ILR 1: Sensor and Motors Lab

Angad Sidhu
Team B: Arcus
Logan Wan, Clare Cui, Maitreya Naik
October 14th 2016

Individual Progress

For the sensors and motors lab I was in charge of the GUI and the final software integration of all the disparate components.

GUI

I decided to implement the GUI as a Web App so that a user could interact with the system with any device with a modern web browser. The stack consists of a front end web page, a middleware server, and a microcontroller as the hardware interface. I wrote the front end web page and its associated scripts in plain HTML and Javascript. For the middle-ware I used python to bridge communication with the Web Page and the Microcontroller. The Microcontroller, an Arduino Uno, acquires sensor data and issues motor commands as well as transfers this information up the stack.

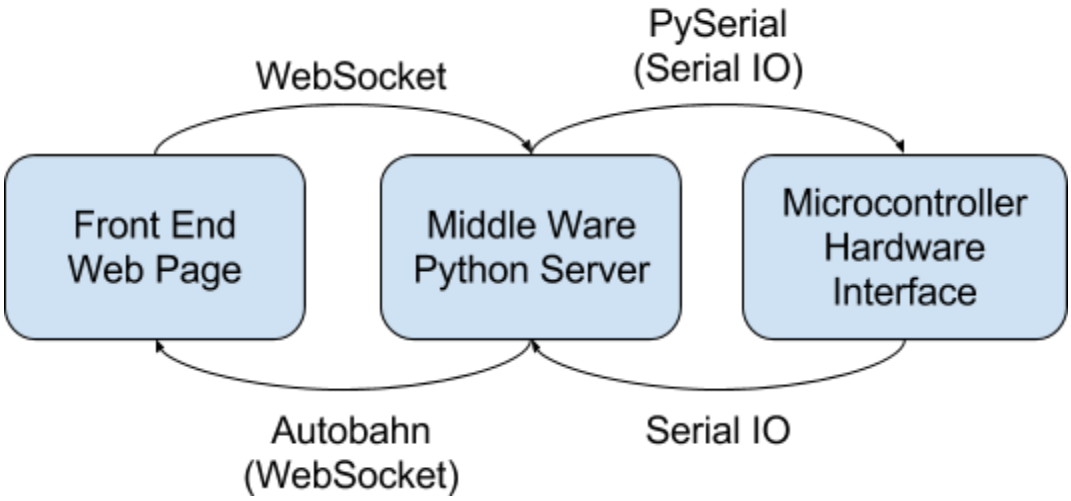

Figure 1. Shows the full stack of software and how data is passed between layers.

The front end web page allows a user to view the outputs and states of each of the sensors and motors. It depicts sensor and motor values in a dynamically updating graph using the Plotly.js library. It allows a user to control which state the microcontroller was in. The state dictates which sensor and motor is enabled. It also allows a user to manually control the settings of the stepper and servo motor. I also made use of the JQuery library in order to reduce the code complexity involved in manipulating DOM elements. The server and the web app communicated with our own protocol layered on top of a WebSocket (RFC-6455) communication channel. The elements used for accepting input are very basic HTML elements like input, and select.

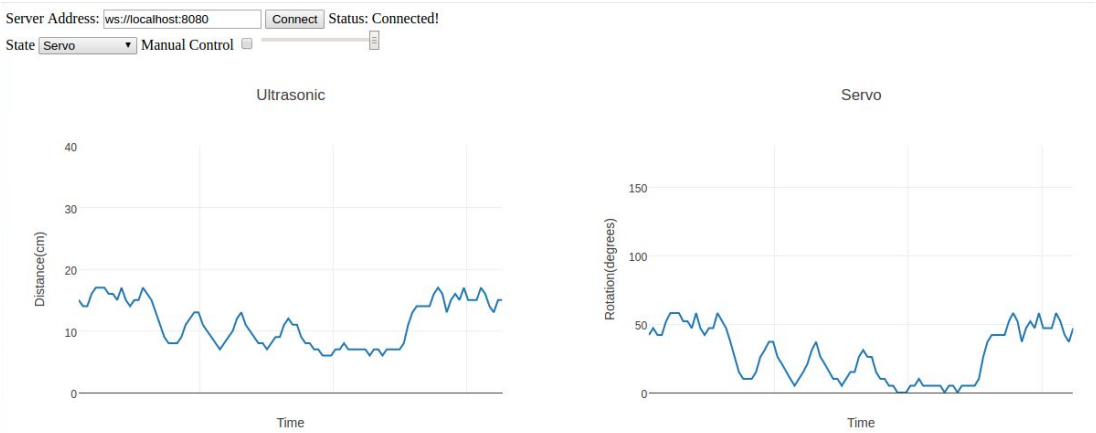

Figure 2. This image depicts the layout of the Web App GUI.

