EE209AS (Winter 2016) Lab Assignment 2

Prof. Ankur Mehta
Team White:
David Hong (Me): 25%: Web-Dev and Online Music Controller
Jamie Lee 25%: Web-Dev and Edison Code Refinement
Haoying Wang: 25%: Buzzer Implementation
Xiaobang Zhang: 25%: Drum Implementation

Tuesday Feb. 2, 2016

1 Introduction

In this lab we designed internet-enabled electromechanical "musical" instruments that produce music with two independent Intel Edison boards mounted on Arduino.AL.K development boards. We began the project by discussing which electromechanical "musical" instruments we would create; eventually, we decided on creating a "drum" with an Allen wrench attached to a servo and a "buzzer" to generate melodies by varying frequencies for different tones and duty cycles for different tone duration. Even though the sample code Haoying and Xiaobang modeled their instrumental code after included web-server controls, we were not able to control both Edison boards from either of the web-servers with a single HTML and JavaScript command. This issue is solved in my contribution of work, described later in the Methods section.

I designed the front-end and back-end code for the web-based user interface that controls the instruments simultaneously. I used JQuery in conjunction with HTML and JavaScript to communicate with the Edison devices within the web-page user interface. We then collaborated and pieced together material and results for Jamie to design a final web-page that fancifully displays an overview of the entire project.

2 Methods

To begin, I started with the simple HTML code generated from the Arduino web-server control samples to control instruments internally. The code is shown in Figure 1:

```
int get_flag(WiFiClient client){
  if(client){
    Serial.println("new client");          // print a message out the serial port
    String currentLine = "";                // make a String to hold incoming data from the client
    while (client.connected()) {            // loop while the client's connected
      if (client.available()) {             // if there's bytes to read from the client,
        char c = client.read();             // read a byte, then
        Serial.write(c);                    // print it out the serial monitor
        if (c == '\n') {                    // if the byte is a newline character

          // if the current line is blank, you got two newline characters in a row.
          // that's the end of the client HTTP request, so send a response:
          if (currentLine.length() == 0) {
            // HTTP headers always start with a response code (e.g. HTTP/1.1 200 OK)
            // and a content-type so the client knows what's coming, then a blank line:
            client.println("HTTP/1.1 200 OK");
            client.println("Content-type:text/html");
            client.println();

            // the content of the HTTP response follows the header:
            client.print("Click <a href=\"/Speed1\">here</a> for speed on<br>");
            client.print("Click <a href=\"/Speed2\">here</a> for speed two<br>");
            client.print("Click <a href=\"/Speed3\">here</a> for speed three<br>");
            client.print("Click <a href=\"/Speed4\">here</a> for speed four<br>");
            client.print("Click <a href=\"/Speed5\">here</a> for speed five<br>");
            client.print("Click <a href=\"/Speed6\">here</a> for speed six<br>");
            client.print("Click <a href=\"/Speed7\">here</a> for speed seven<br>");
            client.print("Click <a href=\"/Beats\">here</a> for speed beats playing<br>");
            client.print("Click <a href=\"/Play\">here</a> for song playing<br>");
            client.print("Click <a href=\"/Pause\">here</a> to pause<br>");
            client.print("Click <a href=\"/VolumUp\">here</a> to increase volumn<br>");
            client.print("Click <a href=\"/VolumDown\">here</a> to decrease volumn<br>");
            client.print("Click <a href=\"/SpeedUp\">here</a> to increase speed<br>");
            client.print("Click <a href=\"/SpeedDown\">here</a> to decrease speed<br>");
            client.print("Click <a href=\"/Stop\">here</a> to turn off <br>");
```

Figure 1: Arduino HTML that maps to Edison music program commands.

