ILR01 - Sensors and Motor Control Lab

Justin Morris
Teammates: Awadhut Thube, Alex Withers
Team G: The Pit Crew

February 13, 2020

1 Individual Progress

1.1 Sensors and Motor Control Lab

My major contribution to the Sensors and Motor Control lab project was the design and implementation of the GUI. I used the Processing language to develop an application that provided a graphical interface to display sensor and motor data from the Arduino and control and adjust the behavior of the motors.

The Processing application communicated with the Arduino through a serial buffer. Both programs sent and received messages over this serial buffer. The messages were formatted as a single letter followed by a number, and the Arduino and Processing programs parsed these messages according to a scheme that I defined.

Arduino to GUI		GUI to Arduino	
Stepper Motor Position	s###	Servo Motor Step Size	t###
Infrared Sensor Reading	i###	DC Motor Direction	d#
Servo Motor Position	r###	DC Motor Position	p###
Force Sensor Reading	n###	DC Motor Velocity	v###
DC Motor Position	p###		
DC Motor Velocity	v###		

Table 1: Allowed formats for messages sent between the program running on the Arduino and the program running on the laptop.

The visual elements of the GUI were designed to communicate the behavior of the Arduino's physical components in a logical and clear way. These features can be seen in Figure 1 on the next page.

The horizontal bar at the top of the GUI window (1) was used to display the distance reading received from the infrared sensor in cm. This feature was implemented using the IFProgressBar class available through the Interfascia library for Processing.

Immediately below that are the GUI elements related to the servo motor and force sensor. The circular element on the left (2) functions as a "warning light" that changes color when the readings from the force sensor exceed the threshold for actuating the servo motor. Directly below this warning light is a numerical readout showing the force detected by the sensor. The semicircle to the right (3) shows the range of positions that the servo motor can reach. The blue arc within that semicircle shows the motor's current position (the counterclockwise edge of the arc) and the size of the step that the motor will take when the force sensor is activated. Just below the semicircle is a text field in which an operator can enter in a number which dictates the number of degrees that the motor will rotate with each step.

All features in the lower half of the GUI window relate to the DC motor, which is controlled primarily through the GUI. The radio buttons on the left side (4), again implemented with the Interfascia library, allow the user to select between two modes of controlling the DC motor. Depending on the mode, the user can either command the DC motor to move to relative positions (i.e. -180 degrees from its current position) or rotate continuously at a given velocity. Regardless of mode, the desired value is entered into the text field immediately to the right of the radio buttons (5). The new instruction is sent to the Arduino when the "Update" button is pressed. As soon as a new command is sent to the Arduino, the graph (6) displays both the desired value and the actual position or velocity of the motor, so that the user can confirm that the actual motor state is converging to the desired value. The graph was created using the Grafica library.

