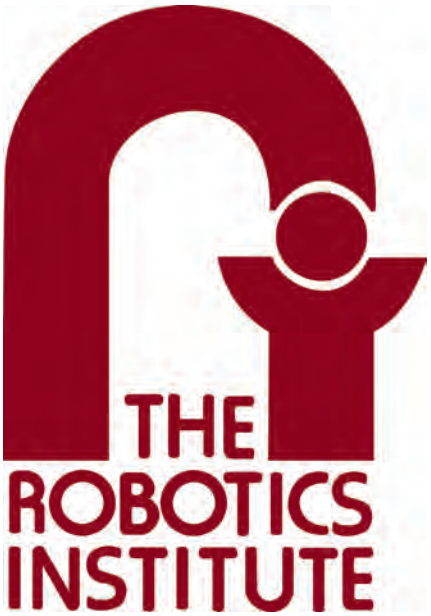

MRSD Project Course

Team I – AIce

Autonomous Zamboni Convoy

Individual Lab Report 1

Team

Rathin Shah

Nick Carcione

Yilin Cai

Jiayi Qiu

Kelvin Shen

Author

Jiayi Qiu

Feb 9, 2022

Contents

1 Individual Progress

1.1 Sensor and Motor Lab

In this sensor and motor lab, I designed a circuit and developed a microcontroller program for a FlexiForce. The FlexiForce sensor serve as a force sensing resistor. Its resistance varies based on the force apply to it. I did experiments to find the transfer function. The output can be transferred into physical unit N instead of volts using this transfer function. I wrote code to control a stepper motor using the physical outputs of the sensor. I integrated the code for FlexiForce sensor, ultrasonic rangefinder sensor, and the stepper motor. With the integrated code, the FlexiForce sensor and the ultrasonic rangefinder sensor can control the rotation of the stepper motor. I used a button to switch which sensor to control the stepper motor. I also applied filter functions to filer the inputs for both the FlexiForce sensor and the ultrasonic rangefinder sensor.

1.1.1 Sensor

The force sensor I used in this lab is a FlexiForce standard model A201. It is a force sensing resistor. When the force sensor is unloaded, its resistance is extremely large. When force is applied to this sensor, its resistance is reduced. The sensor I used has a force range (0N,4.4N). I designed a voltage divider circuit using a resistor (R=10KΩ) in series with the force sensor. The circuit is shown in Figure 1. The circuit divides the 5V between the force sensor and the resistor. I measure the voltage change of the force sensor using one of the Arduino's analog inputs. The analog read is 1023 at 5V. At 0V, it will read 0.

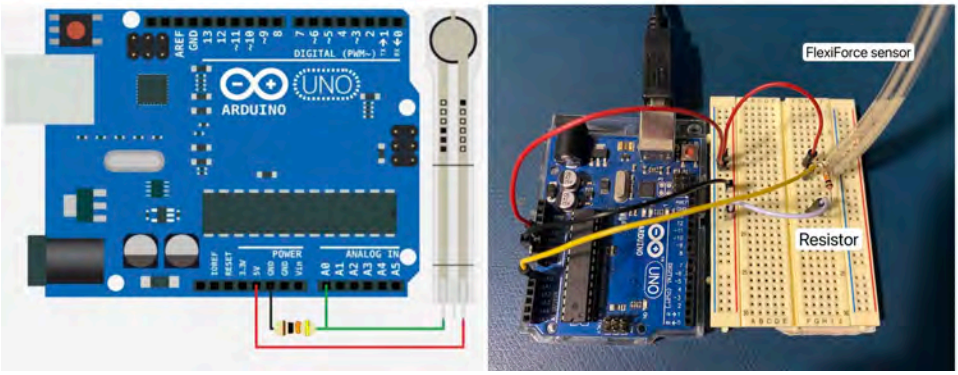

Figure 1: The voltage divider circuit of the FlexiForce sensor

I did experiments by applying different forces on the sensor and record the analog reads. Then, I found a fitting function using Python. The plot I generated that displays the transformation between analog reads and the forces is shown in Figure 2. The transfer function is: $y = 0.0315x$.