The python server was a very simple websocket-serial data exchanger. It connected to a specified serial port at a specified baudrate and hosted a websocket server on a specified port. All of these parameters were specifiable by command line. The server simply forwarded serial data to the websocket clients and any websocket data to the serial port. Python was chosen in order to maintain cross-platform compatibility. I made use of the autobahn library to enable communication over websocket and the pySerial library to communicate over a serial port. The web page itself is not served using this program; the web page can be statically hosted using any type of web server.

The protocol used for communication between the web server and the micro controller consists of a total of 7 messages. The microcontroller only emits 2 types while the web app emits 3 types. The protocol is described in the table below.

Message	Sender	Format	Description
write	Web App	cmd,value[0-100]	Sets the state of the motor to a percentage between 0 - 100
set-manual	Web App	cmd,enabledl[1,0]	Tells microcontroller whether motor should be in manual mode or tracking the sensor
set-state	Web App	cmd,state[0-3]	Change the state of the micro controller causing.
status	Arduino	Cmd, state[0-3],manual[0,1],minlimit[int],maxlimit[int],manualValue[int]	Sends the status of the arduino. Things like state, manual control values, and limits on the current sensor are sent.
read	Arduino	Cmd,id[string],valu	A packet containing the values read from

		e[int/float],timestamp[long]	the sensor or motor.

Figure 3. A table that describes the communication protocol between arduino and web app.

Software Integration

I was in charge of integrating the various parts contributed by my teammates. This was fairly simple and really only involved cleaning the code, switching pins for the final setup, and minor refactoring to enable the setting of manual values. Cleaning the code involved removing any serial prints which could ruin communication and removing delays that were not absolutely necessary. I refactored most of the code given to me into separate cpp and h files so as to avoid any global variable name clashes. Most of the work involved in this portion went into solving the challenges that I will discuss below.

Challenges

One issue that was difficult to circumvent was debugging the Arduino code. Since the protocol monopolizes the serial connection it can be challenging to debug issues with print statements. It is also very hard to debug issues that cause strange bugs in serial communication. A good solution would have been using a hardware debugger and Atmel Studio to actually debug the code and inspect the state of microcontroller. I unfortunately didn't have access to a hardware debugger so I mainly had to rely on inserting print statements and changing code in various places and checking the output.

Another very large issue that cropped up was the limited SRAM memory available on the Arduino Uno. I encountered a few symptoms on the serial line like strangely garbled strings that prompted me to investigate further. What really led me to my diagnosis was commenting out large swathes of code in an effort to discover the problematic lines of code. I eventually got it down to the point where commenting/uncommenting a line of code that didn't even execute would cause the issue to appear. After a bit more research I discovered that this kind of unpredictable behavior can be caused by the low memory limit (2KB) on the Uno. I was able to solve it by making two changes. First I reduced the size of my arbitrarily large read and write buffers. They had been set to 512 bytes each which together took up 1 KB. The second change I made was reducing the verbosity of the protocol command strings I used as those also contributed somewhat to memory size. An alternative to reducing the string size could also have been adding the PROGMEM define for static strings which would have indicated to the compiler to store those strings in program memory rather than SRAM. Yet another solution would have been to use byte packets to send data rather than a string representation as it would have reduced throughput of the stream and memory and cpu complexity of encoding the packets.

Teamwork

Our team divided the work pretty evenly; each member was assigned either a sensor and motor, or the GUI. Clare worked on setting up the IR sensor and Stepper motor. Maitreya worked on the DC motor and the potentiometer. Logan worked on the ultra sonic sensor and servo motor. In addition Maitreya and Clare teamed up to work on electrical integration.

Current Project Progress and Future Plans

We have been working hard to accomplish a few key tasks before the weather becomes prohibitive to testing. We went to La Farge Quarry and setup a rig in the back of a car to collect LiDar and GPS data as drove around. This is essential data we can use when testing our localization and mapping algorithms. We also have calibrated our flight controller and performed preliminary flight tests with our Hex rotor. We are looking to fly it outdoors before the weather becomes an issue.

Our future plans include an initial design for our power distribution board as it will be necessary for final systems integration. We are looking to finalize the mechanical design of the UAV. I will be aiding in efforts to help finalize the mechanical design. I will be responsible for installing software packages onto the onboard drone computer and testing hardware integration with it. This is an important systems integration task that is necessary to complete before being able to fly and collect data. I will be doing research on potential localization and mapping algorithms. This is a critical part in our project and will involve considering many different factors like resistance to motion effects, cpu complexity, and accuracy.

