Robot Usage Guide *

EECS/ME/BIOE C106B/206B Spring 2021

Contents

*Developed by Josephine Koe, Fall 2021.

1 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 106A. Please live up to our expectations.

1.1 Robot Reservations

Robots should be reserved on the robot calendars:

- Rethink Robot Calendar
- TurtleBot Calendar

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 per group.** This is shared hardware and we want to ensure that all groups have enough time to complete the projects.
3. **One computer per group (if needed).** 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 work in parallel by using personal computers, file-sharing software like Git, and/or using simulators like Gazebo or RViz.

1.2 Lab 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 responsible for this person's conduct). Luna is Stella's dog, so Luna would not be an appropriate friend to bring along for safety reasons.
2. **Do not leave the room while the robot is running.** Running robots must be under constant observation.
3. **No food or drink (except water) in the lab.** We already have an ant problem, so do not bring anything into the lab that may attract more. Keep water in a container with a lid and drink it away from the computers and robots.
4. **Keep the walkways clear.** Make sure your fellow classmates can safely move around the lab and access the robots with ease. Do not place anything on the floor that may trip or block someone.
5. **Always check the trajectory before executing MoveIt! trajectories.** You can view a trajectory in RViz. Sometimes MoveIt! plans extremely strange trajectories, and checking them before running them can prevent damage to the robot. If a trajectory looks strange in RViz, do not execute it on the robot. Try giving the robot a different start or end position and then replan the trajectory.
6. **Always operate the robot with the E-Stop within reach.** Be ready to press the E-Stop button at a moment's notice to prevent the robot from crashing into anything. If a robot is going to hit a person, the wall, or a table, press the E-Stop. Luna wears a parka when it is cold, but not everyone has such protective padding around them for safety. Be particularly careful of people who may be walking through the aisle between the Rethink robots and computers.
7. **Terminate all processes before logging out.** Because the lab workstations are shared, we sometimes run into strange bugs. The most common is that someone leaves a process running when they leave the lab, sometimes by forgetting to log out properly. When the next person uses the same workstation, ROS can get confused by the additional processes running from the prior user. A particularly insidious example is leaving a `roscore` master node running in the background; the next person will not be able to run a master node with proper communications. To avoid issues like this in the lab, please do the following before leaving:

- Ctrl+C out of every terminal before closing it. It is critical that you Ctrl+C. **Do not Ctrl+Z**. Ctrl+Z may look like it stops your process, but it really only pauses it. The process will continue to run in the background.
- Instead of simply logging out of your account, you should type

  ```
  pkill -u [username]
  ```

  into your terminal, where `[username]` is your instructional account username (`ee106b-xyz`). This will close out every process on your account before logging you out.

You can also use

```
ps -ef | grep ros
```

to check your currently running processes.

8. **Do not modify robots without consulting lab staff.** In previous semesters we had problems with students losing gripper pieces and messing with TurtleBots. This is inconsiderate and makes things much more difficult for everyone.
9. **Tell the course staff if something is broken.** 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.
10. **If you are unsure of how to do something, ask someone on course staff.** We are all here to learn, so don't be afraid to ask for help. Operating the hardware without knowing exactly what you are doing can be extremely dangerous, so please ask for help whenever you have questions about the hardware.

1.3 COVID Safety Measures

We are still very much in the middle of a pandemic. You should not have to sacrifice your health for your education, so we are implementing a few safety measures:

- Mask-wearing is required inside the lab.
- Before working on the computers, wipe down your workstation (including the keyboard and mouse) and sanitize your hands. Wipes will be available at the front desk in the lab.
- If you feel unwell, stay home and email your lab TA and Stella. We will arrange accommodations for you.
- Students who are assigned to a lab section will always be allowed to work during that section, but otherwise you may be asked to leave if there are too many people in the lab.

If you do not comply with these safety measures, you may be asked to leave the lab.

