INDIVIDUAL LAB REPORT 1

February 15, 2019

John Macdonald

Wholesome Robotics, Team E

Teammates:
Aman Agarwal
Aaditya Saraiya
Hillel Hochsztein
Dung-Han Lee

Contents

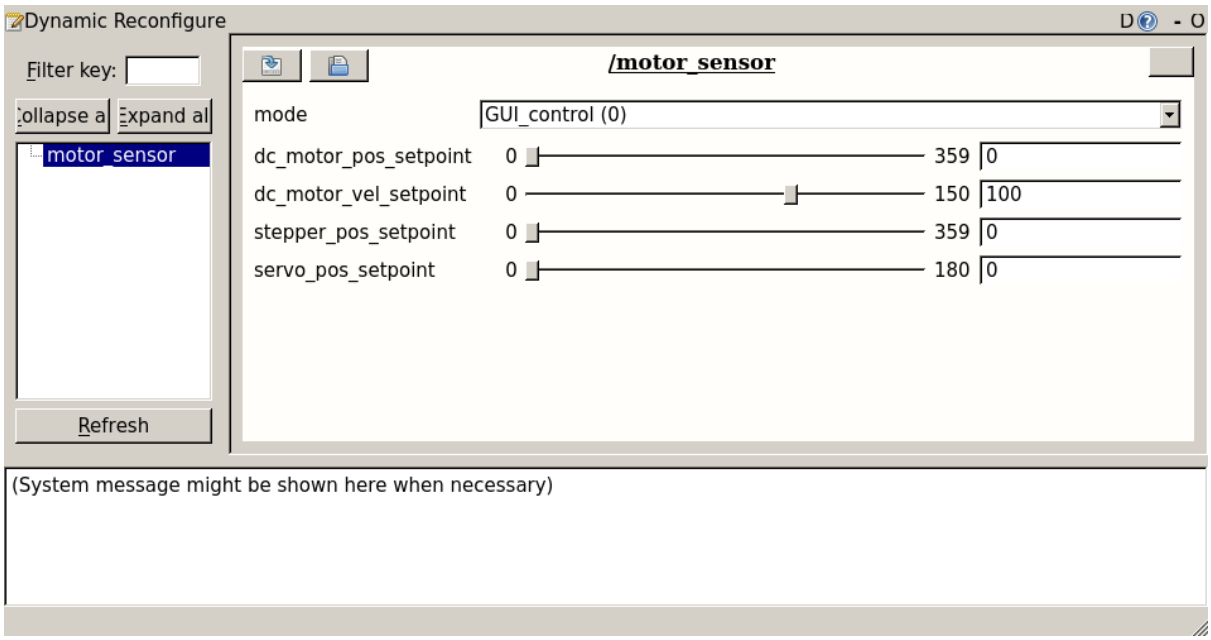


Figure 1: The GUI running my personal laptop.

0.1 Motors and Sensors Lab

0.1.1 Individual Progress

I took on the role of designing and implementing the GUI as well as integrating the various software components. We initially held a meeting in which we extracted system requirements from the prompt and agreed on work distribution. I drafted an initial architecture separating these tasks into software modules which could be assigned to individual members. The software was split into separate modules for: (1) the GUI Server, (2) the GUI Client, (3) Servo control, (4) Stepper control, (5) DC Motor control, (6) reading the temperature sensor, (7) reading the ultrasound rangefinder, (8) reading the IR rangefinder, and (9) reading the slot sensor. The GUI server was implemented as a Python program on my personal computer, whereas components (2) - (9) were implemented on an Arduino Mega as separate files, each with its own header. I wrote the initial versions as the header files to define the interfaces, as well as dummy implementations to build the initial framework.

I worked with team members on revisions to the interfaces as needed, and worked with each team member to integrate their code into the system.

On the GUI server side, I used the `dynamic_reconfigure` ROS package to build a GUI which sent ROS messages to the Arduino using rosserial. On the GUI client side,

I implemented ROS subscribers with callbacks to update the system state in respose to GUI events (e.g. adjusting the position setpoint for the stepper motor).

I also implemented ROS publishers using rosserial on the Arduino side to send the current sensor state to the server. These could be visualized using the `rqt_plot` ROS package.

0.1.2 Challenges

We faced a number of challenges. Firstly, when we set software sprint goals for the sprint starting January 27th and ending Feburary 10th, we underestimated the amount of work the motors and sensors lab would require. This led directly to a difficulty balancing the time requirement of the team project with the lab. In an attempt to meet both sets of goals, I settled for a less comprehensive set of interfaces than I would have liked, which led to a poorly designed architecture. This allowed a sleep-deprived team to stumble into a number of bugs during integration, such as forgetting to call an appropriate `init_dc_motor` function before attempting to use the motor functions. This would have been avoided with an object-oriented design patter, in which, it would be impossible to call methods on the motor without first calling the constructor.