1.1.2 Motor

The motor I used in this lab is a stepper motor. The stepper motor moves 200 steps per revolution. I used the output of the force sensor to determine the desired movement of the motor. The force range of the sensor is (0N,4.4N). The movement range of the stepper motor is 0 to 200 steps, which means 0 to 360 degrees. Therefore, I mapped the force sensor outputs to movement steps. If the desired steps represent a larger angular position than the current position, the motor will rotate in clockwise to reach the desired position. If the desired steps result a smaller angular

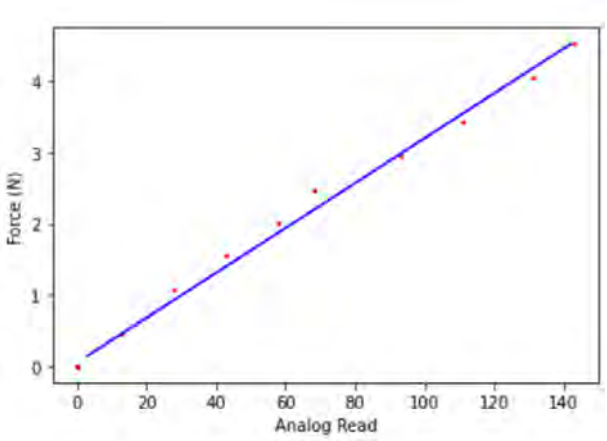

Figure 2: The transfer function plot of the FlexiForce sensor

position than the current position, the motor will rotate in counterclockwise to reach the goal position.

1.1.3 Circuit and Code

In this lab, my team decided to use the FlexiForce sensor and ultrasonic rangefinder sensor to control the stepper motor. I integrated the code for FlexiForce sensor, ultrasonic rangefinder sensor, and the stepper motor. A button was designed to determine which sensor was selected to control the motor. I applied Moving average filters to the inputs of each sensor to reduce the noise. The integrated circuit is shown in Figure 3. The code I contributed to is shown in Appendix A.

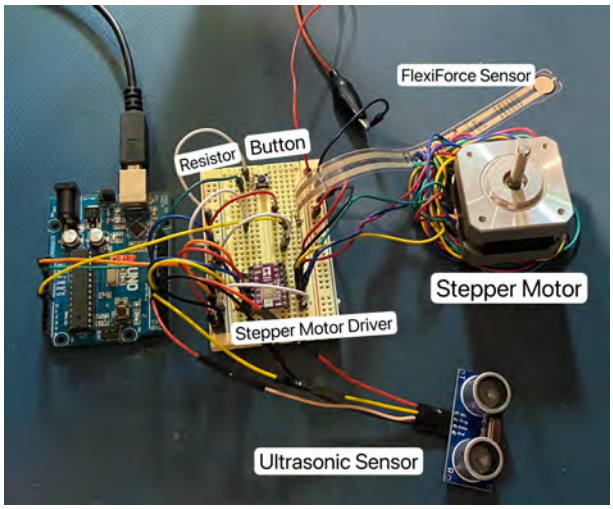

Figure 3: The integrated circuit for FlexiForce sensor, ultrasonic rangefinder sensor, and the stepper motor

1.2 MRSD Project

In the last few weeks, I studied how to simulate an Ackermann vehicle in Gazebo and learned simulation structures. I modified the mesh file of Zamboni links and helped to setup a Zamboni in the Gazebo. I also built the ice rink simulation environment. The ice rink model is in true size and has curved corners. The plane ground has the same patterns as a real ice rink. Currently, I am working on leader trajectory tracking of the follower vehicle. Based on the relative position and

the relative heading angle between leader and follower, a simple PID controller can be applied to maintain the desired distance.

2 Challenges

2.1 Sensor and Motor Lab

The challenge I faced in my own part was to transfer the force sensor data so that the output is in physical unit N. When using the FlexiForce sensor, I had to make sure all weight I applied to the sensor is directed onto the small sensing area. Otherwise, I could not determine how much force I added on the sensor. Therefore, I placed the small sensing area on an electronic digital scale with a small metal cylinder on the top. The cross section of the metal cylinder was completely placed inside the sensing area. Then, I added different numbers of metal pieces on it. In this way, the force I applied to the sensor was shown on the electronic digital scale.

