TASK 4: SENSORS AND MOTOR CONTROL LAB

16-681 MRSD Project 1 (Spring 2021)
Carnegie Mellon University

BY: Feng Xiang[*]
DUE: Thursday, February 25, 2021 11:59 PM

Notes

- N/A

[*]Compiled on Thursday 25th February, 2021 at 21:22

Contents

1 Sensors and Motors Lab **3**
1.1 Individual Progress 3

2 CoBorg **5**
2.1 Individual Progress 5
2.2 Challenges 7
2.3 Future Plans 8

3 Sensor and Motor Control Quiz **9**
3.1 Question 01 9
3.2 Question 02 10
3.3 Question 03 12

4 Appendix **13**
4.1 Sensors and Motor Lab: Written Code 13

1 Sensors and Motors Lab

1.1 Individual Progress

My tasks and responsibilities on the sensors and motor lab assignment included pulling analog data from the potentiometer and tying those analog readings to servomotor outputs, helping tie the potentiometer and flex sensor with the output on the servomotor, and helping integrate code on the Arduino-only section of the program. By helping others, I was in colloboration with other teammates as we worked to develop the respective components of the assignment.

Acquiring readings from the potentiometer included reading the analog output from the middle pin of the potentiometer into the Arduino. One of the two pins were tied to ground and the third pin was tied to 5V. Connecting the servomotor to the Arduino was very similar, where the black and red pins were connected to the 5V power supply and the third pin was connected to a PWM terminal on the Arduino board. The figure shown below (Figure 1.1) displays the pinout schematic of both the potentiometer and servomotors.

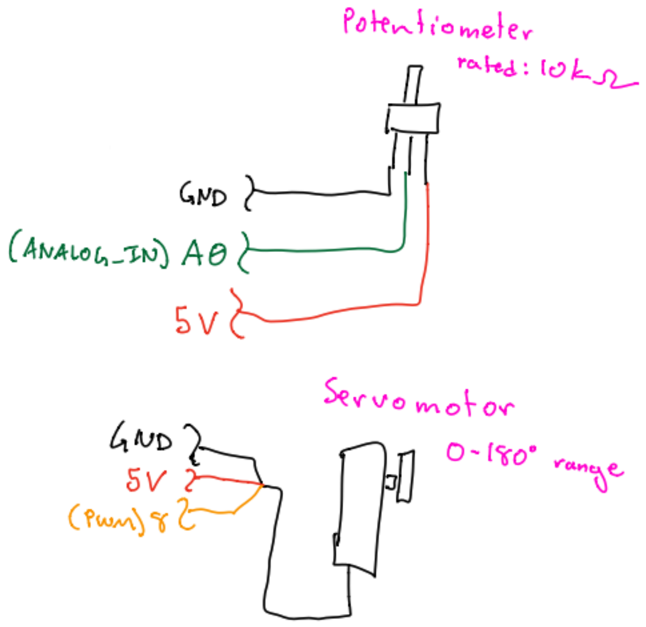

Figure 1.1: Potentiometer Servomotor Pinout Schematic

Once I was done with my section and Yuqing was done wiring and coding the Flex Sensor outputs of the Arduino and tying those readings to servomotor position outputs, we worked to develop a logic where both sensors could control the position of the servomotor. Between the two sensors, the flex sensor is the dominant controller of the servomotor. This means that when the flex sensor is undisturbed, the potentiometer controls the 180-degree position of the servomotor by rotating the sensor from one extreme position to the other. Regardless if the potentiometer is outputting signal to control the servomotor, when the flex sensor is disturbed significantly, the servomotor will follow the signal output from the flex sensor over the potentiometer.

To help integrate the code for the Arduino-only state section, I followed the integrated architecture layout as developed by Jonathon Lord-Fonda. As such, from the provided code description, I injected code snippets from their respective team member into the *main.ino* file. To verify the integration, I ran the integrated code on the production setup to confirm both expected sensor and motor behavior and expected state switches as per the push button. I did not integrate the GUI and ROS integrations to the Arduino code file.

