

DIRECTIONS TO SET UP BAXTER SIMULATOR TO RUN CHESS_SIMULATOR.PY PROGRAM

Being able to model an environment to test as much of your code as possible before trying it on the real robot, in my opinion, is important. This prevents you from putting code into the robot that can potentially cause harm.

Gazebo is the name of the simulator that ROS uses. ROS Indigo has Gazebo 2.x which is an old version. (Gazebo is up to 8.0 now). However, 2.x is the version that is compatible with Indigo and the Baxter simulator. I bring this up because if in the future you do any tutorials on Gazebo realize the limitations of version 2.x.

The chess_simulator.py program uses parts of the program “golf.py” from the visual servoing demo and the baxter pick and place simulator program to demonstrate how baxter finds the board and it’s corners using opencv and canny edge. It modifies the simulated environment by adding a table with a board on it to the environment. If you study the code, you will see where the models are added to the simulated environment.

The two demo programs referenced above are located here:

http://sdk.rethinkrobotics.com/wiki/Worked_Example_Visual_Servoing

http://sdk.rethinkrobotics.com/wiki/Baxter_Simulator

Note: In the folder, the original demo code that did not compile for the worked example visual servoing is in the folder called visual_servoing from the web. The activerobots file contains code that compiles but often “hangs up” because of camera issues which we discussed at our last meeting. It demos both the canny edge and hough circles.

When you are ready to add chess pieces, you may consider using the chess model found in a book available in the UW library as an example. The book is called “Programming Robots with ROS” by Morgan Quigley, Brian Gerkey, and William D. Smart. Chapter 11 is called Chessbot. The authors use the NASA Robot called R2 to play chess using MoveIt. However, included in the chapter is information on how to create chess pieces for a board in a simulated environment. The information is in a subsection of the chapter called “modeling a chessboard”.

This assumes that you have already installed Ubuntu 14.04, ROS Indigo, the Baxter workstation, and followed the directions to install the Baxter simulator. The ros_ws, once all this is done, will have multiple rethink packages in it. I suggest you create a new ros package to put all your own scripts in. At the Baxter lab, you created a package called baby_steps. In case you do not have it at home where you might be working, I made a copy of it and included it with this email. The baby_steps package also includes scripts to help you find baxter’s endpoint (the gripper), a sample program using the ik service and a program to tell you what Baxter’s current joint angles are. Please note that for the real robot, use the joint_positions.py file and if you want to query joint angles in the simulated robot, please use the file I included as an attachment called subscriber_to_jointstate_simulated.py. The need for two different programs just reflects the slight differences between the real robot and the simulated robot.

The package will go in the /ros_ws/src directory

Remember that the .py files in the scripts folder will have to be made executable since they will be on a different machine.

The directions for making a file executable are below when I talk about the chess_simulator.py file.

Also, when you put the package in your ros_ws/src directory, you must rebuild the workspace.

This is done with the catkin_make command. Please see the ros tutorials on how to use the command.

You would open up a terminal ---- ctrl – alt -t and change directory to the ros_ws with the following command:

```
cd ros_ws
```

Make absolutely certain you are in ros_ws directory before typing in the command: catkin_make

You will see your workspace being built.

While still in the ros_ws directory, you should also source the workspace with the command:

```
source ~/ros_ws/devel/setup.bash
```

I previously gave directions on how to put this command on the .bashrc file so that you do not have to source the workspace everytime you open a terminal. If you need the directions again, please let me know and I will be happy to send it again.

SETTING UP FOR THE SIMULATOR

Step 1. The Camera controller does not work in the simulator. However, all three of Baxter's camera will "see" what is in his simulated environment. There is not a bandwidth issue like there is in the real Baxter, so you do not have to worry about closing one of the cameras.

However, since you cannot use the camera services, the only way to change the default resolution of the left hand camera(800 for the width and 800 for the height) is to manually go in and change the baxter_base.gazebo.xacro file.

This is the file that creates the baxter robot in the simulator.

Navigate to the file like this:

```
home/ros_ws/baxter_common/baxter_description/urdf/baxter_base
```

Once there, you will see the baxter_base.gazebo.xacro file.

Open it with gedit and scroll down the document till you see the following:

Note: gedit is a text editor used in Ubuntu. If you do not have it, it is simple to install.

