Fall 2018

IOT Feeder

Jorge Felix
felixjor14@gmail.com

Jorge Felix

Dr Bareiss

Senior Project

Olivet Nazarene University

Fall 2018

IOT Feeder

A busy individual can be problematic for pet owners. Feed your pet remotely from anywhere in the world. I will talk about how to control an item using a web interface. In my case how a digital button from a web browser can interact with hardware to feed a pet in an asynchronous way by the internet. Let’s get started.

Now the first thing that you’ll need is knowledge of a few things. The first is knowledge of a procedural language in this case C. This will be used to program the microcontroller. Second, know about digital systems specifically how circuits work and how to read a datasheet, this will come and handy, so you don’t accidentally break something. The third, thing is knowledge of web technologies and techniques such as HTML, CSS (optional), and Ajax.

The equipment needed will be a microcontroller with a wifi module (see Figure 1.1), servo (see Figure 1.2), 3 wires or optional 9 wires (see Figure 1.3), a micro USB to power the microcontroller (see Figure 1.4), and breadboard (see Figure 1.5). The microcontroller I will be using is the esp8266.

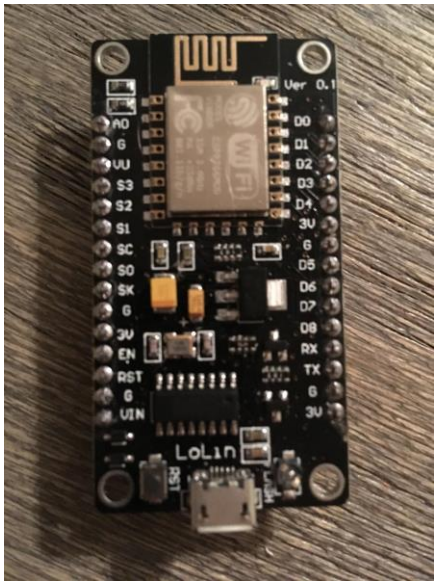

Figure 1.1

Found picture in:

https://www.flickr.com/photos/oskay/22731304279

Figure 1.2

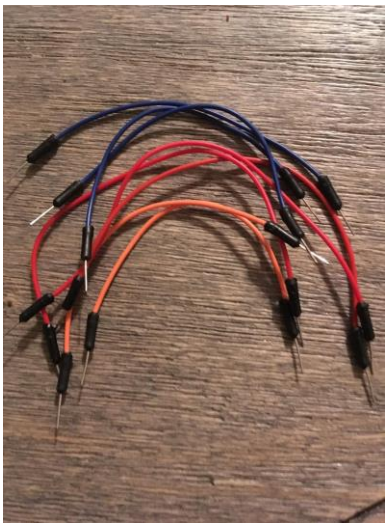

Figure 1.3

Figure 1.4

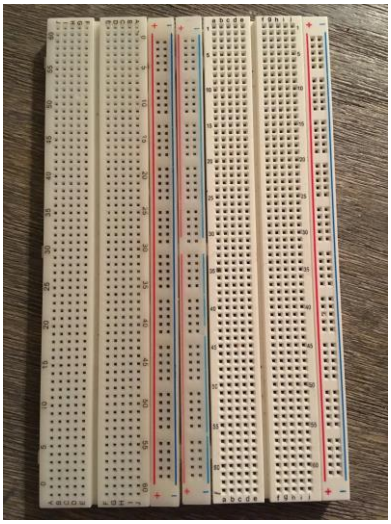

Figure 1.5

Time to program the esp8266 microcontroller, to this we must use C. There are a total of two libraries that need to be included in the code the first is for the wifi which is

<ESP8266Wifi.h> so we can have access to the wifi and the second is for the servo <Servo.h> in order to have access to the servo. In the code, you need to apply two important things just like any device, the ssid, and the password. This will allow the microcontroller to connect to the local wifi; thus, we will be able to control it wirelessly later.