We also interpreted the design document to require a number of features which we later realized the course instructors did not care about, such as the motors and sensors running in parallel. We could have avoided a great deal of effort spent trying to get the sensors and motors to run in parallel without conflicts if we had realized this was not a requirement earlier. In particular, the temperature sensor sampling function used originally (provided by the manufacturer) caused a delay of nearly one second. This caused running this sensor cause problems with other components, most notably the ultrasound sensor which used a sliding window average. Aman had to switch this sensor to analog and determine his own transfer function in order to make this sensor work in parallel, additional effort which could have been saved if requirements were better understood initially and we had designed the system appropriately.

ROS, especially rosserial, was used in an attempt to make the lab work more relevant to our project, which in theory would help us gain relevant experience to further our progress on the team project. While ROS itself was not an issue, rosserial in particular added additional work since every topic which we published or subscribed to required an additional callback function. In addition, rosserial added complexity and unintuitive behaviors, such as resetting the Arduino when it failed to communicate with the server.

A simple space separated and newline terminated list of values sent between the server and client probably would have been just as effective. In retrospect, attempting to make the lab relevant to our team project actually decreased the amount of time available for the project, and simply trying to get the lab completed sooner so that we could return to team project work would have been wiser.

0.1.3 Teamwork

The work was divided as follows:

1. John: Integration & GUI
2. Aman: DC motor & temperature sensor
3. Aaditya: DC motor & IR sensor
4. Hillel: Servo & slot sensor
5. Dung-Han: Stepper & ultrasound sensor

0.2 Team Project

0.2.1 Individual Progress

0.2.1.1 Project Management

Firstly, in my role as the software team manager, I decided that using Agile project management was an absolute necessity in order to make progress under the uncertainty associated with what is essentially a new product development program. We began our first software spring on January 27th, and completed it on February 10th. While I aimed to use Jira as a project management tool, skepticism from the team (which has never used Jira), resulted in us using GitHub Projects, shown in Figure 2 as an alternative for now. We aimed to individually evaluate tools such as Jira or Trello in the meantime.

While the mechanical team consisting of Hillel and Aman was resistant to introducing Agile as well, we agreed on having standup twice weekly on Mondays and Thursdays in order to facilitate communication and ensure continued progress. Aman joined us for our first software sprint, which I believe was very good for team communication. I hope to see the workflows of the mechanical and software teams merge over time.

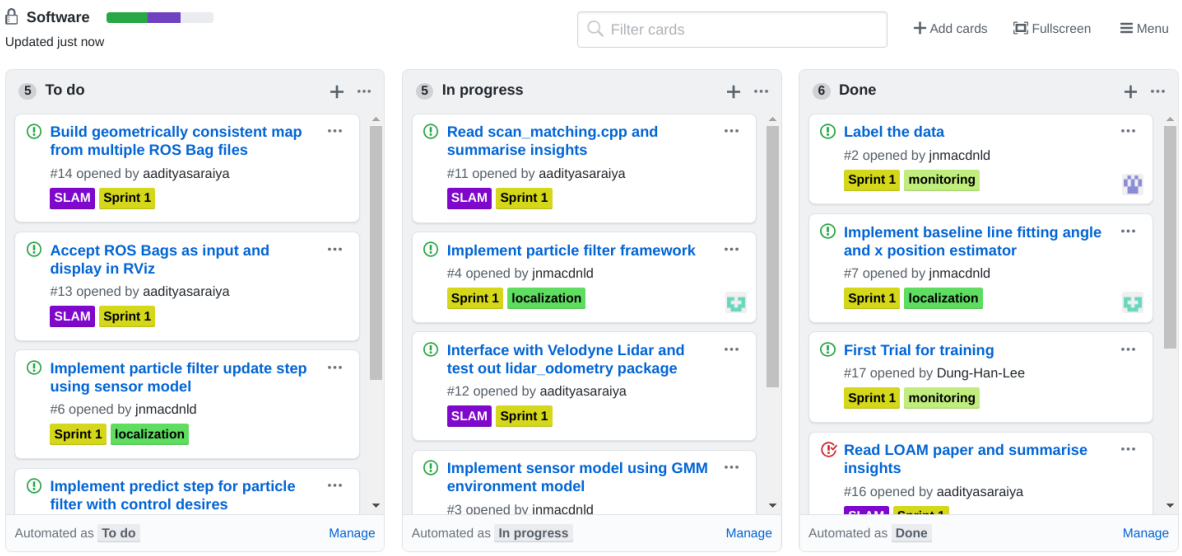


Figure 2: The software team's Kanban board on GitHub.

0.2.1.2 Engineering

My work focused on the navigation subsystem, depicted in Figure 3. I am responsible for processing a fully registered pointcloud, outputted from Aaditya's SLAM system, and generating a simplified map representation. Additionally, I am responsible for then having the robot localize within that map at run-time. This will be utilized by the planning controls subsystem in order to complete a full navigation stack.

My first step was to analyze the data which we collected from Rivendale Farms, our industry co-sponsor. I decided to focus on what was possible with the laser scan data. Since we have limited access to the farm to collect large datasets, it is important that we develop subsystems which can perform well under many environmental conditions, even if we have not encountered them in the dataset. The lidar offers environmental invariance to sunlight which other sensors, for example cameras, do not. I explored ways in which the ground and plants could be segmented into ground points and plant points. I found RANSAC plane fitting effective at this task, with results shown in Figure 4.