Open a terminal and type in: sudo apt-get install gedit

```

</gazebo>
<gazebo reference="left_hand_camera">
  <sensor name="left_hand_camera" type="camera">
    <update_rate>30.0</update_rate>
    <camera name="left_hand_camera">
      <pose>0.0 0.0 0.0 0.0 -1.570796327 1.570796327 </pose>
      <horizontal_fov>1.3962634</horizontal_fov>
      <image>
        <width>960</width>
        <height>600</height>
        <format>R8G8B8</format>
      </image>
      <clip>
        <near>0.02</near>
        <far>300</far>
      </clip>
    </camera>
  </sensor>
</gazebo>

```

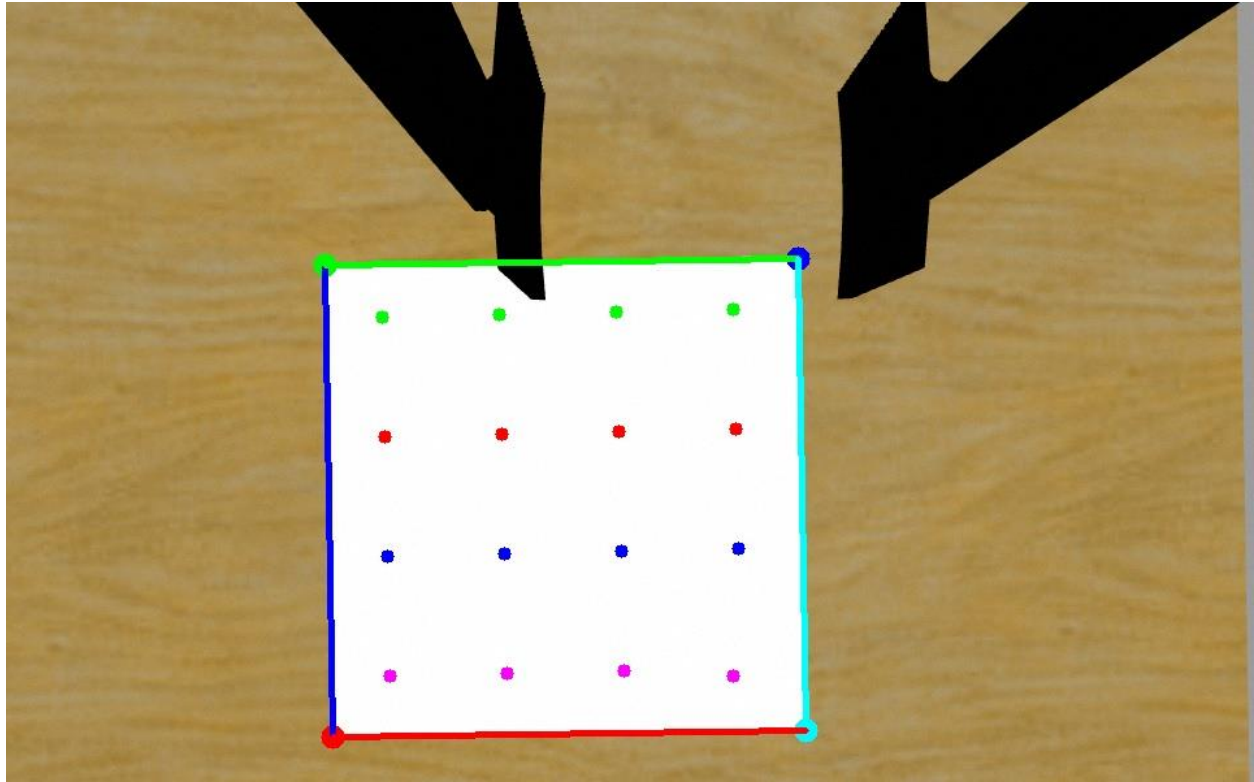
I already changed the height and width. The default was <width> 800 </width> and <height> 800 </height>. Just remember that it needs to be changed back. Save your changes and close the file.

Step 2. In your home directory, paste the folder called Golf. This stores the pictures that the chess_simulator.py takes and also the setup.dat file. The setup.dat file is used by the program to determine which arm you are using and also the height of Baxter's arm above the table.

For the simulator, I just found the distance by trial and error. It was .27. Distances are all in meters. In the real robot, the table in the lab with the green felt top is 30 inches high. I just measured the distance from the table top to Baxter's camera and got .445 meters. You will find as you work with the program that having an accurate distance measure is critical.

The pictures show that the simulator program was pretty accurately determining the corners. I changed the colors so that you could differentiate which corner was which. OpenCV used bgr (blue, green, red) as opposed to rgb (red, green, blue). You will see what I am talking about when you look at the code.