Now we need to specify our objects, one for the servo and the other for wifi. Since we are making a connection to the wifi we are essentially creating a server; this we must specify it as well as the port we are going to use in this case it would port 80. The request variable will be used to read the request. Be sure to connect fill in the ssid and password in order to connect to the local wifi. (See figure 1.6)

```
#include <ESP8266WiFi.h> // wifi
#include <Servo.h> // servo

Servo myservo;
WiFiServer server(80);

const char* ssid = "";
const char* pass = "";
String request = "";
```

Figure 1.6

Next, time for the physical stuff. I am using the esp8266 be sure to look at the datasheet before plugging anything in. (See Figure 1.7)You will need to plug a few

things together to power on any electrical item. In this case it will be a servo. There are three things to get the servo to work. The first will be a positive charge, the second will be negative charge, and the third will be the data. Make sure to plug these wires correctly. It could mean the death of your item. (See Figure 1.2)

https://einstronic.com/wp-content/uploads/2017/06/NodeMCU-ESP8266-ESP-12E-Catalogue.pdf

Figure 1.7

Figure 1.8

Next back we will be using pin D4, we will need to specify it later so make sure you remember it, mind you can use any other pin. The purpose is so that we can control how the servo rotates. We want to control the rotation, so the feeder is able to move; thus, dispersing food. As for power we will use the 3v which is the pin that says 3V, for ground we will use pin GND. In our case we will use a breadboard; however, you can simply connect it directly if you have the proper female to male wires. Figure 3.1 shows how I connected it.

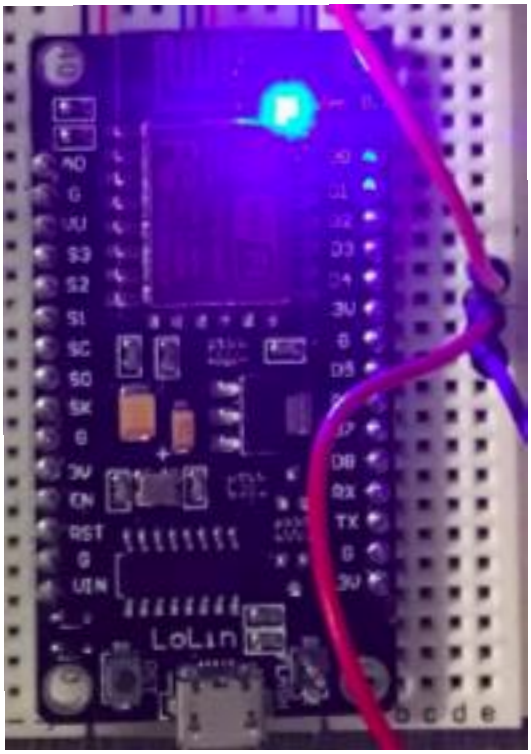

Figure 1.9

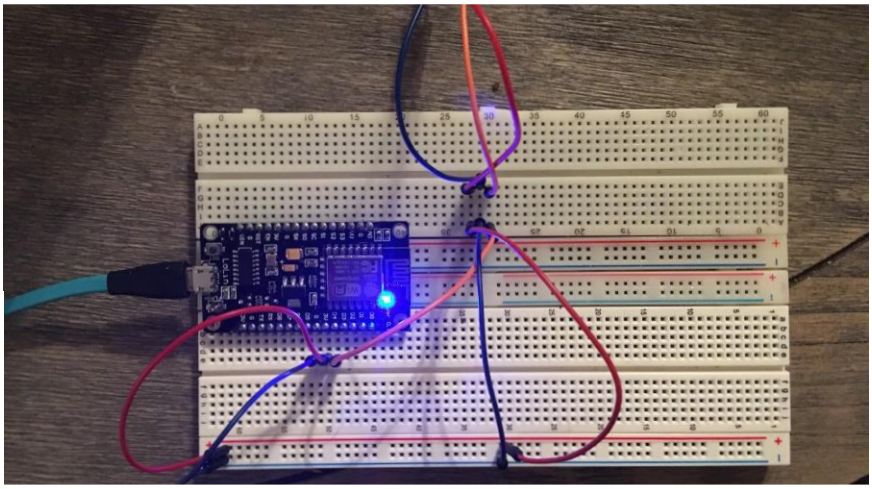

Figure 3.1

Now let's setup the wifi, first we will connect to the wifi later we will start the server. (See figure 2.0). Later we will loop thru a few things, this is important so that we can interact with the request and appropriately attach and detach the servo based on user input from request from the web. However, it is not just any simple request, that will be returned it will be a web-based request, specifically an asynchronous request also known as Ajax! (See Figure 2.1)

```
// @setup , connects to wifi and starts server
void setup(){
    // Connect local WiFi
    WiFi.begin(ssid, pass);
    // start a server
    server.begin();
}
```