The issue with the Arduino HTML requests is that they do not allow for requests to another web address (the second Edison instrument). Therefore, as a next step, I developed a web page hosted through github.com to control both servers externally (viewable here. This started with simple HTML form action code in Figure 2:

```
<!DOCTYPE html>
<html>
    <body>
        <form action="169.232.87.55/Play" method="GET">
            <input type="Submit">
        </form>
        <form action="169.232.87.55/Stop" method="GET">
            <input type="Stop">
        </form>
        <button type="button" onclick="location.href='169.232.87.55/Play'">Click to play</button><br>
    </body>
</html>
```

Figure 2: Simple HTML for GET commands.

These simple commands created a button that was able to Play and Stop music for single Edison devices based on the inputted addresses. It worked great; however, HTML did not

allow me to send GET requests to two different addresses with the click of a single button. This was necessary since sending an HTTP GET request with the command attached to the end of {ip-address}/{command} was the only way to control the instruments from the web page and we needed to control both instruments at the same time so that the instruments would play together with a click of a single button. This resulted in researching JavaScript commands to POST or GET multiple HTTP requests with a single action. JQuery solves this issue by stringing together commands under a single action or button, shown in Figure 3.

The code demonstrates that under a single button, i.e. 'Play1', the click event would trigger two simultaneous HTTP GET requests, ultimately sending commands to control both Edison instruments.

```
<!DOCTYPE html>
<html>
<head>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.0/jquery.min.js"></script>
    <script>
        var ip1="169.232.86.124:50"; //buzzer
        var ip2="169.232.87.9:50"; //servo
        $(document).ready(function(){
            $("#Play1").click(function(){
                $.get("http://"+ip1+"/Play1");
                $.get("http://"+ip2+"/ServoBeats1");
            });
            $("#Play2").click(function(){
                $.get("http://"+ip1+"/Play2");
                $.get("http://"+ip2+"/ServoBeats2");
            });
            $("#Play3").click(function(){
                $.get("http://"+ip1+"/Play3");
                $.get("http://"+ip2+"/ServoBeats3");
            });
            $("#Pause").click(function(){
                $.get("http://"+ip1+"/Pause");
                $.get("http://"+ip2+"/Pause");
            });
            $("#SpeedUp").click(function(){
                $.get("http://"+ip1+"/SpeedUp");
                $.get("http://"+ip2+"/SpeedUp");
            });
            $("#SpeedDown").click(function(){
                $.get("http://"+ip1+"/SpeedDown");
                $.get("http://"+ip2+"/SpeedDown");
            });
            $("#VolumeUp").click(function(){
                $.get("http://"+ip1+"/VolumeUp");
            });
            $("#VolumeDown").click(function(){
                $.get("http://"+ip1+"/VolumeDown");
            });
            $("#Stop").click(function(){
                $.get("http://"+ip1+"/Stop");
                $.get("http://"+ip2+"/Stop");
            });
          });

    </script>
```

Figure 3: JQuery JavaScript dual GET commands per single button.

For each command, 'Play1', 'Play2', ... , 'Stop', the code would provide a button that GETs the commands from the two IP addresses of the Edison boards stored in variables 'var ip1' and 'var ip2.' While Haoying and Zhangbang programmed the Arduino musicality code, Jamie and I worked on a presentable front-end for this back-end code. We used Bootstrap HTML templates to design a beautiful working web page with interactions that connected with my JavaScript code, Figure 4. The actual code for button and control instantiation is in Figure 5.

```
96      <!-- Header -->
97      <header id="top" class="header">
98          <div class="text-vertical-center">
99              <h1>Musical Instruments - Team White</h1>
100             <h3>Control Center for Buzzer and Drum</h3>
101             <br>
102             <a href="#controls" class="btn btn-dark btn-lg">Start Controlling</a>
103         </div>
104     </header>
105
106     <!-- Controls -->
107     <section id="controls" class="controls">
108         <div class="container">
109             <div class="row">
110                 <div class="col-lg-12 text-center">
111                     <h2>Click on any button to send the command to the two Intel Edison boards simultaneously</h2>
112                 </div>
113             </div>
114             <!-- /.row -->
115         </div>
116         <!-- /.container -->
117     </section>
```

