EE106b Lab 2: Grasp Planning with Baxter and Sawyer *

Due date: Monday, March 11th at 11:59pm

Goal

Grasp Planning with Baxter.

The purpose of this lab is to combine many of the topics presented in this course to plan and execute a grasp with Baxter. This will consist of detecting the object to grasp, planning a stable grasp, moving into position, closing the grippers, and lifting. The block diagram in Figure 1 shows the workflow you'll be implementing.

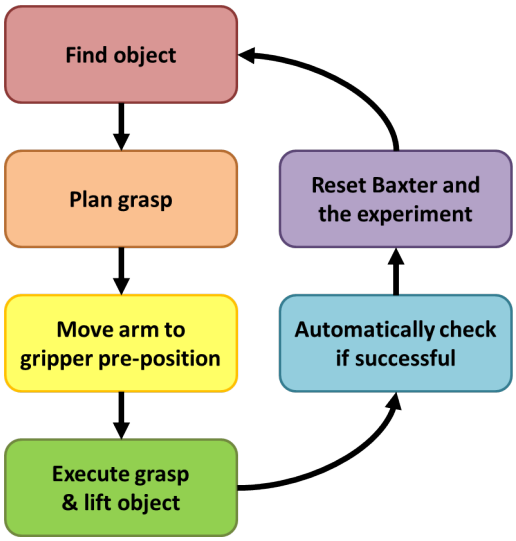

Figure 1: Workflow for Lab 2.

Contents

*Developed by Jeff Mahler, Spring 2017. Expanded and edited by Chris Correa and Valmik Prabhu, Spring 2018, Spring 2019

1 Theory

Here's a quick refresher on grasp theory. We can define a contact as

$$F_{c_i} = B_{c_i} f_{c_i}$$

Where B is the contact basis, or the directions in which the contact can apply force, and f is a vector in that basis. F is the wrench which the contact applies. In our case, we use a soft contact model, which has both lateral and torsional friction components, so the basis is

$$B_{c_i} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

However, in the real world, friction is not infinite. For the contact to resist a wrench without slipping, the contact vector must lie within the *friction cone*, which is defined

$$FC_{c_i} = \{f \in \mathbb{R}^4 : \sqrt{f_1^2 + f_2^2} \leq \mu f_3, f_3 > 0, \|f_4\| \leq \gamma f_3\}$$

However, we want the wrenches a contact point can resist in the world frame, not the contact frame. So we define

$$G_i := Ad_{g_{oc_i}^{-1}}^T B_{c_i}$$

A grasp is a set of contacts, so we define the wrenches (in the world frame) a grasp can resist as:

$$F_o = G_1 f_{c_1} + \cdots + G_k f_{c_k} = \begin{bmatrix} G_1 & \cdots & G_k \end{bmatrix} \begin{bmatrix} f_{c_1} \\ \vdots \\ f_{c_k} \end{bmatrix} = Gf$$

A grasp is in *force closure* when finger forces lying in the friction cones span the space of object wrenches

$$G(FC) = \mathbb{R}^p$$

Essentially, this means that any external wrench applied to the object can be countered by the sum of contact forces (provided the contact forces are high enough).

2 Logistics and Lab Safety

We expect that all students in this class are mature and experienced at working with hardware, so we give you much more leeway than in EECS 106A. Please live up to our expectations.

2.1 Groups, Robots and Accounts

Remember that groups must be composed of 2-3 students, at least one of whom must have completed EECS 106A. Robots should be reserved on the robot calendar here. The rules are

1. **Groups with a reservation have priority.** You can go in and use (or continue using) the hardware whenever you like if no one has a reservation. However, you must pass along the hardware once the next reservation starts. Please be respectful and try to plan ahead; it's not cool to take the first twenty minutes of another group's section winding down.
2. **Do not reserve more than two hours at a time.** This is shared hardware and we want to ensure that all groups have enough time to complete the lab.
3. **One computer per group (if needed)** We only have ten lab computers, and have around thirty students. If more than ten students want to use the computers, please try to limit yourself to one computer per group. In addition, groups with robot reservations have priority on the computers next to their respective robots. Groups can accomplish working in parallel by using personal computers, file-sharing software like git, remoting into the lab computers, and/or using simulators like Gazebo or RViz.

