EECS C106B: (Mini) Project 4 - Soft Robots *

Due date: April 18th at 11:59pm

Goal

The goal of this lab is to become more familiar with soft robotics and the advantages and limitations of using soft robots over rigid manipulators. In addition, this lab will provide hands-on experience with modeling and grasping with soft fingers.

Contents

1 Theory

Soft robots are made out of materials with compliance similar to that of soft biological materials, so they can more readily adapt to a variety of objects and environments compared to rigid robots. This makes tasks like grasping easier, but precisely controlling the infinite dimensional state space of a soft robot is quite difficult. You may not necessarily need to use the theory in this section for this particular (mini) project, but it is good to know in case you want to pursue further projects in this field.

*Developed by Valmik Prabhu and Chris Correa, Spring 2018, Spring 2019. Further expanded by Josephine Koe, Spring 2022.

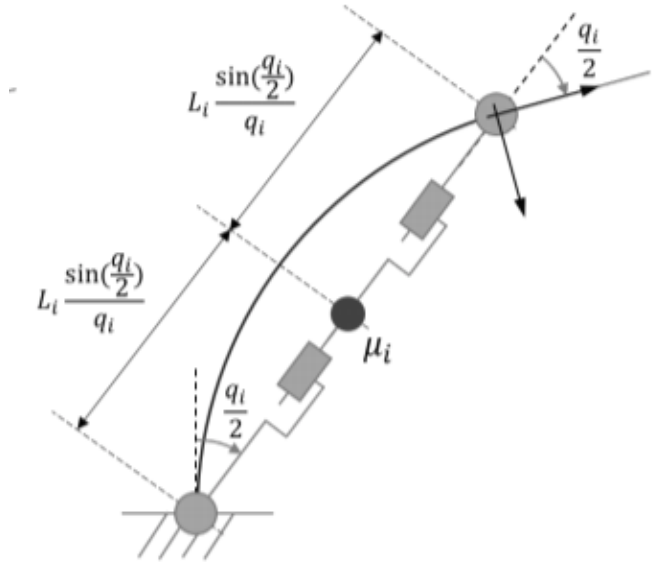

Figure 1: A soft robotic joint modeled as four rigid joints

1.1 Soft Finger Model

To model a soft robotic finger, we can use the methodology from this paper from Daniella Rus's lab at MIT. Each soft joint is assumed to be a constant curvature segment of length L_i and mass μ_i. Each segment can be modeled as four rigid joints as shown in Figure 1 where q_i, degree of curvature for that segment, is the only parameter needed to describe the state of the segment. For the fingers in our lab, $L_i = 7.6$ cm and $\mu_i = 38$ g.

We've already seen how to find the dynamics for a rigid manipulator, so we'll start with this equation

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = \tau + J^\top(q) f_{ext} \tag{1}$$

where

- $M(q)$ is the inertia matrix
- $C(q,\dot{q})$ is the Coriolis matrix
- $G(q)$ is the gravity matrix
- τ represents the torque through which the finger is actuated
- f_{ext} are external wrenches that are mapped through the Jacobian $J(q)$

Of course, this model is only an approximation, so we add stiffness (K) and damping (D) terms to take into account the compliant nature of the joint.

$$M(q)\ddot{q} + [C(q,\dot{q}) + D]\dot{q} + G(q) + Kq = \tau + J^\top(q) f_{ext} \tag{2}$$

For simplicity we can model the stiffness (Young's modulus) and damping as constants, but keep in mind that in reality rubbers are hyperelastic and have a nonlinear stress-strain relationship.[1]

1.2 Input Dynamics Model

The control input u to the soft finger will be a PWM (pulse-width modulation) value to send to the pump. The torque of the finger is related, but not directly, to this PWM input. We can model these input dynamics with a second order filter

[1] An example of a more complex approximation is described in Walsh et al which uses a fourth order stiffness model.

