Carnegie Mellon University

16-681

MRSD Project 1

Individual Lab Report 1
COBORG

Team C:
Gerald D'Ascoli | Jonathan Lord-Fonda | Yuqing Qin | Husam Wadi
Feng Xiang

Sponsor:
Biorobotics Lab

February 25, 2021

Table of Contents

1. Individual Progress

1.1. Sensors & Motors Lab

My responsibilities for the sensors and motors lab was to create the GUI, help with sensor code integration, mount the hardware, and clean up the wiring after the initial prototype was completed. The code integration was the process of taking all the individual sensor components and combining them into one, and since the GUI was to interface with the final integrated sensor code, I helped with the transition between sensor and GUI integration. The hardware mounting was the process of affixing all the components (primarily using thermal glue) to the breadboards. The wiring cleanup consisted of removing each individual wire that was oversized and replacing with smaller, compact wire to increase the reliability of the sensors and motors circuit.

1.1.1. GUI

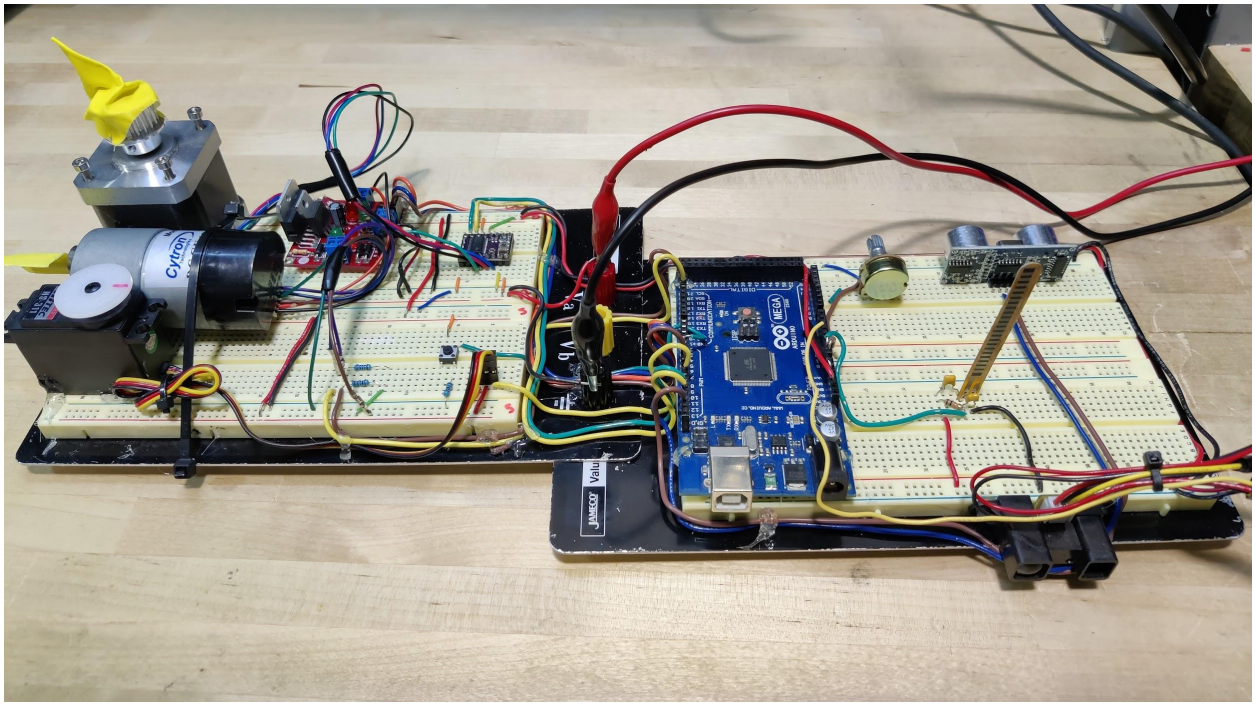
We decided to do the GUI in RVIZ/RQT, as this would allow the sensors and motor lab to relate to our MRSD project. While I am very familiar with Processing, and could have created a GUI within a matter of hours in a single day through Processing, we felt as a team that learning ROS through this lab was something that was worth investing time into. This meant that we had to learn how to incorporate a ROS publisher and subscriber in Arduino, in parallel to the sensor control that already existed. I would estimate that creating the GUI in ROS took well over 40 hours to complete, as there were many components that had to be completed for the GUI to work and display correctly:

