Automated Driving Using External Perception

Individual Lab Report - ILR01

February 9, 2023

Team E - Outersense

Author:

Ronit Hire

Team Members:

Atharv Pulapaka

Dhanesh Pamnani

Jash Shah

Shreyas Jha

Contents

1 Individual Progress

1.1 Sensors and Motors Lab

For this lab, I was responsible for developing the Graphical User Interface (GUI) and defining data packets for exchanging information between the GUI and the micro-controller.

1.1.1 Graphical User Interface

The GUI was developed using the Flask web-framework. It consisted of two modular components, the frontend and the backend. Frontend was responsible for visualizing all the sensor data coming from the Arduino and acting as an interface for receiving user-commands, while the backend was essentially a web-server which communicated with the Arduino using the "PySerial" library.

Figure 1: Team E: Sensor Motor Lab GUI

A screenshot of the GUI can be seen in the image above. The frontend was implemented using a combination of Javascript, HTML and CSS to provide an immersive and an intuitive user-experience. The sensor readings are visualized in the form of real-time graphs which are updated every 100ms with the last 20 readings. Every 100ms, a javascript program makes an AJAX request to the backend server and fetches the latest sensor readings asynchronously and plots them using the ChartJS library. Also, on the GUI dashboard, the user can interact with slider bars associated with different motor types. On selecting the desired state of a motor and toggling the corresponding radio button, the page executes a POST request with the desired state data to the webserver; analogous to submitting a form on websites.

On the backend of the GUI (implemented in python), a SensorData object keeps track of the recent sensor readings and a tokenizer function packs the user-desired motor states into a data packet and sends it to the Arduino. To achieve this two-way communication seamlessly, the webserver runs two parallel threads, one for receiving and decoding incoming sensor data over the serial channel and another one for listening to user-commands, encoding them and forwarding them to the Arduino to execute. The details of the message format used for communication is explained in the next section.

1.1.2 Arduino

Apart from implementing the GUI, I also contributed to establishing a standard message format for exchanging data between the python webserver and the Arduino. In the Arduino code, a struct made of 4 floats, one for each of the sensor (Potentiometer, Ultrasonic, Infrared and Temperature) is used. The struct object is passed over serial in the form of bytes. Using a fixed size (16 bytes) data packet makes it easier to avoid race conditions over serial communication as the buffer can wait until it gets the desired number of bytes. Similarly, when receiving user-desired motor states, a single byte header is first acknowledged, indicating that the next 7 bytes of data need to be unpacked into 3 integers and 1 char. The char is used to uniquely identify a motor for which a control signal is to be sent.

The final circuit can be seen below:

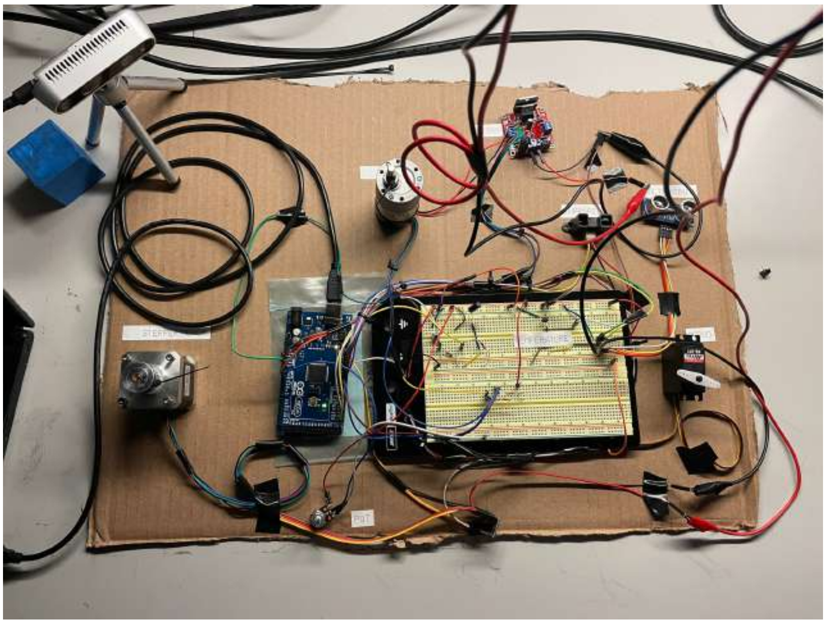

Figure 2: Team E: Circuit for sensor motor lab (Credits: Atharv Pulapaka)