Appendix A. Transfer Plot

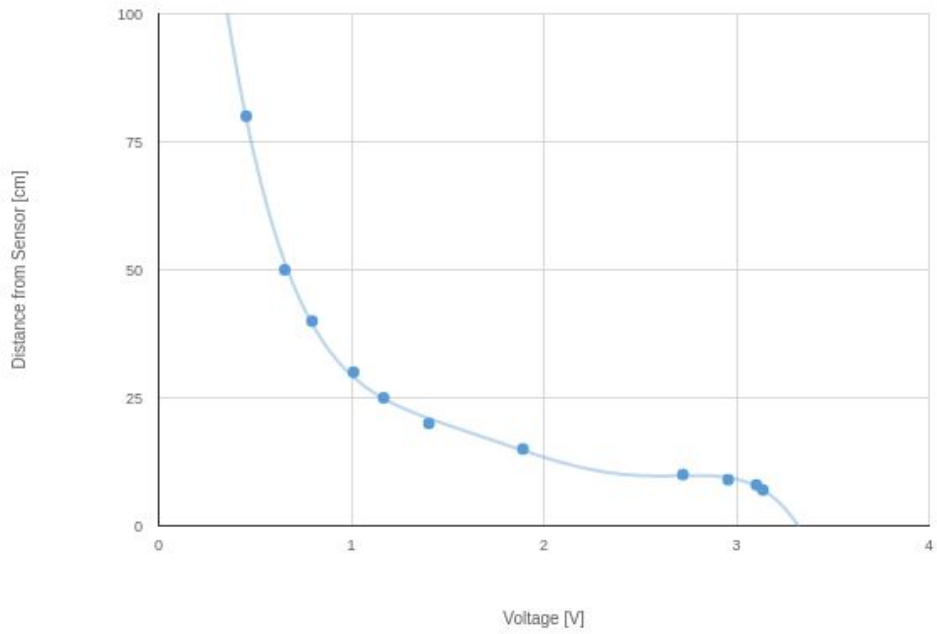

Figure x. Shows a transfer plot between IR analog readings and distance measured courtesy of Clare.

Appendix B. Hardware Setup

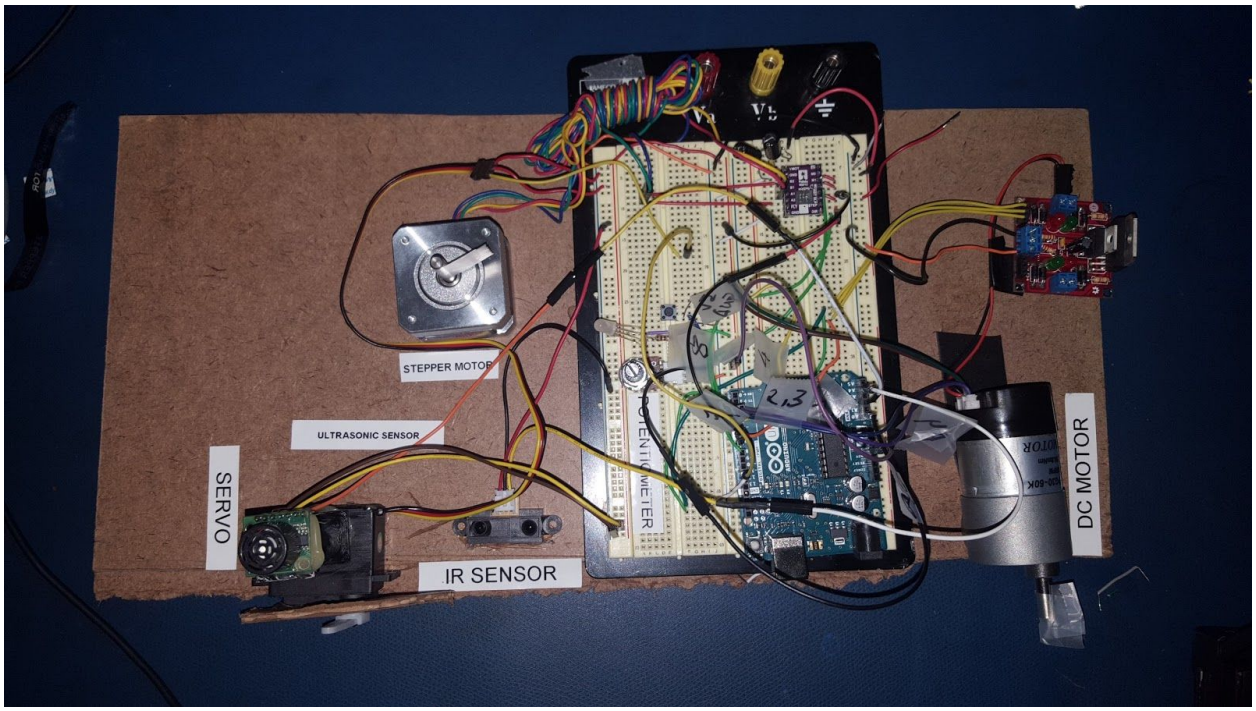

Figure x. Shows the hardware setup that we used for the lab.