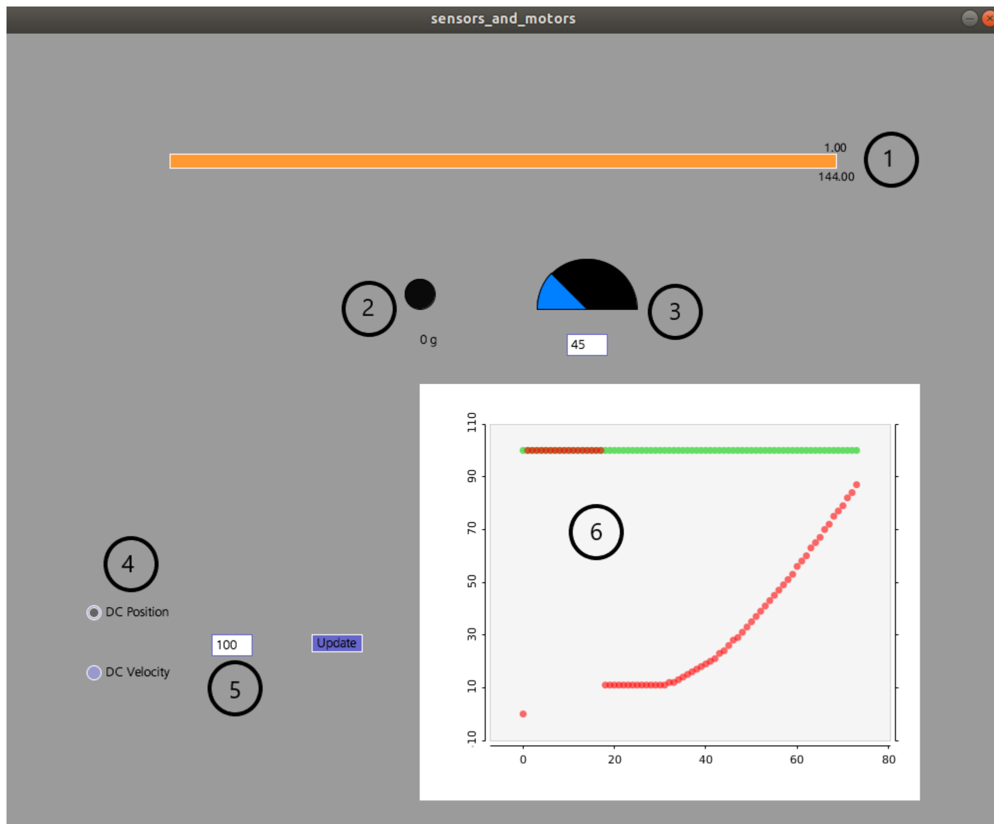

Figure 1: The GUI, with important elements highlighted: (1) Bar showing IR sensor distance reading; (2) indicator light showing when force sensor crosses force threshold, with numerical readout of force; (3) servo motor position display and step size input field; (4) mode selection for DC motor control; (5) input field for DC motor positions and velocities; and (6) graph showing that DC motor tracks desired input.

1.2 Pit Navigator

Much of the work that I have done with regard to the Pit Navigator project thus far has been about refining our top development priorities and taking stock of related work that is available for us to build on. In practice, this has meant meeting with other people involved in the MoonRanger/PitRanger project and reviewing established codebases developed during earlier stages of the project.

I was able to spend some time with a robot called Blue that was built by Jordan Ford and Neil Khera as a platform for collecting camera data for PitRanger development. Blue has a skid-steer drive system and is similar in size to the future flight rover, so we intend to use Blue for our testing as much as possible. Some code to drive Blue and operate its RealSense camera has already been written by other people involved with the MoonRanger project, so I am evaluating that code to determine how it meshes with our needs.

Another existing code resource that we intend to use as a stepping stone is a project done by Neil Khera last semester on the topic of brinkmanship. Brinkmanship is the act of operating close to an edge (in this case the edge of the pit). Neil's project used depth readings from a RealSense camera to detect dangerous edges and prevent an AutoKrawler robot from driving over those edges. We intend to further develop this concept by detecting edges using only stereo image data (since we cannot use a RealSense camera on the moon) and increase the sophistication of the safety criteria used to influence the rover's behavior. As a first step in that process, I

have been communicating with Neil about his work and reviewing his code to understand how he implemented the brinkmanship functionality.

Lastly, I have been studying available methods for detecting features in stereo image data. We continue to anticipate that detecting pit edges in our images will be a major aspect of our project. To that end, it is important that we familiarize ourselves with the existing methods for accomplishing this sort of task, and I personally cannot rely solely on the Computer Vision course to provide the relevant information at the pace or depth that the MRSD project demands.

2 Challenges

2.1 Sensors and Motor Control Lab

In the Sensors and Motor Control lab, I had the challenge of discovering how to implement GUI features in the Processing language that would clearly communicate the sensor and motor values that we were operating on, and allow a user to affect those values in a logical way. This required me to delve into several of the external libraries that exist for use with Processing, some of which had minimal or incomplete documentation. Luckily, these libraries provided access to their source code, so in the worst case I could delve into the underlying class files and discern how the libraries functioned in that way.