$$\tau = \frac{\alpha}{(\gamma s + 1)^2} u \tag{3}$$

which means that

$$\gamma^2 \ddot{\tau} + 2\gamma \dot{\tau} + \tau = \alpha u \tag{4}$$

1.3 Estimating the Finger State

To estimate q, we can combine the two models above and describe the soft finger as a system with states $\begin{bmatrix} q & \dot{q} & \tau & \dot{\tau} \end{bmatrix}^\top$ and input u. Using Equations (2) and (4) and assuming no external wrenches, we can derive the following dynamics equations

$$\begin{bmatrix} \dot{q} \\ \ddot{q} \\ \dot{\tau} \\ \ddot{\tau} \end{bmatrix} = \begin{bmatrix} \dot{q} \\ M(q)^{-1}(-[C(q,\dot{q}) + D]\dot{q} - G(q) - Kq + \tau) \\ \dot{\tau} \\ \frac{2\gamma\dot{\tau} + \tau + \alpha u}{\gamma^2} \end{bmatrix} \tag{5}$$

Integrating these dynamics over time will give you an estimate of q for a given set of system parameters and initial finger states.

1.4 System Identification

To use the above model, you need to identify values for the system parameters K, D, α and γ, and you can do that by taking data on the behavior of the finger and using a nonlinear least squares solver. The least-squares solver will solve the following problem:

$$\min_{K,D,\alpha,\gamma} (\|q_m - \hat{q}(K, D, \alpha, \gamma, u)\|^2) \tag{6}$$

where q_m is your measured state (from your data) and $\hat{q}$ is the estimated state (from your model) given a time series of input PWM values (also from your data) and the set of system parameters you'll be solving for. Solving the nonlinear least-squares problem will give you the best estimates of the system parameters according to your data, and plugging these values into the dynamics equations will give you an approximation of the dynamics for the specific finger. With this, you can design a model-based control input.

2 Project Tasks

You'll be using the soft gripper to do the following tasks:

1. Collect data on the behavior of the soft fingers
2. Fit simple models to the data to describe the behavior of the soft fingers
3. Design a control input to evenly lift a paper cube with the soft gripper

Due to the compliance of the gripper and the simplicity of the last task, this project can potentially be done without implementing the above theory. We encourage you to try to understand the theory behind these soft robots but make the coding as easy on yourselves as possible.

2.1 Collecting the data

Place the backboard with angles through the slot in the top between the soft gripper and the camera. Run the data collection sequence, where for each finger you will repeat the following process:

1. Enter in a PWM value to send to the finger.
2. Wait until the finger stops moving and reaches a steady state.
3. Eyeball the approximate angle that the finger stabilizes at according to the backboard (see Figure 2).

You should collect at least 15 data points for each finger, and the more data you collect the better.

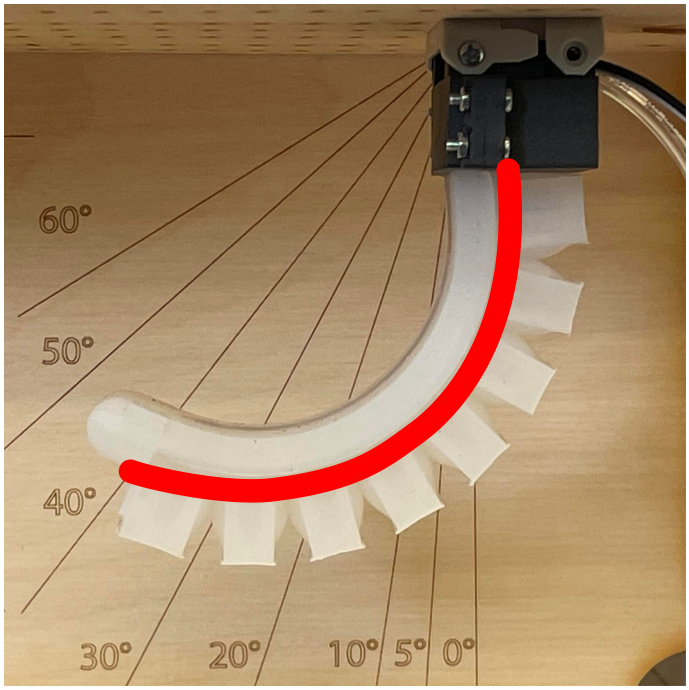

Figure 2: Example of approximating the angle of the soft finger. To best account for the assumption of a constant length of the segment, measure down the center of the finger as shown by the red line and find the angle at which the inflatable portion of the finger ends. In this picture, the angle of the finger is 40 degrees. (Tip: when approximating the angle, close one eye or stand farther away.)