Appendix C. Code

Client.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Client</title>
       <meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1">
    <link rel="stylesheet" href="style.css">
    <script   src="http://code.jquery.com/jquery-3.1.1.min.js"
integrity="sha256-hVVnYaiADRTO2PzUGmuLJr8BLUSjGIZsDYGmIJLv2b8="
crossorigin="anonymous"></script>
    <script src="client.js"></script>
    <script src="https://cdn.plot.ly/plotly-latest.js"></script>
  </head>
  <body>
       <div style="display:none" id="vert-list-template"><div class="vertical-list">
```

```
                <div class="vertical-list manual-controls">
                        <span class="manual-value"/>
                        <input class="slider" type="range" min="0" max="100" value="0"
orient="vertical"/>
                </div>
        </div>
  </div>

  <div>
  <span id="connection-bar">Server Address:
  <input type="text" id="wss_address"/>
  <button type="button" id="connect_button">Connect</button></span>
  <span id="status-label"> Status: <span id="status-label-content">Disconnected</span> </span>
  </div>
  <div id="controls-bar">
                <span class="state-control-label">
                        State
                        <select class="state-control">
                                <option value="0">DC Motor Pos</option>
                                <option value="1">DC Motor Vel</option>
                                <option value="2">Servo</option>
                                <option value="3">Stepper</option>
                        </select>
                </span>
                <span class="manual-control-label">
                        Manual Control
                        <input class="manual-control-box" type="checkbox"/>
                </span>
                <input class="manual-slider" type="range" min="0" max="100" value="0"/>
  </div>
  <div id="graph-container" />
  </body>
</html>
```

client.js

```
$(document).ready(function(){
        var server = null;

        var xaxislayout = {
                title: "Time",
                showticklabels: false,
        };
        var diffModules = {
                "sensor1":{
                        writeable:false,
                        unitsStr:"cm",
                        trace:[{
                                x: [],
```

```
                y: [],
                mode:"lines",
                type:"scatter"
        }],
        layout:{
                title: "Ultrasonic",
                xaxis: xaxislayout,
                yaxis:{
                        title: "Distance(cm)",
                        fixedrange: true,
                        range:[0,40]
                }
        },

},
"motor1":{
        writeable:true,
        unitsStr:"deg",
        trace:[{
                x: [],
                y: [],
                mode:"lines",
                type:"scatter"
        }],
        layout:{
                title: "Servo",
                xaxis: xaxislayout,
                yaxis:{
                        title: "Rotation(degrees)",
                        fixedrange: true,
                        range:[0,180]
                }
        }
},
"sensor2":{
        writeable:false,
        unitsStr:"cm",
        trace:[{
                x: [],
                y: [],
                mode:"lines",
                type:"scatter"
        }],
        layout:{
                title: "IR Distance",
                xaxis: xaxislayout,
                yaxis:{
                        title: "Distance(cm)",
                        fixedrange: true,
                        range:[5,71]
```

```
                }
        }
},
"motor2":{
        writeable:true,
        unitsStr:"deg",
        trace:[{
                x: [],
                y: [],
                mode:"lines",
                type:"scatter"
        }],
        layout:{
                title: "Stepper",
                xaxis: xaxislayout,
                yaxis:{
                        title: "Rotation(degrees)",
                        fixedrange: true,
                        range:[0, 360]
                }
        }
},
"sensor3":{
        writeable:false,
        unitsStr:"mV",
        trace:[{
                x: [],
                y: [],
                mode:"lines",
                type:"scatter"
        }],
        layout:{
                title: "Potentiometer",
                xaxis: xaxislayout,
                yaxis:{
                        title: "Volts(mV)",
                        fixedrange: true,
                        range:[0,5000]
                }
        }
},
"motor3":{
        writeable:true,
        unitsStr:"rpm",
        trace:[{
                x: [],
                y: [],
                mode:"lines",
                type:"scatter",
                name:"RPM"
```