Figure 2.0

```
// @loop manages requests from web browser
void loop(){
    detachServo();
    // Check wifi connection
    WiFiClient client = server.available();
    if (!client){  return; }

    // Read request
    request = client.readStringUntil('\r');

    // Take action on request
    if(request.indexOf("TURNSERVO") > 0 ){
      attachServo();
      rotate();
      client.print( "turn" );
    }else{
      client.flush();
      client.print( web );
      delay(5);
    }
}
```

Figure 2.1

Detaching and attaching the servo is a must, you don't have the servo to stay on when not in use Figure 2.2 fixes that problem. Now since I used pin D4, it will need to be included inside the method “attach” parameters of the object myservo. For detaching there is no need to include a parameter as it simply detached any pin in that method.

```
// @detachServo , Detaches Servo from Pin
void detachServo(){
      myservo.detach();
}
// @attachServo , Attaches servo on Pin D2
void attachServo(){
        myservo.attach(D4);
}
```

2.2

We are almost done with the C code portion; the last thing will be actually doing is the rotation of the servo. The way we want to servo to work is that we want it to rotate from 0 to 180 in order to disperse food. Later I want to close the disbursement in order to do that the opposite needs to be done since by default, we want the rotation to be closed. As a result, we will also need to go from 180 to 0 so that we can stop the disbursement. That is all for the C code. See figure 2.3

```
// @rotate , rotates servo
void rotate(){
    for (int pos = 0; pos <= 180; pos += 1) { // Rotates Servo 0-180 degre
ss
      myservo.write(pos);
      delay(10);
    }
    for (int pos = 180; pos >= 0; pos -= 1) { // Rotates Servo 180-0 degre
ss
      myservo.write(pos);
      delay(10);
  }
}
```

2.3

Do you remember that web variable from Figure 2.1, know we will use it! But first Be sure to create a variable that will contain all of the HTML, CSS, and JavaScript code it will go inside the () as seen in Figure 2.4. It is time for the cool stuff to begin. The actual interaction using asynchronous request from a user input using a browser, starts now! But first we need the actual interface. We will create this with html, all we need in a button I will include it inside the body of the html. (See Figure 2.5).

```
String web = R"=====()=====";
```

Figure 2.4

```
<body>
  <input type="button" id="rotateServoButton" onclick="rotateServo()" value="Rotate Servo"/>
</body>
</html>
)=====";
```

Figure 2.5

Now that we have our button and it is visible in a browser, let's make it interactive. For this we will use JavaScript. As seen in Figure 2.5 there are 3 important things that we will need. The first is an id, the second is an onclick event, and the third the value of the button. The id will allow the identification of a particular element in this case it is input tag. The onclick event will allow us to access the JavaScript function which than we can actually have an interaction, and the value is simply to result the button holds.

Let’s get into the JavaScript now, there will be only be two function now that will focus on getting the user input and the other that will make the ajax request. The function rotateServo, is in charge of paying attention when the button is clicked and making the ajax request. The second function ajaxLoad is in charge of loading the request to a new URL in the background. We don't actually see it, but it happens. This create the asynchronously and allows us to get a real time update without the need of a page refresh which is by the default a normal behavior for submission. This is why ajax is known as a developer’s dream. (See Figure 2.7) It later sends TURNSERVO to the C code’s loop and that processes it as seen in figure 2.1 and later rotates the servo.

```
<script>
var ajaxRequest = new XMLHttpRequest();

function rotateServo(){
  var button_text = document.getElementById("rotateServoButton").value;
   if(button_text=="Rotate Servo"){ ajaxLoad('TURNSERVO');}
}

function ajaxLoad(ajaxURL){
  ajaxRequest.open("GET",ajaxURL,true);
  ajaxRequest.onreadystatechange = function(){
    if(ajaxRequest.readyState == 4 && ajaxRequest.status==200){
      var ajaxResult = ajaxRequest.responseText;
    }
  }
  ajaxRequest.send();
}
</script>
```