2 Setting Up Your Environment

To set up your environment to automatically use ROS and interface with the robots, you need to edit your `~/.bashrc` file. The `~/.bashrc` file is a script that is run when a new terminal window is started. Adding the commands in this section to the end of your `~/.bashrc` file will automatically set up ROS and the environment variables for connecting to the robots in every terminal window you open. Be careful with which commands you include and in what order; some may override or interfere with one another. This is why we suggest you add the commands in this section at the end of your default `~/.bashrc` file.

2.1 Setting up ROS

The first additional command you should add to the end of your `~/.bashrc` file is for setting up ROS. If you are not using the Baxter SDK (the Rethink robot packages for both Baxter and Sawyer), add the following line to your `~/.bashrc` file

```
source /opt/ros/kinetic/setup.bash
```

to set up only ROS. More specifically, this line tells Ubuntu to run a ROS-specific configuration script every time you open a new terminal window. This script sets several environment variables that tell the system where the ROS installation is located. It also adds the directories for all of ROS's built-in packages to the package path.

If you are using the Baxter SDK, you should instead add the following line to your `~/.bashrc` file

```
source /scratch/shared/baxter_ws/devel/setup.bash
```

which sets up ROS and its built-in packages like the `source /opt/ros/kinetic/setup.bash` command does but additionally edits the `ROS_PACKAGE_PATH` environment variable which tells ROS which directories to search for software packages. Any code you want to run with ROS must be located beneath one of the directories specified in the list. Luna has a list of foods that she likes to search for, but she does not like fruit so it is not on that list. You want to be able to use the Baxter SDK packages in addition to ROS's built-in packages, and using this line in the `~/.bashrc` file enables you to do so because it adds the Baxter SDK directory to the package path. The Baxter SDK is located in the `/scratch/shared` directory.

It is good practice to only import the ROS packages that you need, so you should only use this line if you are using the Baxter SDK packages as to not clutter your workspace with the things associated with the Baxter SDK.

2.2 Setting the ROS hostname

After the command to set up ROS, you can set the ROS hostname if you are using a workstation in the lab (not a VM). This node environment variable sets the declared network address of a ROS node or tool, and you usually want this to be the hostname of the computer you are using. Adding the following line will set the ROS hostname to the current computer you are using

```
export ROS_HOSTNAME=$(hostname --short).local
```

If you run into a hostname error at any point, you can instead temporarily use the line

```
export ROS_HOSTNAME=192.168.1.[COMPUTER_NUMBER]
```

where `[COMPUTER_NUMBER]` is your computer number (1 through 10). This sets the ROS hostname to the specific computer that you are working on. Note that you should change the ROS hostname back to `$(hostname --short).local` eventually, otherwise you'll have to edit the `~/.bashrc` file every time you work with a new computer.

2.3 Setting the ROS master URI

If you are running ROS nodes on a robot, you may also need to change the ROS master URI (Uniform Resource Identifier). This is a hostname:port combination that tells nodes where they can locate the master node. Setting this environment variable to point to a robot will allow you to run ROS nodes on the robot remotely by setting the ROS master as the robot's computer rather than your workstation computer. To do this, add the following line to your `~/.bashrc` file

```
export ROS_MASTER_URI=http://[IP_ADDR_OF_ROBOT]:11311
```

You can run the following command in a terminal window

```
cat /etc/hosts
```

to see the list of IP addresses for the workstations and robots and set the `[IP_ADDR_OF_ROBOT]` value accordingly. For example, adding the line `export ROS_MASTER_URI=http://192.168.1.111:11311` to your `~/.bashrc` file will enable you to run ROS nodes on the Black TurtleBot.

For TurtleBots, this can alternatively be done with the line

```
export ROS_MASTER_URI=http://[TurtleColor].local:11311
```

For example, adding the line `export ROS_MASTER_URI=http://black.local:11311` to your `~/.bashrc` file will also enable you to run ROS nodes on the Black TurtleBot.