```
        },
        {
                x: [],
                y: [],
                mode:"lines",
                type:"scatter",
                name:"P_Term"
        },
        {
                x: [],
                y: [],
                mode:"lines",
                type:"scatter",
                name:"I_Term"
        },
        {
                x: [],
                y: [],
                mode:"lines",
                type:"scatter",
                name:"D_Term"
        },
        {
                x: [],
                y: [],
                mode:"lines",
                type:"scatter",
                name:"Error"
        }
        ],
        layout:{
                title: "DC Motor Velocity",
                xaxis: xaxislayout,
                yaxis:{
                        fixedrange: true,
                        range:[-75,75]
                }
        }
},
"motor4":{
        writeable:false,
        unitsStr:"deg",
        trace:[{
                x: [],
                y: [],
                mode:"lines",
                type:"scatter",
                name:"Pos"
        },
        {
```

```
                x: [],
                y: [],
                mode:"lines",
                type:"scatter",
                name:"P_Term"
        },
        {
                x: [],
                y: [],
                mode:"lines",
                type:"scatter",
                name:"I_Term"
        },
        {
                x: [],
                y: [],
                mode:"lines",
                type:"scatter",
                name:"D_Term"
        },
        {
                x: [],
                y: [],
                mode:"lines",
                type:"scatter",
                name:"Error"
        }
        ],
        layout:{
                title: "DC Motor Position",
                xaxis: xaxislayout,
                yaxis:{
                        fixedrange: true,
                        range:[-180,180]
                }
        }
    },
};


function onMessage(evt){
    var splitByLine = evt.data.split("\n");
    for( var i = 0; i < splitByLine.length; i ++){
        var splits = splitByLine[i].split(",");
        if(splits.length > 1){
                var cmd = splits[0].trim();
                if(cmd == "read"){
                        var id = splits[1].trim();
                        var val = parseFloat(splits[2].trim());
                        var time = parseFloat(splits[3].trim());
```

```
                                        var obj = diffModules[id];
                                        var idx = 0;
                                        if(id.includes("-")){
                                                var moresplits = id.split("-");
                                                id = moresplits[0];
                                                idx = parseInt(moresplits[1]);
                                        }
                                        if(obj){
                                                var div = document.getElementById(id);
                                                var data = div.data;
                                                data[idx].x.push(time);
                                                data[idx].y.push(val);
                                                if(data[idx].x.length > 100){
                                                        data[idx].x.shift();
                                                        data[idx].y.shift();
                                                }
                                                Plotly.redraw(div);
                                        }
                                }
                                else if(cmd == "status"){
                                        var state = parseInt(splits[1].trim());
                                        var manual = splits[2].trim();
                                        var minLim = parseFloat(splits[3].trim());
                                        var maxLim = parseFloat(splits[4].trim());
                                        var manualVal = parseFloat(splits[5].trim());
                                        $(".state-control").val(state);
                                        $(".manual-control-box").prop("checked", manual == "1");
                                        $(".manual-slider").val(manualVal / (maxLim - minLim) *
100);
                                }
                        }
                }
        }

        function generateSliderCmd(id){
                return function(e){
                        var obj = diffModules[id];
                        if(server != null){
                                server.send(id + ",write," + obj.slider.val()+"\n");
                        }
                };
        }

        function generateHideControlsFunc(id){
                return function(){
                        var obj = diffModules[id];
                        if(server != null){
                                var newState = DEFAULT_STATE;
                                if(obj.control_box.prop("checked")){
                                        newState = MANUAL_STATE;
```

```
                        }
                        server.send(id + ",set-state," + newState + "\n");
                }
        };
}

function onManualValueChange(){
        if(server == null)return;
        server.send("write,"+$(".manual-slider").val()+"\n");
}

function onManualChange(){
        if(server == null)return;
        server.send("set-manual," + ($(".manual-control-box").prop("checked")? "1" : "0"
) + "\n");
}

function onStateChange(){
        if(server == null)return;
        server.send("set-state,"+$(".state-control").val()+"\n");
}

function connect(){
        if(server == null){
                try{
                        server = new WebSocket($("#wss_address").val());
                        server.onmessage = onMessage;
                        $("#status-label-content").text("Connecting...");
                        server.onopen = function () {
                                $("#status-label-content").text("Connected!");
                        };

                        server.onclose = function(){
                                server = null;
                                $("#status-label-content").text("Disconnected");
                        };

                        // Log errors
                        server.onerror = function (error) {
                                console.log('WebSocket Error ' + error);
                        };
                        console.log("connect to " + $("#wss_address").val());
                }
                catch(e){
                        console.log(e);
                        if(server != null){
                                var x = server;
                                server = null;
                                x.close()
                        }
```