Figure 4: Bootstrap sample code for web page Home Sector.

```
<section id="control buttons" class="services bg-primary">
    <div class="container">
        <div class="row text-center">
            <div class="col-lg-10 col-lg-offset-1">
                <h2>Control Buttons</h2>
                <hr class="small">
                <div class="row">
                    <div class="col-md-3 col-sm-6">
                        <div class="service-item">
                            <span class="fa-stack fa-4x">
                            <i class="fa fa-circle fa-stack-2x"></i>
                            <i class="fa fa-cloud fa-stack-1x text-primary"></i>
                        </span>
                            <h4>
                                <strong>Laputa</strong>
                            </h4>
                            <p>Hayao Miyazaki</p>
                            <a id="Play1" class="btn btn-light">Play</a>
                        </div>
                    </div>
                    <div class="col-md-3 col-sm-6">
                        <div class="service-item">
                            <span class="fa-stack fa-4x">
                            <i class="fa fa-circle fa-stack-2x"></i>
                            <i class="fa fa-cloud fa-stack-1x text-primary"></i>
                        </span>
                            <h4>
                                <strong>Song of Secret Garden</strong>
                            </h4>
                            <p>Secret Garden</p>
                            <a id="Play2" class="btn btn-light">Play</a>
                        </div>
                    </div>
                    <div class="col-md-3 col-sm-6">
                        <div class="service-item">
                            <span class="fa-stack fa-4x">
                            <i class="fa fa-circle fa-stack-2x"></i>
                            <i class="fa fa-cloud fa-stack-1x text-primary"></i>
                        </span>
                            <h4>
                                <strong>Pathetique 3rd Movement</strong>
```

Figure 5: Bootstrap sample code for web page Controls Sector.

The last major roadblock of the back-end design was the actual usage of the internet browser. Since accessing two URLs behind the scenes of a browser is considered "unsafe" and "sketchy" I had to create a .bat file to allow Google Chrome web browser to authorize the JavaScript commands. To do this, I had to execute Chrome.exe under special arguments, structured in Figure 6. This allowed Chrome to crawl to the two HTTP GET requests without browser security blocks. Without executing Chrome.exe under these parameters, Chrome will not push the commands to the Edison devices. Other browsers do not have this security flag to disable or enable. Hence, our Edison devices will only work through Chrome and executing Chrome in this fashion.

```
cd \
cd Program Files (x86)
cd Google
cd Chrome
cd Application
chrome.exe --disable-web-security --allow-running-insecure-content
```

Figure 6: Windows .bat file for Chrome Security Option.

3 Results

Figure 7 is the actual web page Home sector design, resultant from Figure 4. Figure 8 is the control system for the two musical instruments resultant from Figure 5.

