C106B Project 3: Grasp Planning with Baxter and Sawyer *

Due date: Wednesday, April 6th at 11:59pm

Goal

Grasp Planning with Baxter and Sawyer

The purpose of this project is to combine many of the topics presented in this course to plan and execute a grasp with a manipulator in the lab. 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.

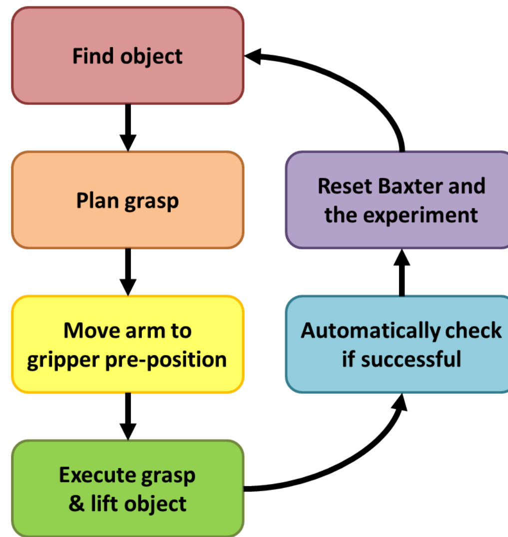

Figure 1: Workflow for a single grasp.

Contents

1 Theory **2**

1.1 Discretizing the Friction Cone . . . 2

1.2 Grasp Metrics . . . 3

1.2.1 Gravity Resistance . . . 3

1.2.2 Ferrari-Canny . . . 4

1.2.3 Robust Force Closure . . . 5

2 Project Tasks **5**

3 Deliverables **7**

4 Getting Started **8**

4.1 GitHub Classroom . . . 8

4.2 Pulling starter code updates . . . 8

4.3 Configuration and Workspace Setup . . . 9

4.4 Working With Robots . . . 9

4.4.1 Working with Baxter . . . 9

4.4.2 Working with Sawyer . . . 10

4.5 Starter Code . . . 11

*Developed by Jeff Mahler, Spring 2017. Expanded and edited by Chris Correa and Valmik Prabhu, Spring 2018, Spring 2019. Further expanded and developed by Amay Saxena and Tiffany Cappellari, Spring 2020. 2020 and 2019 versions reworked into 2022 version by Riddhi Bagadiaa and Jay Monga.

5 Scoring 11

6 Submission 12

7 References 13

1 Theory

Here's a quick refresher on grasp theory drawn from [1]. 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

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

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 that a contact point can resist in the world frame, not the contact frame. So we define

$$G_i := Ad^T_{g_{oc_i}^{-1}} B_{c_i}$$

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).

1.1 Discretizing the Friction Cone

All of the grasp metrics we use are structured as optimization problems with $f \in FC$ as a constraint. However, set of constraints that make up the friction cone for contact i

$$FC_{c_i} = \begin{cases} \sqrt{f_1^2 + f_2^2} \leq \mu f_3 \\ f_3 > 0 \\ |f_4| \leq \gamma f_3 \end{cases}$$

contains a quadratic constraint. Thus, none of the optimization problems will be convex. In order to guarantee a solution, you can approximate the (conical) friction cone as a pyramid with n vertices. The level sets of the friction cone are circles, but the level sets for this convex approximation are n sided polygons circumscribed by the circle. Thus, the interior of this convexified friction cone is a conservative approximation of the friction cone itself.

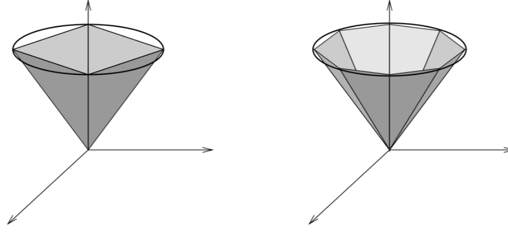

Figure 2: Approximations of the friction cone. From section 5.3 of [1]

Any point in the interior of this pyramid can be described as a sum of

$$f = \alpha_0 f_0 + \sum_{i=1}^{n} \alpha_i f_i = F\alpha$$

where f_i are the edges of the pyramid and f_0 a straight line in z, and the weights α are all non-negative. Here, we can write a composite matrix F with the f_i vectors as its columns.

