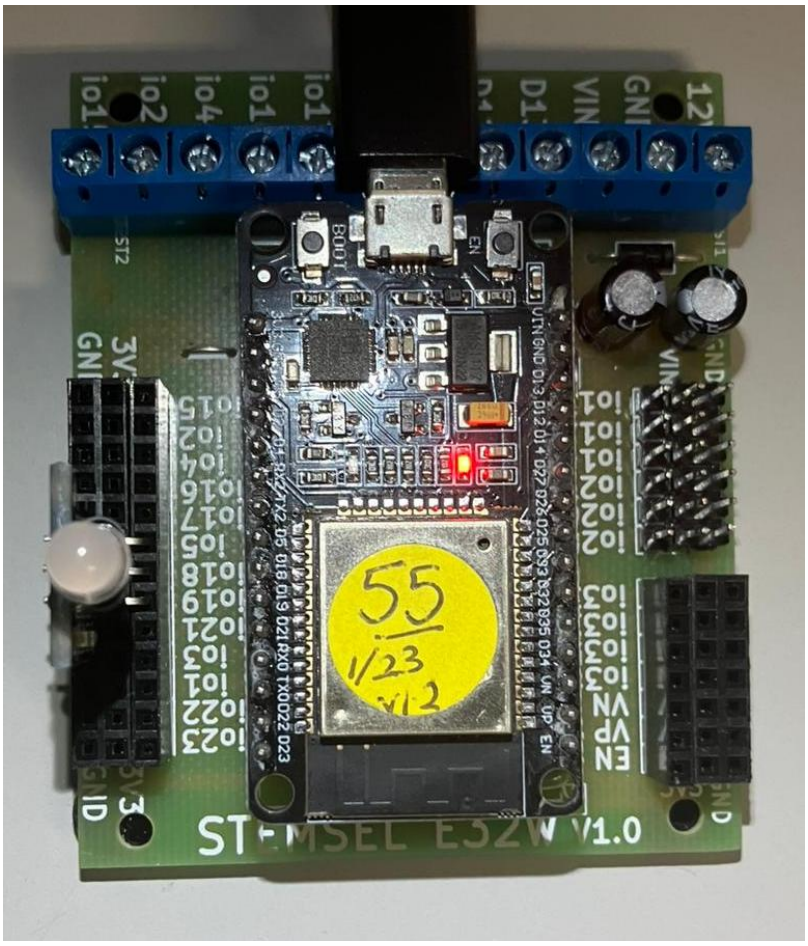

Figure 7: Circuit board connection with I/O parts (top view)

Part C: Program the Circuit

Use the blocks on the right side of the runlinc webpage to program the functions of the traffic light. Use the HTML to add contents, CSS to add style to your taste and Javascript to program the microchip. For this case, we will be using HTML and JavaScript to program our AI chatbot.

After naming the port C4, we are going to program the circuit. First, we need the **HTML block** to add in the graphics for what we are going to do.

In this step, we'll need to give our webpage a title first, let's call it, Voice Greeting App. To do this, we'll need to use the Heading tag <h1>.

```
<h1>Voice Greeting App</h1>
```

After we have our title, we can create the status text line saying the current status of the bot. To do this, we will use the div tag . This will make the text stand alone as a new line. We give it an id so that we can use JavaScript to automatically change the status.

```
<div id="status">Waiting...</div>
```

Now, we can add a line of text to show the last command the bot recognized. We still use the div tag and this time we give it a different id.

```
<div id="lastCommand">Last command: None</div>
```

Now that we have set up all the inputs and outputs, we can now look at creating our AI. To do this, we'll need to use the **JavaScript Loop block.**

To start with, we will create an array called recognition. We will use the contents in this array as the variables of the speech recognition function.

```
let recognition;
```

Now that we have set up the variable, we can give responses to our AI. To do this, it will need to receive what has been said, and then compare it with what it is expecting, which will allow it to give a response. The first block in this function is for the initialization of the speech recognition and setting of the variables that control its functions.

```
function initializeRecognition() {
  recognition = new (window.SpeechRecognition || window.webkitSpeechRecognition)();
  recognition.lang = 'en-US';
```

```
recognition.interimResults = false;
recognition.continuous = false;
```

The next block of code in this function is for the update of the status line and the last command line in the HTML.

```
recognition.onstart = () => {
    console.log("Listening...");
    document.getElementById('status').textContent = "Listening...";
};

recognition.onresult = (event) => {
    const transcript = event.results[0][0].transcript;
    console.log("You said: " + transcript);
    document.getElementById('lastCommand').textContent = "Last command: " + transcript;
```