1.2 MRSD project

On the capstone project front, my focus so far has been split between two activities. First is helping my team with the high level design of the perception and the control blocks in our system. I have been involved in sketching out a series of experiments and tests to help us determine the best way to estimate the pose of our RC car in a bird's-eye perspective using an Intel Realsense. I have also been conducting a literature survey on different camera calibration methods, as in our case we will be needing frequent re-calibrations on account of non-rigidity in our track setup. Second area of focus is a bit managerial, as I am also looking into procurement of materials, budgeting and maintaining documentation

2 Challenges

2.1 Sensors and Motors Lab

While making the GUI, I faced the challenge of keeping the CPU load to a minimum when updating the graphs otherwise the backend webserver would freeze leading to loss of communication. The 100ms refresh rate was chosen such that the graph updates appeared real-time and also ensured that the server is not flooded with update requests. Another issue which bugged the team was that for the RPM control of the DC motor our code and the Servo library for Arduino updated the same TIMER register leading to random interrupt function calls when using the motor encoder. We solved this by using the 8bit TIMER register for our DC motor instead of the 16bit registers which are all blocked by the Servo library.

2.2 MRSD project

When we bought the first RC car for our project, we tried to run a mock run around a makeshift track. We realized that the friction forces in the brushed DC motor and its high rotor inertia make it undrivable at our desired speeds. We shifted to a brushless DC motor from the same manufacturer and decided to use a custom Vedder-ESC (VESC) to control it. Configuring the VESC and getting the firmware installed correctly has been a challenge for us so far. The open-source tool available to help configure the motor parameters is glitching and is unable to write any data to the motor. A couple of labs in CMU use a similar ESC and we are in touch with them for potential solutions.

3 Team Work

We split our work for the sensor motor lab as per the guidelines, with 4 people working on a sensor individually and 1 person working on the GUI. The breakdown for the sensor motor lab and the MRSD project is as follows:

- **Jash Shah**: Jash was responsible for the position and velocity control of DC motor for this lab. He implemented PID controls for both and tuned them. On the project, he is working on detecting markers on the car and ARuCo tags on the track as well as experimenting with different approaches of pose estimation.

- **Shreyas Jha**: For the sensors and motor lab, Shreyas worked on setting up the stepper motor, integrating potentiometer for DC motor control and also integrating the entire code for all the sensors and motors. For the project, he is working on customizing the RC with a BLDC motor and configuring an ESC for the same

- **Dhanesh Pamnani**: Dhanesh worked on the IR sensor for this lab. He computed the transfer function for it and also implemented debouncing for the push button. He is working on setting up the track infrastructure for the MRSD project and has taken up manufacturing activities for the time being.

- **Atharv Pulapaka**: For the sensor motor lab, Atharv contributed to overall circuit design and debugging hardware issues. He also interfaced the servo motor with Arduino and implemented Servo sweep with IR sensor readings. On the project front, he is working on the lane keeping behaviour of the RC car using Model Predictive Control.

4 Plans

Building on some of the work from the sensor motor lab, the team will like to progress on all fronts of the project. The plan to is to get most of the hardware elements finished as soon as possible so we can start testing and experimenting with our car on the track. This will help us gather relevant data and highlight some new issues. Moving ahead, I will be working on setting up the Jetson TX2 device and mounting it on a infrastructure unit. This will cover the edge-computing aspect of our project. The Jetson device will run a calibration and pose estimation algorithm communicate data to a central decision making system.

