EchoMote

Design Document

Dec1720
Gary Tuttle
Sarah Huber - Team Leader
Alex Christenson - Communication Leader
Ross Reicks - Idea Holder, Web Master
dec1720@iastate.edu
http://dec1720.sd.ece.iastate.edu/

Revised: 3/09/17 Version 1.0

Table of Contents

Introduction

Project statement

EchoMote is a hardware module that the Amazon Echo can communicate with to perform basic operations on your television. These operations include turning the volume up and down, changing the channel up, down or to a specific channel, as well as turning off and on your television. When we began this project, there were no others on the market, now many of the top companies in technology have created what our senior design team was able to accomplish.

Purpose

The EchoMote is a project primarily aiming to allow visually impaired people to be able to operate their television on their own.There are handicapped people all over the world that need assistance with daily tasks--this serves as a simple solution.

Goals

We aim to create a device that anyone can use. This includes those with physical handicaps. We hope to deliver a product that is not only incredibly simple, but also physically appealing. We hope to have compatibility with all televisions. We will allow control using everyday language. It must be easier to use than a remote.

Deliverables

We will deliver a hardware module, an application to configure the remote, and a user manual. We will create a design for the packaging the remote will be marketed in.

Implementation Details

The EchoMote will look similar to the Amazon Echo Dot and can be easily placed in one's home in the same room as the television the user would like to operate. The product we are creating is

a web server that is connected to Amazon Web Services that can then run commands that operate a TV. The commands will send the IR codes for different commands on the television. We used the Raspberry Pi Zero W as our web server, and the arduino nano to send the appropriate frequency and codes for the TV commands. In order to create a long range signal on our device, we utilized a transistor to boost the signal and created a PCB board that can hold this circuit. Once we had our web server initialized via Amazon Web Services, we were able to implement the commands of a typical TV. We wanted basic functionality of a TV, but we also wanted to consider how TV would be a different user experience without a remote. Due to this, we wanted to add commands, such as, “Alexa, change to ESPN.” Without a remote, channel numbers become obsolete. In order to allow the user to be able to create their channels for their televisions, we created an app that the user could use to enter this information. The app that we created is also used to initialize the product to a user's amazon alexa upon purchase.

System Specifications

Non-functional

The project will also consist of many preliminary tasks that will not be seen by the user.

Connect To Wifi - The device needs to be able to configured to the user’s current in house wifi network so that alexa can communicate with it. The only interface that the amazon echo currently supports is wifi. We created an app that allows the user to configure the EchoMote to their wifi and amazon echo upon their first use.

Creating a compact device - We want our device to be able to sit on someone’s coffee table. We want it be circular with about the same diameter of the current Amazon Echo. The EchoMote is the same size as the Amazon Echo Dot and has 4 equally spaced IR Transmitters. The casing has been 3D printed.

Infrared power - In the EE 230 lab we build a remote control and receiver but we had troubles with distance and line of site. We don’t want this to be an issue. We have added a transistor and created a simple circuit to boost the output of the IR transmitters.

Power - This device will need power. We decided to have the EchoMote plug into the wall such that a user would not need to replace batteries.

Heat - We ensured that even though we are using a lot of hardware devices in a small space that we did not have too much heat for a user.

