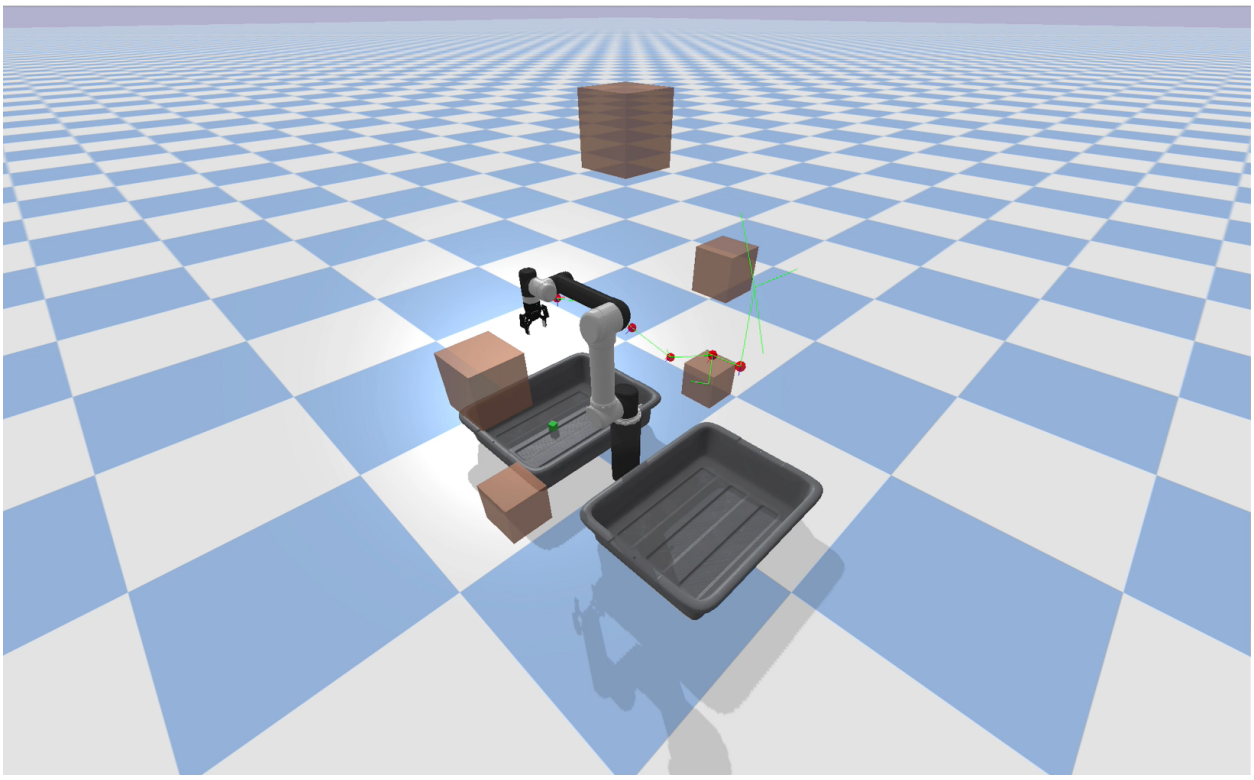
Homework 3

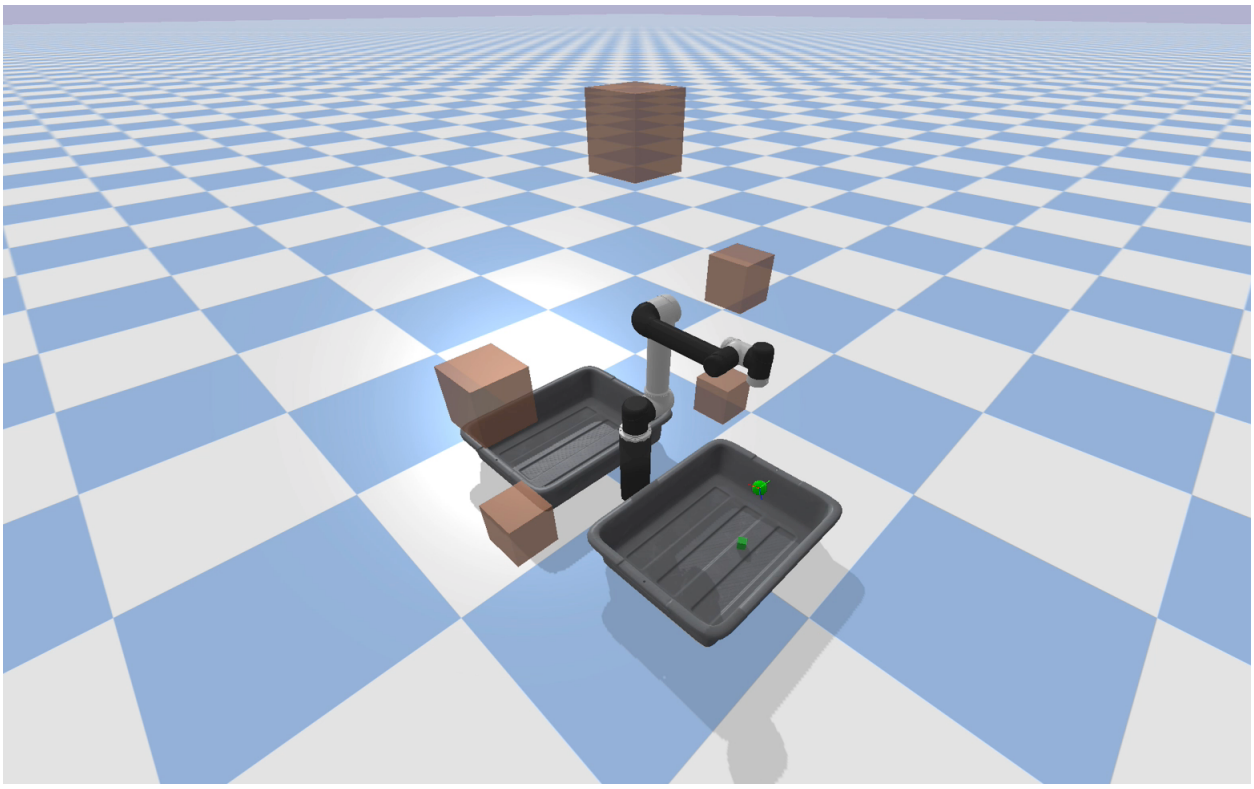
Out: Friday, March 24

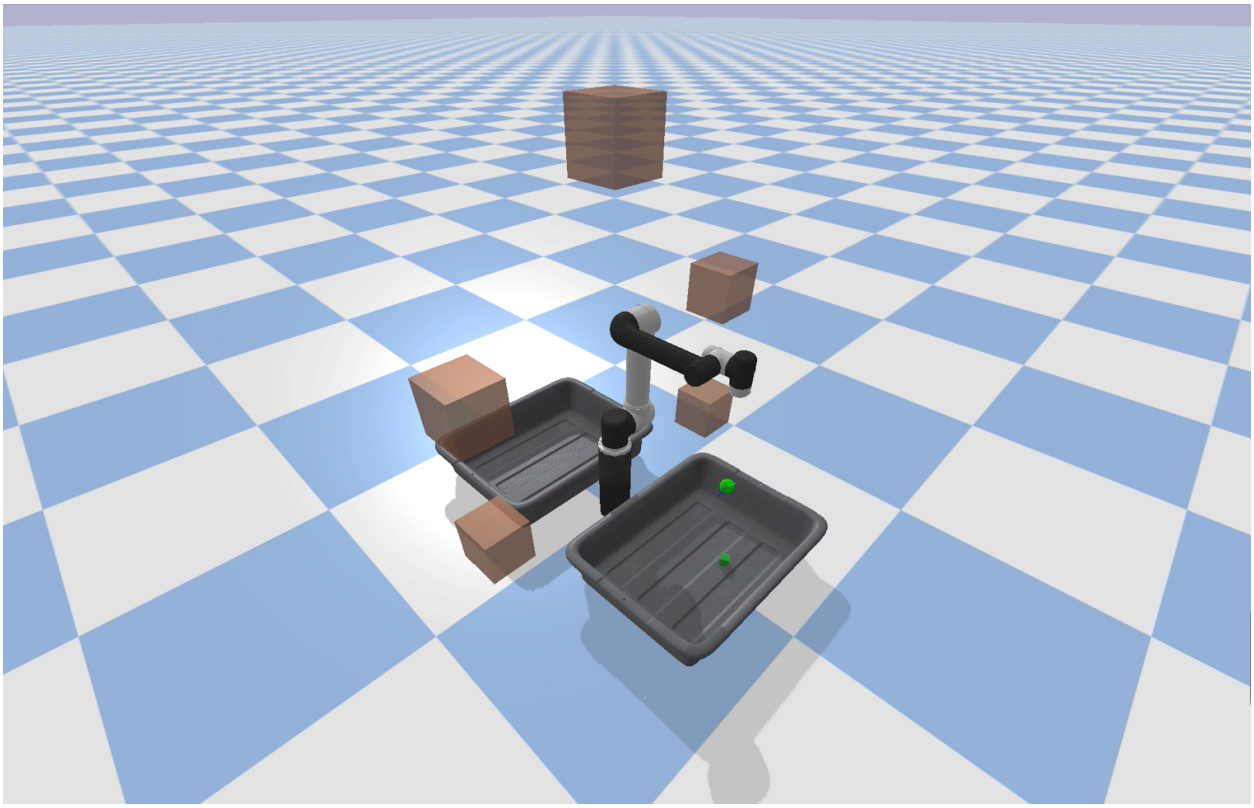
Due: Friday, April 07 @ 5:00 pm EST

1 – Basic Robot Movement

```
Console
<terminated> main.py [/Users/stephen/mambaforge/envs/comsw4733_hw3/bin/python]
pybullet build time: Jul 21 2022 19:56:13
Version = 4.1 ATI-4.8.101
Vendor = ATI Technologies Inc.
Renderer = AMD Radeon Pro 5300 OpenGL Engine
b3Printf: Selected demo: Physics Server
startThreads creating 1 threads.
starting thread 0
started thread 0
MotionThreadFunc thread started
[Grasping] 3 / 3 cases passed
numActiveThreads = 0
stopping threads
Thread with taskId 0 exiting
Thread TERMINATED
destroy semaphore
semaphore destroyed
destroy main semaphore
main semaphore destroyed
```