1. A URDF had to be created to output joint states that would command the motors.
2. A subscriber on the Arduino side had to translate the joint states from radians to the motor's respective input (degrees, pwm, steps).
3. An RVIZ custom configuration to show the URDF
4. The joint states GUI node to generate sliders for the motor control output.
5. A custom RQT layout that displays all 5 states + incorporates the custom RVIZ perspective.
6. A custom message class (named "CMU.msg") to display the names of each output received on
7. A publisher node that publishes to "CMU.msg" the states of the sensors and push button

The appendix contains the Arduino code that I wrote, and also the readme I created to launch the "cmu_motor_lab" package I created for this lab. The final result is the GUI shown in the figure below:

1.1.2. Final Product

In the figure below is the final sensors and motor lab circuit. We had to adjust to an Arduino Mega from an Arduino Uno as our code took too much memory in the Arduino and it became unreliable. While we looked for ways to optimize our code, we thought that it would be safer to upgrade to an Arduino Mega and use 7% of the total memory instead of 98% of the Arduino Uno's memory. The final project consisted of a flexible resistor (flex sensor) and potentiometer which controlled the output of the servo motor, an ultrasonic sensor which controlled the output of the stepper motor, and an IR sensor which controlled the output of the DC motor, and a push button which would switch from sensor control to GUI control. The potentiometer controlled the absolute position of the servo encoder (from 0 - 180 degrees), while the flex sensor would modify the angle from the potentiometer's position (±90 degrees from the current servo position). The ultrasonic sensor would translate 0-100cm into a full revolution of the stepper motor (which was split into the 400 steps per revolution of the stepper). The DC motor was PID controlled based on a desired position generated from the IR sensor translated to an encoder value that the DC motor encoder would feedback to match. The figure below shows the final product as described:



1.2. COBORG Project

As the project manager for the COBORG project, I focus on ensuring that the timeline we promised in the previous semester is met, and that we can adapt as a team when changes arise. We use an Agile framework through Trello to plot one week intervals of work, and every week we review what was accomplished and what was not. By balancing our project workload with our coursework, we have been able to stay on track for the semester.

Outside of project management, I focus with Jonathan on the ROS software architecture of the COBORG platform. Recently I have been researching state managers that may be applied to our system, and we plan on running a pilot test in the upcoming week to see if the addition of a dedicated state manager, named smach, is worth integrating into our project. This package was recommended to us by a ROS mentor and expert who advises the project.

2. Challenges

2.1. Sensors & Motors

There were many challenges in setting up the GUI for the sensors and motors lab. The learning curve to implement “rosserial” to communicate with the Arduino was steep, and specifically creating a custom message type called “CMU” which contained data types of “Potentiometer, Flex_Sensor, IR_Sensor, Ultrasonic_Sensor, and Button_State” was a much greater challenge than I

anticipated. Troubleshooting the message took more than a day for it to operate in the manner that I had anticipated. Also I found RQT to be buggy, as I spent a considerable amount of time working through the program crashes due to matplotlib issues.

2.2. COBORG Project

As a project manager, the last thing I would like to see is feature creep this semester. The original scope of the project was created with two to three weeks of buffer that already seem to be dwindling as we progress through the semester. We did not anticipate that the MRSD Project 1 course would have as many long and strenuous assignments as it currently has, and this could lead to a considerable amount of stress with MRSD project scope as is. Increasing the scope in any way would present a considerable challenge to the project, as between this course, robot autonomy, the business course, and for my teammates "introduction to machine learning", we are overcapacity in terms of working hours per week. Ensuring that our timeline does not slip any further is my greatest challenge.