As a group, we were faced with the challenge of using relatively cheap components, which naturally placed a limit on the accuracy and resolution of our controls. However, my group members were able to minimize the negative effects of this issue by using their knowledge of signal processing and PID control to improve the responses of the motors to our inputs.

2.2 Pit Navigator

The most obvious challenge we have faced thus far has been that there are only three of us, and we have many commitments that we have to balance. This is helped greatly by the fact that there are various other people and groups working on other aspects of this project alongside us, but this situation presents its own challenges. We must integrate and compile the work that has already been done, some of which is poorly documented and organized. We also must occasionally compete with these other contributors for access to test platforms like Blue and the AutoKrawlers.

Another challenge that I have personally faced is that this project has a large computer vision component, and I do not have much experience in that area. I am currently taking the Computer Vision course (16-720), but the pace of that course does not necessarily match the schedule of this project. I am having to improve my knowledge in this area independently in order to achieve the goals of the Pit Navigation project.

3 Teamwork

3.1 Sensors and Motor Control Lab

While I worked on developing the GUI, Awadhut and Alex split the work of wiring the circuits and writing the Arduino code to link the sensor data and motor responses. They began by writing separate programs on different Arduino boards, then combined those programs into a single program that operated all the functions. We all collaborated to come up with logical and interesting ways to control motor behavior using the sensors available to us, and transmit that information between the laptop GUI and the Arduino program. Combining each aspect of the lab resulted in some time spent chasing down bugs and miscommunications, but that is to be expected, and overall it was a very successful and supportive process.

3.2 Pit Navigator

Because of his prior experience in computer vision, Awadhut has taken the initiative in experimenting with applying various feature detection methods to a data set of images taken of a pit in Utah that is ostensibly analagous to the conditions on the moon. While his conclusion has been that detecting pit shape solely from image data is a difficult and potentially impossible task, it has been very helpful to have data to support that conclusion this early on in the project. He has also helped me to deepen my own understanding of computer vision as I work alongside him on this task.

Alex has been hard at work familiarizing himself with the simulator used by the MRSD team that worked on a related project last year. Since we intend to do much of our early work in a simulated environment, it is crucial that we have a detailed and realistic simulation of a lunar pit. We want to build on the work already done in this area by the previous team and others within the MoonRanger organization. I appreciate that Alex has made this a priority.

4 Plans

4.1 Pit Navigator

Our intent in the coming weeks is to attack this project on two fronts. We will continue to develop the simulator, and construct an environment in which a simulated robot can navigate around a lunar pit and collect images of its surroundings. Meanwhile, we will collaborate with others in the MoonRanger organization to get Blue to a point where we can drive it around and capture images using the RealSense camera. In particular, we intend to take Blue to Gascola, an external site that has some terrain which we can use to test brinkmanship routines or collect edge data that we can experiment with as we further develop our software. This latter task will be primarily my responsibility.

5 Quiz

5.1 Question 1 - Reading a Datasheet

1. **What is the sensor's range?** $6g$
2. **What is the sensor's dynamic range?** $\pm 3g$
3. **What is the purpose of the capacitor C_{DC} on the LHS of the functional block diagram on p. 1? How does it achieve this?** It prevents the accelerometer readings from being affected by noise from the voltage source by resisting sudden changes in voltage.
4. **Write an equation for the sensor's transfer function.** $V_out = 1.5V + (300mV/g) * a$
5. **What is the largest expected nonlinearity error in g?** $0.018g$
6. **How much noise do you expect in the X- and Y-axis sensor signals when the sensor is excited at 25 Hz?** $750\mu g$
7. **How about at 0 Hz? If you can't get this from the datasheet, how would you determine it experimentally?** The relation given in the datasheet does not hold at 0 Hz. To determine this value experimentally, one could measure sensor readings when no signal is being input.