Thus you can represent any point in the friction cone as a set of linear constraints with these new grasp forces. One thing you should note is that the magnitude of $f^{\perp}$ is no longer just α_0. You'll have to pay close attention to the way that you define your f_i so that you'll be able to calculate $f^{\perp}$ from α. You'll also have to figure out how to add the soft contact constraint ($|f_4| \leq \gamma f_3$) into your new approximate friction cone.

With this approximation, the condition that $f \in FC$ is equivalent to the pair of linear constraints $\{f = F\alpha, \alpha \geq 0\}$ (where this inequality is understood to be element-wise). In most applications, you can turn a condition in terms of f with the constraint $f \in FC$ into a condition in terms of α with the constraint $\alpha \geq 0$, by substituting f with $F\alpha$. This is a much more tractable constraint and will turn all of our optimization problems into convex problems that can be easily solved by any convex optimization solver.

1.2 Grasp Metrics

When designing a grasp, you'll need some way of determining its quality. Force closure is helpful, but it's a binary metric (yes it's in force closure or no it isn't) so it's often unhelpful in ranking grasps. Instead, you'll be implementing three common grasp metrics: Gravity Resistance, Ferrari-Canny, and Robust Force Closure.

1.2.1 Gravity Resistance

When grasping objects, often the largest force you'll have to counteract is gravity, so the gravity resistance metric calculates how well your grasp will be able to resist the force of gravity. To define this metric, simply compute

$$\begin{aligned} \widehat{f} = \underset{f}{\operatorname{argmin}} \, & \|f\|_2^2 \\ st. \; & f \in FC \\ & Gf = -W_g \end{aligned}$$

where W_g is the wrench on the object's center of mass due to gravity. The metric is

$$J = \sum_i \widehat{f}_{3_{c_i}} = \sum_i \widehat{f}_{c_i}^{\perp}$$

Where $\widehat{f}_{3_{c_i}}$ is the perpendicular finger force at contact i, and is also denoted as $\widehat{f}_{c_i}^{\perp}$. Remember that $\widehat{f}_{3_{c_i}}$ is constrained to be positive (by the friction cone), so this cost will always be positive. This metric calculates the minimum finger forces needed to resist gravity, and a *lower* cost indicates a *more secure* grasp. If the grasp cannot resist the force of gravity (the problem is infeasible), then the cost is infinity.

Once we incorporate the discretized approximation of the friction cone, this optimization problem becomes a quadratic program. There are some ways of simplifying it further by incorporating the discretization of the friction cone directly into the problem and optimizing over $\alpha \geq 0$. We leave the details of these simplifications up to you. You may use your favourite convex optimization solver (we recommend `cvxpy` or `casadi`).

1.2.2 Ferrari-Canny

The Ferrari-Canny metric [2] is an extension to the force closure metric that captures how much grasp effort is needed to maintain force closure. The paper defines the metric as follows. First, define a *local quality metric* LQ, a function of the wrench space, as

$$
\begin{aligned}
LQ(w) = \max_f \frac{\|w\|}{\|f\|} \\
st.\ f \in FC \\
Gf = w
\end{aligned}
$$

$LQ(w)$ is a measure of how efficiently a given wrench w can be resisted. A wrench is resisted more efficiently if the norm of the input force (at the contact points) required is low. Then, we measure the total quality of the grasp as the worst case efficiency ratio.

$$Q = \min_w LQ(w)$$

We consider the worst case efficiency since we don't, in general, have control over the wrench acting on the body that must be resisted.