```
                }
        }
}

function setup(){
        $("#connect_button").click(connect);
        graph_container = document.getElementById("graph-container");
        $(".state-control").on("change", onStateChange);
        $(".manual-control-box").change(onManualChange);
        $(".manual-slider").on("input", onManualValueChange);
        var horizontalDiv;
        divCount = -1;
        //var template = $("#vert-list-template").html();
        for(var prop in diffModules){
                if(divCount == -1 || divCount >= 2){
                        horizontalDiv = document.createElement("div");
                        horizontalDiv.className = "horizontal-graph";
                        graph_container.appendChild(horizontalDiv);
                        divCount = 0;
                }
                var obj = diffModules[prop];
                var newDiv = document.createElement("div");
                newDiv.id = prop;
                horizontalDiv.appendChild(newDiv);
                divCount++;
                /*
                if(obj.writeable){
                        var templateId = prop+"-template";
                        var node = $($.parseHTML(template.trim())[0]);
                        node.attr("id", templateId);
                        node.appendTo(horizontalDiv);
                        obj.slider = node.find(".slider");
                        obj.control_box = node.find(".manual-control-box");
                        obj.controls = node.find(".manual-controls");
                        obj.outputlbl = node.find(".output-value");
                        obj.outputlbl.text("Unknown");
                        obj.outputunits = node.find(".output-units");
                        obj.outputunits.text(obj.unitsStr);
                        obj.controls.css("visibility", "hidden");
                        obj.slider.on("input", generateSliderCmd(prop));
                        obj.control_box.change(generateHideControlsFunc(prop));
                }*/
        }

        for( var prop in diffModules){
                var obj = diffModules[prop];
                Plotly.newPlot(prop, obj.trace, obj.layout,
                        {
                                showLink: false,
                                displaylogo:false,
```

```
                                scrollzoom:true,
                                displayModeBar:false
                        }
                );

        }
    }
    window.onresize = function(){
        for(var prop in diffModules){
            Plotly.Plots.resize(document.getElementById(prop));
        }
    };
    setup();
});
```

style.css

```
@-ms-viewport{
  width: device-width;
}

#graph-container{
}

.horizontal-graph{
    display: flex;
    min-height: 25em;
    align-items: center;
}

.slider{
    writing-mode: bt-lr;
    -webkit-appearance: slider-vertical;
}

.js-plotly-plot{
    width:48vw;
}

.vertical-list{
    display: inline-flex;
    flex-flow: column wrap;
    justify-content: space-between;
}

.vertical-list * {
    margin-top: 1em;
    margin-bottom: 1em;
```

```
}
```

server.py

```
from autobahn.twisted.websocket import WebSocketServerProtocol
from autobahn.twisted.websocket import WebSocketServerFactory
from twisted.internet.task import LoopingCall
from twisted.python import log
from twisted.internet import reactor
import io
import sys
import serial
import argparse
import datetime
import random

connections=[]
serialMsgQueue = []
class SensorMotorLabServerProtocol(WebSocketServerProtocol):

        def __init__(self):
                self.opened = False
                super(WebSocketServerProtocol, self).__init__()

        def onConnect(self, request):
                print("Client Connecting: {}".format(request.peer))

        def onOpen(self):
                connections.append(self)
                self.opened = True

        def onClose(self, wasClean, code, reason):
                print("WebSock closed:{0}".format(reason))
                if(self.opened):
                        connections.remove(self)

        def onMessage(self, payload, isBinary):
                msg = payload.decode('utf8')
                if(msg[0] == '!'):
                        print("WS-DEBUG:" + msg[1:])
                else:
                        serialMsgQueue.append(payload)


class TestServer(SensorMotorLabServerProtocol):

        def onMessage(self, payload, isBinary):
                msg = payload.decode('utf8')
                print (msg)
```

```
                splits = msg.split(",")
                if(len(splits)>1):
                        if(splits[1].strip() == "toggle-control"):
                                sID = splits[0].strip()
                                if states[sID] == "manual":
                                        states[sID] = "sensor"
                                else:
                                        states[sID] = "manual"
read_buffer = ""
def watchSerialPort():
        global read_buffer
        if not (sPort.is_open):
                print ("Serial port is closed! shutting down...")
                sys.exit(1)
        read_buffer += sPort.read(sPort.inWaiting())
        while '\n' in read_buffer:
                line, read_buffer = read_buffer.split('\n', 1)
                if(len(line) == 0): continue
                if(line[0] == '!'):
                        print("SP-DEBUG:" + line[1:])
                else:
                        sendMsg(line)
        for msg in serialMsgQueue:
                sPort.write(msg)
                print( "msg sent to serial port: {}".format(msg.decode('utf8').strip()))
        del serialMsgQueue[:]

states = {
        "motor1":"manual",
        "motor2":"manual",
        "motor3":"manual"
}