Note that for the sixteen points in the square:



Because I changed from a 3 by 4 rectangle to a 4 by 4 square, it changed the mappings in the original code.

Original:

```

# find places for golf balls
def find_places(self, c):
    # find long side of ball tray
    l1_sq = ((c[1][0] - c[0][0]) * (c[1][0] - c[0][0])) + \
            ((c[1][1] - c[0][1]) * (c[1][1] - c[0][1]))
    l2_sq = ((c[2][0] - c[1][0]) * (c[2][0] - c[1][0])) + \
            ((c[2][1] - c[1][1]) * (c[2][1] - c[1][1]))

    if l1_sq > l2_sq:
        cc = [c[0], c[1], c[2], c[3]] # c[0] to c[1] is a long side
    else:
        cc = [c[1], c[2], c[3], c[0]] # c[1] to c[2] is a long side

    # ball tray corners in baxter coordinates
    for i in range(4):
        self.ball_tray_corner[i] = self.pixel_to_baxter(cc[i], self.tray_distance)

    # ball tray places in pixel coordinates
    ref_x = cc[0][0]
    ref_y = cc[0][1]
    dl_x = (cc[1][0] - cc[0][0]) / 8
    dl_y = (cc[1][1] - cc[0][1]) / 8
    ds_x = (cc[2][0] - cc[1][0]) / 6
    ds_y = (cc[2][1] - cc[1][1]) / 6

    p = {}
    p[0] = (ref_x + (3 * dl_x) + (3 * ds_x), ref_y + (3 * dl_y) + (3 * ds_y))
    p[1] = (ref_x + (5 * dl_x) + (3 * ds_x), ref_y + (5 * dl_y) + (3 * ds_y))
    p[2] = (ref_x + (3 * dl_x) + (1 * ds_x), ref_y + (3 * dl_y) + (1 * ds_y))
    p[3] = (ref_x + (5 * dl_x) + (1 * ds_x), ref_y + (5 * dl_y) + (1 * ds_y))
    p[4] = (ref_x + (3 * dl_x) + (5 * ds_x), ref_y + (3 * dl_y) + (5 * ds_y))
    p[5] = (ref_x + (5 * dl_x) + (5 * ds_x), ref_y + (5 * dl_y) + (5 * ds_y))
    p[6] = (ref_x + (1 * dl_x) + (3 * ds_x), ref_y + (1 * dl_y) + (3 * ds_y))
    p[7] = (ref_x + (7 * dl_x) + (3 * ds_x), ref_y + (7 * dl_y) + (3 * ds_y))
    p[8] = (ref_x + (1 * dl_x) + (1 * ds_x), ref_y + (1 * dl_y) + (1 * ds_y))
    p[9] = (ref_x + (7 * dl_x) + (1 * ds_x), ref_y + (7 * dl_y) + (1 * ds_y))
    p[10] = (ref_x + (1 * dl_x) + (5 * ds_x), ref_y + (1 * dl_y) + (5 * ds_y))
    p[11] = (ref_x + (7 * dl_x) + (5 * ds_x), ref_y + (7 * dl_y) + (5 * ds_y))