5.2 Question 2 - Signal Conditioning

5.2.1 Filtering

Moving average filters will not filter out large outliers, and may overemphasize old data points (unless the filter uses a weighted average).
Median filters require knowledge of the sorts of outliers that can be expected in order to set the window size properly, and may slow the signal processing down due to the calculation required to determine the median.

5.2.2 Op-Amps

Case 1: $V_{ref} = -3V$, $\frac{R_f}{R_i} = 1$
Case 2: The calibration cannot be done with this circuit. The calibration requires a unity gain and an offset of $+2.5V$, and no ratio of resistances can provide that in this circuit configuration (the only solution to the system of equations requires a negative resistance, which is impossible).

5.3 Question 3 - Control

Proportional Control: To form a digital input for proportional control, take the difference between the current state and the desired state of the system at each time step, then multiply that value by some constant K_p and use that as your input for the next time step. If a system is responding sluggishly, increasing the proportional term will cause the system to react more aggressively to being far away from the goal state.
Integral Control: To form a digital input for integral control, sum the error of the system from the past several time steps, then multiply the resulting value by some constant K_i and provide that to the plant as an input. If a system has significant steady-state error after the proportional and derivative gains have been tuned, the integral gain can be increased to move the steady-state value closer to the desired state. This works because the integral term adjusts the system based on its previous states, so if error exists in those states it will increase the effect of the integral term.
Derivative Control: To form a digital input for derivative control, take the difference of system errors between the current and previous time steps. Multiply this difference by a constant K_d and use that value to control your system during the next time step. If a system is demonstrating overshoot, the derivative gain should be adjusted to compensate. The derivative term reacts to the rate of change of the system, and can therefore be used to dampen the system response from the P and I terms. Proper damping will allow the system to reach its goal state quickly but with minimal overshoot or oscillation.