2.2 Safety

Remember to follow the lab safety rules:

1. **Never work on hardware alone.** Robots are potentially dangerous and very expensive, so you should always have someone on hand to help if something goes wrong. This person should be another 106B student, though in cases when group schedules are highly incompatible, you may talk to a TA to arrange an alternate solution, such as bringing a friend (note that you would be entirely responsibe for this person's conduct).
2. **Do not leave the room while the robot is running.** Running robots must be under constant observation.
3. **Always operate the robot with the E-Stop within reach.** If Baxter is going to hit a person, the wall, or a table, press the E-Stop.
4. **Power off the robot when leaving.** Unless you're trading off with another group, the robots should be powered down for safety. To power on/off the robot, press the white button on the back of the robot.
5. **Terminate all processes before logging off.** Leaving processes running can disrupt other students, is highly inconsiderate, and is difficult to debug. Instead of logging off, you should type

   ```
   killall -u <username>
   ```

   into your terminal, where `<username>` is your instructional account username (`ee106b-xyz`). You can also use `ps -ef | grep ros` to check your currently running processes.
6. **Do not modify robots without consulting lab staff.** Last semester we had problems with students losing gripper pieces and messing with turtlebots. This is inconsiderate and makes the TAs' lives much more difficult.
7. **Tell the course staff if you break something.** This is an instructional lab, and we expect a certain amount of wear and tear to occur. However, if we don't know something is broken, we cannot fix it.

2.3 Questions and Help

If you experience software-related errors, please perform the following:

1. Document exactly what you did leading up to the error (commands, terminal output, etc.)
2. Post on Piazza with a description of the error, the above materials, and a description of what you did to try and fix it.

Chris and Valmik **Will Not** diagnose any programming errors without the above documentation. However, feel free to ask us theoretical questions on piazza, during office hours, or after lecture or discussion.

3 Project Tasks

You'll be using the Baxters and Sawyers in Cory 111.

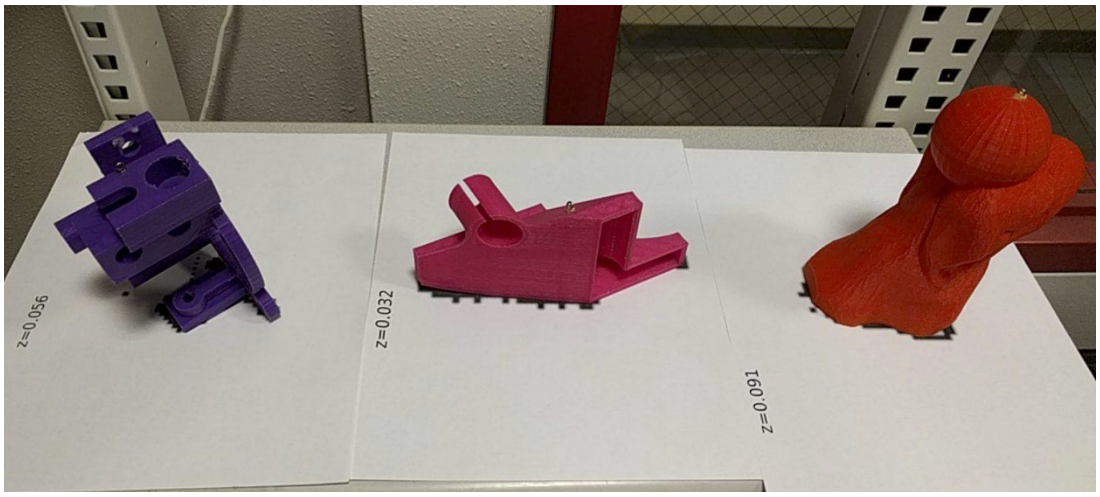

Figure 2: The three test objects with placement templates (gearbox, nozzle, and pawn).

For this lab, we'll ask you to plan five grasps for three different objects pictured in 2. You will then execute each grasp on Baxter or Sawyer, and compare the success rate on the physical system with the success rate predicted by several grasp quality metrics. Note: Please try to use the same robot for all the grasps on each object, to ensure reliable data collection. The tasks are formally listed below:

1. Plan five grasps for each of three objects:
   (a) Sample candidate point grasps from a 3D mesh model of each object by taking pairs of the vertices.
   (b) Compute the quality metrics below for each grasp. Use a soft finger contact models with $\mu = 0.5$ and $\gamma = 0.1$.
      i. Force closure
      ii. Resistance of gravity assuming an object mass of $0.25kg$ and center of mass at the origin.
      iii. A metric of your choosing. Examples include Ferrari Canny or Robust Force Closure. (Grad students should choose Robust Force Closure)
   (c) Compute the gripper pose relative to the object for each grasp.
   (d) Choose a set of five gripper poses based on their grasp metric score. Make sure the gripper will not collide with the object.
   (e) Determine the hand pose in the robot's frame of reference using AR markers.
   (f) Control the baxter arm and move the hand into the proper position.
   (g) Close the grippers.
   (h) Lift the object and determine whether or not the grasp was successful.
2. Execute each grasp 5 times with the object in different locations in each run. Measure the success rate according to:
   (a) Lifting
   (b) Lifting and placing the object in a new location without toppling
3. (Grad Students) Execute some grasps at random x times each. Record the number of successes, and which grasp you executed. Then run a parameter sweep of your Robust Force Closure parameters to find the metric parameters which best fit reality.