def sendDataTest():
        sendMsg("motor1,read,{2},{0},{1}".format(random.randint(0, 360),
(datetime.datetime.now() - startTime).total_seconds(), states["motor1"]))
        sendMsg("motor2,read,{2},{0},{1}".format(random.randint(0, 360),
(datetime.datetime.now() - startTime).total_seconds(), states["motor2"]))
        sendMsg("motor3,read,{2},{0},{1}".format(random.randint(0, 100),
(datetime.datetime.now() - startTime).total_seconds(), states["motor3"]))

def sendMsg(msg):
        bytePayload = msg.encode('utf8')
        for conn in connections:
                conn.sendMessage(bytePayload)

parser = argparse.ArgumentParser(description='Launch ws server that will broadcast information
coming from serial port')
parser.add_argument("-sp", "--serialport")
parser.add_argument("-tcp", "--tcpport", default=8080, type=int)
```

```
parser.add_argument("-br", "--baudrate", default=9600)
args = parser.parse_args()

log.startLogging(sys.stdout)
startTime = datetime.datetime.now()


sPort = serial.Serial(port = args.serialport, baudrate = args.baudrate, timeout = 0)
if not (sPort.is_open):
       raise



serialPortWatcher = LoopingCall(watchSerialPort)
serialPortWatcher.start(0.1)

testDataSender = LoopingCall(sendDataTest)
#testDataSender.start(1)


factory = WebSocketServerFactory()
factory.protocol = SensorMotorLabServerProtocol
reactor.listenTCP(args.tcpport, factory)
reactor.run()
```

Framework.ino

```
#define ENCODER_USE_INTERRUPTS true
#include <Encoder.h>
#include <Servo.h>
#include "io.h"
#include "ServoLib.h"
#include "StepperLib.h"
#include "DCMotorLib.h"
#include "Button.h"
long lastUpdate;
long lastDataDump;
#define DC_MOTOR_POS_STATE 0
#define DC_MOTOR_VEL_STATE 1
#define SERVO_STATE 2
#define STEPPER_STATE 3

#define MAX_STATES 4


ServoLib servo_lib(2, 11);
```

```
int state = 0;
bool manual = false;
int manualValue = 0;

int limits[][2] = {
  {0, 360},
  {-60, 60},
  {0, 180},
  {0, 360}
};

int ir_Last_Dist = 0;
int stepper_Last_Pos = 0;
int uSonic_Last_Dist = 0;
int servo_Last_Pos = 0;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  Serial.setTimeout(0);
  Serial.flush();
  setupIR();
  DCMotorSetup();
  servo_lib.attach();
  lastDataDump = lastUpdate = millis();

}

int* currLimits(){
  return limits[state];
}

int currValue(){
  switch(state){
    case DC_MOTOR_VEL_STATE:
      return M_DC_Speed_Act;
    case DC_MOTOR_POS_STATE:
      return M_DC_Ang_Act;
    case SERVO_STATE:
      return servo_Last_Pos;
    case STEPPER_STATE:
      return stepper_Last_Pos;
  }
}

void parse(char* buf){
  char* curr;
  curr = strtok(buf, ",");
  char* unpack[16];
  int found = 0;
```

```
while(curr != NULL){
  unpack[found] = curr;
  found++;
  curr = strtok(NULL,",");
}
if(found < 1){
  write("!did not understand command!");
  return;
}

if(strcmp(unpack[0],"state-change") == 0){
  state = (state + 1) % MAX_STATES;
  manual = false;
}

if(strcmp(unpack[0],"set-state") == 0){
  if(found > 1){
   int val;
   if(sscanf(unpack[1], "%d", &val) == 1){
     if(val >= 0 && val < MAX_STATES){
       state = val;
       manual = false;
     }
   }
  }
}

if(strcmp(unpack[0],"set-manual") == 0){
  if(found > 1){
   int val;
   if(sscanf(unpack[1], "%d", &val) == 1){
     if(val == 1){
       manual = true;
     }
     else if(val == 0){
       manual = false;
     }
   }
  }
}

if(strcmp(unpack[0], "toggle-manual") == 0){
   manual = !manual;
   manualValue = currValue();
}

if(strcmp(unpack[0], "write") == 0){
  if(found > 1){
   int val;
   if(sscanf(unpack[1], "%d", &val) == 1){
```

```
      int* limits = currLimits();
      manualValue = map(val, 0, 100, limits[0], limits[1]);
      manualValue = constrain(manualValue, limits[0], limits[1]);
     }
    }
  }