3. Teamwork

3.1. Sensors and Motors Lab

Team Member	Sensor	Motor	Motor Lab Contribution
Feng Xiang	Potentiometer	Servo motor	Mapped potentiometer sensor analog output to servo-motor input. Collaborated with Yuqing to develop a control relationship between potentiometer and flex sensor with servo motor output. Helped combine code together to get arduino-only arduino code state.
Jonathan Lord-Fonda	Ultrasonic	Stepper motor	-Soldered DC motor controller board -Wrote switch debouncer code -Wrote code structure -Wrote program for stepper motor -Wrote program for ultrasonic sensor
Gerry D'Ascoli	IR	DC motor	-Wired DC motor and motor controller to each other and arduino -Debugged electrical issues -Wrote PID control of DC motor using encoder for state and IR sensor as input for desired state -Wrote program to control stepper motor position using the ultrasonic sensor as input.
Yuqing Qin	Flex sensor	Servo motor	Mapped a flex sensor output to the servo motor to

			control its movement. Also tested the IR sensor and implemented moving average filtering to improve the sensor outputs.
Husam Wadi	GUI		-Created a ROS publisher and subscriber node in Arduino to send/receive data -Created a serial interface between ROS and the Arduino using “rosserial” -Created a custom message class “CMU” that handled sensor data outputs -Created a URDF that controlled the motor outputs through joint_states publisher -Created a visualization in RVIZ to show the GUI motor control output -Created a RQT GUI that contained RVIZ and 5 plots that displayed all the sensor outputs in real time

3.2. COBORG Project

Team Member	**Work Description for COBORG**
Feng Xiang	Developed pipeline between MoveIt and HEBI API and Coborg motors Developed URDF and initialized moveit to move robot in RViz
Jonathan Lord-Fonda	-Wrote the initial BOM for the current system -Created initial ROS node map -Sketched out main state node -Installed Linux/ROS and completed ROS training -Tested the arm’s max lift at full extension -Reviewed Jason’s 3-D model of Coborg
Gerry D’Ascoli	Developed prototype of voice system. Generated a custom implementation of pocketsphinx on the CoBorg system and created custom libraries and recognition files to tailor the voice system to the CoBorg’s specific requirements and functions.
Yuqing Qin	Work on perception subsystem. Implement YOLO ROS node and integrated with realsense camera node
Husam Wadi	-Manage and updating project timeline to adapt to the semester load -Used scrum/agile methodologies through kanban boards to map out weekly project work -Deepened ROS interconnectivity knowledge by researching state machines in ROS and going through ROS tutorial examples.

4. Plans

4.1. Sensors & Motors

We plan on using the knowledge we gained from doing the GUI in ROS for our project. There were many fundamental aspects of ROS that were cemented by doing the sensors and motors lab GUI. The use of the joint_states and robot_states publishers will come handy for our robot arm execution in the future.

4.2. COBORG Project

In the next few weeks I will focus on encouraging the team to meet a mid March pre demo date, where we will try to go through a limited run of the SVD. This process will allow us to see clearly which aspects of the project need to be focused on and improved. I will also continue providing support across the board for the COBORG team, as my experiences lend well to almost every aspect of the project. Specifically, we have challenges integrating the Intel Realsense cameras with the hebi robot arm in ROS, so we will be doing pilot runs and rapidly iterate and test cross functions between the cameras and the arms using Move-It.

5. Quiz

5.1. ADXL335

- What is the sensor's range?
  - ±3g (minimum used)
- What is the sensor's dynamic range?
  - 6g (dynamic based on minimum)
- What is the purpose of the capacitor CDC on the LHS of the functional block diagram on p. 1? How does it achieve this?
  - The capacitor decouples the accelerometer from noise on the power supply. It does this by carrying a capacitance that provides enough power to keep the accelerometer stable if the voltage drops temporarily, and if the voltage increases the capacitor will be able to absorb the excess energy.
- Write an equation for the sensor's transfer function.
  - $1.5V + (300mV/g) * a$
- What is the largest expected nonlinearity error in g?
  - Based on typical (3.6g) = ±0.0108g
- How much noise do you expect in the X- and Y-axis sensor signals when the sensor is excited at 25 Hz?
  - $rms\ Noise = Noise\ Density * (\sqrt{BW\ x\ 1.6}) = 150 * (\sqrt{25\ x\ 1.6}) = 948.68ug$
- How about at 0 Hz? If you can't get this from the datasheet, how would you determine it experimentally?