This metric is difficult to implement as written, so we will perform some simplifications. First, notice that the the magnitude of the applied wrench scales linearly with the magnitude of the input force vector. Since the grasp metric is the ratio between these two magnitudes, we can equivalently optimize over unit wrenches. Then, LQ can be re-written as

$$
\begin{aligned}
LQ(\widehat{w}) = \max_f \frac{1}{\|f\|} \\
st.\ f \in FC \\
Gf = \widehat{w}
\end{aligned}
$$

defined over unit 6D vectors $\widehat{w}$. Maximizing $1/\|f\|$ is equivalent to minimizing $\|f\|$. So we define a new local quality metric

$$
\begin{aligned}
\widehat{LQ}(\widehat{w}) = \min_f \|f\|_2^2 \\
st.\ f \in FC \\
Gf = \widehat{w}
\end{aligned}
$$

where we have made two important changes. First, we have reciprocated the objective function. Secondly, we have turned the norm into a norm-squared. These two changes do not change the optimal force f, but they do change the value of the objective function. So our final quality metric is

$$Q = \min_{w, \|w\|=1} \frac{1}{\sqrt{\widehat{LQ}(w)}}$$

The advantage of making this change is that now the optimization problem $\widehat{LQ}$ (after incorporating the standard discretization of the friction cone) is a quadratic program and can be solved by any convex optimization solver.

It remains to discuss how to solve the outermost problem Q. Indeed, this overall problem is difficult to solve exactly. So we can find an approximate solution through Monte-Carlo sampling. To implement this metric, we recommend first sampling N 6D unit vectors $\{\widehat{w}_i\}_{i=1}^N$ randomly from the unit sphere, finding $\widehat{LQ}(\widehat{w}_i)$ for each $\widehat{w}_i$, and then picking the smallest $1/\sqrt{\widehat{LQ}}$. This is a randomized approach, so you will get a random result, but by increasing the number of samples N you can reduce the variance of your estimate. You can sample from the unit sphere by first sampling from a rectangle and then normalizing the result. We recommend starting with $N = 1000$, and increasing it if you see high variability. Note that your grasp metrics do not have any tight runtime requirements.

Note that once again, you will find it useful to bring in the discretization of the friction cone into the problem directly and optimizing over α.

You can also think about the Ferrari-Canny metric geometrically. If you define the friction cone when $\|f^\perp\|$ is 1, the Ferrari-Canny metric is the minimum distance between the origin and the convex hull of the friction cone. It's possible to efficiently compute this metric geometrically by examining the polytope you get when you take the level set of the friction cone. However, the implementation is quite involved without using specialized geometry packages, hence the solution above.

1.2.3 Robust Force Closure

This is the grasp metric used in the DexNet paper that you can find here.

The idea behind the Robust Force Closure grasp metric is to quantify *to what extent* a grasp is force closure. For instance, you may imagine that we have two force closure grasps. Naively, there is no way to tell these apart, even though it may be the case if one of the contacts moved even a little bit while executing grasp 1, it would fail to be force closure, whereas with grasp 2, the contacts can handle a fair bit of disturbance before the grasp ceases to be force closure. We would like to quantitatively say that grasp 2 is "better" than grasp 1.

So, instead of simply checking whether a given grasp is force closure or not, we instead introduce some random noise to the grasp, and then ask what the *probability* is that the resultant perturbed grasp is force closure. Let's define this rigorously. We describe a grasp using a pose $\mu \in SE(3)$, which describes where the end effector of the robot should be when the grasp is executed. We additionally have $\mathcal{M} \in \mathcal{O}$, which is a model of the object we are grasping, with $\mathcal{O}$ being the space of all objects. Together, the pose μ and the model $\mathcal{M}$ can be used to check if the specified grasp is in force closure. Let $F : SE(3) \times \mathcal{M} \to \{0, 1\}$ be the proposition:

$$F(\mu, \mathcal{M}) = 1 \text{ if grasp } \mu \text{ on object } \mathcal{M} \text{ is in force closure, else } 0 \tag{1}$$

Let δ be some random disturbance. We are deliberately keeping this vague at this time since there are many ways to perturb a pose in $SE(3)$. Let $\hat{\mu} = \mu + \delta$ be a random variable that models an uncertain pose (our original pose μ with random disturbance δ). The quality Q of the grasp $(\mu, \mathcal{M})$ will be

$$Q(\mu, \mathcal{M}) = \mathbb{P}(F(\hat{\mu}, \mathcal{M}) = 1) \tag{2}$$

In practice, we would implement the above probability by Monte Carlo estimation - taking a large number of samples from the disturbance δ, perturbing μ by each sample, and then checking what fraction of the perturbed grasps are in force closure.