2.2 Fitting models to the data

If you want to use the model outlined in the Theory section, q_m will be approximately double of the steady state angles you observed during the data collection. This would be enough information to use the model from the Theory section, but to get a more intuitive sense of how the finger behaves, plot the following relationships:

- Finger angle vs. PWM value
- Finger angle vs. pressure sensor value
- Finger angle vs. flex sensor value

You will probably care most about the relationship between finger angle and PWM value, but knowing the relationship between finger angle and the sensor values can be useful when incorporating feedback into your control input. For each of these relationships, fit a model of your choice to the data and plot it to verify that it fits your data well.

2.3 Lifting the cube

If a Baxter tried to grasp a delicate paper cube, it would probably either fail to pick it up or crush it. On the other hand, the compliance of the soft fingers makes them a great tool for this task. Based on the models you that you generated from the data, design a controller that takes in a single desired angle for both fingers and outputs a PWM value for each finger (this is your control input). The fingers do not necessarily have to reach the specified desired angle as it grasps the cube (it shouldn't anyways because you assumed no external wrenches when fitting the model), but try to get both fingers to be at approximately the same angle at all times. The requirements for a successful grasp are as follows:

- The cube is lifted at least 5 mm above its starting position for at least 5 seconds
- The cube never tilts more than 0.2 radians around the z-axis from its starting orientation

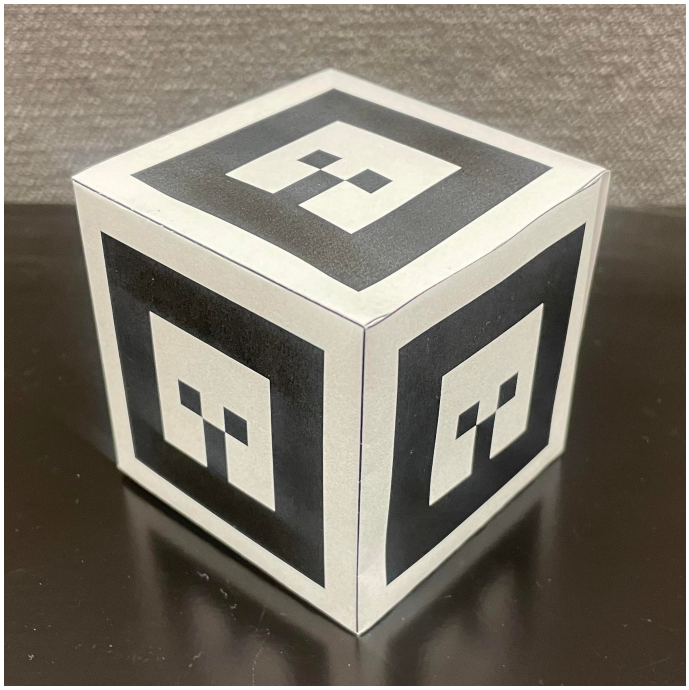

Figure 3: Paper cube with AR Marker 0 on all sides for the grasping task

You will need to remove the backboard so the camera can record the position of the cube. You may not use the camera images as visual feedback for the controller, but you can use the pressure and flex sensor values to provide feedback to the controller.[2]

3 Deliverables

To demonstrate that your implementation works, deliver a report with the following information. For this project, you do NOT need to use the IEEE two-column conference template, but your answers to the prompts should be clearly labeled.

1. GitHub link: Provide the link to your GitHub classroom repo. Simply push your code to the private repository that was created for your team, and we will be able to see any changes you push to your assignment repository.
2. Data and model plots: Attach the figure displayed by `soft_robot_user_interface.py` after you have fit your models to each of the plots.
3. Fitted models: For each plotted relationship, briefly explain why you chose the type of regression that you did and what the final equation was.
4. Grasp plots: Attach the figure displayed when grasping the cube for one grasp attempt.
5. Grasp video: Provide a link to the video corresponding to the same grasp attempt.
6. Control input: Explain how you designed your control input.
    - What did you try that worked? What didn't? Did you incorporate your models from the previous task?
    - What were the starting and ending angles you chose? Why did you choose them?
    - Did you have a feedforward term? If so, how did you design it? If you had to tune it, how did you approach it? What was the final feedforward term?
    - Did you have a feedback term? If so, how did you design it? In particular, which sensor(s) were useful for the feedback term? If you had to tune the feedback term, how did you approach it? What was the final feedback term?