Functional

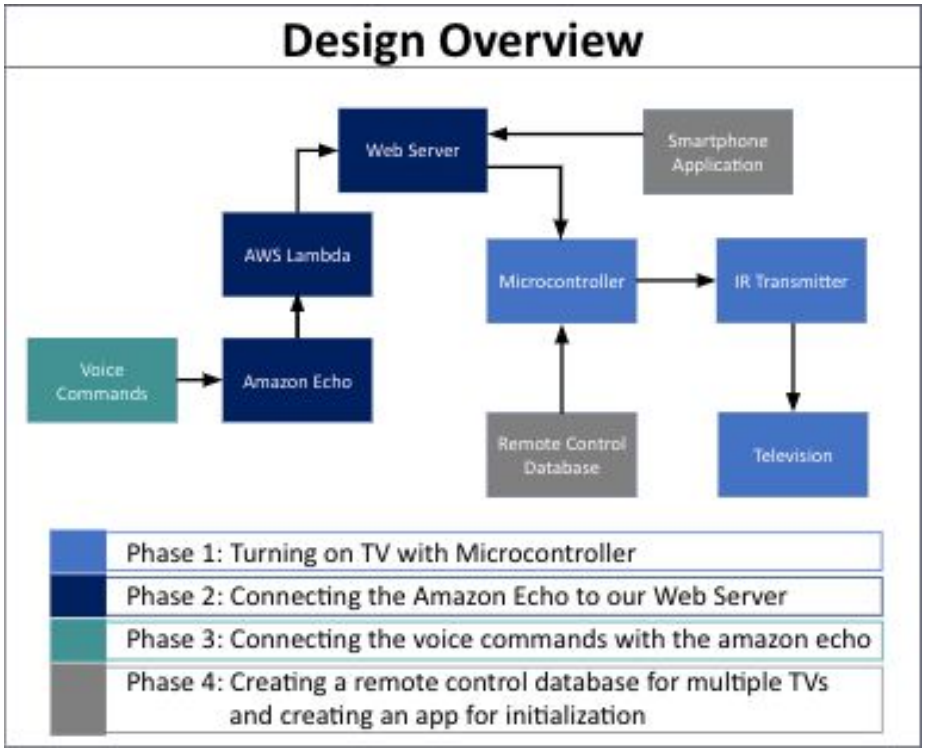
In order to implement the following commands, we will need to utilize amazon's open source software and apply our commands that we would like to the code. The software will then have to communicate with the hardware to tell the TV what to do.

The amazon echo will initially be able to do all basic remote functions via a voice command through the amazon echo.

Functions:

- Power On/Off
- Input
- Channel Up/Down
- Change to channel X
- Volume Up/Down
- Volume Up/Down Increment
- Mute

Proposed Design Method



Design Analysis

The proposed design method above allowed for an efficient and effective way to implement our project. In order to implement our proposed design, we worked backwards. Our first stage is to turn on the TV using an IR transmitter. Next, we used a Raspberry Pi Zero W as a web server and connected that to Amazon Web Services to save our code with the commands. Once this was working, we created an app for initialization and a remote control database.

Development/Testing

INTERFACE specifications

For this project, the interface will be via voice commands. There will be a list of common commands that we will implement. The following initial codes are:

1. “Tell TV to turn On/Off”
2. “Tell TV to change channel”

3. “Tell TV to turn volume up/down”
4. “Tell TV to turn volume up/down X amount”
5. “Tell TV to tune through channels”
    a. “Tell TV to stop tuning”
6. “Tell TV to change input”

Future inputs could be:

1. “Tell TV to turn on Apple TV”
2. “Tell TV to open Netflix/Amazon/Hulu/etc”
3. Be able to change TV settings

Hardware/software

Testing certain aspects of the EchoMote components is easily done with any commercial television and the Amazon Echo. This makes the testing relatively easy to do since there are few instruments needed in order to produce a working prototype.

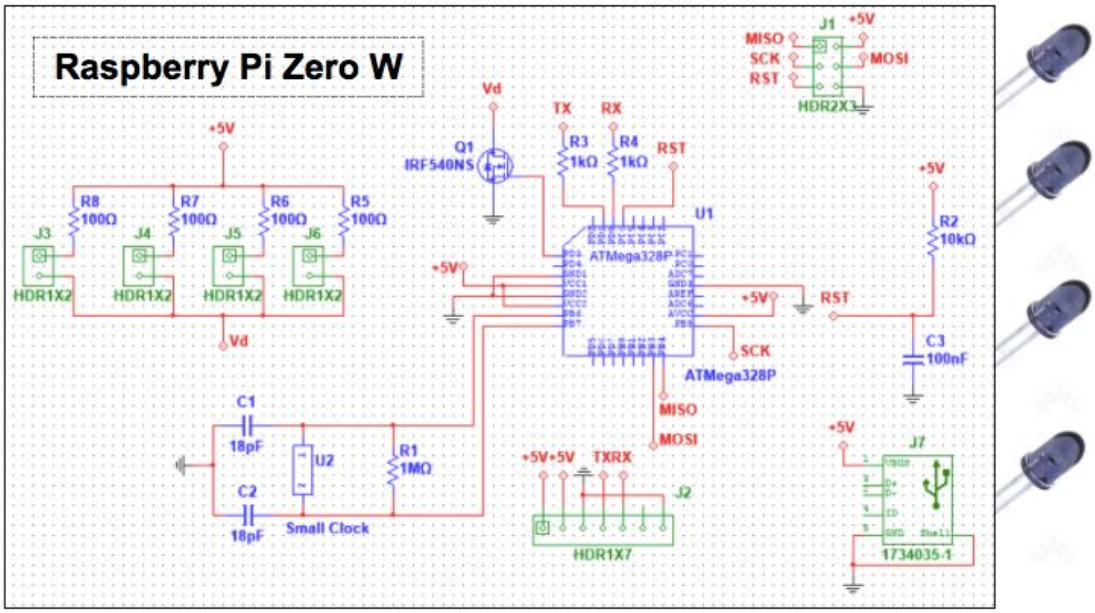
TV:

The television we use to test is a standard Samsung commercial TV. It only tests the IR protocol specific to the Samsung brand, but since all other IR transmission is very similar, testing with other models and brands is unnecessary in the prototype phase.

Digital Multimeter:

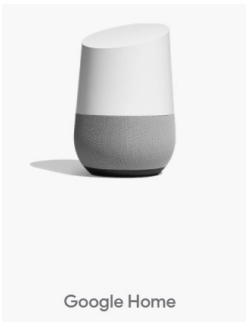
Since it is important to be able to transmit over a relatively long range, we need to have the largest possible current output to the LED transmitter. The DMM is useful to test when attempting to achieve the output on the datasheet.

Below is the circuit used to increase the distance of the LED IR Transmitters and also how it connects with the entire project. This circuit utilizes a mosfet to increase the current going through the IR LEDs and then from the microcontroller, the LEDs are powered.



Related Products and Literature

When we began this project in the spring of 2017, we were excited to take this project to market. As the senior design term progressed, many companies have released items that can do what ours does. The Amazon Fire TV Stick and the Google Home are two examples, and main players in this market. We are excited to say that we were able to create a product that many of the top tech companies have created, however, we were sad to not be able to compete in this market with our product.



Conclusions

In conclusion, this device will be able to control a television fully from the Amazon Echo. It will act as other products that already interact with the Echo via wifi. This product will be used by all

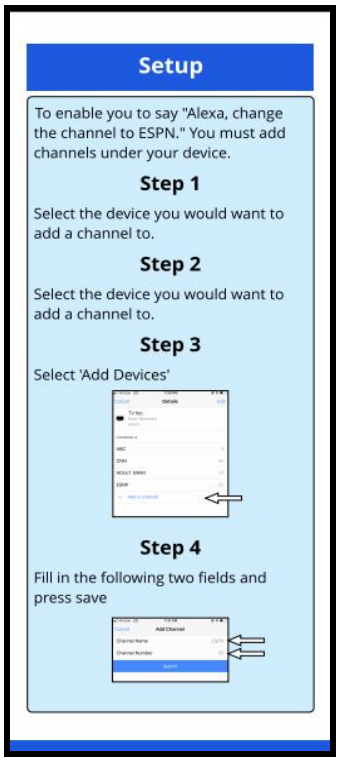
people but will open the door for people with seeing disabilities to be able to control their television. The final product looks like something that would be purchased at a store, the product will not only be functional, but physically appealing. Finally this project taught us how to create a piece of hardware that interacts with wifi, how to utilize open source software, and how to plan, design, and create a product.

References

AnyMOTE- https://www.anymote.io/tutorials/tutorial-how-to-control-your-tv-with-amazon-echo
Codes- https://sourceforge.net/p/lirc-remotes/code/ci/master/tree/remotes/
Protocols- http://www.techdesign.be/projects/011/011_waves.htm
IR protocol explained- https://rusticengineering.com/2011/02/09/infrared-room-control-with-samsung-ir-protocol/

Appendix I -- Operational Manual

The following brochure is an operational manual for our users.



Appendix II -- Code

Below all the raw code that was used for are project is listed. It is seperated by file and location of the file.