- We would get as close as we could to 0 Hz and give it a test. Let's say that 0.01 Hz is close enough:
- $rms\ Noise\ =\ 150\ *\ (\sqrt{0.01\ x\ 1.6})\ =\ 18.97ug$~

5.2. Signal Conditioning

- Filtering
  - Name at least two problems you might have in using a moving average filter.
    - If the moving average filter is large, there will be a lag between the latest value and the average value.
    - The moving average smooths the output of the sensor, causing it to lose granularity/sensitivity.
  - Name at least two problems you might have in using a median filter.
    - You usually have to add padding to the filter to accommodate the boundaries of the median filter. This in itself is noise.
    - A median filter is a nonlinear filter. This means that it is still possible to see large spikes in the data that a moving average would have prevented. Depending on the application this could be an advantage, but in many proximity sensor applications this is a disadvantage.

- Op Amps
  - 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).
    - For this system V2 = Input, and V1 = Reference. We have 2 systems of equations:
      - $5v\ =\ (1.0v\ -\ V1)(Rf/Ri)\ +\ 1.0v$
      - $0v\ =\ (-\ 1.5v\ -\ V1)(Rf/Ri)\ -\ 1.5v$
    - Solving for this system we find:
      - $V1\ =\ \ -\ 3v$
      - $Rf/Ri\ =\ 1$
  - 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).
    - This circuit has no solution. When I tried to solve for the system of equation, using V1 as an input I got a negative Rf/Rivalue. I tried the other way (V2 as input) and found it was the same result. It seems that if the dynamic range is equal to the output range there is no combination of resistors that will give that value.

5.3. 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.
    - For Proportional a motor encoder outputs a positional output so we can directly feed the current encoder output minus the desired output to generate a proportional error.
    - The Integral control input can be created by summing the error of the desired positional encoder position minus the current encoder position multiplied by the timestep.
    - The Derivative control input could be derived from the positional encoder by taking the previous position minus the current position divided by the timestep, but this will also derive the noise causing a somewhat unreliable output. Another method would be to use a dedicated velocity sensor, such as MEMS gyroscope or a hall effect based sensor.
- If the system you want to control is sluggish, which PID term(s) will you use and why?
    - Proportional. By just adding a proportional gain we can decrease the rise time and increase the responsiveness of the system.
- 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?
    - We would add an Integral term to the equation. This will allow us to get a running sum of the steady state error and compensate for it, thus allowing us to reach our desired goal.
- After applying the control in the previous question, if the system still has overshoot, which PID term(s) will you apply and why?
    - The Derivative term will allow us to derive the future error, thus removing the overshoot caused by cranking up the proportional gain. Our system will now “anticipate” the overshoot and correct before it happens.

6. Appendix

6.1. Arduino Code

```
// ROS Headers
#include <Arduino.h>
#include <ros.h>
#include <sensor_msgs/JointState.h>
#include <rosserial_arduino/CMU.h>

// ROS declarations
// Message callback declaration
void messageCb(const sensor_msgs::JointState& msg) { //don't move this function. doesn't work if moved
under void loop().
```

```
  if (state == 1){ //only run this code in GUI state (1)
    //servo msg
    int set_angle = msg.position[0]*(180/3.14); //servo motor convert rad2deg (pi -> 180 deg rotation)
    servo.write(set_angle); // valid inputs: 0-180

    //dc motor msg
    int set_pwm = msg.position[1]*(255/1.57); //dc motor range is from pi/2 to -pi/2. each side denotes
max PWM speed (255) in that direction
    if(set_pwm > 0){//CounterClockwise
        digitalWrite(motorpin1,HIGH);
        digitalWrite(motorpin2,LOW);
        analogWrite(enable_motor, set_pwm);
    }
    else if(set_pwm < 0){//Clockwise
        digitalWrite(motorpin1,LOW);
        digitalWrite(motorpin2,HIGH);
        analogWrite(enable_motor, -set_pwm); //only positive PWM values. --set_pwm = +set_pwm
    }

    //stepper message
    int set_step = msg.position[2]*(stepsPerRevolution/6.28); //convert rad2steps (2 pi = 360 deg
rotation)
    if(set_step > currentStep) digitalWrite(dirPin,HIGH);
    else digitalWrite(dirPin,LOW);
    for(int x = 0; x < abs(set_step-currentStep); x++){
        digitalWrite(stepPin, HIGH);
        delayMicroseconds(1000);
        digitalWrite(stepPin, LOW);
        delayMicroseconds(1000);
    }
    currentStep = set_step; //ensure no discrepancy
  }
}

// ROS node type(s) declaration
ros::NodeHandle nh; //create a NodeHandle called nh
ros::Subscriber<sensor_msgs::JointState> sub("joint_states", messageCb);  //subscribe to joint_states
rosserial_arduino::CMU cmu_msg; //define CMU message type
ros::Publisher pub("CMU", &cmu_msg); //create a publisher to cmu_message of type CMU

void setup(){
    //ROS Setup
    nh.initNode();
    nh.subscribe(sub);
    nh.advertise(pub);
}

void loop(){
    //package sensor date to cmu_msg
    cmu_msg.Potentiometer = anglePot;
    cmu_msg.Flex_Sensor = angleFlex;
    cmu_msg.IR_Sensor = dist;
    cmu_msg.Ultrasonic_Sensor = distanceCm;
    cmu_msg.Button_State = state;

    pub.publish(&cmu_msg); //send it!
    nh.spinOnce(); //check for ROS callbacks for joint_states movement
}
```