[2] In this setup, the camera would be an external sensor (a sensor in the environment), and the pressure and flex sensors are embedded sensors (a sensor in the robot). In general, being able to control a robot without external sensors is valuable because it does not require the environment to be set up in a specific way and you can use the robot in a wider variety of situations.

4 Getting Started

Create a new ROS workspace to store your code for this project. Remember to build and source your workspace after you download the starter code and other packages.

4.1 GitHub Classroom

For projects in this class, we will be using GitHub Classroom. To access the starter code, simply head over to the Project 4 assignment page to accept the assignment. If you are asked to associate yourself with a school identifier, just click "Skip to the next step".
You should now be asked to create a team or join from a list of existing teams. If one of your teammates has already created a team, you should join that team instead of creating a new one. **After you have joined a team, you will not be able to switch teams by yourself. If you make a mistake or something else comes up that requires you to switch teams, let us know.**
Your team should now have a private project repository located at https://github.com/ucb-ee106-classrooms/project-4-your-team-name. If there are any updates to the starter code that you wish to pull you may do so with the commands

```
cd your-repo
git remote add public https://github.com/ucb-ee106/proj4_pkg.git
git pull public main --allow-unrelated-histories
git push origin main
```

Once you have formed your team, fill out this form so we can assign your team to a soft gripper. You can view the robot assignments here, and the soft robot booking calendar is here.

4.2 Other ROS Packages

Clone these ROS packages into your workspace. Make sure you also have the `head_camera.yaml` file in the correct location from Lab 0.

```
git clone https://github.com/ros-drivers/usb_cam.git
git clone https://github.com/srv/srv_tools.git
git clone https://github.com/sniekum/ar_track_alvar.git
cd ar_track_alvar
git checkout kinetic-devel
```

4.3 Soft Gripper Logistics / Safety

In this lab, we will be working with the soft gripper created by Amir Mousavi, David Au, Michael Nguyen, and Xiaoman Wu. It has two opposing silicone fingers which expand on one side when air is passed into the internal chambers. It can wrap around objects when pressurized.
Please handle these robots with care so they will remain in a good condition for other groups. Breaking any of the following rules will put your team at risk for a grade deduction.

1. Don't set the PWM values of the pumps to be higher than 100. This is to prevent damage to the fingers. 100 should be enough to bend the fingers completely.
2. Make sure to always be in reach of the yellow "ON" button on the power supply. If something goes wrong, hit this button immediately.
3. Only turn on the power supply when you need it, and do not power the gripper with more than 12 V.
4. Be careful especially with the tubes and wires connecting to the fingers. It is possible to puncture one of the internal chambers or break the wires if you are too rough with the gripper.
5. Do not touch any of the electronics or valves below the soft robot or detach the webcam without permission from the course staff.
6. Use the computer that your team's assigned robot is plugged into. Do not unplug the robots from the power supplies or computers and move them. The robots are named Leslie, Levi, Lily, Luka, and Luna and are attached to computers 1-5 in that order.

Figure 4: (Left): Power supply when turned on but output is OFF. Note the power button (yellow), the output "ON" button (blue), the set voltage (red), and the keypad(white) Right: Power supply when output is ON. Note the glowing output "ON" button.

4.4 Working with the Soft Gripper

Check that you have done the following before using the soft gripper:

1. Turn on the power supply. It should look like Figure 4 (left).
2. With the set voltage box selected, enter in "12" using the keypad, then press Enter.
3. To power the soft robot, press the yellow "ON" button. It should look like Figure 4 (right) where the yellow "ON" button is glowing. You can also confirm that the soft robot is powered by checking that the red LED below the soft robot is lit. **If something goes wrong with the soft gripper, press the yellow "ON" button immediately to turn off power to the soft gripper.**

Finally, you can start the soft gripper with the following launch file:

```
roslaunch proj4_pkg soft_gripper.launch
```

This will open up the camera and start the node which listens to states and sends commands over Serial.