When you're done working with the robot, you should remove the above command from your `~/.bashrc` file. If you don't, you won't be able to run a `roscore` properly on your computer. Note that removing the command will only take effect after logging out of your computer. To work on other things immediately, type

```
export ROS_MASTER_URI=http://localhost:11311
```

in all terminal windows you plan to use for running ROS nodes on your computer. You can alternatively edit the `~/.bashrc` file to include this line and update the terminal windows by re-sourcing the `~/.bashrc` file in each one, especially if you are running into a `IOError: unsupported XML-RPC protocol` error.

2.4 Sourcing the .bashrc file

When you're done editing the `~/.bashrc` file, save and close the `~/.bashrc` file and re-run it by executing the command

```
source ~/.bashrc
```

in each existing terminal that you want to be updated with the new settings. The `~/.bashrc` file is only automatically run when a new terminal window is started, so you have to run this command to manually "source" (run) the `~/.bashrc` file again in order to update an existing terminal. This will only update the terminal window you run the command in, so make sure to run it in all terminal windows that you want to be updated. Likewise, when Luna got a new hoodie, she had to update her list of things she wears when it is cold to include it. After saving the `~/.bashrc` file, every new terminal window you open will automatically be updated with the new settings and you won't have to source it manually.

3 Rethink (Baxter/Sawyer) Robots

In the lab, there are Baxter (Ayrton, Asimov, and Archytas) and Sawyer (Ada and Alan) robots. The Baxter and Sawyer robots from Rethink Robotics are manufacturing robots designed to operate around people. They are unfortunately no longer supported by the company, so **please be especially careful when using these robots as they are difficult to fix and replace.**

Figure 1: (Left): Baxter robot. (Right): Sawyer robot.

3.1 Hardware

Note: Much of this section is borrowed from the Rethink Baxter Wiki and Rethink Sawyer Wiki. We picked out the material you will find most useful in this class, but feel free to explore other resources if you are interested in learning more.

- Arms
  - Baxter: Two arms, each with seven joints. Note that the Baxter has a left and right arm (from the robot's perspective).
  - Sawyer: One arm with seven joints. Because the Sawyer only has one arm, we default to calling it the right arm.

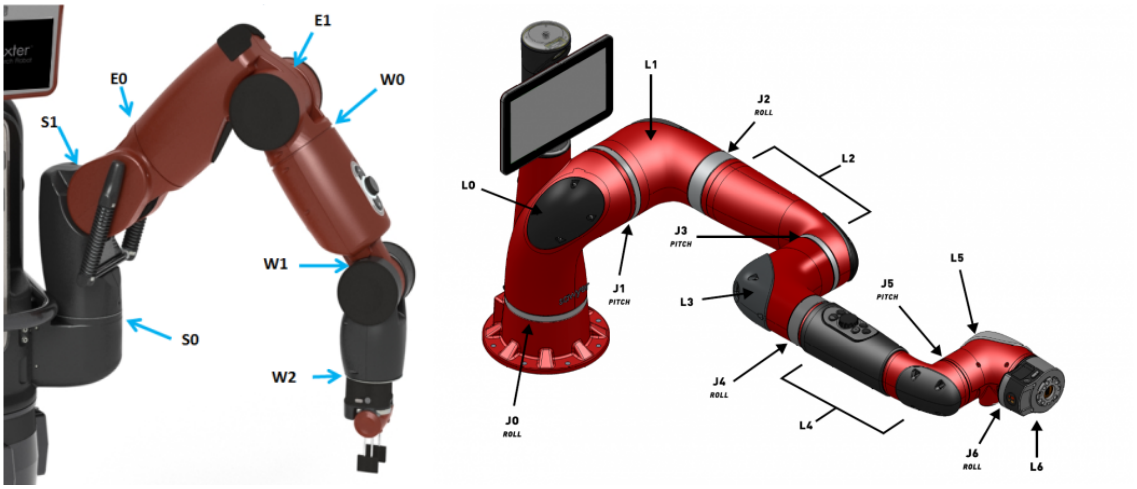

Figure 2: (Left): Labeled Baxter joints. (Right): Labeled Sawyer joints and links.