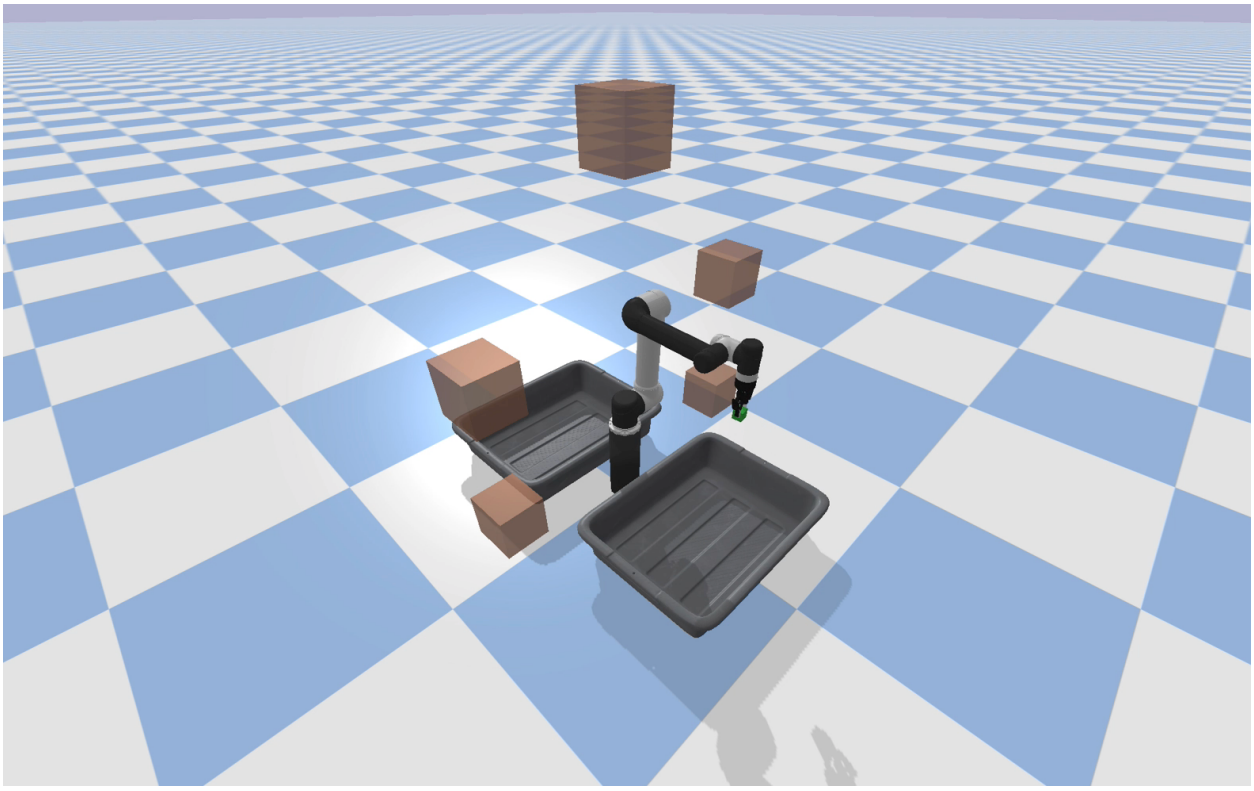


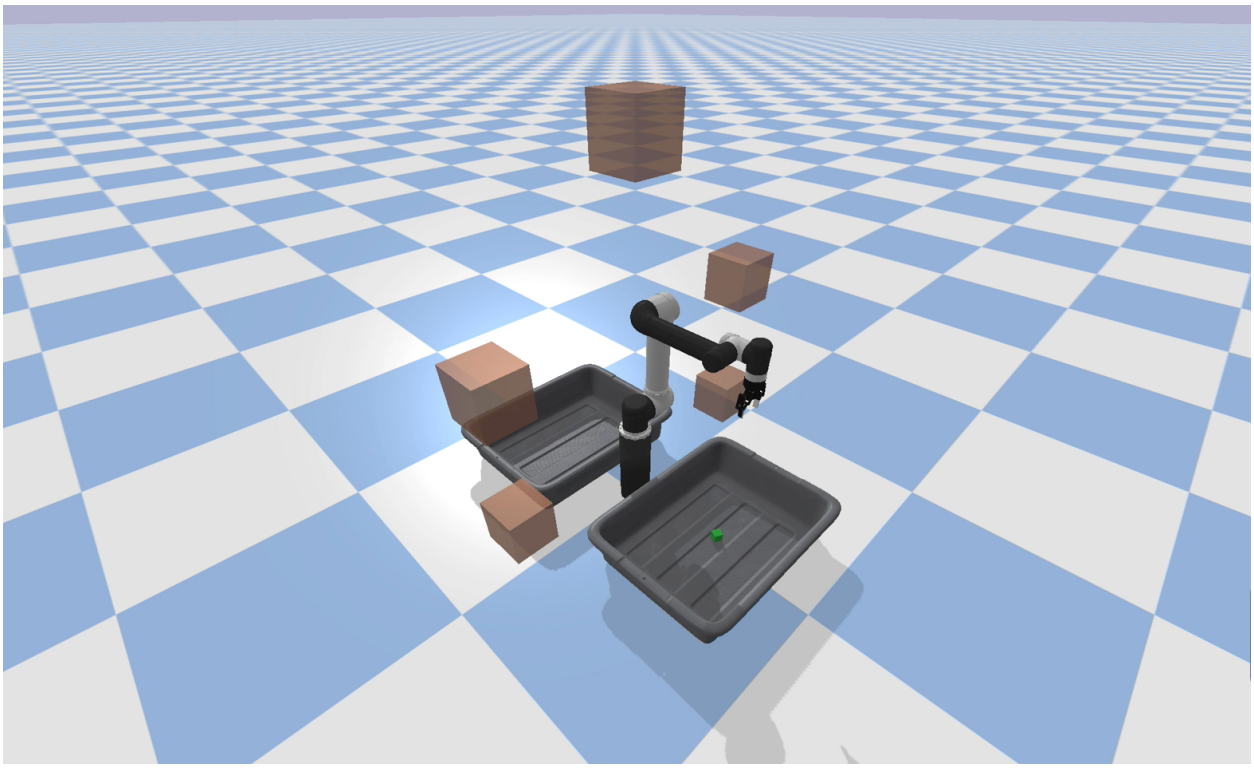


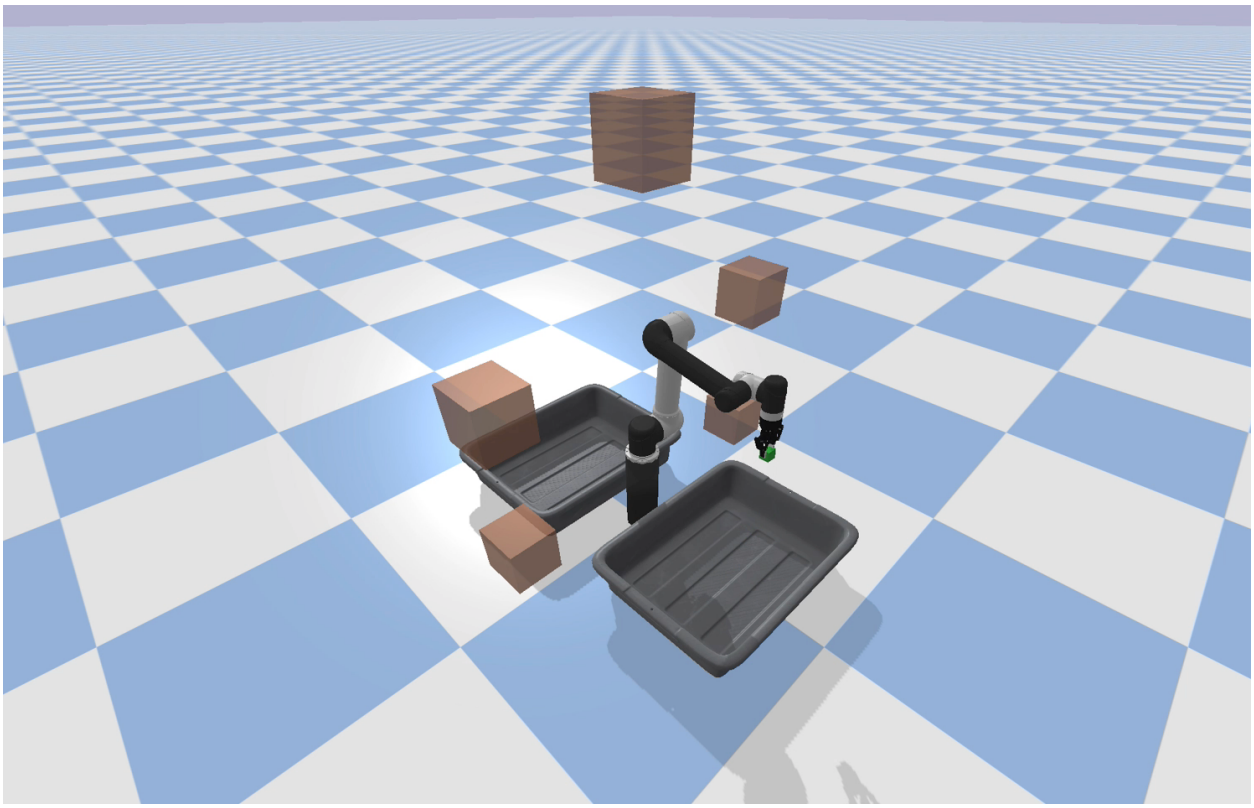


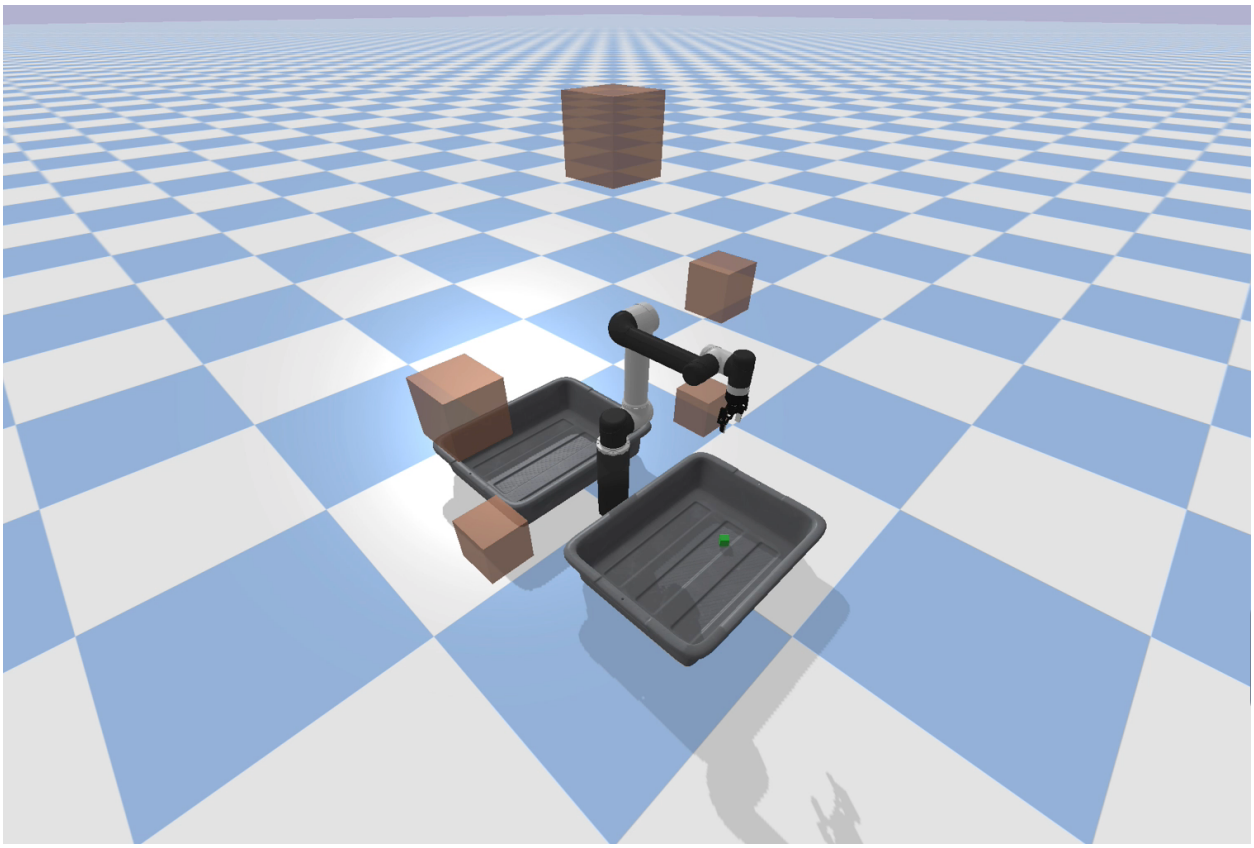
2 – Grasping

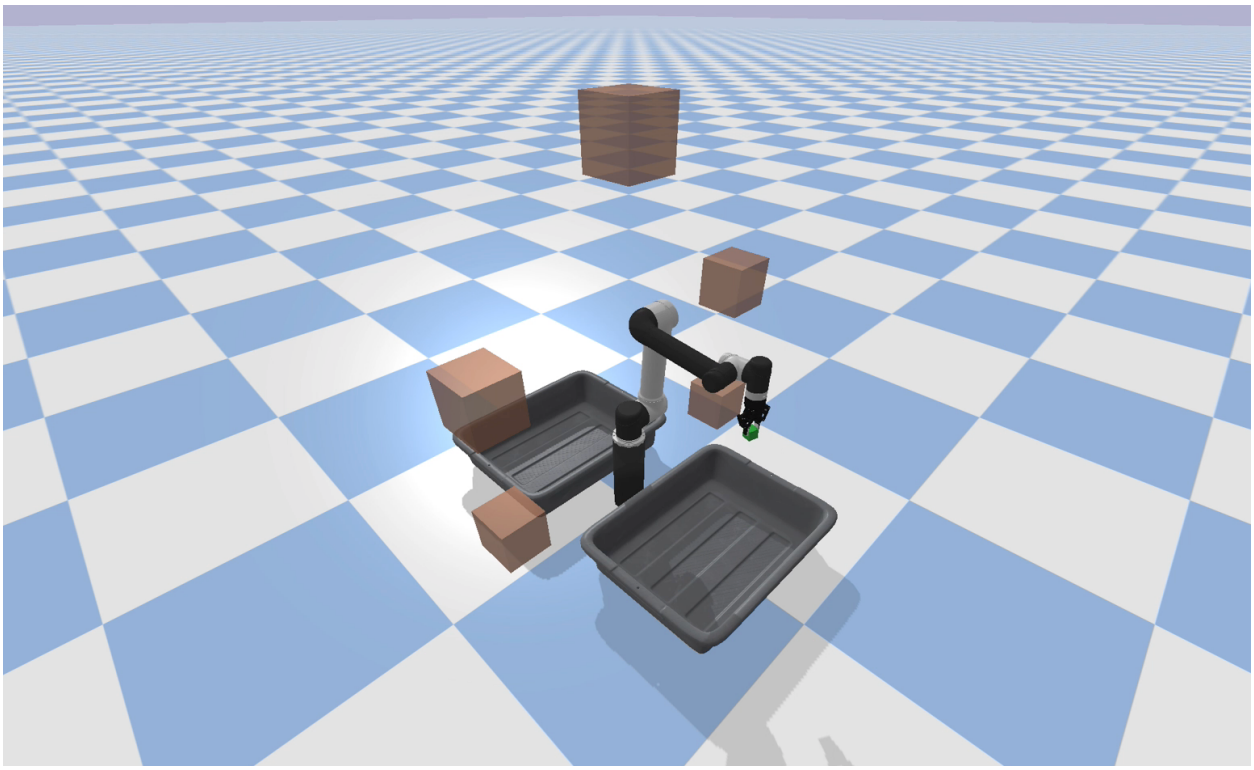
```
Console
<terminated> main.py [/Users/stephen/mambaforge/envs/comsw4733_hw3/bin/python]
pybullet build time: Jul 21 2022 19:56:13
Version = 4.1 ATI-4.8.101
Vendor = ATI Technologies Inc.
Renderer = AMD Radeon Pro 5300 OpenGL Engine
b3Printf: Selected demo: Physics Server
startThreads creating 1 threads.
starting thread 0
started thread 0
MotionThreadFunc thread started
[Robot Movement] 3 / 3 cases passed
numActiveThreads = 0
stopping threads
Thread with taskId 0 exiting
Thread TERMINATED
destroy semaphore
semaphore destroyed
destroy main semaphore
main semaphore destroyed
```