6 Sensors and Motor Control GUI Code

```
import interfascia.*; // Provides GUI elements
import grafica.*; // Provides plots

import processing.serial.*;

GUIController c;
IFProgressBar p; // Display IR sensor reading
IFLabel force_reading; // Display force sensor reading
IFTextField servo_step_tf, dc_tf; // Field for inputting motor step size
IFRadioController rc; // Radio button controller
IFRadioButton dc_pos_control, dc_vel_control; // Radio buttons for setting DC control mode
```

```
IFButton bGo; // Set new value for DC motor control
GPlot plot;

int pbarhpos, pbarwid; // Progress bar position and width
float pnumpos, pnumpos2, pnumwid; // Progress bar number position and width
float percent = 0.5; // Progress bar fill percentage
PFont prog;

boolean f_t = false;
float force = 0;

int mposvpos = 275;
boolean servo_step_change = true;
float arcpos, arcwid, arcwid_temp; // Motor position and step size
String aw = "45"; // Default motor step size text field contents
int aw_int; // Motor step size

int dc_radbut_hpos; // Radio button horizontal position
int dc_tf_hpos; // Text field horizontal position
int bGohpos; // Button position
int dc_midline; // Horizontal centerline for all DC-related elements

boolean track_error, track_error_start, pos_control;
String dc_goal = "";
String out = "";
float goal = 50;
int arr_len = 100;
int point_count = 0;
GPointsArray goals = new GPointsArray();
GPointsArray actual = new GPointsArray();

float stepper_pos, ir_dist, servo_pos, dc_pos_err, dc_vel;

Serial port;
String serial_data;

void setup() {
  size(1000, 800); // Window size
  background(155); // Background color

  pbarhpos = width/6; // Progress bar horizontal location (left edge)
  pbarwid = width*2/3; // Progress bar width
  pnumwid = textWidth(nf(percent,1,2)); // Progress bar number width
  pnumpos = pbarhpos + (pbarwid*percent) - (pnumwid/2); // Progress bar number horizontal
  pnumpos2 = pnumpos;

  dc_radbut_hpos = width / 12;
```

```
dc_tf_hpos = (width*5/24);

bGohpos = (width / 3) - 25; // "Go" button horizontal location
dc_midline = height * 3/4; // "Go" button vertical location

// Initialize values of Arduino input variables
stepper_pos = 0; // Stepper motor position
servo_pos = 0; // Servo motor position TODO: change default
dc_pos_err = 0; // DC motor position
dc_vel = 0; // DC motor velocity
ir_dist = (percent * 120) + 20; // IR sensor distance reading

prog = createFont("Ubuntu",12,true);

// GUI element constructors
c = new GUIController(this);

rc = new IFRadioController("DC Mode Selector");

bGo = new IFButton("Update", bGohpos, dc_midline, 50, 17); // "Go" button

p = new IFProgressBar(pbarhpos, 120, pbarwid); // IR distance display

force_reading = new IFLabel("", width*5/12 - 15, mposvpos+25);

servo_step_tf = new IFTextField("Arc Width", width*7/12 - 20, mposvpos+25, 40, "45"); //

dc_pos_control = new IFRadioButton("DC Position", dc_radbut_hpos, dc_midline - 30, rc);
dc_vel_control = new IFRadioButton("DC Velocity", dc_radbut_hpos, dc_midline + 30, rc);

dc_tf = new IFTextField("DC Motor Input", dc_tf_hpos, dc_midline, 40, "0"); // DC motor

plot = new GPlot(this, width * 5/12, 350, width / 2, width * 5/12);

track_error = false;
track_error_start = false;
pos_control = false;

// Add two sets of points to the plot
 goals.add(0, goal);
actual.add(0, 0);
point_count++;

plot.setPoints(goals);
plot.addLayer("actual", actual);

// Change the color of the points in the first layer
```

```
  plot.setPointColor(color(100, 220, 100));

  // ActionListeners to respond to button presses
  bGo.addActionListener(this);

  // Add GUI elements to GUI
  c.add(bGo);
  c.add(p);
  c.add(force_reading);
  c.add(servo_step_tf);
  c.add(dc_pos_control);
  c.add(dc_vel_control);
  c.add(rc);
  c.add(dc_tf);

  // Set initial progress bar fill percentage
  p.setProgress(percent);

  // Open serial port at speed 9600 bps
  port = new Serial(this, Serial.list()[0], 9600);
}

void draw() {
  background(155);

  // If there is data in Serial buffer, pass it to parseSerialData()
  if (0 < port.available()) {
    serial_data = port.readStringUntil(';');
    parseSerialData(serial_data);
  }

  // Set progress bar fill percentage
  p.setProgress(percent);

  // Update progress bar number
  pnumwid = textWidth(nf(percent,1,2));
  pnumpos = pbarhpos + (pbarwid*percent) - (pnumwid/2);

  pnumwid = textWidth(nf(ir_dist,1,2));
  pnumpos2 = pbarhpos + (pbarwid*percent) - (pnumwid/2);

  textFont(prog,12);
  fill(0);
  text(nf(percent, 1, 2),pnumpos,118);
  text(nf(ir_dist, 1, 2),pnumpos2,147);