- Grippers
  - Please ask someone on course staff for assistance when installing or removing a gripper.
  - Electric parallel-jaw grippers: 44 mm throw and attachment points for a variety of finger configurations, meant for lifting payloads up to five pounds
  - Pneumatic suction grippers: can attach either a single vacuum cup or a multi-cup vacuum manifold

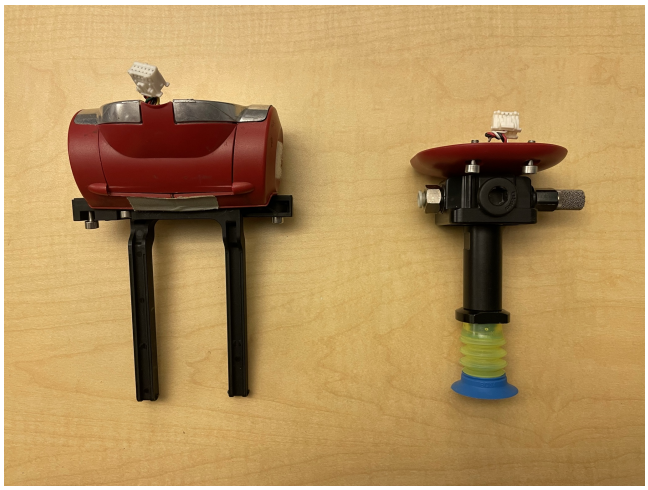

Figure 3: Electric parallel-jaw gripper (left) and pneumatic suction gripper (right).

- Cameras
  - One camera per arm (Figure 4)
  - One camera in the head
  - Baxter has three cameras in total, but only two cameras can be powered on at once

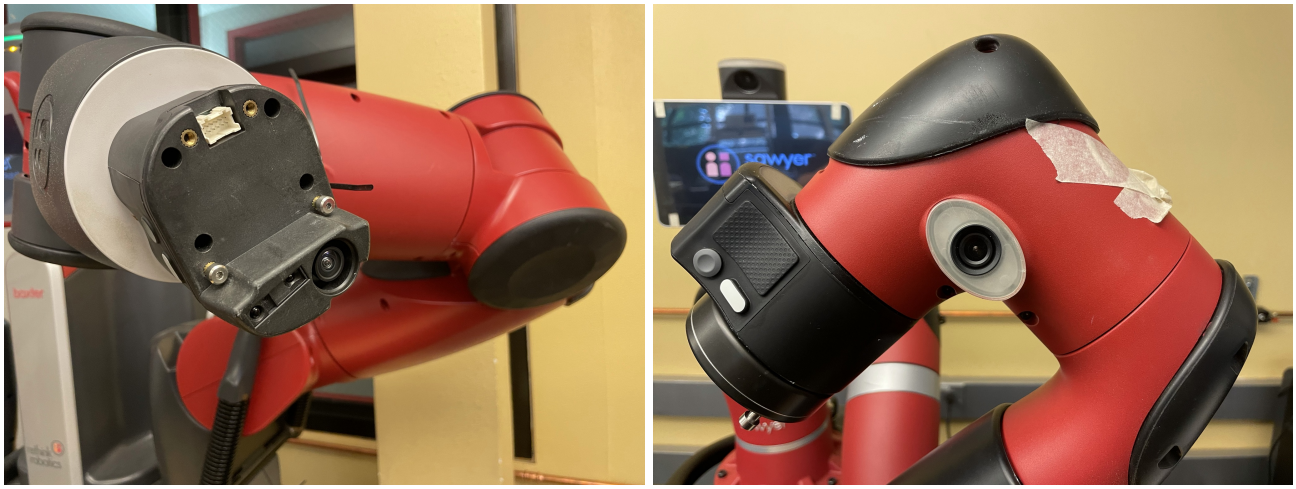

Figure 4: (Left): Baxter wrist camera. (Right): Sawyer wrist camera.