It remains to be discussed how we should construct the random variable δ. We are looking for some way to perturb a given grasp. One straightforward way to do this is to instead perturb the two contact points directly. Let $x, y \in \mathbb{R}^3$ be the two contact points of your grasp. Let $\delta_x, \delta_y \overset{i.i.d}{\sim} \mathcal{N}(0, I\sigma^2)$ be two independent Gaussian random variables in $\mathbb{R}^3$, and construct a new grasp $\hat{\mu}$ by picking contact points $(\hat{x}, \hat{y}) = (\pi(x + \delta_x), \pi(y + \delta_y))$ where π is an operator that projects a point in $\mathbb{R}^3$ to its nearest point on the outer surface of the object $\mathcal{M}$. You could get cleverer by sampling Gaussians from only the planes tangent to the object at the points x and y to come up with your perturbation. The starter code provides some basic utilities that can get you started in finding intersections and projections onto the meshes of objects.

Alternatively, we could find a way to perturb the pose μ as we stated in the algorithm definition. One way of doing this is to sample δ randomly from $se(3)$, and then compute

$$\hat{\mu} = \mu \cdot e^{\hat{\delta}} \tag{3}$$

For instance, you may sample δ as a zero mean Gaussian with uniform variance σ^2 in $\mathbb{R}^6$. This is how robust force closure is implemented in the DexNet paper. Beware, however, that if you do it in this way, then the random variable $\hat{\mu}$ in fact does *not* have expectation μ. Nevertheless, this is a commonly used noise model for $SE(3)$ valued variables.

The one parameter left to tune is the variance σ^2 of the disturbance. We want to pick a variance that reasonably mimics the variation in pose in the real world when a grasp is executed multiple times. If the variance is too low, then practically every sampled grasp will be force closure if the original grasp was. If the variance is too high, then the metric will not be representative of disturbances in the real world.

When implementing this grasp metric, feel free to use either of the above techniques, or design your own disturbance. You should discuss in your report how you went about implementing your metric.

2 Project Tasks

For this project, we'll ask you to plan five grasps for three different objects pictured in Figure 3.
The orientation of the gripper's coordinate frame and the gripper dimensions is shown in Figure 4.

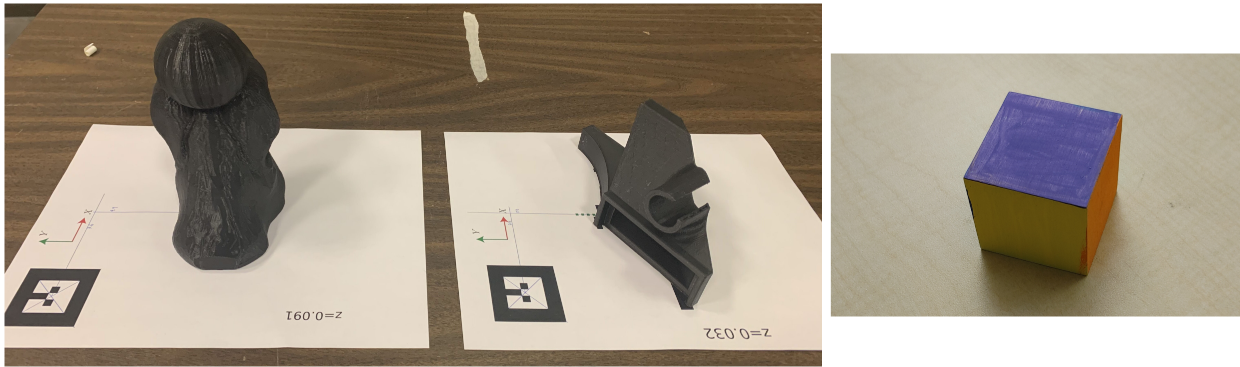

Figure 3: The three test objects with placement templates (pawn, nozzle, cube).