References

- N/A

2 CoBorg

2.1 Individual Progress

Description Since the Conceptual Design Review submission, my main progress made was in developing the actuated manipulation subsystem. This includes developing the HEBI ROS API, MoveIt framework , and developing a unit test to demonstrate the pipeline between Moveit and the HEBI ROS API.

Because the motors being used on the CoBorg are actuators manufactured by HEBI. Integrating the HEBI API into ROS would be crucial for the project. With the HEBI API already being integrated with the ROS Melodic framework, the integration steps were well documented by both ROS and HEBI. This meant that installing and configuring the HEBI API packages into ROS were straightforward.

Another integral part of the actuated manipulation subsystem is the integration of the ROS MoveIt node. Installation and configuration instructions into ROS Melodic was well documented, and so the installation was performed successfully. A preliminary URDF model was built to incorporate a rough polygon model of the As-Is CoBorg. A figure of the URDF model is shown below:

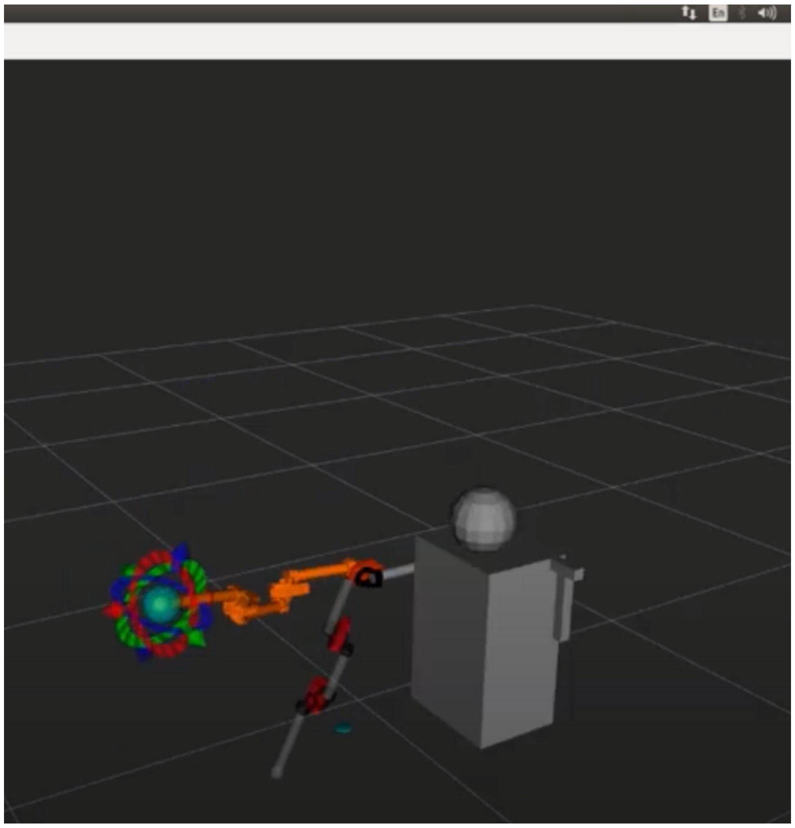

Figure 2.1: CoBorg URDF Model as seen through RViz

The .xacro file of the URDF model incorporates several macros from the downloaded HEBI API, namely the .xacro macros for the X-Series actuators and mounting brackets. Once the URDF model was built, a MoveIt installation was performed with the URDF model. The installation consisted of defining rotating joint agents in the URDF model and their frames, calculating the collision space of the robot arm with the rest of the URDF model, and defining a motion planning algorithm from the OMPL motion planning library provided by MoveIt. The selected motion planning algorithm was chosen to be RRT-Connect.

Once the MoveIt installation was completed, a demo.launch was performed on the MoveIt-integrated URDF file of the CoBorg arm as visualized through RViz. The end effector was moved to a point in 3D space, and the MoveIt motion planning component visualized the path between start and end. The /joint_states rostopic was examined and confirmed that the joint states were changing as per the visualizations in RViz.