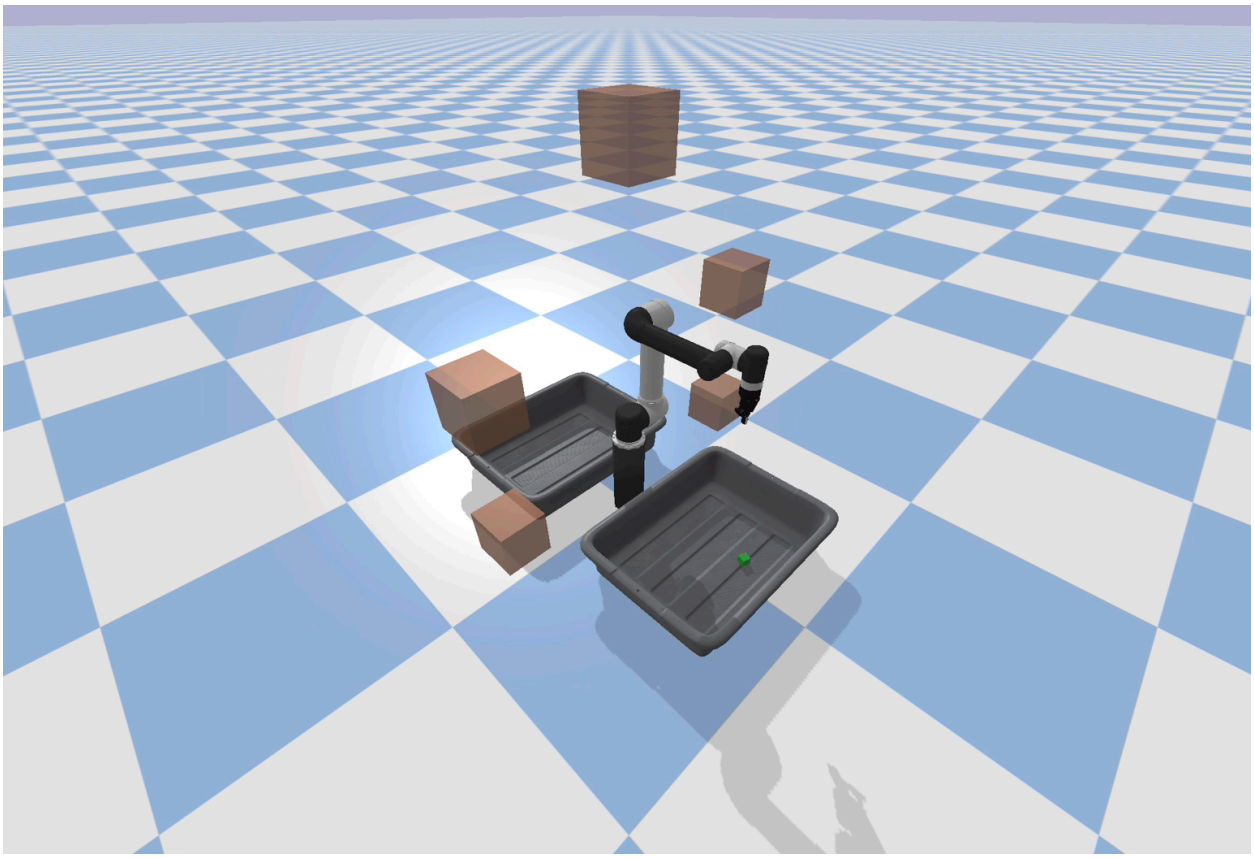






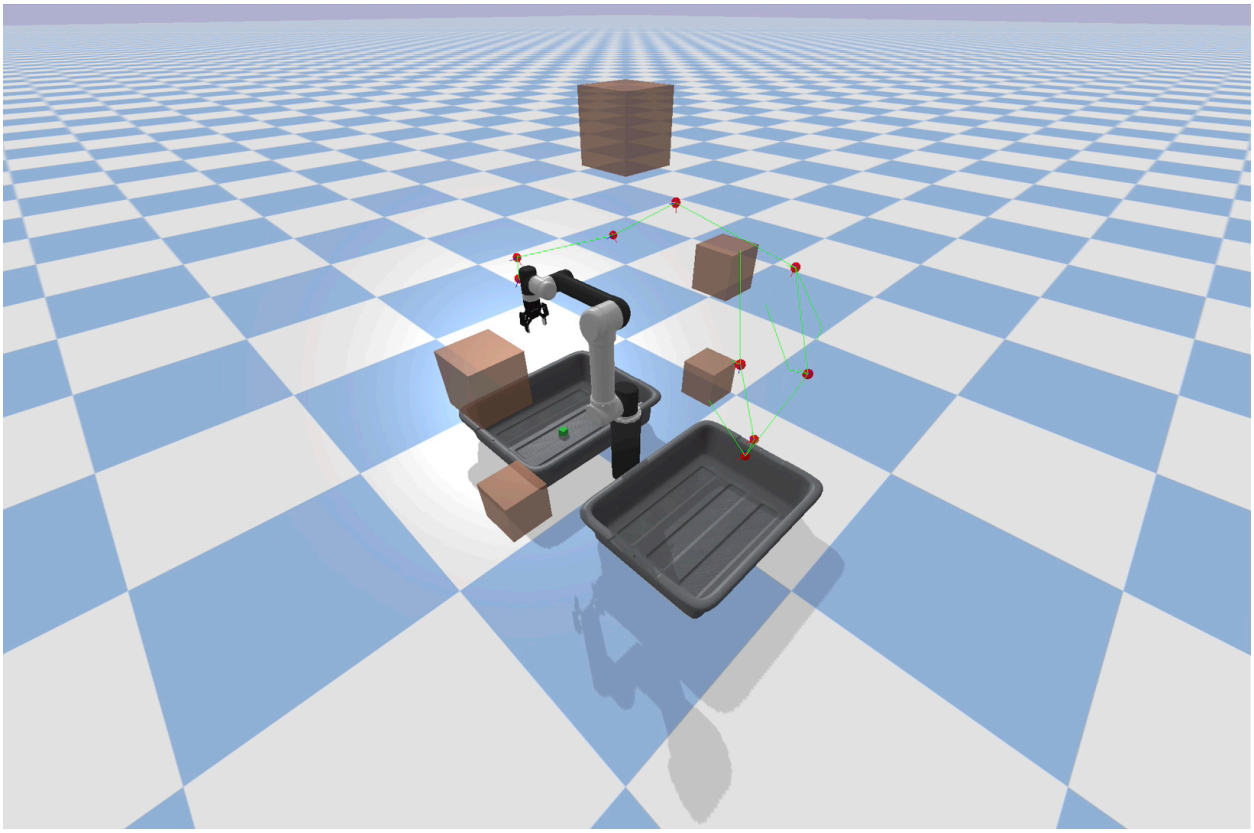


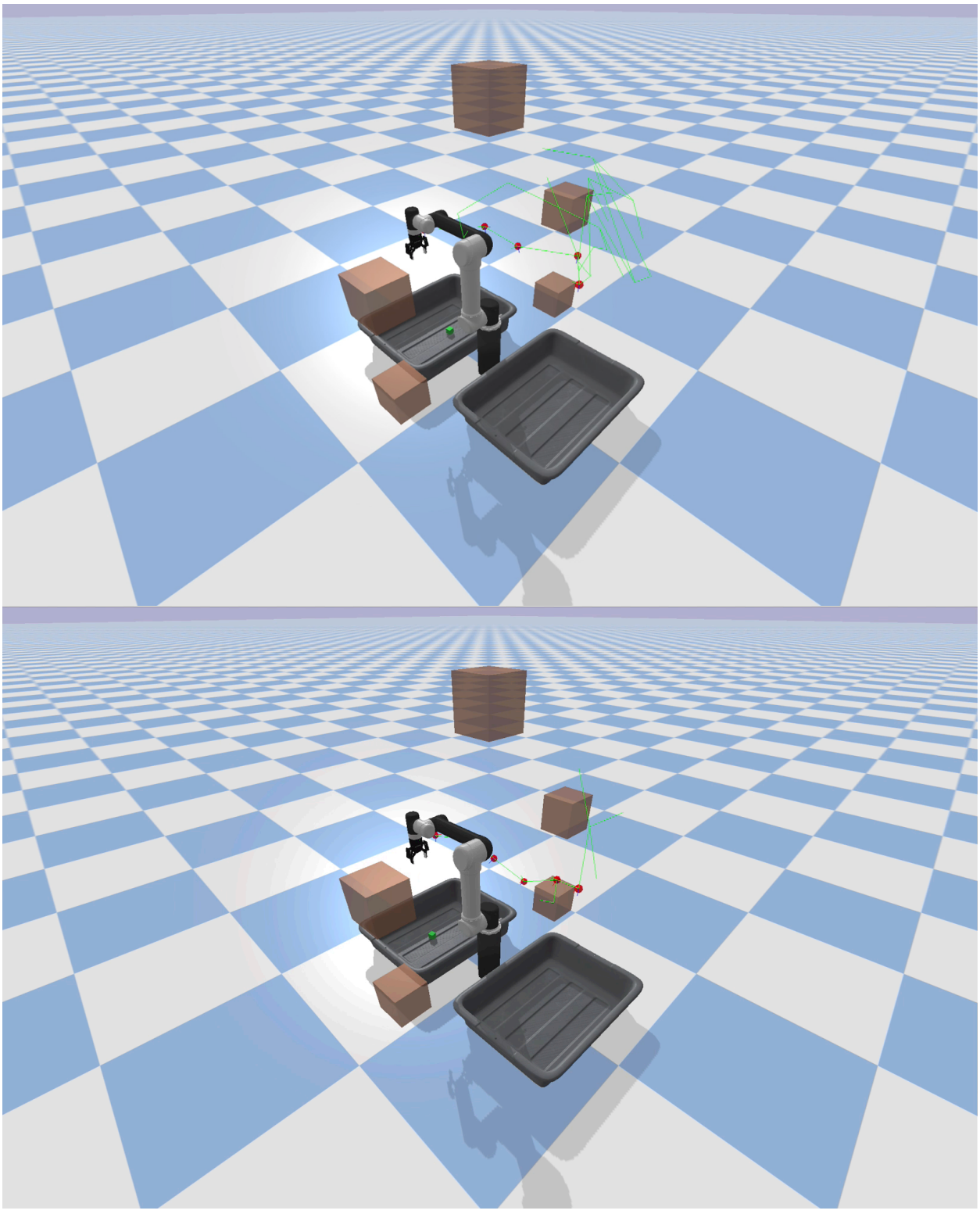




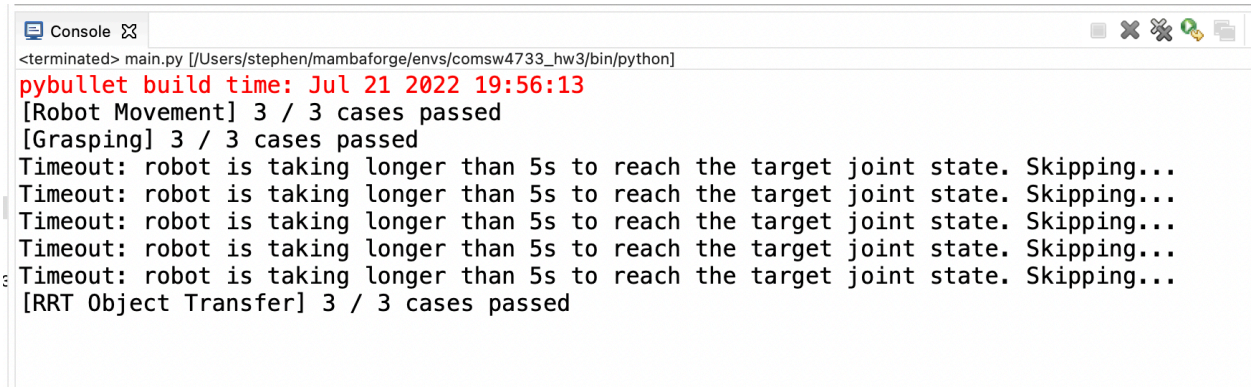
3 – Path Planning Using RRT

```
Console
<terminated> main.py [/Users/stephen/mambaforge/envs/comsw4733_hw3/bin/python]
pybullet build time: Jul 21 2022 19:56:13
Version = 4.1 ATI-4.8.101
Vendor = ATI Technologies Inc.
Renderer = AMD Radeon Pro 5300 OpenGL Engine
b3Printf: Selected demo: Physics Server
startThreads creating 1 threads.
starting thread 0
started thread 0
MotionThreadFunc thread started
Timeout: robot is taking longer than 5s to reach the target joint state. Skipping...