One requirement of our mapping system is that it must produce a map which is usable as the plants grown over time. I therefore explored ways in which the pointcloud could be reduced to a map representation which would be invariant to these changes. I wrote a script to rotate and translate pointclouds into a consistent global reference frame using the recorded RTK GPS and IMU data. The result is shown in Figure 5. I then fit a line to the plant segments. In this case, the intended map representation would be a set of

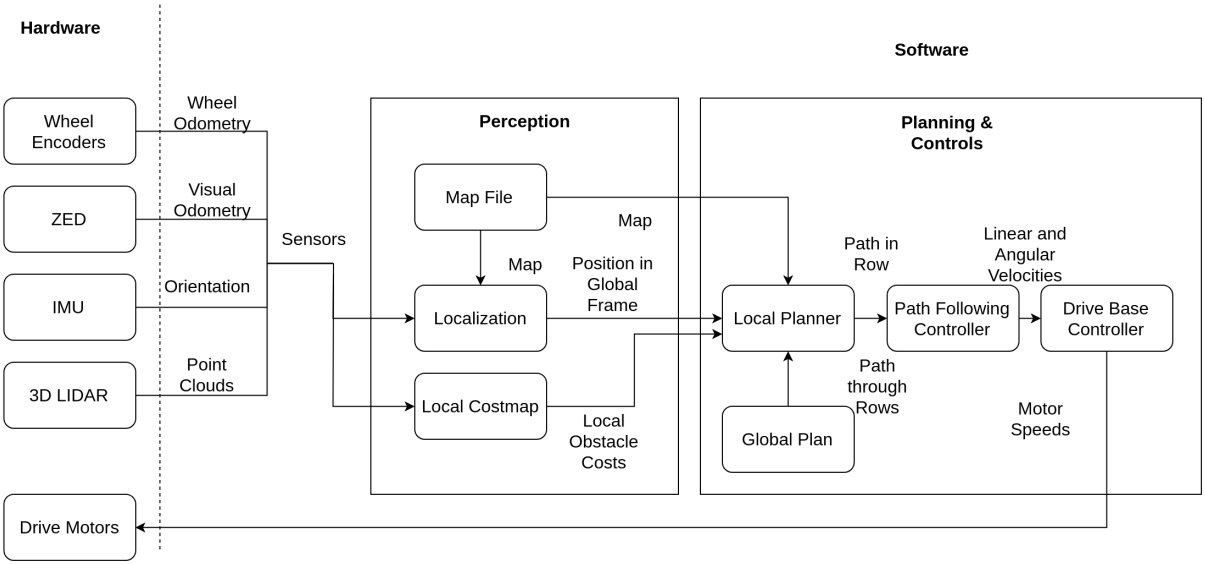


Figure 3: Block diagram of the navigation subsystem. I am responsible for the Perception group of blocks.

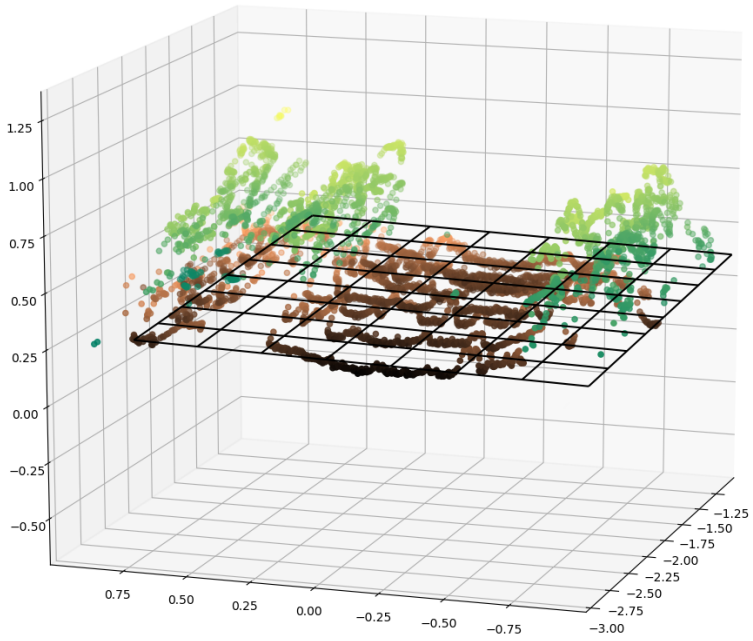


Figure 4: Segmentation of ground and plants with RANSAC.