2.2 MRSD Project

I had problems with models when I tried to build the simulation environment. We generated COLLADA files (.dae) of Zamboni based on the Zamboni Solidworks model. These files need to be used in the Zamboni URDF model for simulation. However, due to extremely large number of faces and vertices, these COLLADA files were not small enough for Gazebo simulation. Gazebo got stuck when loading the URDF model. Finally, I used the decimate modifier in Blender to reduce the number of faces and vertices. The appearance of the final model was well maintained while its size was greatly reduced. It also challenged me when building the ice rink world environment. It took me a while to build an ice rink model in true size with curved corners in Solidworks, wrote sdf files for this model, and wrote a world file for simulation.

3 Team Work

3.1 Sensor and Motor Lab

Each team member's distributions are shown below:

- Rathin: Designed the circuit and code for a potentiometer to control a servo motor. Wired a button to switch between GUI control and sensor control.

- Nick: Designed the circuit and code for a IR sensor to control a DC motor. Wrote PID code for the DC motor. Wrote button code to switch between position control and velocity control. Integrated the final circuit.

- Jiayi: Designed a FlexiForce sensor circuit. Wrote code for the FlexiForce sensor to control a stepper motor. Integrated code for FlexiForce sensor, Ultrasonic sensor and stepper motor. Wrote button code to switch between these two sensors to control the stepper motor. Wrote the moving average filter.

- Yilin: Soldered stepper motor driver. Designed the circuit and code for a ultrasonic sensor to control a stepper motor. Adjusted the current of the stepper motor driver. Integrated the final circuit.

- Kelvin: Developed a GUI program using ROS. Created custom messages for sensor data outputs. Programmed to Control the motor outputs through joint states publisher. Integrated code for sensors and motors with GUI program. Debugged the final code.

The final circuit is shown in Figure 4.

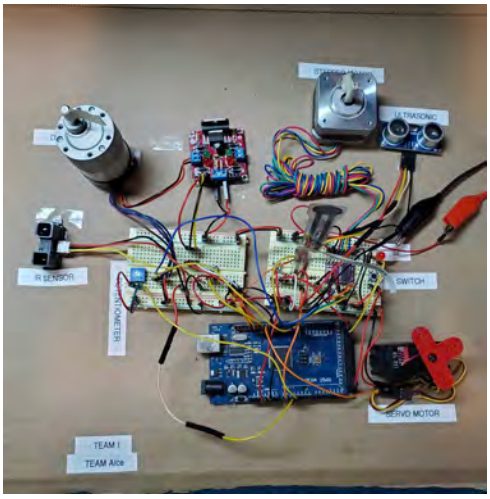

Figure 4: Final circuit

3.2 MRSD Project

Each team member's distributions are shown below:

- Rathin: Developed the Pure Pursuit Controller on Simulink for Ackermann Geometry; began developing package for fusing IMU + Wheel Odometry for zamboni using ros; began working on developing the SIMULINK-ROS Interface.

- Nick: Researched methods and packages for fusing wheel encoder and IMU data to obtain accurate velocity estimation of follower; began looking at DBW conversion hardware.

- Yilin: Built up the simulation environment;created URDF (XACRO) file of the Ackermann steering Zamboni with sensor and controller plugins; simplified mesh files and colorized them for better visualization; realized vehicle motion command with keyboard teleoperation; realized multi-robot spawn in Gazebo and individually control with keyboard teleoperation; completed Tf tree setup, odometer frame setup and visualization in Rviz.

- Kelvin: Learned cv bridge that communicates between ROS camera topic and OpenCV; generated a mesh file of a board of ArUco markers with appropriate size; tested the ArUco wall with Zamboni model inside the Gazebo environment.

- Jiayi: Investigated vehicle simulation in Gazebo; simplified COLLADA files of Zamboni model using Blender; helped to simulate the Zamboni; built the ice rink simulation environment.