- Head
  - Panning and nodding abilities
  - Camera
  - Sonar ring (Baxter)
  - Display
- Control
  - On-board computer running Linux and ROS Kinetic
  - Can run ROS nodes remotely or on-board
  - Low-level controllers prevent self-collisions and enforce limits on acceleration, torque, and position
  - Zero-g mode: If you wish to move the robot arms manually, enable the robot and grasp the cuff of the arm over its grooves (Figure 5). This will enable the zero-g mode where the controllers are disabled and so the arm can be freely moved across without much resistance. Luna likes going to the park where she can also run around freely. **Do not try to push or pull on the arms without enabling zero-g mode.**

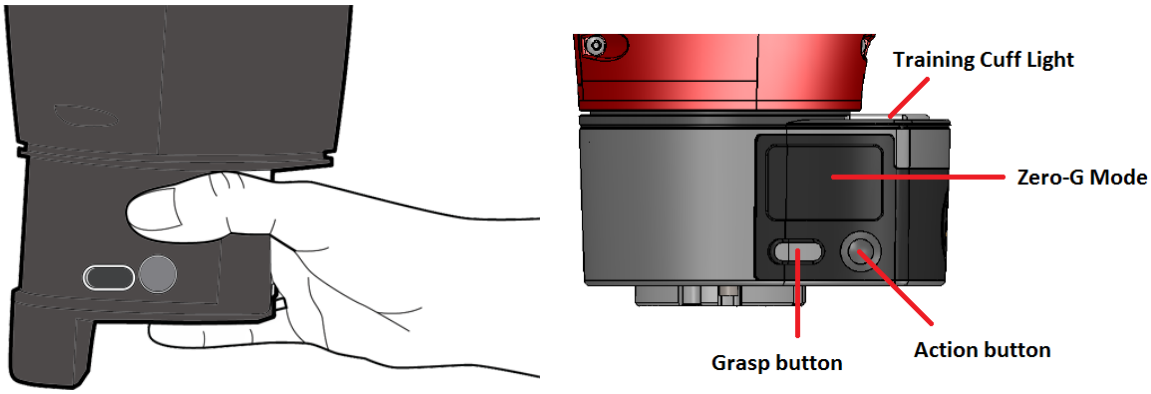

Figure 5: (Left): Illustration of enabling zero-g mode on Baxter. (Right): Diagram of zero-g mode indentations on Sawyer cuff.

- Sensors
  - 3-axis accelerometer inside each cuff
  - Baxter: two IR range sensors in each hand

- Baxter: sonar ring around head

- E-Stop and power buttons
  - E-Stop button: Both the Baxters and Sawyers have an emergency stop (E-Stop) button (Figure 6). **If you have even the slightest feeling that the robot is behaving or about to behave dangerously, push down on this button**; the robot will be immediately disabled and the arms will lower slowly. Dangerous behavior includes but is not limited to any of the following:
    - the arm is about to crash into someone/something
    - the arm is moving very quickly or jerkily
    - the arm is not behaving as expected

    In any case, keep a hand on the E-Stop button whenever you are running code on the robot and don't be afraid to press it if you think a situation is or is becoming unsafe. You do not need to hold down on the E-Stop button; after pressing it, it will be mechanically held down until you release it. To release the E-Stop button from the held-down state, turn the button clockwise until it pops up. Luna can jump 3 feet in the air, but the E-Stop button will not pop up that high. Note that the robot can only be enabled after the E-Stop button is released, and you will need to re-enable the robot before moving it again. Refer to 3.3.1 on how to enable the robot.

Figure 6: E-Stop button with arrows indicating how to turn the button to release it.