4 Deliverables

To demonstrate that your implementation works, deliver the following:

1. Videos of your implementation working. Provide a link to your video in your report.
2. Provide the code by adding "berkeley-ee106b" as a collaborator to a *private* github repo.
3. Submit a detailed report with the following:
   (a) Describe your approach and methods in detail, including a formal description of the metric you decided to use.
   (b) Let us know what of any difficulties and how you overcame them.
   (c) Summarize your results in both words and with figures. In particular, show:
   - Methods for implementing each of the blocks in Figure 1
   - Descriptions of how effective your approach and your grasps were.
   - Plots of the success rates for the different objects.
   - Discussion of physical principles that appear to differentiate the successful grasps from the failures.
   - Discussion of how performance differed across the objects. Which object was the hardest? What was the easiest?

   (d) Propose an extension to improve performance. This can be a new grasp quality metric, a new way of framing the problem (grasping based on images), or hardware modifications to the parallel jaws (must use two fingers). Find and summarize two publications supporting your proposal.
4. **BONUS:** In addition to your report, write a couple paragraphs detailing how the lab documentation and starter code could be improved. This course is evolving quickly, and we're always looking for feedback. This should be a separate section in the report, and only well-considered, thoughtful feedback will merit full credit.

5 Getting Started

We assume that you've already completed all the configuration steps from labs 0 and 1, but there are some additional things you need for lab 2.

5.1 Configuration and Workspace Setup

Note: *we expect all students to already be familiar with configuration procedures, either through EECS 106A, Lab 0, or some other place. Therefore, we won't be providing any explanations in this section. Please consult Lab 0 if you require a refresher*
Type the following commands:

```
cd ~/ros_workspaces
mkdir ~/ros_workspaces/lab2_ws
mkdir ~/ros_workspaces/lab2_ws/src
cd ~/ros_workspaces/lab2_ws/src
catkin_init_workspace
git clone https://github.com/chriscorrea14/lab2_pkg.git
cd lab2_pkg
python setup.py develop

cd ..
catkin_make
source devel/setup.bash
```

5.2 Working With Robots

For this lab, you'll be working with Baxter or Sawyer. In Lab 1, we only used Baxters because of the non-existant Sawyer Kinematics and Dynamics Library. Here we can use both platforms. The initial setup steps are the same. First link the `baxter.sh` file to your workspace. Type

```
ln -s /scratch/shared/baxter_ws/baxter.sh ~/ros_workspaces/lab2_ws/
```

To setup your robot, run `./baxter.sh [name-of-robot]` in your folder (where `[name-of-robot]` is either `asimov`, `ayrton`, or `archytas` for the Baxters; or `alan` or `ada` for the Sawyers). Note, you should no longer need to add the `.local` extension. Then run `source devel/setup.bash` so the workspace is on the `$ROS_PACKAGE_PATH`.

5.3 Setting up Dependencies

5.3.1 AUTOLAB Python Modules

You will use three Berkeley AUTOLAB Python modules:

1. *autolab_core.* Common functions and convenient wrappers for rigid transformations.
2. *trimesh.* Great mesh processing package.
3. *visualization.* Visualization tools in 3D.

You could either install the following in a virtual environment or use the –user flag. If you don't know how to use a virtual environment, read this tutorial. It'll tell you what they're used for, and why.
You'll be installing these in a folder called `modules` in your home directory. To install these, run the following commands:

```
cd
mkdir modules
cd modules
# INSTALL AUTOLAB_CORE
git clone https://github.com/BerkeleyAutomation/autolab_core.git
cd autolab_core
python setup.py develop --user
cd ..
# INSTALL trimesh
git clone https://github.com/BerkeleyAutomation/trimesh.git
cd trimesh
python setup.py develop --user
cd ..
# INSTALL VISUALIZATION
git clone https://github.com/BerkeleyAutomation/visualization.git
cd visualization
python setup.py develop --user
```

5.4 Object Tracking with AR Markers

Baxter will determine the position of the objects by tracking the AR marker on a template pictured in Figure 2 and using a calibrated transformation from the AR marker to the object. The objects and templates are stored in the black cabinet on the Baxter side of the lab. Please put the objects and templates away after you use them.