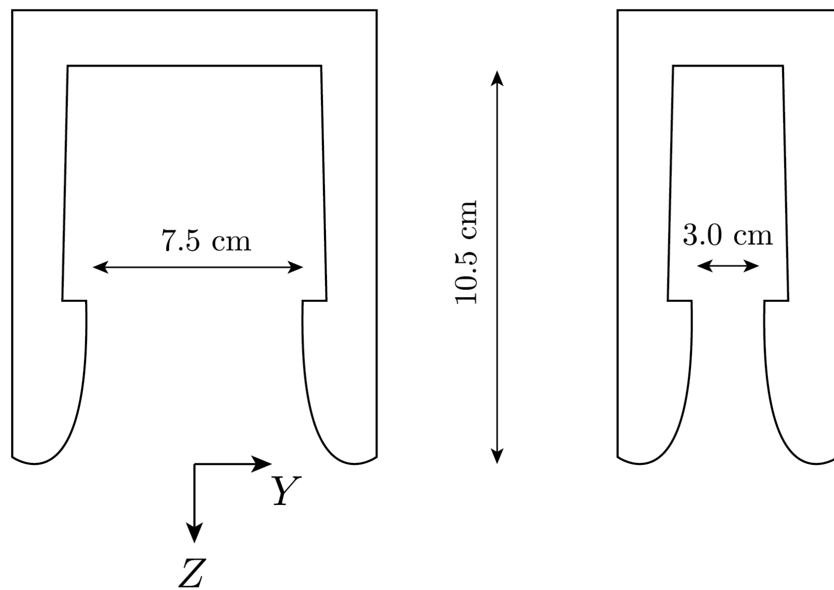

Figure 4: Orientation of the gripper's frame and its dimensions

You will then execute each grasp on a Baxter or Sawyer, and compare the success rate on the physical system with the success rate predicted by several grasp quality metrics. Note: Please use the same robot for all the grasps on each object to ensure reliable data collection.
The tasks are formally listed below:

1. For the cube object only, you will need to use vision to determine the object's size and pose without use of an AR tag. Once these parameters have been determined, you can feed the cube's known geometry into the rest of your grasping pipeline.

2. Plan five grasps for each of three objects with the following three quality metrics (45 grasps total!). Use a soft finger contact models with $\mu = 0.5$ and $\gamma = 0.1$:

   (a) Resistance of gravity assuming an object mass of 0.25 kg and center of mass at the origin

   (b) Ferrari Canny

   (c) Robust Force Closure

3. The general workflow for planning an executing a grasp is as follows:

   (a) Sample candidate point grasps from a 3D mesh model of each object by taking pairs of the vertices.

(b) Compute the quality metric for each grasp.
(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.

4. Measure the (binary) success rates of each grasp according to:
   (a) Lifting
   (b) Lifting and placing the object in a new location without toppling

For graduate students (or extra credit for undergradutes), you will be required to achieve some success in the cube reconstruction and grasping pipeline for full credit (being able to locate, pickup, and place the cube). Every project team must at least attempt this pipeline, but only graduate student teams need to have success for full credit on this portion of the project.

3 Deliverables

To demonstrate that your implementation works, deliver a report with the following. The purpose of these project reports are to gradually scale up to a full conference-style paper, which you'll be writing for your final project. Please format your report using the IEEE two-column conference template. Column suggestions do not account for the figures.

1. Abstract: An abstract, of at most 150 words, describing what you did and what results you achieved. Conveying information concisely is a difficult skill, and we want you to practice it here.