  - Power button: The power button is located on the left side of each robot, but you should leave the robots on most of the time and should not need to use this button. **Do NOT hold down the power button;** holding down the power button can cause hard drive corruption on the next boot. Just press the power button once to turn on or once to turn off.

3.2 Setting up your environment for Rethink robots

To automatically set up the Baxter and Sawyer packages in each new terminal, make sure that your `~/.bashrc` file includes the line

```
source /scratch/shared/baxter_ws/devel/setup.bash
```

If you edited the `~/.bashrc` file, make sure to source it in any existing terminal windows that you plan on using. Refer to the Setting Up Your Environment section for more information about what this line does in the `~/.bashrc` file.

To set up your environment for interacting with Baxter, make a shortcut (symbolic link) in the root of your catkin workspace to the Baxter environment script `/scratch/shared/baxter_ws/baxter.sh` using the command

```
ln -s /scratch/shared/baxter_ws/baxter.sh [path-to-workspace]
```

From the root of the catkin workspace, use the following line to connect to one of the Baxter robots:

```
./baxter.sh [name-of-robot].local
```

where [name-of-robot] is either ayrton, asimov, or archytas.

To set up your environment for interacting with Sawyer, make a shortcut (symbolic link) in the root of your catkin workspace to the Sawyer environment script /scratch/shared/baxter_ws/intera.sh using the command

```
ln -s /scratch/shared/baxter_ws/intera.sh [path-to-workspace]
```

From the root of the catkin workspace, use the following line to connect to one of the Sawyer robots:

```
./intera.sh [name-of-robot].local
```

where [name-of-robot] is either ada or alan.

When you are done using any robot, make sure all of the terminal windows where you connected to the robot are no longer running any processes. Then in each terminal window that is connected to the robot, close the connection to the robot with the command

```
exit
```

3.3 Basic functions

3.3.1 Enabling the robot

In order to control Baxter's arms, the robot must be put in the "Enabled" state. Enabling the robot provides power to the joint motors, which are initially in the "Disabled" state on start-up or after a serious error, such as an E-Stop. If you are using a Baxter, enable the robot by running

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

If you are using a Sawyer, enable the robot by running

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

3.3.2 Pre-set positions

If you are using a Baxter, running the tuck_arms.py script is an easy way to test that the robot can move after you have enabled it. The untucked position is also useful as a good pre-set starting configuration where the arm joints are not at awkward angles and the wrists are generally above your physical workspace. Untuck the Baxter arms by running

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

If you are using a Sawyer, you can send the joint angles to a pre-set position by running

```
rosrun intera_examples go_to_joint_angles.py
```

3.3.3 Controlling the joints from the keyboard

The joint_position_keyboard.py script allows you to move the robot's limbs from the keyboard. Note that you do not need to start roscore; it is already running on the Baxter/Sawyer robot itself. If you are using a Baxter, start the joint_position_keyboard.py script by running

```
rosrun baxter_examples joint_position_keyboard.py
```

If you are using a Sawyer, start the joint_position_keyboard.py script by running

```
rosrun intera_examples joint_position_keyboard.py
```

3.3.4 Reading the transform

Read the rigid body transformation from the robot base to one of the hands by running

```
rosrun tf tf_echo base right_hand
```

or

```
rosrun tf tf_echo base left_hand
```

3.4 Interfacing with the robot

Instead of publishing directly to a topic to control Baxter/Sawyer's arms, the respective SDKs provide a library of functions that take care of the publishing and subscribing for you. Baxter and Sawyer have different packages (`baxter_examples`, `baxter_interface` and `intera_examples`, `intera_interface` respectively), but they are virtually identical. Luna's original name was Cauliflower, but she is virtually the same dog as before. Don't forget to check that you have the correct package imported! The main difference is the obvious one: Sawyer only has one arm! This means that whenever you try to move an arm on Sawyer, it must be the **right** one. On Baxter, you may use either arm.