4.5 Starter Code

4.5.1 Data collection and fitting models to the data

- `src/soft_gripper_user_interface.py` This script is what you will use to interface with the soft gripper. You can use the command line arguments to specify a file name and one of 3 modes:
  - `play`: You can use this mode to control both fingers and get a sense of how the robot moves. We encourage you to have fun with this mode and try to manually control the gripper to grasp the cube. If you specify a file name, the script will record the states to that file.
  - `record_angles`: You should specify a file name and use this mode to run the data collection sequence and record the angle data.
  - `display_data`: You should specify a file name to read the angle data from. This mode will plot the data for you. The initial starter code will plot a guess at the type of regression for the data, but this may not necessarily match your data. You will need to edit this file, specifically the `display_data` function, if you want to test out other types of regressions to see if they fit better.

4.5.2 Grasping the cube

- `launch/soft_gripper_cube_grasper.launch` When you want to grasp the cube, run this launch file. This file launches the grasp controller and the animation that plots the grasp. It takes in a starting angle and an ending angle (see `src/grasp_controller.py`) and a file name to save the grasp data to as command line arguments.

- `src/grasp_controller.py` This script commands the grasp. More specifically, it takes in a starting angle and an ending angle and commands the fingers to linearly ramp up from the starting angle to the ending angle for about 10 seconds. Then it continuously commands the fingers to go to the ending angle for another 10 seconds. You may not change this behavior, but you can specify your own start and end angles as command line arguments in the above launch file. This file is also where you will implement your control input.
- `src/grasp_plotter.py` This script plots data from the grasp with the bounds for success in red. It also saves the camera feed to a bag file in the `/data` folder with the file name you specified when running the launch file. You will not need to edit this file, but you will need to run

  ```
  rosbag reindex [bag-file]
  ```

  and then

  ```
  rosrun bag_tools make_video.py usb_cam/image_raw [bag-file] --fps 16 --output [video-file]
  ```

  after running a grasp to convert the bag file into a video for your submission. `[bag-file]` should be the path to the bag file from the grasp trial with a `.bag` extension and `[video-file]` should be the path where you want to save the video to with a `.mp4` extension. Make sure each grasp attempt is given a different file name to ensure the bag file and video only contains information for one grasp attempt. If this still doesn't work, just take a video of the same grasp yourself and submit it.

5 Scoring

Table 1: Point Allocation for Project 4

Section	Points
Code	3
Data and Model Plots	6
Fitted Models	6
Grasp Plots	6
Grasp Video	4
Control Input	15

Summing all this up, this mini-project will be out of 40 points. There is no additional graduate student portion of this project.

6 Submission

You'll submit your write-up on Gradescope and your code to GitHub Classroom. Your code will be checked for plagiarism, so please submit your own work. Only one submission is needed per group. Please add your groupmates as collaborators on both GitHub and Gradescope.

7 Common Issues

- **EOFError in the terminal:** Open a new terminal window and try running it again.
- **The cube height and angle are not plotting in the animation:** Try running

  ```
  rostopic echo /usb_cam/image_raw
  ```

  and if no output gets printed, check if you have the file

  ```
  ~/.local/lib/python2.7/site-packages/easy-install.pth
  ```

  If you have this file and a line in the file looks like this

```
import sys; new = sys.path[sys.__plen:]; del sys.path[sys.__plen:]; p = getattr(sys, '__egginsert', 0); sys.path[p:p] = new; sys.__egginsert = p + len(new)
```

comment out this line.

- **The grasp plots close before the grasp finishes and there are no visible errors in the terminal:** Just run the grasp again. It does this sometimes and we don't know why, but it thankfully doesn't happen too often in the solution.
- **There are spikes in the grasp plots for the cube height and angle:** For the most part this is acceptable for your submission if there are only very few brief spikes (the AR marker tracker is not completely foolproof). You can also try rotating the cube so that the AR marker is in a different orientation.