Figure 3: Example object detections with AR markers.

To set up object tracking, first configure the Baxter cameras to be the correct resolution for AR tracking. This must be run **every time** you start the AR marker tracking.

```
rosrun baxter_tools camera_control.py -o left_hand_camera -r 1280x800
rosrun baxter_tools camera_control.py -o right_hand_camera -r 1280x800
```

Then launch the AR tracking for the left hand by running

```
roslaunch lab2_pkg baxter_left_hand_track.launch
```

You may want to use one arm to locate the AR tag, and the other to manipulate the object. There are two ways to find the object position:

1. Use the ros node that is automatically started when you run `baxter_left_hand_track.launch` You will have to modify it to publish the right transform.
2. Manually compute the pose of the object in your `main.py` script, according to the ar tag, using the `lookup_tag()` function provided.

Note: When first running your code, do not put the objects on the templates. The objects can break easily, so be careful with them.

5.5 Starter Code

Main There are many parameters at the beginning of `main.py`. You should make sure you know what these are, so you can change them as you see fit. You will need to change the part which selects which OBJ file to load and which tag to look for. You could also use the `nodes/object_pose_publisher.py` script to constantly broadcasts a tf to the object. If you use this script, you'll have to ensure the `t_ar_object` variables are set correctly.
You should fill out `execute_grasp()`. This function takes in the desired hand position relative to the base frame. Then it moves the gripper from its starting orientation to some distance behind the object, then move to the hand pose in world coordinates, closes the gripper, then moves up. You will use moveit to control the robot, using the `PathPlanner()` class from Lab 1 (we've already added it to the `scripts` folder for you). **You should first execute the grasp without the object, so as to not break the object if something goes wrong.**

Policy Some object meshes have a very sparse number of vertices. A mesh of a cylinder will likely only have vertices on the edges between the round part and the circular face. Therefore, you should sample vertices on the surface of the mesh using `trimesh.sample.sample_surface_even()`
You should first start by implementing `sample_grasps()`. This function samples N pairs of mesh vertices, and considers a grasp where each of the Baxter's / Sawyer's fingers contact the mesh at the vertices. Then you should fill out `score_grasps()`, which should return a score for each grasp, based on grasp quality (computed in `grasp_metrics.py`). Then you should take the top n, and return each pair of points into a hand pose for the baxter (computed in `vertices_to_baxter_hand_pose()`).

Metrics You should then implement the metrics in `grasp_metrics.py` to score these randomly sampled pairs of vertices, and then return the K best pairs. Each metric will take in two contact points and return the grasp quality at these points using what you learned from Chapter 5 of MLS.
The Force Closure metric is defined in the book. Gravity Resistance is simply computing whether the grasp can resist gravity pulling down on the object, assuming a mass in kg. You can use whatever metric you would like for the custom metric, however we recommend Ferrari Canny or Robust Force Closure.

Robust Force Closure To compute the Robust Force Closure metric for a single grasp, you should take n samples of each grasp. For each sample, you should vary parameters like friction parameters, hand position, hand orientation, and any other parameters you see fit. You'll have to decide how much noise to add, what kind of noise (gaussian, uniform, etc). You should report the fraction of samples which are in force closure. To tune the parameters, you should execute the same grasp a certain number of times on the baxter / sawyer. Do this for multiple objects and multiple grasps. Record exactly which grasp you executed (contact points, normals, etc). Then run a search over the parameters you varied to find the set of parameters which matches the fraction of grasps which succeeded in the real world.

Remember to look at `utils.py` for any useful functions.

Table 1: Point Allocation for Lab 2

Section	Points
Video	3
Code	3
Methods	10
Results: Force Closure Metric	5
Results: Gravity Resistance Metric	5
Results: Custom Metric	10
Discussion of Results	10
Proposed Extensions	4
Parameter Sweep	10*
Bonus Difficulties section	5*

Summing all this up, for undergrads this mini-project will be out of 50 points, with an additional 15 points possible. For grad students, the mini-project will be out of 60 points, with an additional 5 points possible.

6 Submission

You'll submit your writeup(s) on Gradescope and your code by adding "berkeley-ee106b" as a collaborator to a github repo. 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 Improvements

If you notice any typos or things in this document or the starter code which you think we should change, please let us know here. Next year's students will thank you.