# *EECS 4421/5324 Intro to Robotics*

## *Assignment 1*

*Joshua Abraham – 214334643*
*Olga Klushina – 215176811*
*Elie Lithwick – 214824254*
*Julia Paglia – 214762926*
*Arvin Tangestanian – 214195796*

# Question 1

In order to refactor the blockrobot.urdf to exploit xacro we started by adding the second line in the code snippet below to tell the program we are using xacro. Then, we added global variables for numbers/dimensions that we saw were repeating. We created a macro for creating the box and cylinder inertia calculations. This allowed the code to look cleaner and change the inertia calculation automatically if we ever changed any of the global dimensions. A snippet of the macros can be seen below, but the full source code for this question can be found in `q1_blockrobot.xacro` on GitHub (see Appendix).

```
<?xml version="1.0"?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro" name="block_robot">

<xacro:property name="base_height" value="0.05" />
<xacro:property name="base_width" value="0.1" />
<xacro:property name="base_length" value="0.1" />
<xacro:property name="cylinder_length" value="0.05" />
<xacro:property name="radius" value="0.05" />


<xacro:macro name="boxInertia" params= "w d h M">
  <inertial>
     <mass value="${M}"/>
     <inertia ixx="${1/12 * M * (h*h+d*d)}" iyy="${1/12 * M * (h*h+w*w)}"
izz="${1/12 * M * (w*w+d*d)}" ixy="0" ixz="0" iyz="0"/>
  </inertial>
</xacro:macro>

 <xacro:macro name="cylinderInertia" params= "h r M">
  <inertial>
    <mass value="${M}"/>
     <inertia ixx="${1/12 * M * h*h + 1/4 * M * r*r}" iyy="${1/12 * M * h*h +
1/4 * M * r*r}" izz="${1/2 * M * r*r}" ixy="0" ixz="0" iyz="0" />
  </inertial>
</xacro:macro>
```

In order to launch the xacro file from the gazebo launch file the following line need to changed from:

```
<param name="robot_description" textfile="$(find cpmr_apb)/urdf/
blockrobot.urdf" />
```

to:

```
<param name="robot_description" command="$(find xacro)/xacro $(find cpmr_apb)
/urdf/blockrobot.xacro" />
```

In the `rviz` launch file the below line was deleted:

```
<arg name="model" value="$(find cpmr_apb)/urdf/blockrobot.urdf" />
```

and the below line was added:

```
<param name="robot_description" command="$(find xacro)/xacro $(find cpmr_apb)
/urdf/blockrobot.xacro" />
```

# Question 2

The objective of this problem was to create a suitable hat for the simple block robot. While it was possible to utilize 3D modeling software and export the hat as a mesh object or .stl file, the team decided it would be beneficial to use the basic 3D solid primitives that can be defined in a URDF file. The geometry classes in a URDF description are a sphere, block (rectangular prism), cylinder, and mesh. A hat was constructed using only spheres, blocks, and cylinders.

It was decided to create a propeller baseball cap, as the cap visor clearly indicates the “front” of the robot, and the design has obvious aesthetic merits. The hat itself is made up of links with rigid joints connecting everything. It would be fun to change the propeller stem to a rotational joint, with 1 DOF about the Z axis, but that was considered too complex for the problem description. Figure 1 displays a side-by-side comparison of the hat rendered in `RVIZ` with the reference image.

This question was a good exercise in working through and understanding a URDF file. Although all the joints in this example were rigid, it would not be too much more of an extension to apply revolute and prismatic joints. The real concern with this question was calculating the appropriate sizes for each shape, as well as the required origin offsets and rotations. By going through the ROS documentation, it was understood that joints in URDF not only “connect” links, but also provide the coordinate transformations necessary to go from the parent frame to the child frame. While not very important when designing a rigid hat, this will be crucial when calculating forward kinematics for something like a manipulator. Figure 2 displays the hierarchy of joints and links for this modified robot, generated using `urdf_to_graphiz`. See `q2_blockrobot.urdf` on GitHub for the full source code for this question.