Figure 2.7

Now time to make things look pretty, this is obviously optional. But will do it anyway as it gives interface a more user-friendly look. This is done using CSS. To do this I will access the button id this will allow me to pinpoint the exact element that I want to style. (See figure 2.8) The result can be seen in figure 2.9.

```
<style>
#rotateServoButton{
  border: none;
  color: white;
  padding: 15px 32px;
  min-width: 100%;
  text-align: center;
  font-size: 16px;
  margin: 4px 2px;
  background-color: purple;
}
</style>
```

Figure 2.8

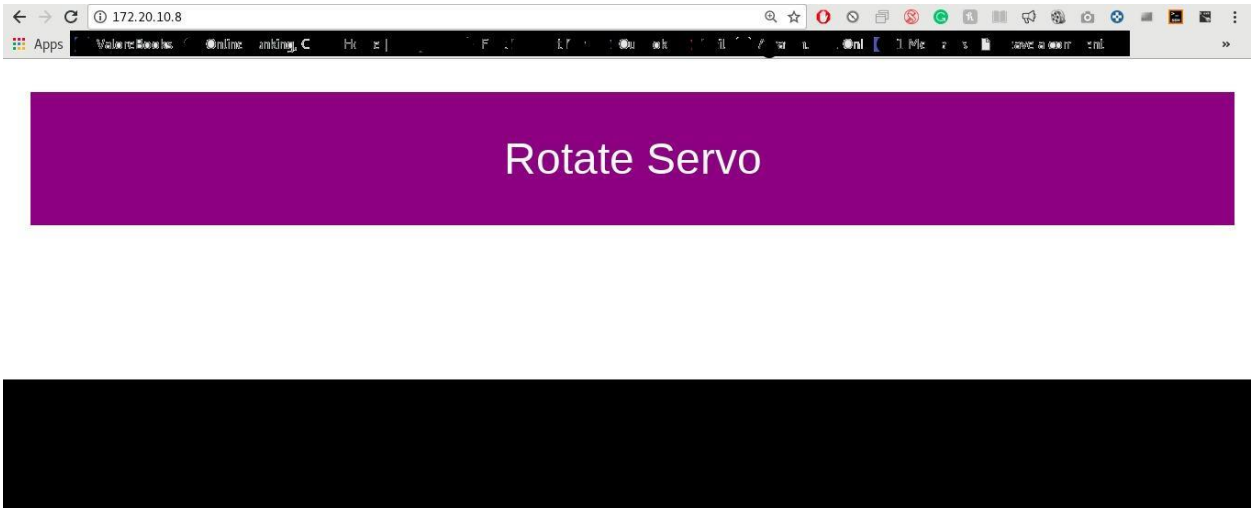

Figure 2.9

Now you may be wondering how the heck did I get the ip address. I simply used a linux command called nmap to scan my local wifi. After that it will display the ip address of the esp8266m, like any other device connected to the wifi. Sine the esp8266 also serves as a server it will be main location that the UI will be held. Essentially, it will be where you find website that the device is hosting.

Finally, that is how you are able to remotely control a feeder using the internet in a asynchronous fashion. However, to truly control it from anywhere in the world there are some other things that need to be done such as setup dns and have port forwarding setup in order to access the feeder or any device from outside of you home, but those service are dependent on the router and the type of dns chosen.

In figure 3.0 it shows the feeder that I built. I 3D printed the top portion that contains the food and used 2 tuna cans for the button half and a party cup to control the way food falls out of the feeder.

Figure 3.0

The final result of the feeder is below, (See Figure 3.1) I will also provide the code and the .lst file that I made for printing the top half portion of the feeder. It will be found in: https://github.com/felixjorge/ajaxEnabledDevice.