  force_reading.setLabel(nf(force) + " g");
```

```
force_reading.setX(width*5/12 - (force_reading.getWidth() / 2));

// Turn "warning light" on if force sensor/potentionmeter value exceeds threshold
if (f_t) {
  fill(255,128,0);
} else {
  fill(10, 10, 10);
}
ellipse(width*5/12, mposvpos - 15, 30, 30); // "Warning light"

// Remove leading/trailing spaces from text field contents
aw = trim(servo_step_tf.getValue());

// Convert text field contents to integer
if (aw == null || aw.equals("")) {
  aw_int = 0;
} else {
  aw_int = Integer.parseInt(aw);
}

// Set motor position and motor step size
arcpos = ((servo_pos / 180) * PI) + PI;
arcwid_temp = arcwid;
arcwid = (aw_int * PI) / 180;
if (arcwid != arcwid_temp) {
  servo_step_change = true;
}

// Send servo motor data through serial port
if (servo_step_change) {
  port.write("t" + nf(aw_int) + ";");
  servo_step_change = false;
}

// Draw motor position and step size display
fill(0);
arc(width*7/12, mposvpos, 100, 100, PI, 2*PI, PIE);

// Update motor position and step size in display
fill(0, 128, 255);
arc(width*7/12, mposvpos, 100, 100, max(arcpos, PI), min(arcpos+arcwid, 2*PI), PIE);

// Remove leading/trailing spaces from text field contents
dc_goal = trim(dc_tf.getValue());

if (track_error) {
  GPoint cur_goal = new GPoint(point_count, goal);
```

```
    plot.addPoint(cur_goal);

    if (pos_control) {
      GPoint cur_actual = new GPoint(point_count, goal - dc_pos_err);
      plot.addPoint(cur_actual, "actual");
    } else {
      GPoint cur_actual = new GPoint(point_count, dc_vel);
      plot.addPoint(cur_actual, "actual");
    }
    point_count++;

    if (point_count > arr_len) {
      plot.removePoint(0);
      plot.removePoint(0, "actual");
    }
  }

  // Send DC motor data out through serial port
  if (track_error_start) {
    if (goal >= 0) {
      if (pos_control) {
        out = "d" + nf(1) + ";p" + nf(goal) + ";";
      } else {
        out = "d" + nf(1) + ";v" + nf(goal) + ";";
      }
    } else {
      if (pos_control) {
        out = "d" + nf(0) + ";p" + nf(abs(goal)) + ";";
      } else {
        out = "d" + nf(0) + ";v" + nf(abs(goal)) + ";";
      }
    }
    port.write(out);
  }

  if (point_count > arr_len * 10 || track_error_start || !track_error) {
    if (track_error_start) {
      track_error = true;
    } else {
      track_error = false;
    }
    track_error_start = false;
    goals.set(0, new GPoint(0, goal));
    actual.set(0, new GPoint(0, 0));

    plot.setPoints(goals);
    plot.removeLayer("actual");
```

```
    plot.addLayer("actual", actual);
    point_count = 1;
  }

  plot.beginDraw();
  plot.drawBackground();
  plot.drawBox();
  plot.drawXAxis();
  plot.drawYAxis();
  plot.drawRightAxis();
  plot.drawTitle();
  plot.getMainLayer().drawPoints();
  plot.getLayer("actual").drawPoints();
  plot.endDraw();
}

void actionPerformed(GUIEvent e) {
// Checks if button has been pressed, and enables error plot in appropriate mode
  if (e.getSource() == bGo) {
    if (!(dc_goal == null || dc_goal.equals(""))) {
      if (dc_pos_control.isSelected()) {
        track_error_start = true;
        pos_control = true;
        goal = Integer.parseInt(dc_goal);
      } else if (dc_vel_control.isSelected()) {
        track_error_start = true;
        pos_control = false;
        goal = Integer.parseInt(dc_goal);
      } else {
        track_error_start = false;
        track_error = false;
        pos_control = false;
      }
    } else {
      track_error_start = false;
      track_error = false;
      pos_control = false;
    }
  }
}

void parseSerialData(String data) {
  String[] coms = split(data, ',');

  if (coms == null) {
    return;
  }
```

```
  for (int c = 0; c < coms.length; c++) {
    String cmd = coms[c].trim();

    if (cmd == null || cmd.equals(";")) {
      return;
    }

    char type = cmd.charAt(0);
    int val = int(cmd.substring(1, cmd.length()-1));

    switch(type) {
      case 's':
        stepper_pos = val;
        break;
      case 'i':
        ir_dist = val;
        percent = (ir_dist - 20) / 120;

        // Restricts value of percent to between 0 and 1
        percent = max(0, min(1, percent));
        break;
      case 'r':
        servo_pos = val;
        break;
      case 'n':
        force = val;
        if (force > 3) {
          f_t = true;
        } else {
          f_t = false;
        }
        break;
      case 'p':
        dc_pos_err = val;
        break;
      case 'v':
        dc_vel = val;
        break;
      default:
        break;
    }
  }
}
```