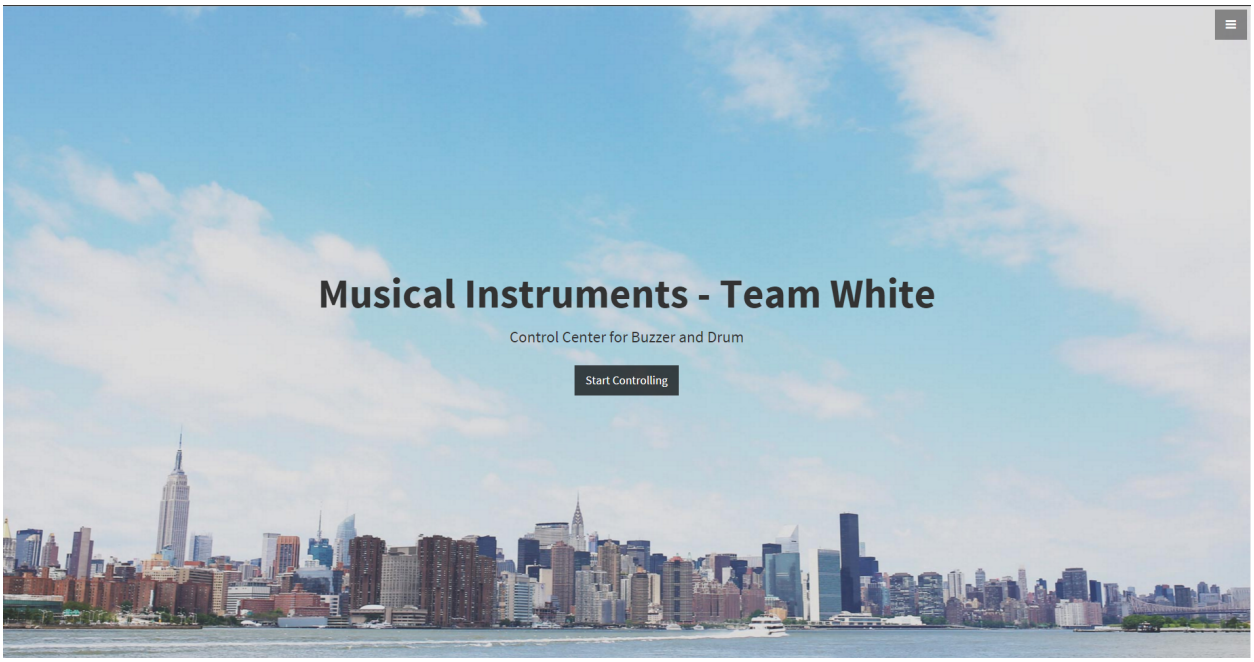

Figure 7: Bootstrap design: Home Sector.

Figure 8 is the control system for the two musical instruments resultant from Figure 5.