2. Methods: Describe your approach and methods for implementing each of the blocks in Figure 1 in detail. Let us know of any difficulties you encountered and how you overcame them. ($\sim$ 2-4 columns)
   (a) Specifically describe your procedure for reconstructing the size and pose of the cube.
   (b) Describe your implementations of the grasp metrics. You do not need to repeat theory that is already present in the lab document, but you should mention any details necessary for understanding your implementation of the metrics. For instance, if you modified the optimization problem to explicitly incorporate the discretization of the friction cone, explain how you did that. Explain the software you are using as your optimization backend. Explain what parameters you chose for your algorithms and why.
   (c) A detailed description of your grasp planning algorithm.
   (d) For each of 45 grasps, include images of the grasp point visualization outputted by your code and a video of the robot attempting your grasp. All videos should be stitched together into a single video no longer than 3 minutes with each grasp labeled and speed-up factor included.

3. Experimental results: Summarize your results in both words and with figures. The figures should all have descriptions so that they are understandable without needing to read the paper or context. ($\sim$ 1-2 columns)
   (a) For the cube object, compare the size and pose computed by your computer vision algorithm to the actual size and pose of the cube. How accurate (qualitatively and quantitatively) is your algorithm? You may use an AR tag to help measure accuracy of your algorithm.
   (b) For each object, record the scores of each of your planned grasps for each of the three different grasp metrics (15 scores total per object). Summarize your results in a table along with the success rate of each grasp (success or fail for the object).
   (c) Summarize the results for your cube recognition and grasping pipeline. **This should have some success for grad students.**

4. Discussion: ($\sim$ 2-3 columns)

(a) Discuss how performance differed across the objects. Which object was the hardest? Which was the easiest? Why?
(b) Discuss what difficulties you faced when designing your grasp planning algorithm. Were there any edge cases you did not anticipate?
(c) Discuss any physical principles that appear to differentiate the successful grasps from the failures.
(d) Discuss how predictive each of your grasp metrics were for the success or failure rate of the given grasps. Do the grasp metrics seem to have predictive power? Or do the grasps seem to succeed/fail in real life for other reasons?
(e) Discuss the strengths and limitations of your custom grasp planning algorithm. Do you use any heuristics that depend on assumptions that may not always hold? Is your planner computationally efficient? Can you think of any failure cases for your planner?
(f) Discuss what worked well for the cube reconstruction and grasping pipeline, and what could be improved.

5. A bibliography section citing any resources you used. This should include any resources you used from outside of this class to help you better understand the concepts needed for this project. Please use the IEEE citation format (https://pitt.libguides.com/citationhelp/ieee).

6. Page Limit: To prepare you for conference-style papers (and to protect the graders' time), reports are **limited to 6 pages**, not including the bibliography or appendix. While we are not enforcing a figure limit, we expect your figures to be concise, informative, and readable, with detailed captions for each.

7. Appendix:

   (a) 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.
   (b) Videos of your implementation working. Provide a link to your video in your report.
   (c) Bonus: What do you think the learning goals of this assignment are? How effective was this assignment at fostering those learning goals for you? How can the lab documentation and starter code be improved for the future?

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.

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 3 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-3-your-team-name. Let us know if anything went wrong.

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

4.2 Pulling starter code updates

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/proj3_pkg.git
git pull public main
git push origin main
```

4.3 Configuration and Workspace Setup

Set up your workspace by navigating to the proj3_pkg directory and running

```
pip install trimesh[easy]==3.6.0 vedo==2020.3.2 shapely casadi
```

You will also need to download the AR tracking package into your workspace.

```
git clone https://github.com/sniekum/ar_track_alvar.git
cd ar_track_alvar
git checkout kinetic-devel
```

4.4 Working With Robots

Set up your workspace for using the type of robot you have been assigned. If you need a refersher on how to do this, refer to the “ Setting up your environment for Rethink robots” section of the Robot Usage Guide.

4.4.1 Working with Baxter

To test that the robot is working, run the following commands:

1. Enable the robot

```
rosrun baxter_tools enable_robot.py -e
```

2. Test motion

```
rosrun baxter_tools tuck_arms.py -u
```

3. Start the trajectory controller

```
rosrun proj3_pkg start_joint_trajectory_action_server.py
```

4. Check that MoveIt! works (Note: use this launch file when using MoveIt! to check your trajectories in the first part of this project instead of Baxter’s built in launch file).