AWS Lamba functions:

```
const https = require('https');
const Client = require('node-rest-client').Client;
const client = new Client();

const hostname = "ec2-34-229-46-76.compute-1.amazonaws.com";

function handler(request, context) {
   if (request.directive.header.namespace === Namespaces.Discovery && request.directive.header.name
=== Names.Discover) {
      handleDiscovery(request, context);
   }
   else if (request.directive.header.namespace === Namespaces.Volume) {
      if (request.directive.header.name === Names.AdjustVolume) {
         handleVolume(request, context);
      }
   }
   else if (request.directive.header.namespace === Namespaces.Power) {
      handlePower(request, context);
   }
   else if(request.directive.header.namespace === Namespaces.Channel) {
      handleChannel(request, context);
   }
}
exports.handler = handler;

function handleDiscovery(request, context) {
   var response = {
            event: {
              header: {
                 namespace: "Alexa.Discovery",
                 name: "Discover.Response",
                 payloadVersion: "3",
                 messageId: "5f8a426e-01e4-4cc9-8b79-65f8bd0fd8a4"
```

```
            },
            payload:{}
          }
        };

    client.get('http://'+ hostname +':5000/get-devices', (data, res) => {

        response.event.payload.endpoints = data;
        request.directive.header.name = Names.DiscoverResponse;
        context.succeed(response);
    });
}

function handleVolume(request, context) {
    // get device ID passed in during discovery
    var endpointId = request.directive.endpoint.endpointId;
    // get user token pass in request
    var token = request.directive.endpoint.scope.token;
    // get the volume set
    var volume = request.directive.payload.volume;
    var args = {
        data: { deviceId: endpointId, direction: volume > 0 ? "UP" : "DOWN" },
        headers: { "Content-Type": "application/json" }
    };
    client.post('http://'+ hostname +':5000/volume', args, function (data, response) {
        var header = request.directive.header;
        header.name = Names.Response;
        header.namespace = Namespaces.Response;
        var response = {
            context: {
                properties: [{
                        namespace: Namespaces.Volume,
                        name: Names.PowerState,
                        value: volume,
                        timeOfSample: new Date(),
                        uncertaintyInMilliseconds: 500
                    },
                    {
                        namespace: Namespaces.Volume,
                        name: Names.Muted,
                        value: false,
                        timeOfSample: new Date(),
                        uncertaintyInMilliseconds: 500
```

```
            }]
        },
        event: {
            header: header,
            payload: {
                volume: volume,
                muted: false
            }
        }
    };
    context.succeed(response);
});
}
;
function handlePower(request, context) {
    // get device ID passed in during discovery
    var endpointId = request.directive.endpoint.endpointId;
    // get user token pass in request
    var token = request.directive.endpoint.scope.token;
    // at this point you should make the call to your device cloud for control
    // set content-type header and data as json in args parameter
    var args = {
        data: { deviceId: endpointId },
        headers: { "Content-Type": "application/json" }
    };
    client.post('http://' + hostname + ':5000/power', args, function (data, response) {
        var header = request.directive.header;
        header.name = Names.Response;
        header.namespace = Namespaces.Response;
        var response = {
            context: {
                properties: [{
                        namespace: Namespaces.Power,
                        name: Names.PowerState,
                        value: "OFF",
                        timeOfSample: new Date(),
                        uncertaintyInMilliseconds: 500
                    }]
            },
            event: {
                header: header,
                payload: {}
            }
```

```
        };
        context.succeed(response);
    });
}

function handleChannel(request, context) {
    var endpointId = request.directive.endpoint.endpointId;
    var channel = request.directive.payload.channel.number;
    if(!channel) {
        channel = request.directive.payload.channelMetadata.name;
    }

    var args = {
        data: { deviceId: endpointId, channel: channel },
        headers: { "Content-Type": "application/json" }
    };

    client.post('http://' + hostname + ':5000/channel', args, function (data, response) {
        var header = request.directive.header;
        header.name = Names.Response;
        header.namespace = Namespaces.Response;
        var response = {
            context: {
                properties: [{
                        namespace: Namespaces.Channel,
                        name: Names.ChannelResponse,
                        value: request.directive.payload.channel,
                        timeOfSample: new Date(),
                        uncertaintyInMilliseconds: 500
                    }]
            },
            event: {
                header: header,
                payload: {}
            }
        };
        context.succeed(response);
    });
}

var Namespaces;
(function (Namespaces) {
    Namespaces["Volume"] = "Alexa.Speaker";
```

```
    Namespaces["Power"] = "Alexa.PowerController";
    Namespaces["Channel"] = "Alexa.ChannelController";
    Namespaces["Input"] = "Alexa.InputController";
    Namespaces["Response"] = "Alexa";
    Namespaces["Discovery"] = "Alexa.Discovery";
})(Namespaces = exports.Namespaces || (exports.Namespaces = {}));
var Names;
(function (Names) {
    Names["SelectInput"] = "SelectInput";
    Names["Response"] = "Response";
    Names["DiscoverResponse"] = "Discover.Response";
    Names["ChannelResponse"] = "channel"
    Names["ChangeChannel"] = "ChangeChannel";
    Names["Discover"] = "Discover";
    Names["AdjustVolume"] = "AdjustVolume";
    Names["TurnOn"] = "TurnOn";
    Names["TurnOff"] = "TurnOff";
    Names["PowerState"] = "powerState";
    Names["Muted"] = "muted";
})(Names = exports.Names || (exports.Names = {}));
```