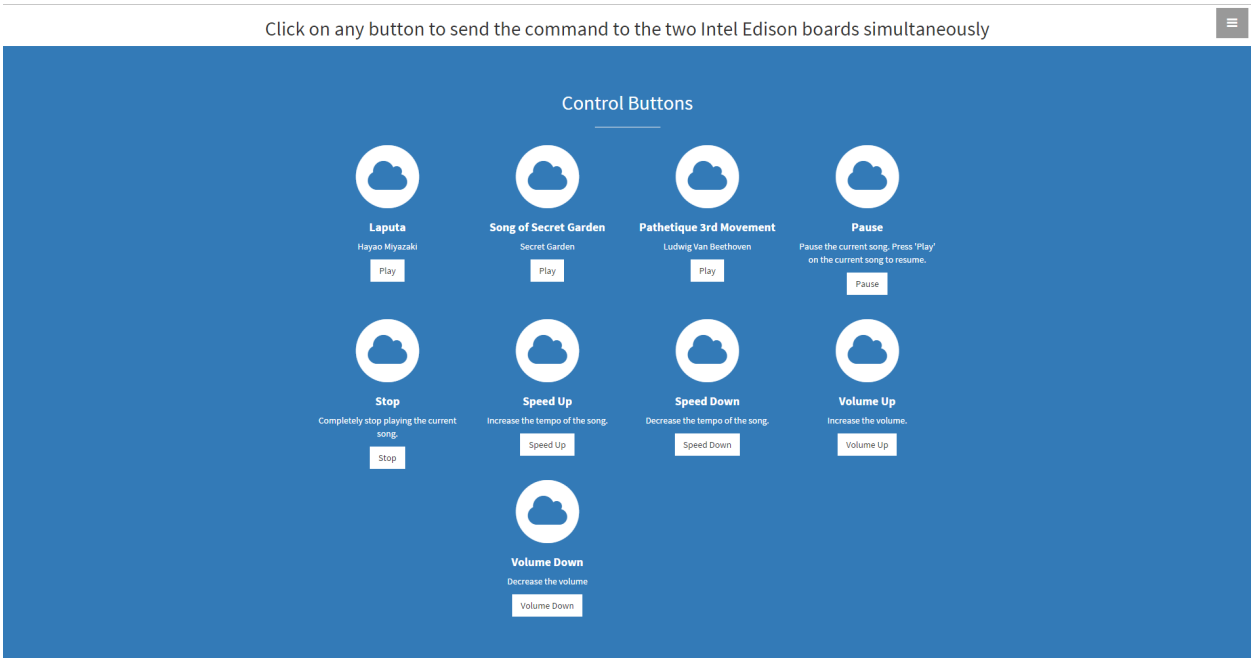

Figure 8: Bootstrap design: Control Sector.

Clicking on Play under Laputa, results in the web page sending HTTP GET requests to both addresses with commands to play song 1 (Laputa by Hayao Miyazaki) respectively per instrument, Figure 9. At the time of creating this document, both Edison boards were not connected on Eduroam WiFi network so the HTTP results in ERR_CONNECTION_TIMED_OUT. When the boards are connected to the Eduroam SSID, the instruments will play Song 1 as expected because the GET requests for {ip-address}/Play1 (for buzzer) and {ip-address}/ServoBeats1 (for drum) were transmitted but only returned with time-out since both boards were powered off. The full result can be seen in the Demo tab at: Play Demo.

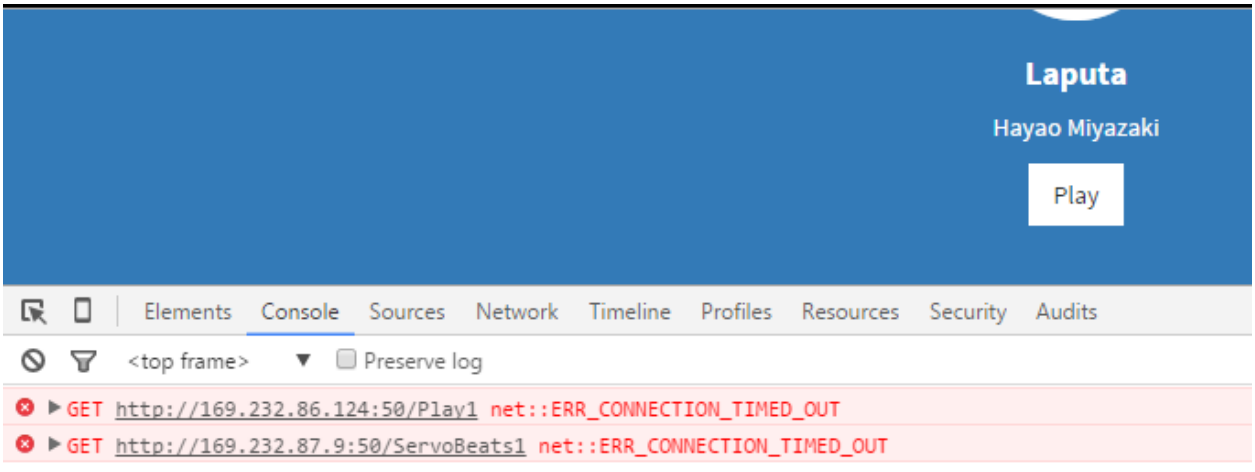

Figure 9: JQuery design: Control Command is sent.

4 Conclusion

All code for my portion of the assignment can be viewed from My Code. Overall, the experience of designing this product or system was exciting and new. This was my first experience with the Intel Edison board, and with my previous internship experience with UART devices, driver, and firmware functionality I was able to help the whole team with driver installation and COM port hardware debugging. The difficult portion of this assignment was the division of the tasks. Since the overall scheme was pretty basic, the responsibilities of each part of the project, front-end, back-end, buzzer, drum, were delegated to each individual; however, we all aided each other and worked together to develop a working project and fancy web designs. We all learned from each other and are glad the team cooperates well with one another.