In-lab computer:
- Atlas 4
- group-specific account: alpha/alpha
- don't forget to kill your roscore when you leave

Final Project Tips/Tricks (Th 12/8)
inal project hints from Luc

0. **Check Dr. Newman's Part_6 folder of the repo and re-use as much as you can!**

(This probably will help: https://github.com/wsnew/learning_ros/tree/master/Part6/coordinator)

1. **Creating the transform from the torso to the kinect_link**

- In order to visualize data appropriately, you will need to *create a launch file that starts freenect (the kinect driver) along with an appropriate static transform from Baxter's torso to the kinect_link*. You will need to determine the XYZ RPY translation and orientation from the torso to the kinect link, which is normally done manually then optimized using an external program using the errors between the robot's real position in space (as given by the joint states and encoders) versus where the camera sees some specific markers on the arm. For the purposes of this class, you will simply be configuring the transform by hand.

- This can be done one of two ways. Either by editing the static transform publisher parameters manually, or by using `tf_keyboard_cal`. tf_keyboard_cal is a very useful package written by David Coleman, a PhD student / recent graduate of UC Boulder in the Correll Lab. His tool allows you to publish a dynamic transform between two links along with an interactive marker that can be used to change the transform by hand. This makes it a lot easier.

- **(all on baxter_master…)**

- To begin, start a baxter_master and roslaunch freenect_launch freenect.launch.

- Then, start publishing a tf_keyboard_cal transform using rosrun tf_keyboard_cal tf_interactive_marker.py torso kinect_link 0 0 1 0 0 0 1

- The 0 0 1 in the beginning correspond to 1m above the torso. The 0 0 0 1 corresponds to the quaternion rotation between the two frames. (I believe this is no rotations?)

- Now in rviz, add an interactive_marker object and select the appropriate topic. You should now be able to pan and rotate the marker to calibrate the transform. You should be able to see the PointCloud with respect to Baxter as long as it is displayed.

- Getting a good transform takes some time. Some tricks are to: make sure the bar around Baxter's torso is aligned with the model and Kinect projection, and check to see if the arms, when put in front of the camera, match up with the Kinect projection as well.

- Once you are happy with the transform, do a *rosrun tf tf_echo torso camera_link and find the*

appropriate X Y Z R P Y or equivalent quaternion numbers. Don't trust the numbers output by the tf_keyboard_cal node. They are wrong for some reason.

- Take these numbers and put them in a launch file along using the following line: `kinect_static_tf" pkg=" tf" type="static_transform_args="X Y Z W1 W2 W3 W4 torso camera_link 50" />` (replacing X, Y, Z, and the W's with the appropriate numbers (you can do R P Y instead of the W's))

- Inside the launch file, add a line to launch freenect.launch, as well.

- And that should do it!

2. Determining the height of the stool with respect to the torso (not sure if this is given to you, but this is how I'd do it...)

- You should transform all of the points from the kinect_link frame into the torso frame. You can do this using math or the tf2 library (or tf if that's easier for you)

- You should use the Point Cloud Library to write some kind of "plane detector" using a patch of selected points.

- In rviz, you should use the 'select points tool' (tutorial and installation coming) to select a patch of points that correspond to the stool's surface.

- After you get the patch of points, you should devise some way to filter out all points in the point cloud that do not correspond to some kind of cluster of points with the same Z height and are connected by a certain distance metric. One way to do it would be to filter out all points that are not within the patch's avg Z height +/- some error + max block height, and then add each point that is also in the path to a cluster, and do a 'fan out' search using some distance metric. If a point's, say, XY (not Z) euclidian distance from some point in that group is not too big, add it to the cluster. This should grow to a size corresponding to the stool. (unproven/untested. Dr. Newman (and maybe you) probably has/have a better way)

- After getting everything but the stool and blocks filtered out, you should know the height average height of the stool, so you should be able to determine which points likely belong to a type of block. You may wish to use a clustering algorithm in order to group points together, then use some data on the clusters to determine which cluster of points corresponds to which block.

Gripper transform info (M 12/12)

You should not need to edit the rethink gripper function--rather, re-use it as-is. This function says that the gripper fingertips should be moved to a height mid-way between the stool surface and the top of the block. That is, since the origin of the block is in the middle of the block, it is desirable to place the fingertips at z=0 relative to the block's frame. This is still correct for the Yale gripper--we just need to inform the robot of a good tool transform.
You want to inform the system how the gripper is mounted to the tool flange using a transform

publisher (just like the Kinect camera calibration). You can see how this is used within the cartesian planner (for Baxter's right arm), the tool transform is looked up using tf as follows:

See cartesian_planner/baxter_rt_arcpp. Lines 341-357:

```
    ROS_INFO("waiting for tf between generic gripper frame and tool
flange...");
    while (tferr) {
        tferr=false;
        try {
                tfListener_->lookupTransform("ros::Time(0),
generic_toolflange_frame_wrt_gstf_);
            } catch(tf::TransformException &exception) {
                ROS_WARN("%s; retrying...", exception.what());
                tferr=true;
                ros::Duration(0.5).sleep(); // sleep for half a second
                ros::spinOnce();
                ntries++;
                if (ntries>5) ROS_WARN("did you launch
baxter_static_transforms.launc
            }
    }
    ROS_INFO("tf is good for generic gripper frame w/rt right tool
flange");
    xformUtils.printStampedTf(genetoolflange_frame_wrt_grippstf_);
```

Upon start-up, the class constructor of this class (which is started inside the cartesian planner) looks for a transform between generic_gripper_frame and right_hand.
This start-up will complain if you do not launch "baxter_static_transforms.launch." The launch file is in:
cartesian_planner/launch/baxter_static_transforms.launch. Note in this launch file the line:
tf" type="static_transform_publishargs="0 0 0 0 0 0 1 right_gripper generic_gripper_frame 100" />
this creates a transform between right_gripper and generic_gripper_frame (and hopefully, this leads to a transform between right_hand, i.e. flange, and generic_gripper_frame). All computed moves will be based on generic_gripper_frame.
BUT, we do not have a "right_gripper" (Rethink gripper). Instead, **make up a transform from right_hand (which is really the tool flange frame) to generic_gripper_frame**. This should take into account the dimensions and mounting pose of the Yale gripper. You want a transform that should result in the following:
when the robot is in the "grasp" position for the block, the Yale-gripper fingers should straddle the block (such that you can grasp it, once the gripper is ready).
Be careful--don't let the fingertips touch the stool at this pose, or the gripper won't close correctly.
So: **you should only need to edit baxter_static_transforms.launch** (or make your own, with a new name, and include it in your start-up code).

Find magic numbers for the transform publisher, and the object grabber should perform correctly. Note that there are *several sources of error:* camera calibration, relatively poor Baxter arm control precision (not much you can do about that), gripper transform calibration, and potential PCL block-localization imprecision. All sources of error combined must be less than the clearance between block and open fingers in order to be successful.