3.4.1 Joint Trajectory Action Server

The joint trajectory action server within the Baxter's/Sawyer's SDK allows users to command the robot's arm(s) through multiple waypoints and track the trajectory execution. The main benefit of this server is its ability to interpolate between supplied waypoints, command Baxter's joints accordingly, and ensure the trajectory is being followed within a specified tolerance. You will need to run this before executing a trajectory on the robot. If you are using a Baxter, start the joint trajectory action server by running

```
rosrun baxter_interface joint_trajectory_action_server.py
```

If you are using a Sawyer, start the joint trajectory action server by running

```
rosrun intera_interface joint_trajectory_action_server.py
```

3.4.2 Baxter/Sawyer MoveIt!

MoveIt! is an open source robotics manipulation platform that runs on top of ROS and uses kinematics, motion planning, and collision checking to plan and execute trajectories for robot manipulators. You will need to run this before executing a trajectory on the robot. If you are using a Baxter, start MoveIt! by running

```
roslaunch baxter_moveit_config demo_baxter.launch right_electric_gripper:=true left_electric_gripper:=true
```

On a Baxter, the command above may fail depending on the types of attached grippers; if it does, try running

```
roslaunch baxter_moveit_config baxter_grippers.launch
```

If you are using a Sawyer, start MoveIt by running

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

Omit the gripper argument if the robot does not have a gripper.

3.4.3 Opening and closing the wrist cameras (Baxter)

For the Baxters, 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 camera before you can open the other one.

Open a camera by running

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

and close it by running

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

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

4 TurtleBots

TurtleBot is one of the classic platforms for mobile robotics research and teaching. We will be using the classic TurtleBot 2 in this class, since it is the most common and best supported out of the three TurtleBot versions. Each is color-coded according to the acrylic platforms and has a different AR tag number.

Each TurtleBot is assigned to one workstation as follows. You should not use any other TurtleBot besides the one assigned to your current workstation, but you may use the red TurtleBot (AR tag #0) if the TurtleBot assigned to your workstation is broken.

Workstation #	Color	AR Tag #
1	green	4
2	pink	5
3	blue	6
4	black	7
5	yellow	8

Figure 7: TurtleBots with identifier colors

4.1 Hardware

Note: Much of this section is borrowed from the TurtleBot User Guide. We picked out the material you will find most useful in this class, but feel free to explore other resources if you are interested in learning more.

- Kobuki drive base
  - Pick up the TurtleBot from under the base on both sides; do not hold the TurtleBot by the acrylic platforms.
  - "Unicycle model" tank drive (can rotate in place)
  - Encoders on 2 main drive wheels (can measure how far each wheel has turned)
  - Gyroscope
  - Front bumper (can detect collisions)
  - AR (augmented reality) tag on top

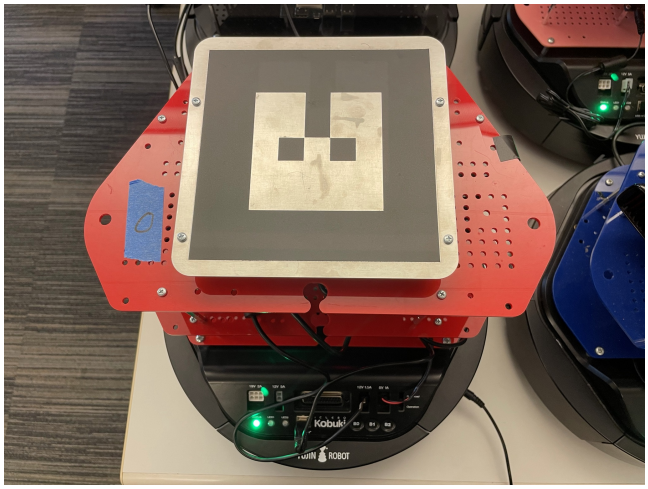

Figure 8: AR tag on top of TurtleBot.

- Kinect sensor
  - RGB color camera
  - Depth camera

Figure 9: Kinect on TurtleBot.