Web Server Code:

```
const express = require('express')
var cors = require('cors')

const app = express()
app.use(cors())
const http = require('http');
const url = require('url');
const WebSocket = require('ws');
var mysql = require('mysql');
var bodyParser = require('body-parser');
var expressSanitized = require('express-sanitize-escape');

var con = mysql.createConnection({
        host: "localhost",
        user: "root",
        password: "",
        database: "echomote"
  });
```

```
 con.connect(function(err) {
        if (err) throw err;
        console.log("Connected!");
 });

// parse application/x-www-form-urlencoded
app.use(bodyParser.urlencoded({ extended: false }))

// parse application/json
app.use(bodyParser.json());
app.use(expressSanitized.middleware());

// start the server
app.listen(5000, function() {
        console.log('Listening on port 5000');
});

const wss = new WebSocket.Server({ port: 3000});

function heartbeat() {
        this.isAlive = true;
}

wss.on('connection', function(ws, req) {
        const location = String(url.parse(req.url, true).query['id']);
        if(location != null) {
                let query = "SELECT * FROM pis where piID = '" + location + "'";
                con.query(query, (err, result) => {
                        if (err) throw err;
                        if(result.length === 0) {
                                con.query("INSERT INTO pis(piID, tvId) values ('" + location + "', 1)",
(err, result) => {
                                        if(err) throw err;
                                        console.log(result);
                                });
                        } else {
                                console.log("found " + result.length  + " devices")
                        }
                })
                ws.location = location;
        }
        ws.isAlive = true;
        ws.on('pong', heartbeat);
```

```
});

const interval = setInterval(function ping() {
  wss.clients.forEach(function each(ws) {
    if (ws.isAlive === false) return ws.terminate();

    ws.isAlive = false;
    ws.ping('', false, true);
  });
}, 30000);

// base route to navigate to see if it works
app.get('/api/pis', function (req, res) {
        con.query('SELECT * FROM pis', (err, result) => {
                if(err) throw err;
                result.forEach(res => {
                        wss.clients.forEach(ws => {
                                if(ws.location == res.piID) {
                                        res.active = true;
                                }
                        })
                        Object.keys(res).forEach(key => {
                                if(typeof res[key] === "string") {
                                        res[key] = res[key].replace('&#39;', "'")
                                        console.log(res[key]);
                                }
                        })
                })
                res.send(result);
        });
})

app.get('/', (req, res) => {
        res.send('<h1>Echomote</h1>')
})

app.post('/api/channel', function (req, res) {
        let piId = req.body.piId;
        let channelId = req.body.channelId;
        let channelName = req.body.channelName;

        con.query(`INSERT INTO channels(channelId, channelName, piId) values ( ${channelId},
'${channelName}', '${piId}' )`, (err, channel) => {
```

```
                if(err) throw err;
                res.send('{ Message: Success }')
        });
});

app.get('/api/channel', (req, res) => {
        con.query('SELECT * FROM channels', (err, result) => {
                if(err) throw err;
                res.send(result);
        });
});

app.get('/api/tvs', (req, res) => {
        con.query('SELECT * FROM tvs', (err, result) => {
                if(err) throw err;
                res.send(result);
        });
});

app.put('/api/pis', (req, res) => {
        let piId = req.body.piID;
        let tvId = req.body.tvId;
        let location = req.body.location;
        let friendlyName = req.body.FriendlyName
        con.query(`UPDATE pis SET tvId = ${tvId}, location = '${location}', FriendlyName =
'${friendlyName}' WHERE piID = '${piId}'`, (err, result) => {
                if(err) throw err;
                res.send(result);
        });
});

// power function, req has echo id
app.post('/power', function(req, res) {
        console.log('POWER');
        // connect to database, grab all pis associated with this echo
        // Send a power signal to the pi who has this name
        let query = "SELECT * FROM pis WHERE piID = '" + req.body.deviceId + "'";
        con.query(query, (err, result) => {
                if(err) throw err;
                if(result.length !== 0) {
                        wss.clients.forEach(ws => {
                                if(ws.location == req.body.deviceId) {
                                        ws.send('POWER');
```

```
                        }
                    })
                } else {
                    console.log('no devices found')
                }
        })
        res.send('success');
})

app.post('/volume', function(req, res) {
        let query = "SELECT * FROM pis WHERE piID = '" + req.body.deviceId + "'";
    con.query(query, (err, result) => {
        if(err) throw err;
                    if(result.length !== 0) {