5 Sensor Motor Lab Quiz

1. **From ADXL335 accelerometer datasheet**

   - **What is the sensor's range?**
     A. Typical range is from -3.6g to 3.6g
   - **What is the sensor's dynamic range?**
     A. Dynamic range is 7.2g
   - **What is the purpose of the capacitor CDC on the LHS of the functional block diagram on p. 1? How does it achieve this?**
     A. The role of the capacitor is to act as a filter and prevent or reduce the effect of noise from power supply on sensor readings. The capacitor is invariant to sudden changes in voltage across its terminals.
   - **Write an equation for the sensor's transfer function.**
     A. $V_{out} = 1.5 + 0.3 * a$
   - **What is the largest expected non-linearity error in g?**
     A. Typ. Nonlinearity is 0.3% of full scale, which is equal to $0.3\% * 7.2 = 0.0216g$
   - **What is the sensor's bandwidth for the X- and Y-axes?**
     A. The sensor's bandwidth is 1600Hz for the X- and Y- axes
   - **How much noise do you expect in the X- and Y-axis sensor signals when your measurement bandwidth is 25 Hz?**
     A. Typical noise of ADXL335 is given by $rmsNoise = NoiseDensity * \sqrt{1.6 * BW}$
     Thus, expected noise is 948.68 μg
   - **If you didn't have the datasheet, how would you determine the RMS noise experimentally? State any assumptions and list the steps you would take.**
     A. First place the sensor on a flat surface and ensure that no motion occurs. Second, any reading from the sensor at this point is noise. Taking squareroot of the mean of those values will give us the RMS noise

2. **Signal Conditioning**

   - **Filtering**
     1. Name at least two problems you might have in using a moving average filter.
     A.
     i) Requires large window size and thus data when dealing with high frequency signals
     ii) If the window size is too large, the moving average filter can cause lags.
     2. Name at least two problems you might have in using a median filter.
     A.
     i) Computing median in a stream of data in computationally expensive
     ii) The filter cannot keep track of peaks in data, thus losing those features
   - **Opamps**
     In the following questions, you want to calibrate a linear sensor using the circuit in Fig. 1 so that its output range is 0 to 5V. Identify in each case: 1) which of V1 and V2 will be the input voltage and which the reference voltage; 2) the values of the ratio Rf/Ri and the reference voltage. If the calibration can't be done with this circuit, explain why.

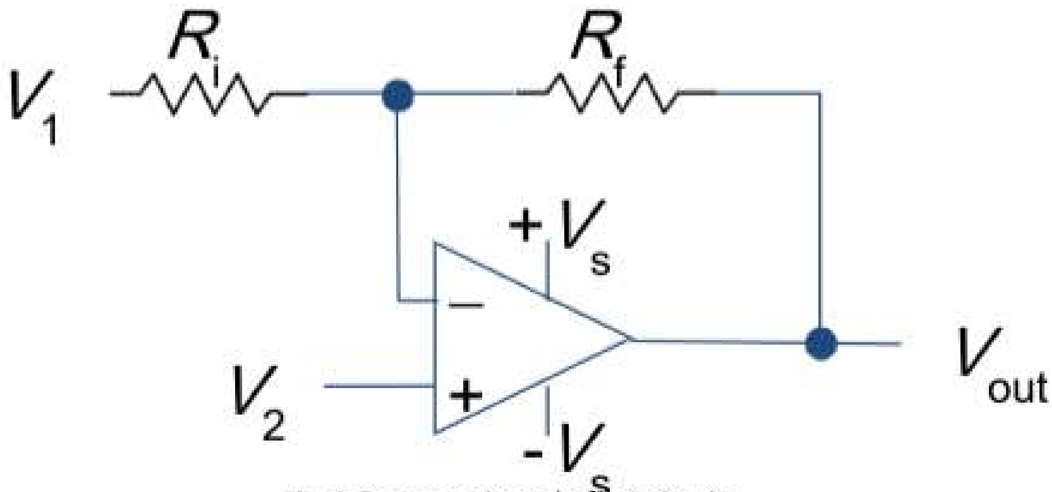

Fig. 1 Opamp gain and offset circuit

Case 1: Your uncalibrated sensor has a range of -1.5 to 1.0V
Going by the opamp equation:

$$\frac{V_1 - V_2}{R_i} = \frac{V_2 - V_{out}}{R_f}$$

Using, $V_1 = V_{in}$ and $V_2 = V_{ref}$, we get,

$$\frac{R_f}{R_i} = -2k\Omega$$

which is not possible, alternatively, we can use $V_1 = V_{ref}$ and $V_2 = V_{in}$

$$\frac{R_f}{R_i} = 1k\Omega$$

Thus, using V_1 as reference and V_2 as input can lead to calibration

Case 2: Your uncalibrated sensor has a range of -2.5 to 2.5V
Going by the opamp equation:

$$\frac{V_1 - V_2}{R_i} = \frac{V_2 - V_{out}}{R_f}$$

Using, $V_1 = V_{in}$ and $V_2 = V_{ref}$, we get,

$$\frac{R_f}{R_i} = -1k\Omega$$

which is not possible, alternatively, we can use $V_1 = V_{ref}$ and $V_2 = V_{in}$

$$\frac{R_f}{R_i} = 0$$

Either R_f has to be 0 or R_i has to be infinite, which is not possible. So calibration is not possible in this case

3. **Control**

- **If you want to control a DC motor to go to a desired position, describe how to form a digital input for each of the PID (Proportional, Integral, Derivative) terms.**
  A. Proportional term is simply the error between the desired position and the current position. For the integral term, we incrementally accumulate the errors at each time step and it forms the input for the next step. For the derivative term, the input is the rate of change of error calculated with respect to the previous error.
- **If the system you want to control is sluggish, which PID term(s) will you use and why?**
  A. The system might feel sluggish because of the high rise time. A way to reduce rise time it to use high proportional gains. Since, K_p directly scales the error for input it can help the system reach the desired state quickly.
- **After applying the control in the previous question, if the system still has significant steady-state error, which PID term(s) will you use and why?**
  A. Using the 'I' term can help in this case because it constantly accumulates past errors and drives the steady state error goes to zero.
- **After applying the control in the previous question, if the system still has overshoot, which PID term(s) will you apply and why?**
  A. The derivative term can help tackle overshoots because the 'D' term is equivalent to a damper and keeps the gains in check as the system reaches the desired state.

6 Snippets of GUI code

```python
# -*- encoding: utf-8 -*-

from apps.home import blueprint
from flask import render_template, request, jsonify
from flask.wrappers import Response
from jinja2 import TemplateNotFound
# from apps.home import pipeline
from apps.home import streaming, realsense_setup, arduino_setup, read_struct, send_mode_to_arduino
import threading
import queue
import time
import struct

class DashboardState():

    def __init__(self):

        # variables for outgoing data
        self.servo_pos = 12
        self.dc_pos = 25
        self.dc_rpm = 20
        self.stepper_pos = 10

        # variables for incoming data
        self.ultrasonic_reading = queue.Queue(maxsize=21)
        self.temperature_reading = queue.Queue(maxsize=21)
        self.rpm_reading = queue.Queue(maxsize=21)
        self.infrared_reading = queue.Queue(maxsize=21)

        # camera state
        self.object_in_frame = False

        # 'S' for sensor mode and 'G' for GUI mode
        self.arduino_mode = 'S'


# bad idea to use global states but it is what it is
ds = DashboardState()

# realsense image pipeline
pipeline = None

# serial communication port for Arduino
ser = None

def __make_threads__(ds, ser, types):

    thrds = []
    if 'rec' in types:
        rec_thrd = threading.Thread(target=read_struct, name='get_data', args=(ds, ser, True))
        thrds.append(rec_thrd)

    # not used
```

```
    if 'send' in types:
        send_thrd = threading.Thread(target=send_mode_to_arduino, name='send_data', args=(ser, ds))
        thrds.append(send_thrd)

    return thrds

@blueprint.route('/index')
def index():
    global pipeline
    global ser
    global ds

    # pipeline = realsense_setup()

    if ser is None:
        ser = arduino_setup()
    ser.reset_input_buffer()
    threads_to_make = []
    for t in threading.enumerate():
        if 'get_datas' not in t.name:
            threads_to_make.append('rec')
        # if 'send_mode' in t.name:
            # threads_to_make.append('send')

    thrds = __make_threads__(ds, ser, threads_to_make)
    for t in thrds:
        t.start()

    for th in threading.enumerate():
        print(th.name)
    return render_template('home/index.html', segment='index')


@blueprint.route('/get_data', methods=['GET'])
def get_sensor_data():
    global ds
    current_data = {}
    current_data['ultrasonic_reading'] = list(ds.ultrasonic_reading.queue)
    current_data['temperature_reading'] = list(ds.temperature_reading.queue)
    current_data['rpm_reading'] = list(ds.rpm_reading.queue)
    current_data['infrared_reading'] = list(ds.infrared_reading.queue)

    return jsonify(current_data)


@blueprint.route('/change_arduino_mode', methods=['GET'])
def change_arduino_mode():
    global ds
    if ds.arduino_mode=='S':
        ds.arduino_mode = 'G'
    elif ds.arduino_mode=='G':
        ds.arduino_mode = 'S'
    return jsonify({})