4 Plan

In the next few weeks, we plan to work on sensor fusion for localization of follower in the world frame. We are going to detect the leader and estimate its pose based on the board of ArUco markers. I plan to complete the leader trajectory tracking algorithm and generate follower velocity profile. I will also work on ROS node and integrate the algorithm with simulation. We need to work on waypoint estimation and smooth path generation with zamboni kinematic constraints. We will also achieve the communication between ROS Master and Matlab for Zamboni Curvature/Steering Controller based on pure pursuit. What's more, we plan to start working on hardware as soon as possible. We need to detect the moving leaderZamboni using RealSense Camera mounted on follower vehicle in the ice rink and get the waypoints in world frame.

5 Quiz

1.

(a) The range is ±3g (minimum), ±3.6g (typical).

(b) The dynamic range is 6g (minimum), 7.2g (typical).

(c) The purpose of the capacitor is to reduce the input voltage noise. It does this by filtering out high-frequency noises from the power supply. The capacitor discharges when the voltage drops and charges when the voltage increases to make the voltage stable.

(d) $V_{out} = 1.5V + (300mV/g) * a$

(e) $0.3\% * 7.2g = 0.0216g$

The largest expected nonlinearity error in g is 0.0216g.

(f) Bandwidths can be selected to suit the application, with a range of 0.5 Hz to 1600 Hz for the X and Y axes.

(g) rms Noise = Noise Density × $(\sqrt{(BW{\times}1.6)}) = 150{\times}(\sqrt{(25{\times}1.6)}) = 948.68g$

(h) Assume that there is no noise from the power supply. The RMS noise can be determined by placing the accelerometer on a static surface and recording its readings over a very long duration. The RMS noise can be calculated with equation $\sqrt{\frac{1}{n}\sum_i z_i^2}$, where z_i is the reading.

2.

(a)

- Moving average filter:

1) Moving average filter is sensitive to individual large jumps (outliers). An occasional large number may greatly influence the result.

2) It averages last n values of the sensor inputs. If n is large, there will be a lag behind the instantaneous data.

- Median filter:

1) Median filter has high computational complexity, because it needs to sort each n-sized window and take the median value.

2) The median value of the n-sized window can be an outlier itself.

(b)

- 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) v_2 is the input voltage and v_1 is the reference voltage.

2)

$$v_{out} = (v_2 - v_1)\frac{R_f}{R_i} + v_2$$
$$0 = -1.5(1 + \frac{R_f}{R_i}) - v_{ref}(\frac{R_f}{R_i})$$
$$5 = 1(1 + \frac{R_f}{R_i}) - v_{ref}(\frac{R_f}{R_i})$$
$$\rightarrow \frac{R_f}{R_i} = 1, v_{ref} = -3V$$

- 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):

1) The circuit has no solution.

2) If v_1 is the input voltage and v_2 is the reference voltage:

$$v_{out} = -v_{in}(\frac{R_f}{R_i}) + v_{ref}(1 + \frac{R_f}{R_i})$$
$$0 = 2.5(\frac{R_f}{R_i}) + v_{ref}(1 + \frac{R_f}{R_i})$$

$5 = -2.5(\frac{R_f}{R_i}) + v_{ref}(1 + \frac{R_f}{R_i})$

There is no solution.

If v_2 is the input voltage and v_1 is the reference voltage:

$0 = -2.5(1 + \frac{R_f}{R_i}) - v_ref(\frac{R_f}{R_i})$

$5 = 2.5(1 + \frac{R_f}{R_i}) - v_ref(\frac{R_f}{R_i})$

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

It's not possible to calibrate. Thus, the circuit has no solution.

3.

(a) The result of current encoder output minus the desired position can be used for the proportional terms.

The integral terms can be formed by accumulating the error between the current encoder position and its desired position multiplied this by the timestep.

The derivative term can be formed by dividing the result of previous position minus the current position by the time step.

(b) I will increase the proportional term. Increase the proportional gain, the rise time can be decreased. The desired position can be reached in a shorter time.

(c) I will increase the integral term. This will compensate for the accumulated errors and make the steady state error closer to zero.