- Control
  - On-board NVIDIA Jetson, Qualcomm DragonBoard, or Raspberry Pi running Linux and ROS Kinetic
  - Can run ROS nodes remotely or on-board
- Power
  - Status LED
    - Green: Kobuki is turned on and battery at high voltage level
    - Orange: Kobuki is turned on and battery at low voltage level (please charge soon)
    - Green blinking: Kobuki is turned on and battery charging
    - Off: Kobuki is turned off
  - Charging port in the drive base

- On-off switch: Please charge the TurtleBots when they are not in use. Similarly, Luna needs to rest after a long day at the beach, where she likes to go. When charging, make sure that the power cord is plugged into the charging port in the drive base and the TurtleBot is switched on. **The TurtleBot must be switched on in order to charge.**

Figure 10: (Left): Status LED on Kobuki base, indicating on and high battery. (Right): Charging port and on-off switch on Kobuki base.

4.2 Setting up your environment for TurtleBots

To automatically set up the TurtleBot packages in each new terminal, make sure that your `~/.bashrc` file includes the lines

```
source /opt/ros/kinetic/setup.bash
export ROS_MASTER_URI=http://[TurtleColor].local:11311
```

where `[TurtleColor]` is the network name of your TurtleBot (`blue`, `orange`, `black`, etc.). If you edited the `~/.bashrc` file, make sure to source it in any existing terminal windows that you plan on using. Refer to the Setting Up Your Environment section for more information about what these lines do in the `~/.bashrc` file.

Before you can listen for messages or give commands to the TurtleBot, you'll have to turn it on. Begin by turning on the power and checking that you can `ping` the onboard computer by running

```
ping [TurtleColor]
```

and making sure that there are no errors when trying to connect to the TurtleBot. If pinging doesn't work, try turning the TurtleBot off and on again. Note that some of the TurtleBots take several minutes to wake up. Now, before you can do anything interesting with the TurtleBot, you'll have to `ssh` (Secure Shell) into the onboard computer by running

```
ssh turtlebot@[TurtleColor].local
```

The password is `[TurtleColor]2022`. When you are done using the robot, make sure all of the terminal windows where you connected to the robot are no longer running any processes. Luna does not like brushing her teeth but still does it to kill any germs in her mouth, and likewise you should also kill any unnecessary running processes on the robot. Then in each terminal window that is connected to the robot, close the connection to the robot with the command

```
exit
```

4.3 Basic functions

4.3.1 Start the base functionality

Start the TurtleBot basic nodes (including the master node) by running

```
roslaunch turtlebot_bringup minimal.launch
```

The TurtleBot is now launched, along with all of its sensors, and it is ready to receive motion commands.

4.3.2 Controlling the TurtleBot from the keyboard

Keep the `minimal.launch` running and open a new terminal window. You can optionally `ssh` into the TurtleBot in this new terminal window. Then drive the TurtleBot from the keyboard by running

```
roslaunch turtlebot_teleop keyboard_teleop.launch
```

4.3.3 Visualizing the Kinect

Kinect is an incredibly powerful sensor, so even though we will not be making extensive use of it in the labs for this course, we expect you to get some familiarity with it, and we hope that some of you will use one in your final project! You could even use one to help Luna find and chase squirrels, which she likes to do. There are actually several different versions of the Kinect, and they each work a little bit differently. If you're interested, we encourage you to figure out how the depth sensing actually works in each one.

For now, let's just examine the ROS interface. Open up a new terminal, ssh into the TurtleBot, and run

```
roslaunch turtlebot_bringup 3dsensor.launch
```

Then, on your local computer, run

```
roslaunch turtlebot_rviz_launchers view_navigation.launch
```

Change Global Options > Fixed Frame to `base_link`, then modify the view type to `ThirdPersonFollower (rviz)` (upper right corner). Next, use Add > By Topic > `/camera/depth_registered/image` to add a visualization of the Kinect data.