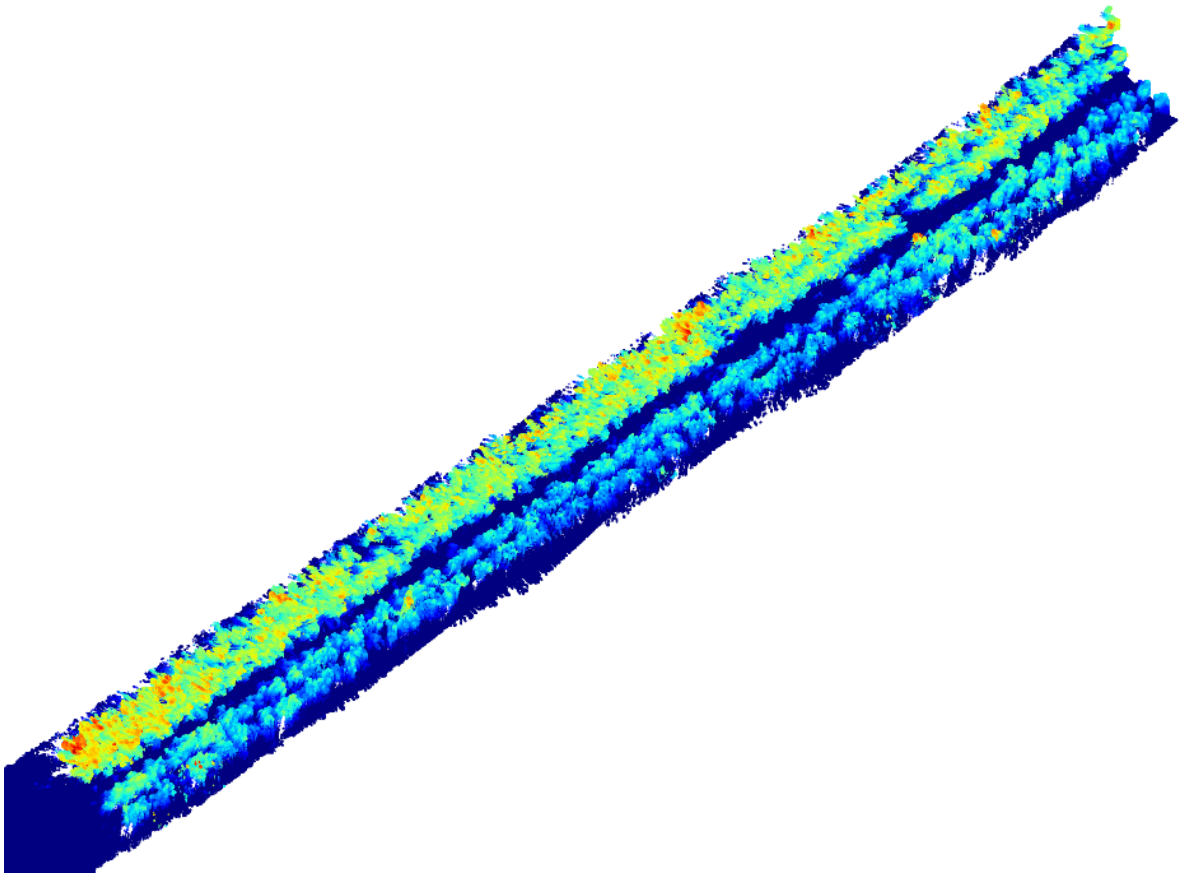


Figure 5: Pointcloud of one row of brassica at rivendale farms, top-down view, generated from IMU and RTK GPS data.

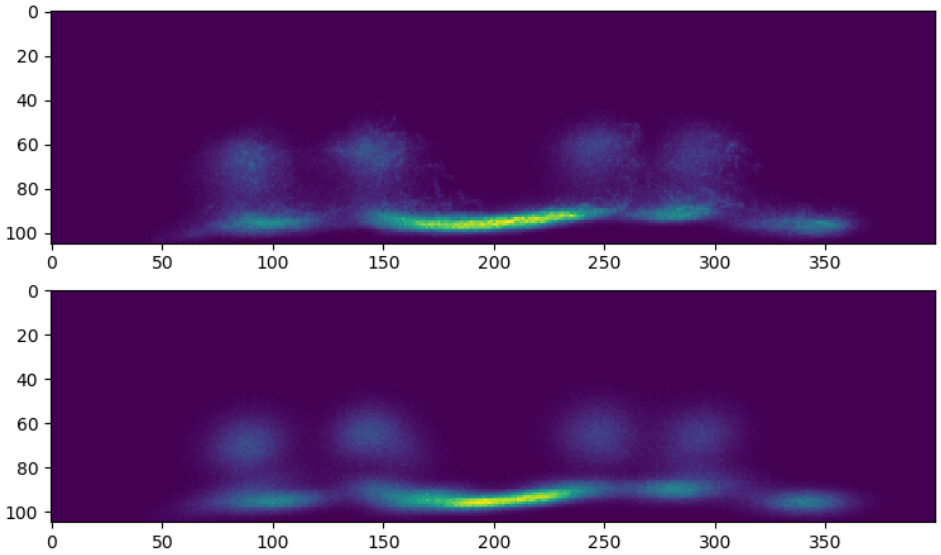


Figure 6: Top: Histogram of laser points looking down the row of plants (into the page is walking forward in the row, the y axis is up, and the x axis is lateral movement between rows). Bottom: Histogram of 1e6 samples from generative gaussian mixture model with 8 components fit to the row distrbution data.

lines representing rows, and the robot would have to determine the center line of the row it is currently in order to localize in real-time. This estimate of the robot position could then be fused with e.g. odometery and IMU data in order to provide a state estimate.