                        wss.clients.forEach(ws => {
                            if(ws.location == req.body.deviceId) {
                                if(req.body.direction == 'UP') {
                                    ws.send('VOLUMEUP');
                                    console.log('VOLUMEUP');
                                } else if(req.body.direction == 'DOWN') {
                                    ws.send('VOLUMEDOWN');
                                    console.log("VOLUMEDOWN");
                                }
                            }
                        })
                    } else {
                        console.log('no devices found')
                    }
    })
        res.send('success');
})

app.post('/channel', function(req, res) {
        let query = "SELECT * FROM pis WHERE piID = '" + req.body.deviceId + "'";
    con.query(query, (err, result) => {
        if(err) throw err;
                    if(result.length !== 0) {
                        // we have all the devices in the database
                        if(isNaN(req.body.channel)){
                            con.query("SELECT channelId FROM channels WHERE
channelName LIKE '%" + req.body.channel + "%' AND piId = '" + req.body.deviceId + "'", (err,
channels) => {
                                if(channels.length === 0) {
```

```
                                        return;
                                    }
                                    wss.clients.forEach(ws => {
                                        if(ws.location == req.body.deviceId) {
                                            ws.send(channels[0].channelId);
                                        }
                                    })
                                })
                            } else {
                                wss.clients.forEach(ws => {
                                    if(ws.location == req.body.deviceId) {
                                        ws.send(req.body.channel);
                                    }
                                })
                            }
                        } else {
                            console.log('no devices found')
                        }
        })
            res.send('success');
})

/*
* Used to register a new echo device and pair it with a corresponding pi
* should only need a echo Id, will do a lookup on IP address and try to match it with an pi
*/
app.post('/register', (req, res) =>{

});



// will return a list of devices connected to this IP
app.get('/get-devices', (req, res) => {
        con.query('SELECT piID as piID FROM pis', (err, result) => {
            if(err) {
                res.statusCode = 500;
                res.send('An error occured grabbing the devices');
            }
            var responseObject = [];
            for(var i = 0; i < result.length; i++) {
                responseObject.push({
                    manufacturerName: "echomote",
```

```
displayCategories: ['TV'],
endpointId: result[i].piID,
friendlyName: "TV" + i,
description: "A TV controlled by my echo",
applianceId: "appliance-001",
displayCategories: [
 "TV"
],
capabilities: [
        {
                type: "AlexaInterface",
                interface: "Alexa.PowerController",
                version: "3",
                properties: {
                 supported: [
                        {
                         name: "powerState"
                        }
                 ],
                 proactivelyReported: true,
                 retrievable: true
                }
        },
        {
                type:"AlexaInterface",
                interface:"Alexa.Speaker",
                version:"3",
                properties:{
                  supported:[
                        {
                                name:"volume"
                        },
                        {
                                name:"muted"
                        }
                  ]
                }
        },
        {
                interface: "Alexa.ChannelController",
                type: "AlexaInterface",
                version: "3",
                properties: {
```

```
                                                supported:[
                                                        {
                                                                name: "channel"
                                                        }
                                                ]
                                        }
                                },
        ]
                        });
                }
                res.send(responseObject);
        })
});
```

Echomote device code

```
const WebSocket = require('ws');
const { exec } = require('child_process');
const SerialPort = require('serialport');

var port = new SerialPort('/dev/ttyAMA0', {
    baudRate: 9600
});

exec("cat /proc/cpuinfo | grep Serial | cut -d ' ' -f 2", (err, stdout, stderr) => {
  if(err) {
    console.error(err);
    return;
  }
  let id = String(stdout.replace(/(\r\n|\n|\r)/gm,""));
  const ws = new WebSocket('ws://ec2-34-229-46-76.compute-1.amazonaws.com:3000?id='+id);

  ws.on('message', (message) => {
    port.write(message, function(err) {
        if(err) {
            return console.log('error: ', err.message);
        }
    });
  });
});
```

Arduino IR Code