Timeout: robot is taking longer than 5s to reach the target joint state. Skipping...
Timeout: robot is taking longer than 5s to reach the target joint state. Skipping...
Timeout: robot is taking longer than 5s to reach the target joint state. Skipping...
[RRT Object Transfer] 3 / 3 cases passed
numActiveThreads = 0
stopping threads
Thread with taskId 0 exiting
Thread TERMINATED
destroy semaphore
semaphore destroyed
destroy main semaphore
main semaphore destroyed
```





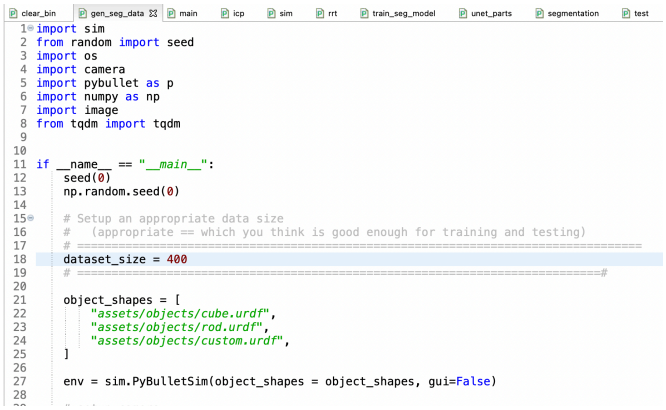
Running python3 main.py -part=all in the project root should run all three parts.