The next block of code in this function is to setup the several reactions we want the bot to do for the different keywords we say.

```
        const userMessage = transcript.toLowerCase();

        if (userMessage.includes("hello")) {
            setTimeout(() => {
                speak("Hii, I am glad to meet you");
            }, 1000);
        } else if (userMessage.includes("too dark")) {
            speak("turning on light");
turnOn(lamp);
        } else if (userMessage.includes("too bright")) {
            speak("turning off light");
turnOff(lamp);
        } else {
            speak("");
turnOff(lamp);
        }
    };
```

The final block of the code in this function is for the error and the self-fixing code, which

```
    recognition.onerror = (event) => {
        console.error("Error occurred in recognition: " + event.error);
        setTimeout(() => {
            try {
                recognition.start();
            } catch(e) {
                console.error("Failed to restart recognition:", e);
            }
        }, 1000);
    };

    recognition.onend = () => {
        if (!window.speechSynthesis.speaking) {
            setTimeout(() => {
                try {
                    recognition.start();
                } catch(e) {
                    console.error("Failed to restart recognition:", e);
```

uses the restart of the recognition to fix it.

```
                }
            }, 1000);
        }
    };
}
```

Now, we need to let the system speak for some reactions to our keywords. The following code will convert the written sentences we set for the bot to speak into voice outputs, and update the status to “Listening” after the speech is finished.

```
function speak(message) {
    document.getElementById('status').textContent = "Speaking: " + message;
    const utterance = new SpeechSynthesisUtterance(message);
    utterance.onend = () => {
        document.getElementById('status').textContent = "Listening...";
        try {
            recognition.start();
        } catch(e) {
            console.error("Failed to restart recognition:", e);
        }
    };
    window.speechSynthesis.speak(utterance);
}
```

The next function is the function to use and begin voice recognition. It uses the previous functions we wrote to initialize itself and process inputs to give outputs.

```
window.addEventListener('load', () => {
    initializeRecognition();
    try {
        recognition.start();
    } catch(e) {
        console.error("Failed to start recognition:", e);
    }
});
```

At last, we put this code into the HTML block:

```
<h1>Voice Greeting App</h1>
<div id="status">Waiting...</div>
    <div id="lastCommand">Last command: None</div>
```

And this code into the JavaScript Loop block:

```
let recognition;
      function initializeRecognition() {
         recognition = new (window.SpeechRecognition || window.webkitSpeechRecognition)();
         recognition.lang = 'en-US';
         recognition.interimResults = false;
         recognition.continuous = false;
         recognition.onstart = () => {
            console.log("Listening...");
            document.getElementById('status').textContent = "Listening...";
         };

         recognition.onresult = (event) => {
            const transcript = event.results[0][0].transcript;
            console.log("You said: " + transcript);
            document.getElementById('lastCommand').textContent = "Last command: " + transcript;
            const userMessage = transcript.toLowerCase();

            if (userMessage.includes("hello")) {
               setTimeout(() => {
                  speak("Hii, I am glad to meet you");
               }, 1000);
            } else if (userMessage.includes("too dark")) {
               speak("turning on light");
turnOn(lamp);
            } else if (userMessage.includes("too bright")) {
               speak("turning off light");
turnOff(lamp);
            } else {
               speak("");
turnOff(lamp);
            }
         };
         recognition.onerror = (event) => {
            console.error("Error occurred in recognition: " + event.error);
            setTimeout(() => {
               try {
                  recognition.start();
               } catch(e) {
                  console.error("Failed to restart recognition:", e);
               }
            }, 1000);
         };

         recognition.onend = () => {
            if (!window.speechSynthesis.speaking) {
               setTimeout(() => {
                  try {
                     recognition.start();
                  } catch(e) {
                     console.error("Failed to restart recognition:", e);
                  }
               }, 1000);
            }
         };
      }
```

```
function speak(message) {
    document.getElementById('status').textContent = "Speaking: " + message;
    const utterance = new SpeechSynthesisUtterance(message);
    utterance.onend = () => {
        document.getElementById('status').textContent = "Listening...";
        try {
            recognition.start();
        } catch(e) {
            console.error("Failed to restart recognition:", e);
        }
    };
    window.speechSynthesis.speak(utterance);
}
window.addEventListener('load', () => {
    initializeRecognition();
    try {
        recognition.start();
    } catch(e) {
        console.error("Failed to start recognition:", e);
    }
});
```