6.2. GUI Launch Readme

arduino_ws is a catkin workspace for the sensors and motors lab

Arduino:

In "Coborg-Platform/arduino_ws/src/arduino_files" is the main.ino for the arduino.

Point your arduino IDE sketch folder to "Coborg-Platform/arduino_ws/src/arduino_files" to be able to compile and upload main.ino. This allows it to see the libraries folder which contains the PID and ros_lib libraries.

GUI:

To run the ROS package for the gui, follow these steps:
`cd Coborg-Platform\arduino_ws`
`catkin_make install`
`source devel/setup.bash`

Now we have to recompile the message library for the (jank) rosserial package to work:
`cd Coborg-Platform/arduino_ws/src/arduino_files/libraries`
`rm -rf ros_lib`
`rosrun rosserial_arduino make_libraries.py .` <- that period is important. include it in the command.

You should be ready to launch the gui. plug in the arduino to a usb port and run:
`sudo chmod a+rw /dev/ttyACM0`
`roslaunch cmu_motor_lab demo.launch`

The demo.launch file runs these components:

- joint_state publisher + GUI sliders
- robot_state publisher
- rosserial python to communicate with the arduino

rqt with a custom perspective for this project:

- rqt_plot of the custom CMU message to read the sensor inputs
- RVIZ
    - The rqt_rviz plugin may not automatically load the urdf.rviz perspective file.
      To do this manually, click on "file/open config" tab in the RVIZ section and load
      the file located in "cmu_motor_lab/rviz/urdf.rviz".
      You may have to pull out the USB plug to allow the "open file" prompt to pop up.
- If for some reason rqt does not load it's custom perspective, click on perspective/import
  from the top drop down list and load the file "cmu_motor_lab/rqt/motor_lab.perspective".

Additional information:

rosserial install info: http://wiki.ros.org/rosserial_arduino/Tutorials/Arduino%20IDE%20Setup

Note: under "src/rosserial/rosserial_arduino/msg" is the custom message type "CMU.msg" Ensure that this file is present when you run catkin_make to allow rosserial to compile that message type.

Common ROS errors:

[ERROR] [1614183628.727014]: Tried to publish before configured, topic id 125

[ERROR] [1614183338.789682]: Error opening serial: [Errno 13] could not open port /dev/ttyACM0: Permission denied: '/dev/ttyACM0' -run: `sudo chmod a+rw /dev/ttyACM0` to allow read/write access to the USB port

Common Arduino compile errors: In file included from Coborg-Platform/arduino_ws/src/arduino_files/libraries/ros_lib/std_msgs/Time.h:7:0, from Coborg-Platform/arduino_ws/src/arduino_files/libraries/ros_lib/ros/node_handle.h:40, from Coborg-Platform/arduino_ws/src/arduino_files/libraries/ros_lib/ros.h:38, from Coborg-Platform/arduino_ws/src/arduino_files/archive/adc_test/adc_test.ino:10: Coborg-Platform/arduino_ws/src/arduino_files/libraries/ros_lib/ros/msg.h:40:10: fatal error: cstring: No such file or directory #include

Basically the `rosrun rosserial_arduino make_libraries.py .` command installs a broken ros_lib library (I know. Why ROS, Why...). Copy the contents from ros_lib_backup into ros_lib and you will be able to compile the arduino code (overwrite).