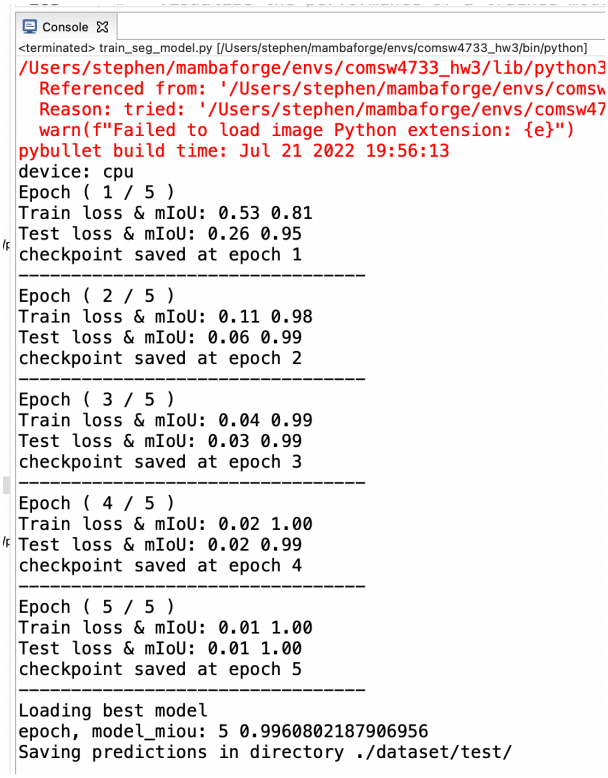
4 – Pose Estimation and Grasping

Report dataset size, test loss and mean IoU of your final model.

For the dataset size, I tested a variety of sizes but ultimately settled on a 400 image set that was likely somewhat larger than necessary but delivered the most desirable results. When I adjusted and tested datasets between 100-300 images, these also trained the segmentation model between the IoU of ≥ 90. In the end, I preferred the larger dataset to generate as much training data as possible without adding significant time to train the model.



This dataset was run across 5 epochs, and likely could have completed in as few as 3. The test loss and mean IoU of the final model are provided below:



Report ground truth and predicted mask images from your final model on any two test set images. For generating these images, you should use save prediction function in train seg model.py file.

Ground truth and predicted mask images are reported below. There are three variations displayed here, based on what tends to work and does not work when grasping and moving objects to another bin. In *Image 1*, the three objects are separated evenly to allow the gripper to select items with interference. In *Image 2*, the three objects are clustered in a way that creates inconsistency when the gripper selects an item from the bin. In *Image 3*, the three objects are presented in the most ideal scenario where the blue object (yellow during simulation) is standing upright, allowing the gripper to form a more secure hold on the object during retrieval.

Image 1:

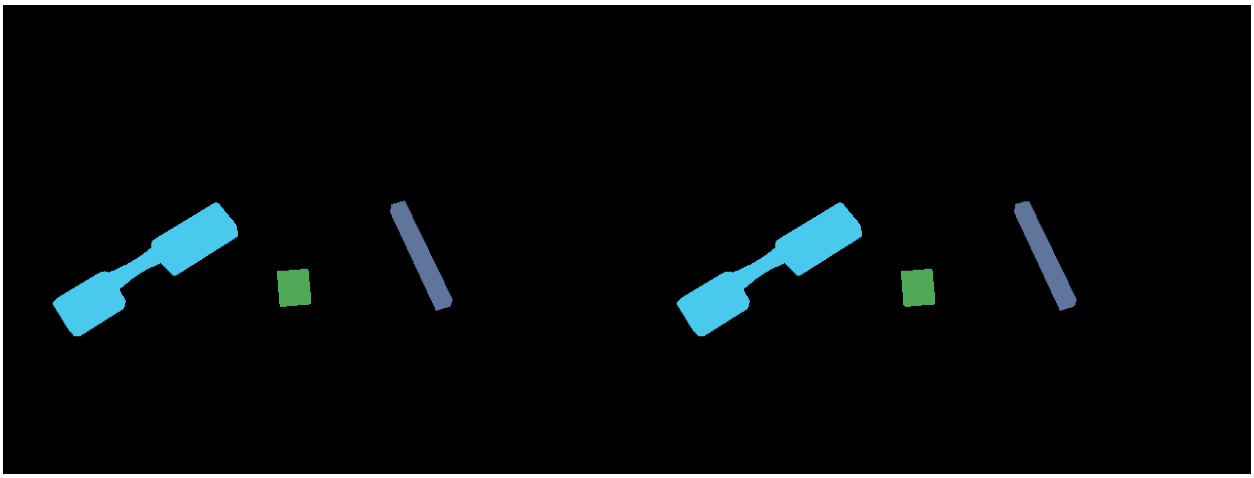


Image 2:

Image 3:

When running the simulation, the objects assembled in a pattern similar to Image 2 often resulted in retrievals that collected (and delivered) multiple items at the same time. While this might seem like a more efficient extraction, it ultimately creates a data error because the machine does not record that a second item has been removed. So when the arm returns after delivering two items at once, the machine tries to look for the now missing item, repeatedly prompting the error message "No points are present in segmented cloud…"

There were a number of successful runs for objects assembled in a pattern similar to *Image 1*, but I found the scenarios where the blue object (yellow in the simulation) is placed vertically resulted in the most consistent retrievals of the blue object. When the blue object is laid flat, there are often multiple grip failures before a successful connection and retrieval occurs. Additionally, the weaker connection often resulted in a FALSE flag despite giving the appearance that the gripper has gripped and lifted the object. Because of the inconsistent shape of the blue object, it was routinely the most inconsistent retrieval with the highest likelihood of failure. Whereas, the green cube and gray (brown in the simulation) stick were retrieved and delivered to the second bin with relatively ease.

There are two video simulations included in my submission: one follows the RRT path with red spheres included, while the other is a direct path with the spheres removed. In general, the direct path worked more consistently, but I believe some of that is a result of the bugs in the assignment associated with adding the spheres. In a real world setting, the RRT path is necessary to avoid collisions in an environment where there might not be a direct, repeatable path to the second bin. I also tested various movement speeds and found that items can be thrown around the room when the speed is too fast. Ultimately, I settled on speeds between 0.005-0.007 as the most efficient, while speeds faster than 0.01 were erratic and unreliable.

**unet_parts.py has been created from the recommended milesial GitHub references and is included in my final submission, selected parts have been imported into train_seg_model and implemented as uparts.