I therefore then explored ways in which the laser scans could be used to determine the center line of the current row. When visualizing the sensor data, looking down the row, I recognized that the distribution of points seemed to approximiate a mixture of gaussians. I thought that it could be possible to fit a gaussian mixture model to the data in order to create a generative model. I could then use that model to compute a sensor model providing $P(z|x)$ for each sensor reading. By plugging that into a particle filter, I could localize the robot. I implemented such a mixture model, shown in Figure 6. The generative model does quite a good job of approximating the distribution of sensor readings, which was super exciting and very encouraging!

0.2.2 Challenges

However, utilizing my generative model in order to localize the robot has proven to be challenging. I first decided to compute the probability $P(z|x)$ with the knowledge

of the angle and azimuth of the laser marginalized out i.e. without incorporating the knowledge that each reading must lie on a particular line. My first issue related to computing the joint probability of all the sensor measurements (i.e. laser scan points). Multiplying thousands of numbers less than 1 together naturally causes the result to go to zero quite quickly; this was solved by changing to computing $log(P(z|x))$ instead, whose individual components can be summed instead of multiplied. However, the output of this function is strange. Where I would expect to see a maximum, I instead have a minimum. Furthermore, I do not have one single peak, in reality, there is a diagonal line through the (y,θ) space which has roughly equal probability (the y-axis translates between rows, not along rows.). I need to further investigate the source of these issues if I am to use this function.

Additionally, due to underestimating the amount of work in the motors sensors lab, I failed to reach my initial sprint goal of implementing the sensor model with a particle filter. This was disheartening and a bit embarrassing. I hope to meet my goals in the future in order to maintain good team culture and morale.

0.2.3 Teamwork

I mainly collaborated with Dung-Han Lee (DHL), who suggested to use log probability instead, thus solving my numerical instability issue; as well as Aaditya in my efforts to design an appropriate localization scheme. In particular, I am communicating closely with Aaditya in order to work out a suitable map representation and overall mapping navigation scheme. Aaditya is investigating lidar odometry-based approaches, which I find fascinating. DHL's inital results on object detection semantic segmentation have also been encouraging. Aaditya, DHL, and I have met with several members of George Kantor's lab as a team in order to gain insights from previous agricultural robotics projects with similar goals.

I have additionally discussed how sensors might optimally be mounted with Aman, who is working on the robot platform, and to a lesser extent with Hillel, who is working on the weeding mechanical subsystem.

0.2.4 Plans

I plan to implement a particle filter in the coming week and use this as a framework to compare a number of sensor models. In this way, I can select the best possible localization approach. I'm looking forward to an exciting semester of robotics!