Lastly, a pipeline between the URDF model of the CoBorg and the actual CoBorg robot was developed. This was done by pulling down the RobotModel object from the /robot_description rosparam and outputting a joint state goal from a desired 3D point position. Once the joint states were acquired, they were passed into the HEBI API which firstly connects to the HEBI motors on the network, then outputs the desired joint states in the robot. A simple unit test was performed on the robot based on a hard-coded 3D point and was successful in moving the CoBorg to that 3D point in space.

References

- HEBI C++ API ROS Integration
- ROS URDF Configuration Tutorials
- ROS MoveIt Configuration Tutorials

2.2 Challenges

CoBorg Currently, my main challenge in CoBorg is to integrate a global frame standard for the MoveIt mode. The CoBorg will be detecting and perceiving its environment through information from two cameras, so as such the actuated manipulation subsystem must comply and with cameras' frame of reference. This involves both integration with the Intel RealSense T265 and D435 software as well as configuration of that frame into the MoveIt URDF model. Additional development and testing needs to be performed in order to ensure the frames of reference between the actuated manipulation and RealSense cameras are the same. In addition, edge case testing would have to be performed to understand the robustness of the frame integration between the two components.

References

- N/A

2.3 Future Plans

CoBorg My future plans for the CoBorg include integrating the Intel RealSense cameras with the actuated manipulation subsystem. This would mean collaboration between the Vision and Actuated Manipulation subsystems, as both systems utilize information from the camera to achieve their goal. Once the cameras are integrated with the MoveIt URDF model, information from both the localization camera (T265) and depth camera (D435) would need to be processed and understood by the MoveIt model in order to perceive and move around the 3D world. In addition, ROS subscriber nodes need to be developed to pull in 3D goal positions and frame information from the camera nodes. ROS publisher nodes need to be developed to output the current joint state of the CoBorg.

References

- N/A

3 Sensor and Motor Control Quiz

3.1 Question 01

Prompt

1. What is the sensor's range?
2. What is the sensor's dynamic range?
3. What is the purpose of the capacitor C_{DC}
4. Write an equation for the sensor's transfer function.
5. What is the largest expected nonlinearity error in g?
6. How much noise do you expect in the X- and Y-axis sensor signals when the sensor is excited at 25 Hz?
7. How about 0 Hz? If you can't get this from the data sheet, how would you determine it experimentally?

Answer

1. The range of the sensor is $\pm 3V$.
2. The dynamic range of the sensor is $6V$.
3. The purpose of the C_{DC} capacitor is to decouple noise from the power supply input to the sensor.
4. Given a measurand reading x, the equation for the sensor's transfer function is shown below:

$$Sensitivity = S = 270 \; [mV/g]$$
$$Bias = B = 1.35 \; [V]$$
$$V_{out} = 0.27x + 1.35 \; [V]$$

5. The largestest expected nonlinearity is $0.108g$.
6. The equation for the *rms Noise* is equal to *Noise Density*$(\sqrt{BW * 1.6})$. Given a supply input of $3V_s$, the X-axis and Y-axis noise density values are $150\frac{\mu g}{\sqrt{Hz}}$, A noise signal at 25 Hz is equal to $150(\sqrt{25 * 1.6}) = 948.7\mu g$.
7. One can determine the noise at 0 Hz by running constant voltage tests on the accelerometer. Given an constant voltage level, one can connect the sensor to an oscilloscope to measure the variation of the output. The fluctuations given by the power supply provides a baseline noise at 0 Hz. Collected multiple votage levels through time, average those values, and subtract the mean voltage to determine the noise at 0 Hz.

References

- ADXL335 Accelerometer Datasheet

3.2 Question 02

Prompt Filtering

1. Name at least two problems you might have in using a moving average filter.
2. Name at least two problems you might have in using a median filter.

Opamps

1. Your uncalibrated sensor has a range of -1.5V to 1.0V (-1.5V should give a 0V output and 1.0V should give a 5V output).
2. 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).