```

```
@blueprint.route('/send_user_input', methods=['GET', 'POST'])
def send_user_input():
    global ser
    if request.method == 'POST':
        stepper_pos = request.form.get("stepper_pos")
        servo_pos = request.form.get("servo_pos")
        dc_pos = request.form.get("dc_pos")
        dc_rpm = request.form.get("dc_rpm")
        motor_sel = request.form.get("motor_sel")
        if(motor_sel=="STPOS"):
            stype = 'T'
            dc_val = stepper_pos
        elif(motor_sel=="SRPOS"):
            stype = 'M'
            dc_val = servo_pos
        if(motor_sel=="DPOS"):
            stype = 'P'
            dc_val = dc_pos
        elif(motor_sel=="RPM"):
            stype = 'R'
            dc_val = dc_rpm

        send_packet = struct.pack('hhhc', int(stepper_pos), int(servo_pos), int(dc_val), str.encode(stype))
        ser.write(str.encode('I'))
        ser.write(send_packet)


        return jsonify({})
    if request.method =='GET':
        return 'Submission success'


@blueprint.route('/video_feed')
def video_feed():
    return 'Hello world'
    # return Response(streaming.stream(pipeline, ds), mimetype='multipart/x-mixed-replace; boundary=frame')

@blueprint.route('/<template>')
def route_template(template):
    try:
        if not template.endswith('.html'):
            template += '.html'

        # Detect the current page
        segment = get_segment(request)

        # Serve the file (if exists) from app/templates/home/FILE.html
        return render_template("home/" + template, segment=segment)

    except TemplateNotFound:
        return render_template('home/page-404.html'), 404

    except:
        return render_template('home/page-500.html'), 500
```

```python
# Helper - Extract current page name from request
def get_segment(request):
    try:
        segment = request.path.split('/')[-1]

        if segment == '':
            segment = 'index'

        return segment

    except:
        return None
```