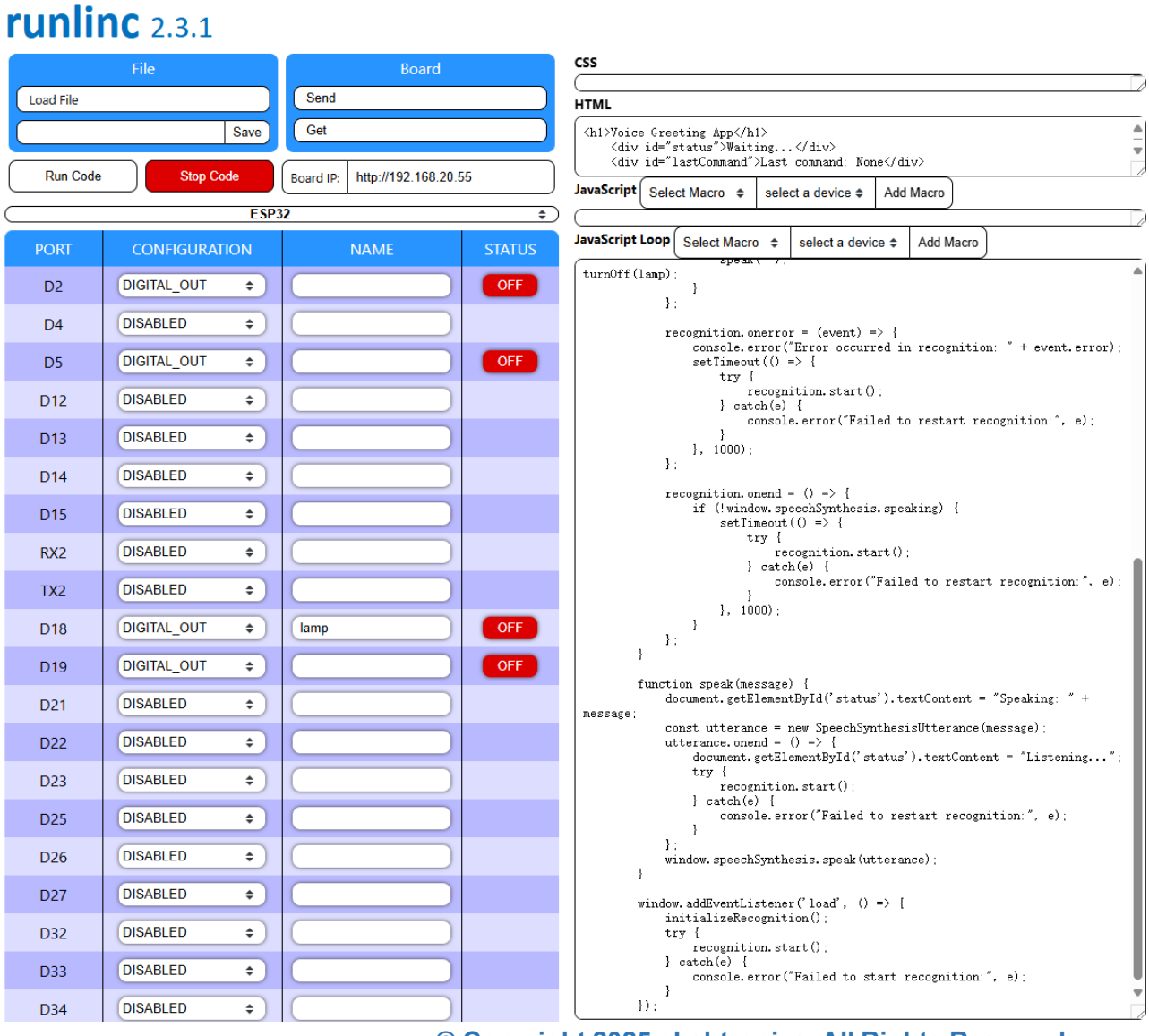

Figure 8: runlinc webpage screenshot

Now we can test the function. Press “Run Code”. If the browser asks for microphone permission, permit it.

Then when the status line shows “Listening”, you can start to say the keywords.

If you say “Hello”, the PC’s dedicated voice output will say “Hii, I am glad to meet you”;

If you say “too dark”, the PC’s dedicated voice output will say “turning on light” and turn on the LED;

If you say “too bright”, the PC’s dedicated voice output will say “turning off light” and turn off the LED.

Extension

After we tested some basic functions, we can add more functions to the chatbot. Here’s an example.

In this example, we will give the chatbot the function to react to the keyword “danger”. When it hears “danger”, it will turn a buzzer on. And when it hears “Safe”, it will turn the buzzer off. To do this, we need to work on the following steps.