Answer Filtering

1. One problem that would occur when using moving average filters is the increased latency of the measurement as a function of the size of the moving average window. Another problem is the moving average filter filters out higher frequency patterns and content, which may prove to be difficult if higher frequency signals are desired.
2. One problem with median filters is the slow computational time of the filter compared to other filters. Another problem is that the filter is not as effective at combating Gaussian noise as compared to the mean filter.

Opamps

1. To begin, V_1 was assumed to be the reference voltage and V_2 is the input voltage. The value of V_1 and V_2 were derived to be the following:

$$V_{out} = (V_{in} - V_{ref})\frac{R_f}{R_i} + V_{in}$$
$$0 = (-1.5 - V_{ref})\frac{R_f}{R_i} - 1.5$$
$$(eq.1)V_{ref} = -1.5 - 1.5\frac{R_i}{R_f}$$
$$5 = (1 - V_{ref})\frac{R_f}{R_i} + 1$$
$$(eq.2)V_{ref} = 1 - 4\frac{R_i}{R_f}$$
$$-1.5 - 1.5\frac{R_i}{R_f} = 1 - 4\frac{R_i}{R_f}$$
$$\frac{R_f}{R_i} = 1$$
$$V_{ref} = 1 - 4 = -3V$$

2. Applying the inverting and noninverting configurations of the circuit, it was found that where is no solution to the given opamp circuit.

References

- Research Gate: Problems with Moving Average Filter
- The Median Filter

3.3 Question 03

Prompt

1. If you want to control a DC motor to got to a desired position, describe how to form a digital input for each of the PID (Proportional, Integral, Derivative) terms.
2. If the system you want to control is sluggish, which PID term(s) will you use and why?
3. 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?
4. After applying the control in the previous question, if the system still has overshoot, which PID term(s) will you apply and why?

Answer

1. In order to measure position of the DC motor, I would acquire readings from an attached encoder. I would set the desired motor position to an aggregated encoder value from the current position of the DC motor, given that the encoder is an incremental encoder. To implement P component of the controller, I would compute the current aggregated encoder error from the desired error and multiply that error by a gain value. An appropriate PWM signal will be sent to the DC motor. To implement the I controller, I would maintain a static variable at the start of the program. At every loop, I would sum up the error between the current encoder position and the desired position. From that sum, I would multiply accumulated error by an I gain and output an appropriate PWM signal to the motor. To implement the D controller, I would compute a forward difference: taking the sum of the current with the previous encoder value and dividing by 2. Then I would divide by the time step of the system and multiply that value by a derivative gain and output an appropriate PWM signal to the motor.
2. If the system is sluggish, I will implement a proportional controller with higher gains. Increasing the P gains will ensure that the controller will output a more intense response to the positional error.
3. If the system has significant steady-state error, I will implement an integral controller. The memory component of the integral control will ensure that the system will reach a steady state error of zero over a non-infinite amount of time.
4. If the system has significant overshoot, I will implement a derivative controller. Setting the gains of the derivative controller will reduce overshoot.

References

- N/A

4 Appendix

4.1 Sensors and Motor Lab: Written Code

```
#include <Servo.h> // servomotor control

// Potentiometer declarations
#define WINDOW_SIZE_Pot 10
int POTENTIOMETER_PIN = A0;
int readingsPot[WINDOW_SIZE_Pot];
int sumPot = 0;
int indexPot = 0;

void setup(){
    // Servo Motor attachments
    pinMode(SERVO_PIN,OUTPUT);
    servo.attach(SERVO_PIN);
}

void loop(){
    //Potentiometer
    int analogValuePot = analogRead(POTENTIOMETER_PIN);
    int avgValuePot= avgFilterPot(analogValuePot);
    float anglePot = map(avgValuePot, 0, 1024, 0, 180.0);
    if(anglePot > 180.0){//bounds check
        anglePot = 180.0;
    }
    if(anglePot < 0){
        anglePot = 0;
    }


    //Update Servo Motor
    if (angleFlex < 15.0){ //threshold between flex sensor and potentiometer
    servo.write(anglePot);
    }else{
    servo.write(angleFlex);
    }
}
```