(d) I will apply the derivative term. This term will counteract the rate of change of the error. It can add damping to the system. Therefore, it can reduce the overshoot.

A Appendix

```
#define window_size 5

// Flexiforce Sensor variables
int flexiForcePin = A0;

// Ultrasonic Sensor variables
const int TrigPin = 7;
const int EchoPin = 13;
float distanceCm;
int duration;

// Stepper Motor variables
const int stepperEnable = 6;
const int stepperStep = 5;
const int stepperDir = 4;
const int stepsPerRevolution =200;
int currentStep;

// Button variables
const int button = 0;
int buttonState;
int buttonState_last = 0;
int mode_count = -1;
int mode=-1;

unsigned long lastDebounceTime = 0;
unsigned long debounceDelay = 50;


// Filter function
int sensorReadings[window_size];
int sum = 0;
int index = 0;
int RunningAverage_filter(int reading){
  sum -= sensorReadings[index];
  sensorReadings[index] = reading;
  sum += sensorReadings[index];
  index = (index+1) % window_size;
  return sum/window_size;
}

// Ultrasonic Sensor function
float Ultrasonic(){
  digitalWrite(TrigPin, LOW);
  delayMicroseconds(2);
```

```
  digitalWrite(TrigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(TrigPin, LOW);

  duration = pulseIn(EchoPin, HIGH);
  int filtered_duration = RunningAverage_filter(duration);
  distanceCm = filtered_duration / 58.0;
  distanceCm = (int(distanceCm * 100.0)) / 100.0;

  Serial.print("Distance:");
  Serial.print(distanceCm);
  Serial.print("cm");
  Serial.println();
  delay(10);
  return distanceCm;
}

// Flexiforce Sensor function
float Flexiforce(){
  int flexiForceReading = analogRead(flexiForcePin);
  int filtered_FFReading = RunningAverage_filter(flexiForceReading);

  float force = 0.0315*filtered_FFReading;
  Serial.print("Flexi Force sensor: ");
  Serial.print(force);
  Serial.println("N");
  return force;
}

// Stepper Motor function
void stepper(int desiredStep){
  if(desiredStep > currentStep) digitalWrite(stepperDir,HIGH);
  else digitalWrite(stepperDir,LOW);

  for(int x = 0; x < abs(desiredStep-currentStep); x++){
  digitalWrite(stepperStep, HIGH);
  delayMicroseconds(1000);
  digitalWrite(stepperStep, LOW);
  delayMicroseconds(1000);
  }
  currentStep = desiredStep;
//return;
}

void setup(){
  Serial.begin(9600);
  pinMode(TrigPin, OUTPUT);
  pinMode(EchoPin, INPUT);
```

```
  pinMode(stepperEnable,OUTPUT); // Enable
  pinMode(stepperStep,OUTPUT); // Step
  pinMode(stepperDir,OUTPUT); // Dir
  digitalWrite(stepperEnable,LOW); // Set Enable low
  pinMode(button, INPUT);
}

void loop(){
  Serial.println(mode);
  buttonState = digitalRead(button);
  if ((buttonState == HIGH) && (buttonState_last == LOW)){
    mode_count ++;
    mode = mode_count % 2;
    if (mode ==0){
      Serial.println("Change to Ultrasonic Sensor");
    }
    if (mode ==1){
      Serial.println("Change to Flexiforce Sensor");
    }

    sum = 0;
    index = 0;
    memset(sensorReadings, 0, sizeof(sensorReadings));
  }

  // debouncing
  if (buttonState != buttonState_last ){
    delay(50);
  }
  // update the last state of Button 0
  buttonState_last = buttonState;

  // Ultrasonic Sensor
  if (mode==0){
    float distanceCm = Ultrasonic();
    int desiredStep = round(map(distanceCm, 0, 100, 0, stepsPerRevolution
    stepper(desiredStep);
  }
  if (mode==1) {
    float force = Flexiforce();
    int desiredStep = round(map(force, 0, 4.4, 0, 200));
    stepper(desiredStep);
  }

  delay(100);
}
```