```

Updated for Simulator:

```

# find places for golf balls
def find_places(self, c):
    # find long side of ball tray
    # this was commented out because you are using a square
    #l1_sq = ((c[1][0] - c[0][0]) * (c[1][0] - c[0][0])) + \
    #((c[1][1] - c[0][1]) * (c[1][1] - c[0][1]))
    #l2_sq = ((c[2][0] - c[1][0]) * (c[2][0] - c[1][0])) + \
    #((c[2][1] - c[1][1]) * (c[2][1] - c[1][1]))

    #if l1_sq > l2_sq:                # c[0] to c[1] is a long side
    #    cc = [c[0], c[1], c[2], c[3]]
    #else:                            # c[1] to c[2] is a long side
    #    cc = [c[1], c[2], c[3], c[0]][0]

    cc = [c[0], c[1], c[2], c[3]]

    # ball tray corners in baxter coordinates
    for i in range(4):
        self.ball_tray_corner[i] = self.pixel_to_baxter(cc[i], self.tray_distance)

    # ball tray places in pixel coordinates
    ref_x = cc[0][0]
    ref_y = cc[0][1]
    dl_x = (cc[1][0] - cc[0][0]) / 8
    dl_y = (cc[1][1] - cc[0][1]) / 8
    ds_x = (cc[2][0] - cc[1][0]) / 8 # was 6
    ds_y = (cc[2][1] - cc[1][1]) / 8 # was 6
    #please see the diagram with the map of the functions location and how
    # they correlate to the colors. The order below is different than that of
    # the original.
    p = {}
    p[0] = (ref_x + (7 * dl_x) + (1 * ds_x), ref_y + (7 * dl_y) + (1 * ds_y))
    p[1] = (ref_x + (7 * dl_x) + (3 * ds_x), ref_y + (7 * dl_y) + (3 * ds_y))
    p[2] = (ref_x + (7 * dl_x) + (5 * ds_x), ref_y + (7 * dl_y) + (5 * ds_y))
    p[3] = (ref_x + (7 * dl_x) + (7 * ds_x), ref_y + (7 * dl_y) + (7 * ds_y))
    p[4] = (ref_x + (5 * dl_x) + (1 * ds_x), ref_y + (5 * dl_y) + (1 * ds_y))
    p[5] = (ref_x + (5 * dl_x) + (3 * ds_x), ref_y + (5 * dl_y) + (3 * ds_y))
    p[6] = (ref_x + (5 * dl_x) + (5 * ds_x), ref_y + (5 * dl_y) + (5 * ds_y))
    p[7] = (ref_x + (5 * dl_x) + (7 * ds_x), ref_y + (5 * dl_y) + (7 * ds_y))
    p[8] = (ref_x + (3 * dl_x) + (1 * ds_x), ref_y + (3 * dl_y) + (1 * ds_y))
    p[9] = (ref_x + (3 * dl_x) + (3 * ds_x), ref_y + (3 * dl_y) + (3 * ds_y))
    p[10] = (ref_x + (3 * dl_x) + (5 * ds_x), ref_y + (3 * dl_y) + (5 * ds_y))
    p[11] = (ref_x + (3 * dl_x) + (7 * ds_x), ref_y + (3 * dl_y) + (7 * ds_y))
    #added these extra rows
    p[12] = (ref_x + (1 * dl_x) + (1 * ds_x), ref_y + (1 * dl_y) + (1 * ds_y))
    p[13] = (ref_x + (1 * dl_x) + (3 * ds_x), ref_y + (1 * dl_y) + (3 * ds_y))
    p[14] = (ref_x + (1 * dl_x) + (5 * ds_x), ref_y + (1 * dl_y) + (5 * ds_y))
    p[15] = (ref_x + (7 * dl_x) + (7 * ds_x), ref_y + (1 * dl_y) + (1 * ds_y))

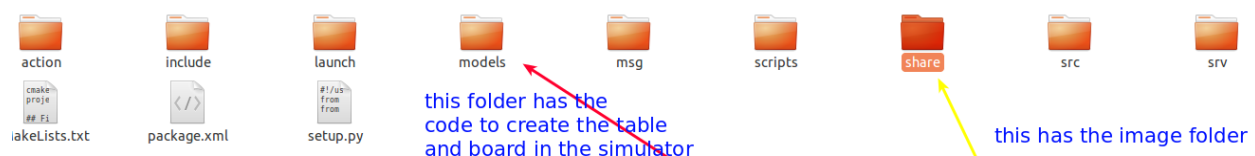
```

Notice the numbers are different from the original due to having 16 instead of 12 positions

Step 3. Navigate to your ros package where you keep your working scripts. To navigate to this /ros_ws/src.

Inside the src directory should be your package. At home, I use a package called test where I do all my “experimenting”.

Here is a picture of mine. It has a lot more folders than yours will have. But the important things to have are the CMakeLists.txt file, the package.xml, scripts, and then you will copy/paste in the model and share folders that I emailed you.



Step 4. Copy/paste the chess_simulator.py file inside the scripts folder. Remember than you must make it executable. So, open a terminal. ctrl -alt- t. Type in cd /ros_ws/src/the name of your package/scripts.

Once you are in the scripts folder, in the terminal, type in: chmod +x chess_simulator.py. That should make the file executable.

Step 5. Open up the chess_simulator.py file. You will need to modify two places with the name of your package.

One is for the path to the image folder and the other is for the path to the model folder.