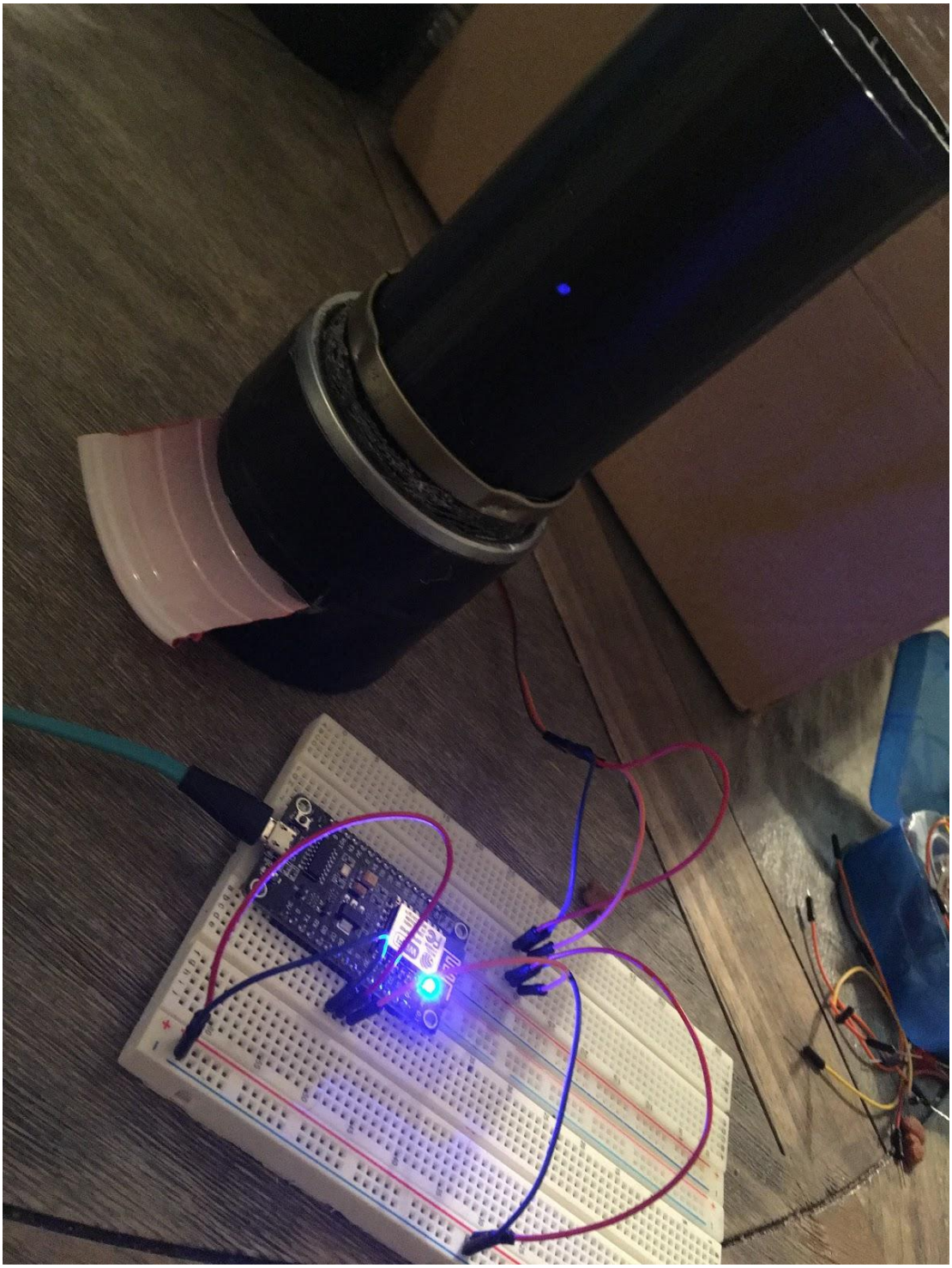

Figure 3.1