0.3 Appendix A: Motors and Sensors Lab Code

0.3.1 motor_sensor.ino - Client

This was the file on the Arduino responsible for receiving commands from the GUI server and sending them to the appropriate modules, as well as sending sensor updates to the ROS server.

```
#include <ros.h>
#include <std_msgs/Bool.h>
#include <std_msgs/Float32.h>
#include <std_msgs/Int32.h>
#include <std_msgs/String.h>

#include "ultrasound.h"
#include "ir.h"
#include "temp.h"
#include "slot.h"
#include "dc_motor.h"
#include "stepper.h"
#include "servo.h"

volatile bool motors_enabled = true;
unsigned long last_motor_enable_time = 0;
int motors_enabled_button_pin = 20;

int slot_pin = 6;

// Debounce pararmeter
const unsigned long delay_time = 300;

void debounce(volatile unsigned long* last_change_time, void (*func)()) {
   unsigned long current_time = millis();
   unsigned long delta = current_time - *last_change_time;

   if ((current_time - *last_change_time) > delay_time) {
     (*func)();
     *last_change_time = current_time;
```

```
   }
}

void toggle_motors_enabled() {
  motors_enabled = !motors_enabled;
  ros_log("Toggle motor enable");
}

void toggle_motors_enabled_debounced() {
  debounce(&last_motor_enable_time, &toggle_motors_enabled);
}

ros::NodeHandle nh;

std_msgs::Float32 temp_msg;
ros::Publisher temp_pub("temp", &temp_msg);
float temp;

std_msgs::Float32 pid_out_msg;
ros::Publisher pid_out_pub("pid_out", &pid_out_msg);
float pid_out;

std_msgs::Float32 pid_in_msg;
ros::Publisher pid_in_pub("pid_in", &pid_in_msg);
float pid_in;

std_msgs::Float32 ultrasound_msg;
ros::Publisher ultrasound_pub("ultrasound_dist", &ultrasound_msg);
float ultrasound_dist;

std_msgs::Float32 ir_msg;
ros::Publisher ir_pub("ir_dist", &ir_msg);
float ir_dist;

std_msgs::Bool slot_msg;
ros::Publisher slot_pub("slot", &slot_msg);
bool slot;
```

```
std_msgs::Float32 dc_speed_msg;
ros::Publisher dc_speed_pub("dc_speed", &dc_speed_msg);

// Logging
std_msgs::String log_msg;
ros::Publisher log_pub("log", &log_msg);

void ros_log(const char* str) {
  log_msg.data = str;
  log_pub.publish(&log_msg);;
}

void ros_log(double num) {
  sprintf(log_msg.data, "%f", num);
  log_pub.publish(&log_msg);;
}

// Mode
enum Mode { GUI = 0, servo_ir = 1, stepper_ultrasonic=2, dc_vel = 3, dc_pos =
  4, dc_slot = 5, dc_temp = 6};
Mode mode = servo_ir;

// Mode subscriber
void mode_cb(const std_msgs::Int32 msg) {
  mode = msg.data;
  ros_log("Changed mode");
}
ros::Subscriber<std_msgs::Int32> mode_sub("mode", &mode_cb);

// Stepper motor
struct stepper_data stepper;
int stepper_gui_setpoint = 0;
void stepper_pos_setpoint_cb(const std_msgs::Int32 msg) {
  ros_log("got stepper request");
  stepper_gui_setpoint = msg.data;
}
```

```
ros::Subscriber<std_msgs::Int32>
    stepper_pos_setpoint_sub("stepper_pos_setpoint",
  &stepper_pos_setpoint_cb);

// Servo motor
struct servo_data servo;
int servo_gui_setpoint = 0;
void servo_pos_setpoint_cb(const std_msgs::Int32 msg) {
  servo_gui_setpoint = msg.data;
  ros_log("got servo request");
}
ros::Subscriber<std_msgs::Int32> servo_pos_setpoint_sub("servo_pos_setpoint",
  &servo_pos_setpoint_cb);

// DC motor
struct dc_motor_data dc_motor;

// DC motor mode subscriber
void dc_motor_mode_sub(const std_msgs::Int32 msg) {
  if (msg.data == 0 || msg.data == 1) {
    dc_motor.mode = msg.data;
  }
}

// DC motor setpoint
int dc_motor_gui_pos_setpoint = 0;
void dc_pos_setpoint_cb(const std_msgs::Int32 msg) {
  dc_motor_gui_pos_setpoint = msg.data;
  ros_log("got setpoint for pos dc motor");
}
ros::Subscriber<std_msgs::Int32> dc_pos_setpoint_sub("dc_motor_pos_setpoint",
&dc_pos_setpoint_cb);

// DC motor setpoint
int dc_motor_gui_vel_setpoint = 0;
void dc_vel_setpoint_cb(const std_msgs::Int32 msg) {
  dc_motor_gui_vel_setpoint = msg.data;
```

```
}
ros::Subscriber<std_msgs::Int32> dc_vel_setpoint_sub("dc_motor_vel_setpoint",
&dc_vel_setpoint_cb);

void setup()
{
  /* Serial.begin(9600); */
  /* Serial.println("begin"); */

  pinMode(motors_enabled_button_pin, INPUT);

  nh.initNode();

  nh.advertise(log_pub);
  nh.advertise(temp_pub);
  nh.advertise(ir_pub);
  nh.advertise(ultrasound_pub);
  nh.advertise(slot_pub);

  nh.advertise(pid_out_pub);
  nh.advertise(pid_in_pub);
  nh.advertise(dc_speed_pub);

  nh.subscribe(mode_sub);
  nh.subscribe(dc_pos_setpoint_sub);
  nh.subscribe(dc_vel_setpoint_sub);
  nh.subscribe(dc_pos_setpoint_sub);
  nh.subscribe(dc_vel_setpoint_sub);
  nh.subscribe(stepper_pos_setpoint_sub);
  nh.subscribe(servo_pos_setpoint_sub);

  // Sensor initialization
  /* init_ultrasound(); */
  init_temp();
  init_ir_dist();
  init_slot(slot_pin);
  /*  */
```

```
  /*  */
  init_servo(&servo);
  init_stepper_motor();

  init_dc_motor(&dc_motor);
  attachInterrupt(digitalPinToInterrupt(motors_enabled_button_pin),
      toggle_motors_enabled_debounced, RISING);
}

unsigned long last_send_time;

void loop() {
  /* Serial.println("loop"); */
  // Get sensor values
  temp = get_temp();
  ultrasound_dist = get_ultrasound_dist();
  ir_dist = get_ir_dist();
  slot = get_slot(slot_pin);

  // Publish data
  temp_msg.data = temp;
  temp_pub.publish(&temp_msg);

  ultrasound_msg.data = ultrasound_dist;
  ultrasound_pub.publish(&ultrasound_msg);

  ir_msg.data = ir_dist;
  ir_pub.publish(&ir_msg);

  slot_msg.data = slot;
  slot_pub.publish(&slot_msg);

  if (mode == GUI) {
    if (motors_enabled) {
      handle_servo_position(servo_gui_setpoint, &servo);
      stepper_GUI_control(stepper_gui_setpoint);
    }
```

```
} else if (mode == servo_ir) {
  // Control servo with IR
  int angle;
  if (ir_dist > 15.0) {angle = 180;}
  if (ir_dist < 15.0) {angle = 0;}
  handle_servo_position(angle, &servo);

  /* Serial.println("ir_dist:"); */
  /* Serial.println(ir_dist); */
} else if (mode == stepper_ultrasonic) {
  stepper_with_ultrasonic();
} else if (mode == dc_vel) {
  Setpoint = (float) dc_motor_gui_vel_setpoint;
  dc_motor.mode = velocity;
  handle_dc_motor(&dc_motor);

  pid_out_msg.data = (float) OutputNet;
  pid_out_pub.publish(&pid_out_msg);

  dc_speed_msg.data = (float) speedMot;
  dc_speed_pub.publish(&dc_speed_msg);

} else if (mode == dc_pos) {
  dc_motor.mode = position;
  SetpointPos = (float) dc_motor_gui_pos_setpoint * (756. / 360);
  ros_log("position mode");
  Setpoint = 0;
  handle_dc_motor(&dc_motor);

  dc_speed_msg.data = (float) speedMot;
  dc_speed_pub.publish(&dc_speed_msg);

  pid_in_msg.data = (float) InputPos;
  pid_in_pub.publish(&pid_in_msg);

  pid_out_msg.data = (float) OutputPos;
  pid_out_pub.publish(&pid_out_msg);
```

```
  } else if (mode == dc_slot) {
    if (slot) {
      SetpointPos = 0.;
    } else {
      SetpointPos = 180. * (756. / 360);
    }

    dc_motor.mode = position;
    handle_dc_motor(&dc_motor);
  } else if (mode == dc_temp) {
    Setpoint = (get_temp()-20) * 10.;
    dc_motor.mode = velocity;
    handle_dc_motor(&dc_motor);

    pid_in_msg.data = (float) Input;
    pid_in_pub.publish(&pid_in_msg);
  }

  delay(10);
  nh.spinOnce();
}
```