I put in two pictures to show you where it is in the code.

```
5 # Locate class
6 class Locate():
7     def __init__(self, arm, distance):
8         global image_directory
9         # arm ("left" or "right")
10        self.limb = arm
11        self._rp = rospkg.RosPack()
12        #my package is called test. You need to change this to the name of your package.
13        self._images = (self._rp.get_path('test') + '/share/images')
14        self.limb_interface = baxter_interface.Limb(self.limb)
15        self._joint_names = self.limb_interface.joint_names()
16        print("Getting robot state.... ")
17        self._rs = baxter_interface.RobotEnable(CHECK_VERSION)
18        self._init_state = self._rs.state().enabled
19        print("Enabling robot... ")
20        self._rs.enable()
```

Note that in the function snip below that the gazebo model is being loaded into a certain pose. You can modify the x, y, z, coordinates and put the table and board where you want.

```
#this function loads the models used in gazebo
def load_gazebo_models(table_pose=Pose(position=Point(x= .75, y=0.2, z=0.0)),
                       table_reference_frame="world",
                       block_pose=Pose(position=Point(x=0.6, y=0.25, z=0.7825)),
                       block_reference_frame="world"):

    # Get Models' Path
    model_path = rospkg.RosPack().get_path('test')+"/models/"
    # Load Table SDF
    table_xml = ''
    with open (model_path + "cafe_table/model.sdf", "r") as table_file:
        table_xml=table_file.read().replace('\n', '')
    # Load Block URDF
    block_xml = ''
    with open (model_path + "block/chessboard.sdf", "r") as block_file:
        block_xml=block_file.read().replace('\n', '')
```

Change test to the
of your package

You will need to build the workspace again once everything is in place.

Open a terminal.

cd ros_ws

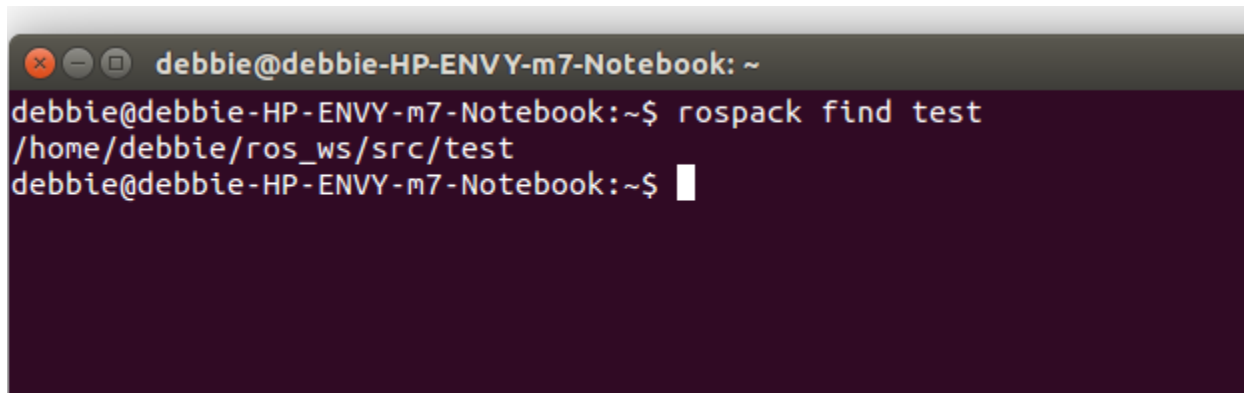
catkin_make

If your .bashrc is set up properly, ros should be able to find your package that contains your scripts.

You can test this by typing the following command in a terminal:

rospack find "the name of your package."

This is what I see when I type the command in and ask it to find my package called test

A terminal window with a dark purple background. The title bar shows 'debbie@debbie-HP-ENVY-m7-Notebook: ~'. The prompt is 'debbie@debbie-HP-ENVY-m7-Notebook:~\$'. The command 'rospack find test' has been entered, and the output is '/home/debbie/ros_ws/src/test'. The prompt is now 'debbie@debbie-HP-ENVY-m7-Notebook:~\$' with a cursor.

```
debbie@debbie-HP-ENVY-m7-Notebook: ~
debbie@debbie-HP-ENVY-m7-Notebook:~$ rospack find test
/home/debbie/ros_ws/src/test
debbie@debbie-HP-ENVY-m7-Notebook:~$
```

If you get an error message that it cannot find the package, then in the terminal type:

```
source ~/ros_ws/devel/setup.bash
```

in the same terminal type:

```
rospack find "your package name"
```

if it finds the package this time, then it means that your .bashrc file is not set up correctly.

Step 6.

To start the simulator:

Make sure you are using an ethernet cable. Wireless does not work.

Check your ip address by opening a terminal and typing in ifconfig.

The address should be in the eth0 section inet addr:

There are two baxter.sh files that should have the correct internet address.