Listing 1: Code to launch server

```javascript
type = ['primary', 'info', 'success', 'warning', 'danger'];


DASH_STATE = {
    plot_sensor_data: true,
    py_arduino_mode: "sensor",
};

function toggle_graphs(input_mode) {
    if(input_mode=="gui")
    {
        DASH_STATE.plot_sensor_data = true;
        DASH_STATE.py_arduino_mode = "gui";
    }
    if(input_mode=="sensor")
    {
        DASH_STATE.plot_sensor_data = true;
        DASH_STATE.py_arduino_mode = "sensor";
    }
    fetch('/change_arduino_mode')
        .then(function(response) {
            return;
        });

};

$(document).ready(function() {

    var stepper_pos_slider = document.getElementById("stepper_inp");
    var stepper_pos_val = document.getElementById("stepper_val");

    var servo_pos_slider = document.getElementById("servo_inp");
    var servo_pos_val = document.getElementById("servo_val");

    var dc_pos_slider = document.getElementById("dc_pos_inp");
    var dc_pos_val = document.getElementById("dc_pos_val");

    var dc_rpm_slider = document.getElementById("dc_rpm_inp");
    var dc_rpm_val = document.getElementById("dc_rpm_val");

    stepper_pos_slider.oninput = function() {
        stepper_pos_val.innerHTML = this.value;
    };
```

```
    servo_pos_slider.oninput = function() {
        servo_pos_val.innerHTML = this.value;
    };
    dc_pos_slider.oninput = function() {
        dc_pos_val.innerHTML = this.value;
    };
    dc_rpm_slider.oninput = function() {
        dc_rpm_val.innerHTML = this.value;
    };

    var tooltip_config = {

        backgroundColor: '#f5f5f5',
        titleFontColor: '#333',
        bodyFontColor: '#666',
        bodySpacing: 4,
        xPadding: 12,
        mode: "nearest",
        intersect: 0,
        position: "nearest"
    };

    Chart.defaults.global.tooltips.enabled = false;

    var gradientChartOptionsConfigurationWithTooltipGreen = {
      maintainAspectRatio: false,
      legend: {
        display: false
      },

      responsive: true,
      scales: {
        yAxes: [{
          barPercentage: 1.6,
          gridLines: {
            drawBorder: false,
            color: 'rgba(29,140,248,0.0)',
            zeroLineColor: "transparent",
          },
          ticks: {
            suggestedMin: 50,
            suggestedMax: 125,
            padding: 20,
            fontColor: "#9e9e9e"
          }
        }],

        xAxes: [{
          barPercentage: 1.6,
          gridLines: {
            drawBorder: false,
            color: 'rgba(0,242,195,0.1)',
            zeroLineColor: "transparent",
          },
          ticks: {
            padding: 20,
            fontColor: "#9e9e9e"
          }
```

```
      }]
    }
  };

  var gradientChartOptionsConfigurationWithTooltipBlue = {
    maintainAspectRatio: false,
    legend: {
      display: false
    },

    responsive: true,
    scales: {
      yAxes: [{
        barPercentage: 1.6,
        gridLines: {
          drawBorder: false,
          color: 'rgba(29,140,248,0.0)',
          zeroLineColor: "transparent",
        },
        ticks: {
          suggestedMin: 60,
          suggestedMax: 125,
          padding: 20,
          fontColor: "#2380f7"
        }
      }],

      xAxes: [{
        barPercentage: 1.6,
        gridLines: {
          drawBorder: false,
          color: 'rgba(29,140,248,0.1)',
          zeroLineColor: "transparent",
        },
        ticks: {
          padding: 20,
          fontColor: "#2380f7"
        }
      }]
    }
  };

  var gradientChartOptionsConfigurationWithTooltipPurple = {
    maintainAspectRatio: false,
    legend: {
      display: false
    },

    responsive: true,
    scales: {
      yAxes: [{
        barPercentage: 1.6,
        gridLines: {
          drawBorder: false,
          color: 'rgba(29,140,248,0.0)',
          zeroLineColor: "transparent",
        },
        ticks: {
```

```
            suggestedMin: 60,
            suggestedMax: 125,
            padding: 20,
            fontColor: "#9a9a9a"
          }
        }],

        xAxes: [{
          barPercentage: 1.6,
          gridLines: {
            drawBorder: false,
            color: 'rgba(225,78,202,0.1)',
            zeroLineColor: "transparent",
          },
          ticks: {
            padding: 20,
            fontColor: "#9a9a9a"
          }
        }]
      }