0.3.2 MotorSensor.cfg - Server

This file was used by `dynamic_reconfigure` to display the appropriate widgets in the GUI.

```
#!/usr/bin/env python

PACKAGE = "motor_sensor"

from dynamic_reconfigure.parameter_generator_catkin import *

gen = ParameterGenerator()

mode_enum = gen.enum([
    gen.const("GUI_control",    int_t, 0, "Motors controlled by GUI"),
```

```
    gen.const("Servo_IR", int_t, 1, "Servo controlled by IR "),
    gen.const("Stepper_Ultrasound", int_t, 2, "Stepper controlled by IR "),
    gen.const("DC_vel", int_t, 3, "DC motor controlled by velocity"),
    gen.const("DC_pos", int_t, 4, "DC motor controlled by position"),
    gen.const("DC_temp", int_t, 6, "DC motor controlled by position"),
    gen.const("DC_slot", int_t, 5, "DC motor controlled by slot")],
    "An enum to choose operation mode")

gen.add("mode", int_t, 0, "Mode of operation which is edited via an enum", 1,
    0, 6, edit_method=mode_enum)

control_enum = gen.enum( [
    gen.const("Position_control",  int_t, 0, "Position control"),
    gen.const("Velocity_control", int_t, 1, "Velocity control")],
"An enum to choose control method")

gen.add("dc_motor_pos_setpoint", int_t, 0, "Position setpoint for dc motor", 0,
    0, 359)
gen.add("dc_motor_vel_setpoint", int_t, 0, "Velocity setpoint for dc motor",
    100, 0, 150)

gen.add("stepper_pos_setpoint", int_t, 0, "Position setpoint for stepper", 0,
    0, 359)
gen.add("servo_pos_setpoint", int_t, 0, "Position setpoint for servo", 0, 0,
    180)

exit(gen.generate(PACKAGE, "motor_sensor", "MotorSensor"))
```

0.3.3 server.py - Server

```
#!/usr/bin/env python

import rospy
from dynamic_reconfigure.server import Server
from motor_sensor.cfg import MotorSensorConfig
from std_msgs.msg import Int32, Float32
```

```
class CfgServer(object):
    def __init__(self):
        rospy.init_node("motor_sensor", anonymous = False)

        topics = [
            (Float32, 'dc_motor_kP_vel'),
            (Int32, 'stepper_pos_setpoint'),
            (Int32, 'dc_motor_pos_setpoint'),
            (Int32, 'servo_pos_setpoint'),
            (Float32, 'dc_motor_kD_vel'),
            (Int32, 'dc_motor_vel_setpoint'),
            (Float32, 'dc_motor_kP_pos'),
            (Float32, 'dc_motor_kI_pos'),
            (Float32, 'dc_motor_kD_pos'),
            (Int32, 'mode'),
            (Int32, 'dc_motor_mode'),
        ]

        self.pubs = {}
        for typ, topic in topics:
            self.pubs[topic] = rospy.Publisher(topic, typ, queue_size=2)

        srv = Server(MotorSensorConfig, self.callback)

    def run(self):
        rospy.spin()

    def callback(self, config, level):
        for param in config:
            if param in self.pubs.keys():
                self.pubs[param].publish(config[param])

        return config

if __name__ == "__main__":
    s = CfgServer()
```

```
s.run()
```