The first is in /ros_ws

The second is in /ros_ws/src/baxter

Open the file and make sure the ip address matches with yours.


```

baxter.sh x
1 #!/bin/bash
2 # Copyright (c) 2013-2015, Rethink Robotics
3 # All rights reserved.
4
5 # This file is to be used in the *root* of your Catkin workspace.
6
7 # This is a convenient script which will set up your ROS environment and
8 # should be executed with every new instance of a shell in which you plan on
9 # working with Baxter.
10
11 # Clear any previously set your_ip/your_hostname
12 unset your_ip
13 unset your_hostname
14 #-----#
15 #                USER CONFIGURABLE ROS ENVIRONMENT VARIABLES                #
16 #-----#
17 # Note: If ROS_MASTER_URI, ROS_IP, or ROS_HOSTNAME environment variables were
18 # previously set (typically in your .bashrc or .bash_profile), those settings
19 # will be overwritten by any variables set here.
20
21 # Specify Baxter's hostname
22 baxter_hostname="baxter_hostname.local"
23
24 # Set *Either* your computers ip address or hostname. Please note if using
25 # your_hostname that this must be resolvable to Baxter.
26 your_ip="192.168.1.116"
27 #your_hostname="my_computer.local"
28
29 # Specify ROS distribution (e.g. indigo, hydro, etc.)
30 ros_version="indigo"
31 #-----#
32
33 tf=$(mktemp)
34 trap "rm -f -- '${tf}'" EXIT
35
36 # If this file specifies an ip address or hostname - unset any previously set

```

Save the file.

If you have open terminal, I would close them all and open a new one.

Launch the baxter simulator with the following commands

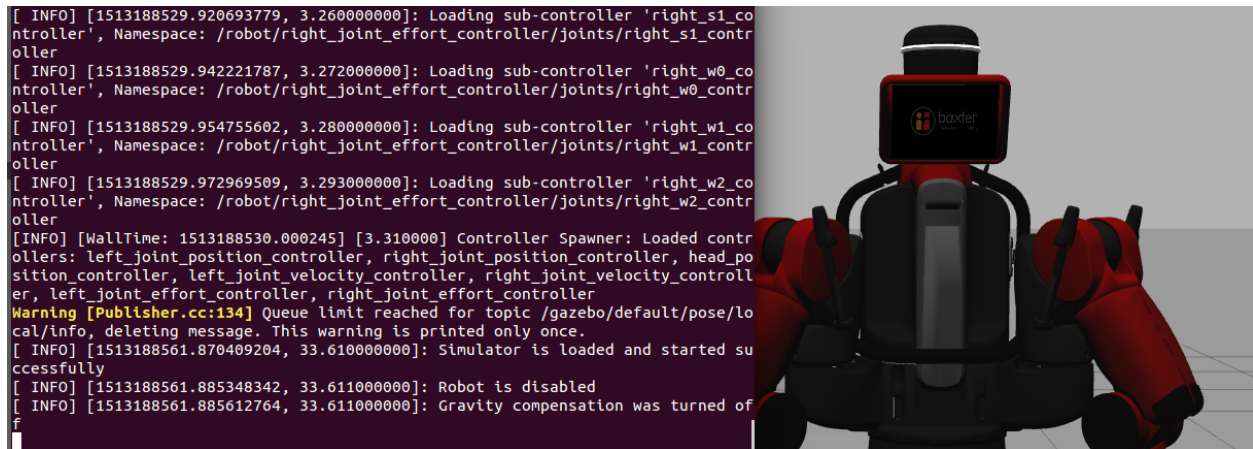
```
cd ros_ws
```

```
./baxter.sh sim
```

```
roslaunch baxter_gazebo baxter_world.launch
```

(don't forget to use the tab key after typing in baxter_gazebo....it will usually auto fill in the rest of the command)

Baxter's simulator should appear. Sometimes there are errors and you have to ctrl c to stop the program and then launch it again.



Sometimes it takes awhile for the simulator to load up. You should see the last two lines in the picture.

Also, Baxter's face screen will have the rethink logo.

When Baxter is done loading, he will probably not be in an orientation that you want. You can move him right/left and up and down by right clicking on your mouse when you are on the baxter robot and moving him. If you have a wheel in your mouse, pressing in allows rotation of Baxter.

Once the simulator is running, run the chess_simulator program with the following in a NEW TERMINAL

```
cd ros_ws
```

```
./baxter.sh sim
```

```
roslaunch "your package name" chess_simulator.py
```

Please let me know if you have any questions.