    };

    var gradientChartOptionsConfigurationWithTooltipPurpleTS = {
      maintainAspectRatio: false,
      legend: {
        display: false
      },

      responsive: true,
      scales: {
        yAxes: [{
          barPercentage: 1.6,
          gridLines: {
            drawBorder: false,
            color: 'rgba(29,140,248,0.0)',
            zeroLineColor: "transparent",
          },
          ticks: {
            min: 20,
            max: 120,
            padding: 20,
            fontColor: "#9a9a9a"
          }
        }],

        xAxes: [{
          barPercentage: 1.6,
          gridLines: {
            drawBorder: false,
            color: 'rgba(225,78,202,0.1)',
            zeroLineColor: "transparent",
          },
          ticks: {
            padding: 20,
            fontColor: "#9a9a9a"
          }
        }]
      }
```

```
};

var gradientChartOptionsConfigurationWithTooltipOrange = {
  maintainAspectRatio: false,
  legend: {
    display: false
  },

  responsive: true,
  scales: {
    yAxes: [{
      barPercentage: 1.6,
      gridLines: {
        drawBorder: false,
        color: 'rgba(29,140,248,0.0)',
        zeroLineColor: "transparent",
      },
      ticks: {
        suggestedMin: 50,
        suggestedMax: 110,
        padding: 20,
        fontColor: "#ff8a76"
      }
    }],

    xAxes: [{
      barPercentage: 1.6,
      gridLines: {
        drawBorder: false,
        color: 'rgba(220,53,69,0.1)',
        zeroLineColor: "transparent",
      },
      ticks: {
        padding: 20,
        fontColor: "#ff8a76"
      }
    }]
  }
};



//------------------- DC motor RPM ----------------------------------
var ctx_dcrpm = document.getElementById("grpm").getContext("2d");
//purple colors
var gradientStrokePurple = ctx_dcrpm.createLinearGradient(0, 230, 0,
50);
gradientStrokePurple.addColorStop(1, 'rgba(72,72,176,0.2)');
gradientStrokePurple.addColorStop(0.2, 'rgba(72,72,176,0.0)');
gradientStrokePurple.addColorStop(0, 'rgba(119,52,169,0)');

var data_dcrpm = {
  labels: ['', '', '', '', '', '', '', '', '', '','', '', '', '', '',
 '', '', '', '', ''],
  datasets: [{
    label: "DC motor RPM",
    fill: true,
    backgroundColor: gradientStrokePurple,
```

```
        borderColor: '#d048b6',
        borderWidth: 2,
        borderDash: [],
        borderDashOffset: 0.0,
        pointBackgroundColor: '#d048b6',
        pointBorderColor: 'rgba(255,255,255,0)',
        pointHoverBackgroundColor: '#d048b6',
        pointBorderWidth: 20,
        pointHoverRadius: 4,
        pointHoverBorderWidth: 15,
        pointRadius: 4,
        data: [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
      }]
    };

    var chart_dcrpm = new Chart(ctx_dcrpm, {
      type: 'line',
      data: data_dcrpm,
      options: gradientChartOptionsConfigurationWithTooltipPurple
    });
    //
----------------------------------------------------------------------

    //
    //
    //
    //------------------- Ultrasonic reading
--------------------------------
    var ctx_us = document.getElementById("gus").getContext("2d");
    //green colors
    var gradientStrokeGreen = ctx_us.createLinearGradient(0, 230, 0, 50);
    gradientStrokeGreen.addColorStop(1, 'rgba(66,134,121,0.15)');
    gradientStrokeGreen.addColorStop(0.4, 'rgba(66,134,121,0.0)');
    gradientStrokeGreen.addColorStop(0, 'rgba(66,134,121,0)');

    var data_us = {
      labels: ['', '', '', '', '', '', '', '', '', '','', '', '', '', '',
    '', '', '', '', ''],
      datasets: [{
        label: "Ultrasonic distance",
        fill: true,
        backgroundColor: gradientStrokeGreen,
        borderColor: '#00d6b4',
        borderWidth: 2,
        borderDash: [],
        borderDashOffset: 0.0,
        pointBackgroundColor: '#00d6b4',
        pointBorderColor: 'rgba(255,255,255,0)',
        pointHoverBackgroundColor: '#00d6b4',
        pointBorderWidth: 20,
        pointHoverRadius: 4,
        pointHoverBorderWidth: 15,
        pointRadius: 4,
        data: [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
      }]
    };

    var chart_us = new Chart(ctx_us, {
```

```
    type: 'line',
    data: data_us,
    options: gradientChartOptionsConfigurationWithTooltipGreen
  });
  //----------------------------------------------------------------------

  //
  //
  //