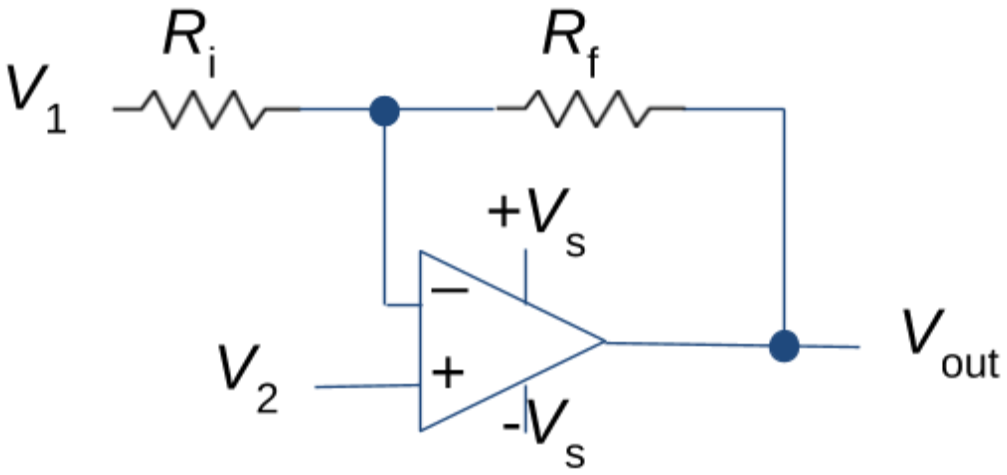
0.4 Appendix B: Quiz

Task 4 (Sensors and Motor Control Lab) Quiz

1. Reading a datasheet. Refer to the ADXL335 accelerometer datasheet (https://www.sparkfun.com/datasheets/Components/SMD/adxl335.pdf) to answer the below questions.
   - What is the sensor's range?
     - The typical range is rated as +/- 3.6g and the minimum is rated as +/- 3g.
   - What is the sensor's dynamic range?
     - DR = max / (noise floor). Since the noise floor is not listed, an experiment is needed first in order to determine it.
   - 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?
     - The capacitor acts as a low pass filter which reduces noise in the circuit. This is accomplished though the time delay which is required to charge the capacitor. If there is a spike in the signal, the capacitor will act as a damper, which will not immediately react as time is required to charge the capacitor.
   - Write an equation for the sensor's transfer function.
     - 0.3 * accel + 1.5 = voltage
     - 0.3 * accel = voltage - 1.5
     - accel = (voltage - 1.5) / 0.3
     - accel = 3.3 * voltage - 5
   - What is the largest expected nonlinearity error in g?
     - 0.3% * full_scale = 0.003 * (3.6 * 2) = 0.216
   - How much noise do you expect in the X- and Y-axis sensor signals when the sensor is excited at 25 Hz?
     - The RMSE is given by *noise_density * sqrt(BW * 1.6) = 150 * sqrt(25 * 1.6) = 948.683 µg*
   - How about at 0 Hz? If you can't get this from the datasheet, how would you determine it experimentally?
     - I could set the sensor on a tabletop, with the Z axis pointing down, at which point the X and Y accelerations should both be zero. I would sample the sensor over some time interval, use the transfer function to convert the voltage to acceleration, and then compute the mean squared error of these samples given the known accelerations.

2. Signal conditioning
   - Filtering
     - What problem(s) might you have in applying a moving average?
       - A moving average will introduce time delay to the signal.
     - What problem(s) might you have in applying a median filter?
       - A median filter will typically not produce a smooth output signal; this could be an issue depending on the application.
   - 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: 1) which of V1 and V2 will be the input voltage and which the reference voltage; 2) the value of the reference voltage; and 3) the value of Rf/Ri in each case. If the calibration can't be done with this circuit, explain why.
  - Your uncalibrated sensor has a range of -1.5 to 1.0V (-1.5V should give a 0V output and 1.0V should give a 5V output).
    - 1) V1 = Vref, V2 = Vin
    - 2) Vref = -3 V
    - 3) Rf/Ri = 1
  -



  - Your uncalibrated sensor has a range of -2.5 to 2.5V (-2.5V should give a 0V output and 2.5V should give a 5V output).
    - Because the range of the input and output voltage is the same, the gain must be 1, however, this makes computing the offset impossible. Therefore this is not possible.

Fig. 1 Opamp gain and offset circuit

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.
    - I would use a an encoder which will produce a digital square wave as the motor turns. I would connect the encoder to an interrupt pin on a microcontroller, and set the interrupt to increment a counter. By counting the encoder ticks, I can determine the position of the motor by dividing the count by number of encoder ticks by the ticks per rotation. I would then have PID controller loop which computes each of the terms. By subtracting the current position of the motor from the desired position, I can compute error, which I would multiply by kP in order to get the P term. I will keep a running total of the errors stored in an integral variable, which I will multiply by kI to get the integral term. Finally, I will compute the change in error for every iteration and multiply this by kD to get the D term.
- If the system you want to control is sluggish, which PID term(s) will you use and why?
    - I would increase the P term, as the P term is directly proportional to error, and will therefore result in a higher response when the measured signal deviates from the setpoint.
- 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?
    - I will increase the integral term, because if there is error in the steady state, this will cause the integral to rise, which will cause the integral term to rise in response to drive the error 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?
    - I would increase the D term, because since the D term is proportional to the derivative of error, the D term will act as a damper to reduce the rate of change of the output, therefore slowing it down as it reaches the setpoint.