}

void loop() {
  DCMotorRead();
  switch(state){
   case DC_MOTOR_POS_STATE:
     DCMotorPosPID();
   break;
   case DC_MOTOR_VEL_STATE:
     DCMotorVelPID();
   break;
   case SERVO_STATE:
     servo_lib.read();
     servo_lib.write(manual ? manualValue : servo_lib.servoPosition);
   break;
   case STEPPER_STATE:
    readIR();
    writeIR(manual ? manualValue :C_angleTargetNew);
   break;
  }

  if(millis() - lastDataDump > 150){
    switch(state){
      case STEPPER_STATE:
     //if(ir_Last_Dist != C_exposedDistAvg || stepper_Last_Pos != C_angle){
       ir_Last_Dist = C_exposedDistAvg;
       write("read,%s,%d,%ld\n", "sensor2", ir_Last_Dist, millis());
       stepper_Last_Pos = C_angle;
       write("read,%s,%d,%ld\n", "motor2", stepper_Last_Pos, millis());
     //}
     break;
     case SERVO_STATE:
     //if(uSonic_Last_Dist != servo_lib.uSonic_dist || servo_Last_Pos !=
servo_lib.servoPosition){
       uSonic_Last_Dist = servo_lib.uSonic_dist;
       write("read,%s,%d,%ld\n", "sensor1", uSonic_Last_Dist, millis());
       servo_Last_Pos = servo_lib.actual_value;
       write("read,%s,%d,%ld\n", "motor1", servo_Last_Pos, millis());
     //}
     break;

     case DC_MOTOR_POS_STATE:
       write("read,%s,%d,%ld\n", "sensor3", map(M_potval, 0, 1023, 0, 5000), millis());
```

```
      write("read,%s,%d,%ld\n", "motor4", M_DC_Ang_Act, millis());
     break;

     case DC_MOTOR_VEL_STATE:
       write("read,%s,%d,%ld\n", "sensor3", map(M_potval, 0, 1023, 0, 5000), millis());
       write("read,%s,%d,%ld\n", "motor3", M_DC_Speed_Act, millis());
     break;
    }
     lastDataDump = millis();
  }

  if(M_button_pushed == 1){
    state = (state + 1) % MAX_STATES;
    manual = false;
    M_motorBrake();
    M_button_pushed = 0;
  }

  char strBuffer[64];
  if(readStringFromSerial(strBuffer)){
    parse(strBuffer);
  }
  if(millis() - lastUpdate > 250){
    int* lims = currLimits();
    write("status,%d,%d,%d,%d,%d\n", state, manual ? 1 : 0, lims[0], lims[1], manualValue);
    lastUpdate = millis();
  }
  button_debounce();
  delay(10);
}
```

io.h

```
#ifndef io_h
#define io_h

// Reads a string from serial. Only emits a string to string handler when "\n" is reached.
Similar to readStringUntil but uses extra buffer to augment Serial Buffer and still functions
when timeout is 0
bool readStringFromSerial(char* buffer);

//Adds a string to write buffer but returns false if no space on buffer. Augments
functionality of write as it will not try and put s on buffer if there is not enough space.
bool write(const char* s, ...);
#endif
```

io.cpp

```
#include "io.h"
```

```
#include "Arduino.h"
#define BUFFER_SIZE 128

//Should use circular array buffer instead of shifting over all the bytes

char read_buff[BUFFER_SIZE];
int buffRead = 0;


char write_buff[BUFFER_SIZE];
int buffWritten = 0;

bool write(const char* s, ...){
  va_list args;
  va_start(args, s);
  int needed = vsnprintf(write_buff, BUFFER_SIZE, s, args);
  va_end(args);
  Serial.write((unsigned char*)write_buff, min(needed, BUFFER_SIZE));
  return true;
}

bool readStringFromSerial(char* buff){
    bool stringFound = false;

 if(Serial.available() > 0  ){
    //Read bytes into big buffer as Serial only buffers 64 chars.
    int numRead = Serial.readBytes(read_buff + buffRead, BUFFER_SIZE - buffRead);
    char* strStart = read_buff;
    for(int i = 0; i < numRead; i++){
      //Look for \n in newly read characters
      if(read_buff[buffRead + i] == '\n'){
        //once found null terminate, and parse it
        read_buff[buffRead + i] = '\0';
        //write("Str recv:%d\n", strlen(strStart));
        strcpy(buff, strStart);
        //Keep track of the latest "valid" buffer
        strStart = read_buff + (buffRead + i + 1);
        stringFound = true;
      }
    }

    //How many bytes total were read including already parsed
    buffRead = buffRead + numRead;
    //How many bytes left after parsing
    int numLeft = buffRead - (strStart - read_buff);
    //Shift over
    for(int i = 0; i < numLeft; i++){
      read_buff[i] = strStart[i];
    }
    //correct numread
```

```
    buffRead = numLeft;
  }
    return stringFound;
}
```