  //------------------- TEMPERATURE  ----------------------------------
  var ctx_gts = document.getElementById("gts").getContext("2d");
  //purple colors
  var gradientStrokePurple = ctx_gts.createLinearGradient(0, 230, 0, 50);
  gradientStrokePurple.addColorStop(1, 'rgba(72,72,176,0.2)');
  gradientStrokePurple.addColorStop(0.2, 'rgba(72,72,176,0.0)');
  gradientStrokePurple.addColorStop(0, 'rgba(119,52,169,0)');

  var data_ts = {
    labels: ['', '', '', '', '', '', '', '', '', '','', '', '', '', '', '', '', '', '', ''],
    datasets: [{
      label: "Temperature",
      fill: true,
      backgroundColor: gradientStrokePurple,
      borderColor: '#d048b6',
      borderWidth: 2,
      borderDash: [],
      borderDashOffset: 0.0,
      pointBackgroundColor: '#d048b6',
      pointBorderColor: 'rgba(255,255,255,0)',
      pointHoverBackgroundColor: '#d048b6',
      pointBorderWidth: 20,
      pointHoverRadius: 4,
      pointHoverBorderWidth: 15,
      pointRadius: 4,
      data: [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
    }]
  };

  var chart_gts = new Chart(ctx_gts, {
    type: 'line',
    data: data_ts,
    options: gradientChartOptionsConfigurationWithTooltipPurpleTS
  });
  //----------------------------------------------------------------------

  //
  //
  //
  //------------------- Infrared Reading --------------------------------
  var ctx_ir = document.getElementById("gir").getContext("2d");
  //green colors
  var gradientStrokeGreen = ctx_ir.createLinearGradient(0, 230, 0, 50);
  gradientStrokeGreen.addColorStop(1, 'rgba(66,134,121,0.15)');
```

```
    gradientStrokeGreen.addColorStop(0.4, 'rgba(66,134,121,0.0)');
    gradientStrokeGreen.addColorStop(0, 'rgba(66,134,121,0)');

    var data_ir = {
      labels: ['', '', '', '', '', '', '', '', '', '','', '', '', '', '', '', '', '', '', ''],
      datasets: [{
        label: "Infrared",
        fill: true,
        backgroundColor: gradientStrokeGreen,
        borderColor: '#00d6b4',
        borderWidth: 2,
        borderDash: [],
        borderDashOffset: 0.0,
        pointBackgroundColor: '#00d6b4',
        pointBorderColor: 'rgba(255,255,255,0)',
        pointHoverBackgroundColor: '#00d6b4',
        pointBorderWidth: 20,
        pointHoverRadius: 4,
        pointHoverBorderWidth: 15,
        pointRadius: 4,
        data: [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
      }]
    };

    var chart_ir = new Chart(ctx_ir, {
      type: 'line',
      data: data_ir,
      options: gradientChartOptionsConfigurationWithTooltipGreen
    });
    // ----------------------------------------------------------------------


    let ultrasonic_reading = []
    let temperature_reading = []
    let rpm_reading = []
    let infrared_reading = []

    function get_arduino_data() {
        if(DASH_STATE.plot_sensor_data === true) {

            fetch('/get_data')
                .then(function(response) {
                    return response.json();
                })
                .then(function (sdata){
                    ultrasonic_reading = sdata.ultrasonic_reading;
                    temperature_reading = sdata.temperature_reading;
                    rpm_reading = sdata.rpm_reading;
                    infrared_reading = sdata.infrared_reading;
                });
        }
    };
    setInterval(get_arduino_data,100);

    function updategraph(chart, newdata) {
```

```
        chart.data.datasets.forEach((dataset) => {
            dataset.data = newdata;
        });
        chart.update();
    };
    setInterval(function() {updategraph(chart_us, ultrasonic_reading)}, 100);
    setInterval(function() {updategraph(chart_ir, infrared_reading)}, 100);
    setInterval(function() {updategraph(chart_gts, temperature_reading)}, 100);
    setInterval(function() {updategraph(chart_dcrpm, rpm_reading)}, 100);


    // submit user input asynchronously
    $('#user_input').submit(function(e) {

        var sensor_inputs = $(this).serialize();
        console.log(sensor_inputs);

        $.ajax({
            type: "POST",
            url: "/send_user_input",
            data: sensor_inputs,
            success: function() {
                $('#message').html("Data posted");
            }
        });

        e.preventDefault();

        return false;
    })



});
```

Listing 2: Frontend JS code