```
roslaunch proj3_pkg demo_baxter.launch right_electric_gripper:=true
```

Omit the last argument if the robot lacks a gripper.

Baxter will determine the position of the objects by tracking the AR marker on a template pictured in Figure 3 and using a calibrated transformation from the AR marker to the object.

Figure 5: Example object detections with AR markers.

To set up object tracking on a Baxter, first configure the Baxter cameras to be the correct resolution for AR tracking. Note that you can only turn on one of the wrist cameras at a time (either left or right). You will need to close a running wrist camera before you can open the other one. Open a camera by running

```
rosrun baxter_tools camera_control.py -o [name-of-camera] -r 1280x800
```

and close it by running

```
rosrun baxter_tools camera_control.py -c [name-of-camera]
```

where `[name-of-camera]` is `left_hand_camera`, `right_hand_camera`, or `head_camera`.

Then launch the AR tracking for the left hand by running

```
roslaunch proj3_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.

4.4.2 Working with Sawyer

The Sawyers conveniently have Logitech cameras mounted to them. You should first download the `usb_cam` package into your workspace.

```
git clone https://github.com/ros-drivers/usb_cam.git
```

If you haven't already done so, move the `camera_info` folder from Lab 0 into your `~/.ros` directory.
To test that the robot is working, run the following commands:

1. Enable the robot

   ```
   rosrun intera_interface enable_robot.py -e
   ```

2. Start the trajectory controller

   ```
   rosrun intera_interface joint_trajectory_action_server.py
   ```

3. Run MoveIt! to test motion and planning

   ```
   roslaunch sawyer_moveit_config sawyer_moveit.launch electric_gripper:=true
   ```

4. To start AR tracking make sure the webcam is plugged into your computer and run

   ```
   roslaunch proj3_pkg sawyer_webcam_track.launch
   ```

On RViz you should be able to see images in the Camera display from the `/usb_cam/image_raw` topic. If there is no image displayed after selecting the topic, try placing an AR tag in view of the webcam and restart RViz. We have provided two other launch files for AR tracking with Sawyer: `sawyer_head_track.launch` and `sawyer_wrist_track.launch`. You may use any of these to track AR tags but we recommend `sawyer_webcam_track.launch` (just remember to be careful with the webcam cord).
When running `main.py` with a Sawyer don't forget to add the `--sawyer` flag!

4.5 Starter Code

main.py 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.

Take a look at `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 Project 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.**

You will be filling out `locate_cube` in order to determine the size and pose of the cube in view of the camera. We recommend using builtin OpenCV tools for locating corners and edges of the cube (each face of the cube is a different color to facillitate this).

policies.py 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()`).

grasp_metrics.py 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 and we have already provided the code for you. You should take a look and make sure you understand how it works. You will need to implement the Gravity Resistance, Ferrari Canny, and Robust Force Closure metrics.

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

5 Scoring

Table 1: Point Allocation for Project 3

Section	Points
Page Limit	5
Figures: Quality and Readability	5
Abstract	5
Methods	10
Results: Cube Recognition	5
Results: Gravity Resistance Metric	5
Results: Ferrari Canny Metric	5
Results: Robust Force Closure Metric	10
Results: Cube Recognition and Grasping Pipeline	10*
Discussion	10
Bibliography	4
Code	3
Video	3
Bonus: Future Improvements	5*

Summing all this up, for undergrads this project will be out of 70 points, with an additional 15 points possible. For

grad students, the project will be out of 80 points, with an additional 5 points possible.

6 Submission

You'll submit your writeup 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 References

[1] R. M. Murray, S. S. Sastry, and L. Zexiang. *A Mathematical Introduction to Robotic Manipulation*. 1st. USA: CRC Press, Inc., 1994.

[2] C. Ferrari and J. F. Canny. “Planning optimal grasps.”