1. First, we need to set the output on the control page. For port D15 set it as “DIGITAL_OUT” and name it as “buzzer”.

Figure 9: I/O configuration connection

2. We need a buzzer for this extension, the respective pins are shown below.

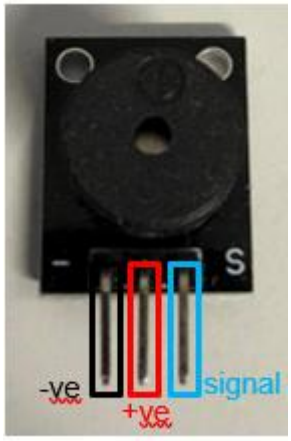

Figure 10: I/O parts with negative,positive and signal pins indicated

We need to plug the buzzer into the port io15 in this way:

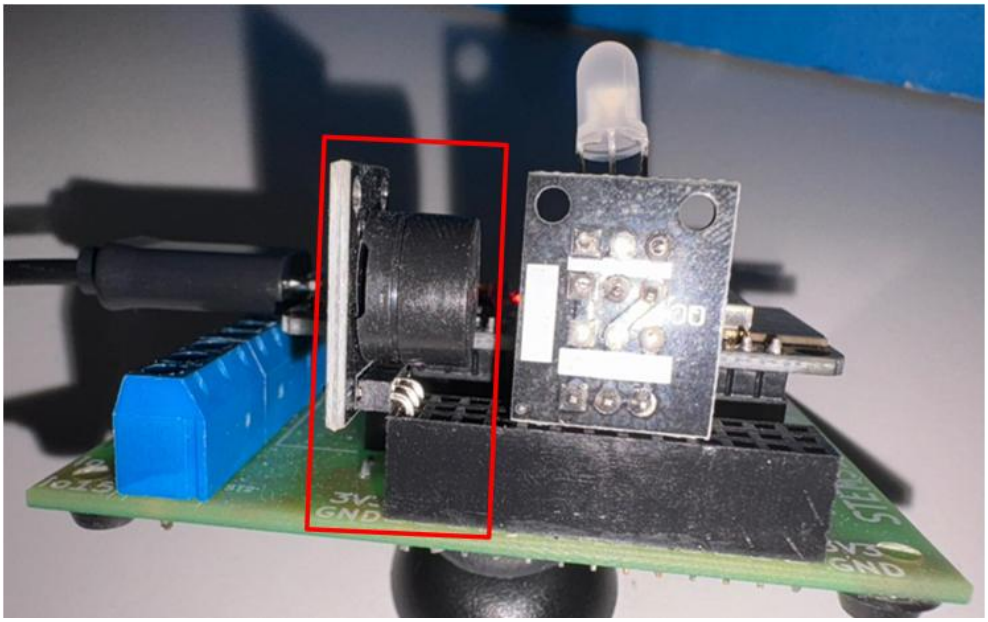

Figure 11: Circuit board connection with I/O parts (side view)

3. We need to add these lines of code to the original code section between “else if (userMessage.includes("too bright")) {speak("turning off light"); turnOff(lamp);}” and “else {speak(""); turnOff(lamp);}.

```
 else if (userMessage.includes("danger")) {
                speak("turning on alarm");
turnOn(buzzer);
            }else if (userMessage.includes("safe")) {
                speak("turning off alarm");
turnOff(buzzer);
            }
```

and we need to add this line of code after the last “turnOff(lamp);” to turn off buzzer in case.

```
turnOff(buzzer);
```

the final code in that chunk should be like this:

```
            const userMessage = transcript.toLowerCase();

            if (userMessage.includes("hello")) {
               setTimeout(() => {
                  speak("Hii, I am glad to meet you");
               }, 1000);
            } else if (userMessage.includes("too dark")) {
               speak("turning on light");
turnOn(lamp);
            } else if (userMessage.includes("too bright")) {
               speak("turning off light");
turnOff(lamp);
            } else if (userMessage.includes("danger")) {
               speak("turning on alarm");
turnOn(buzzer);
            }else if (userMessage.includes("safe")) {
               speak("turning off alarm");
turnOff(buzzer);
            }else {
               speak("");
turnOff(lamp);
turnOff(buzzer);
            }
         };
```

Now we've added the function, we can run the program and test it.

when the status line shows "Listening", you can start to say the keywords.

If you say "Hello", the PC's dedicated voice output will say "Hii, I am glad to meet you";

If you say "too dark", the PC's dedicated voice output will say "turning on light" and turn on the LED;

If you say "too bright", the PC's dedicated voice output will say "turning off light" and turn off the LED.

If you say "danger", the PC's dedicated voice output will say "turning on alarm" and turn on the buzzer;

If you say "safe", the PC's dedicated voice output will say "turning off alarm" and turn off the buzzer.

Note: If you set the response to "Hello" as "Hello and I'm glad to…" (Do you notice the "hello" in the response?) The system will keep saying "hello" to itself and keep reacting by saying "Hello" back and then creating a loop. So you should avoid this kind of response in the code for now…

Summary

People can use programming to tell AI what to do. However, sometimes those AIs can be programmed to have conversations with people, so it is important to program them correctly. In this project, we learned that we can make an AI talk to people and turn on and off devices.