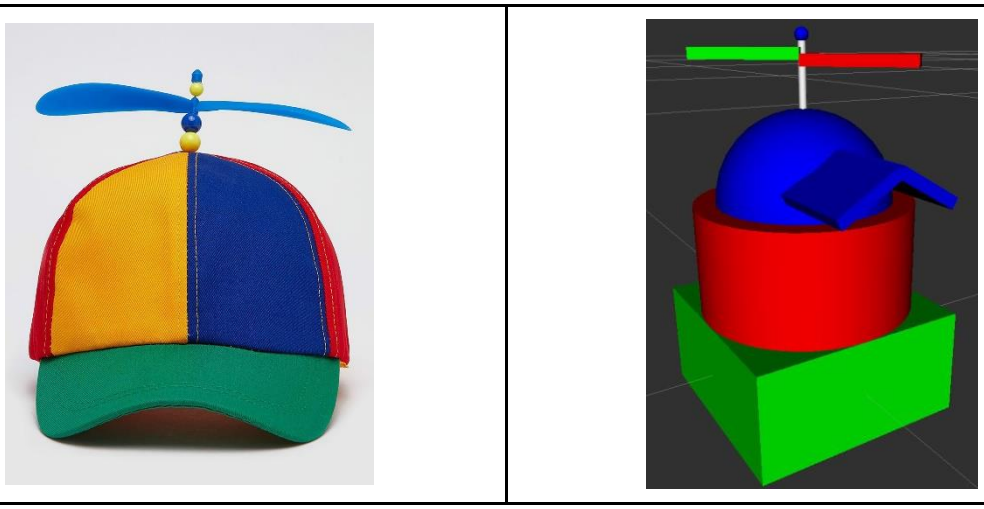

***Figure 1:*** Reference image (left) and `RVIZ` rendering of the robot with hat (right). The bill of the cap points along the local x axis of the robot’s base.

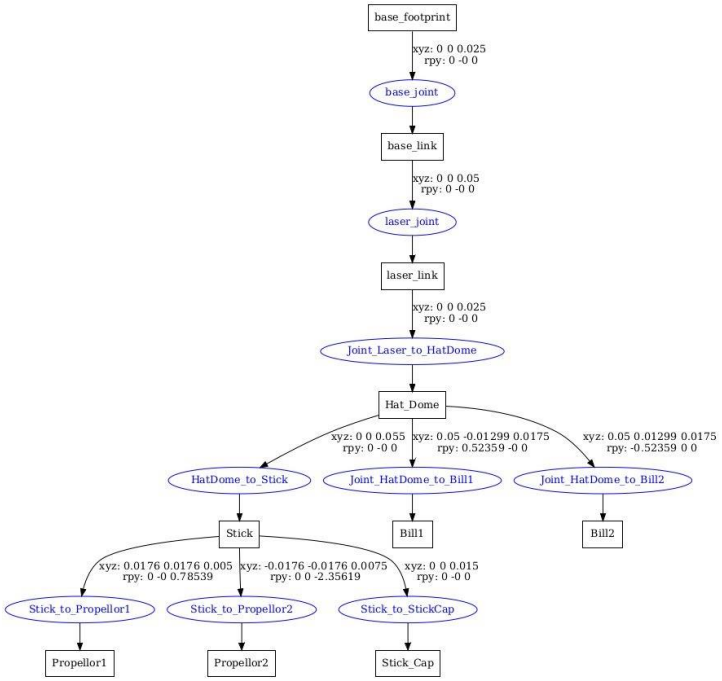


***Figure 2:*** *`Graphiz`-generated tree indicating links (in blocks), and joints (in blue ovals), as well as the coordinate transformations from each parent to child link.*

# Question 3

For this problem, our team decided to use the Husky UGV created by Clearpath Robotics. This robot is an outdoor unmanned ground vehicle that is intended for research usage. The original source code for the Husky UGV including the description files can be found at: https://github.com/husky/husky.

These description files were originally written in xacro, so our team needed to run additional commands to export them to a standard URDF file, which we have included in the code block at the end of the document. Source code files for this question can be found in `q3-husky-description.zip` on GitHub (see Appendix).

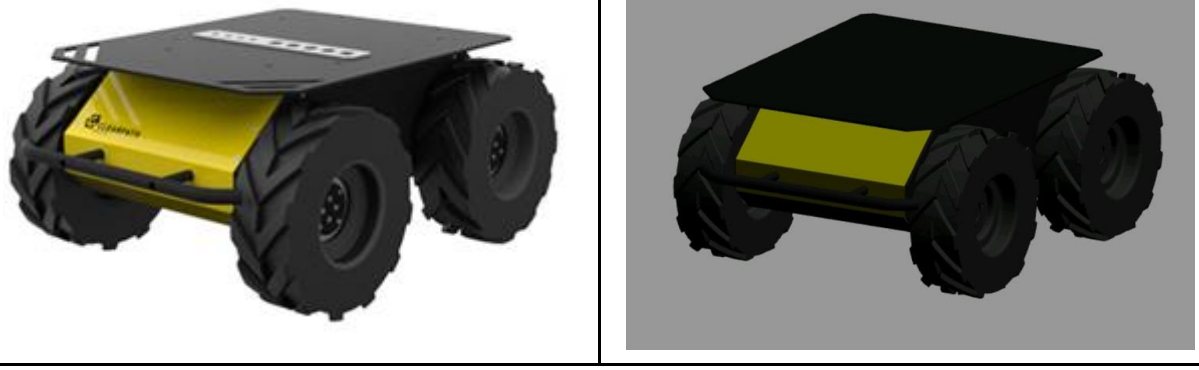

***Figure 3:*** Original robot from Clearpath's website (left) and its respective Gazebo simulation (right).

After this, our team worked on adding motion to the robot by connecting it to the teleop-twist library. This required additional configurations because of the specialized naming within the URDF links.

Figure 4 shows two different movements that we demonstrated. The first image demonstrates the robot travelling in a circular pattern about the origin. The second image was to demonstrate linear motion, which allowed the robot to drive in a straight line parallel to the gridlines. A video was also taken to demonstrate this motion, and can be seen at: https://youtu.be/iggSqLHsu5E

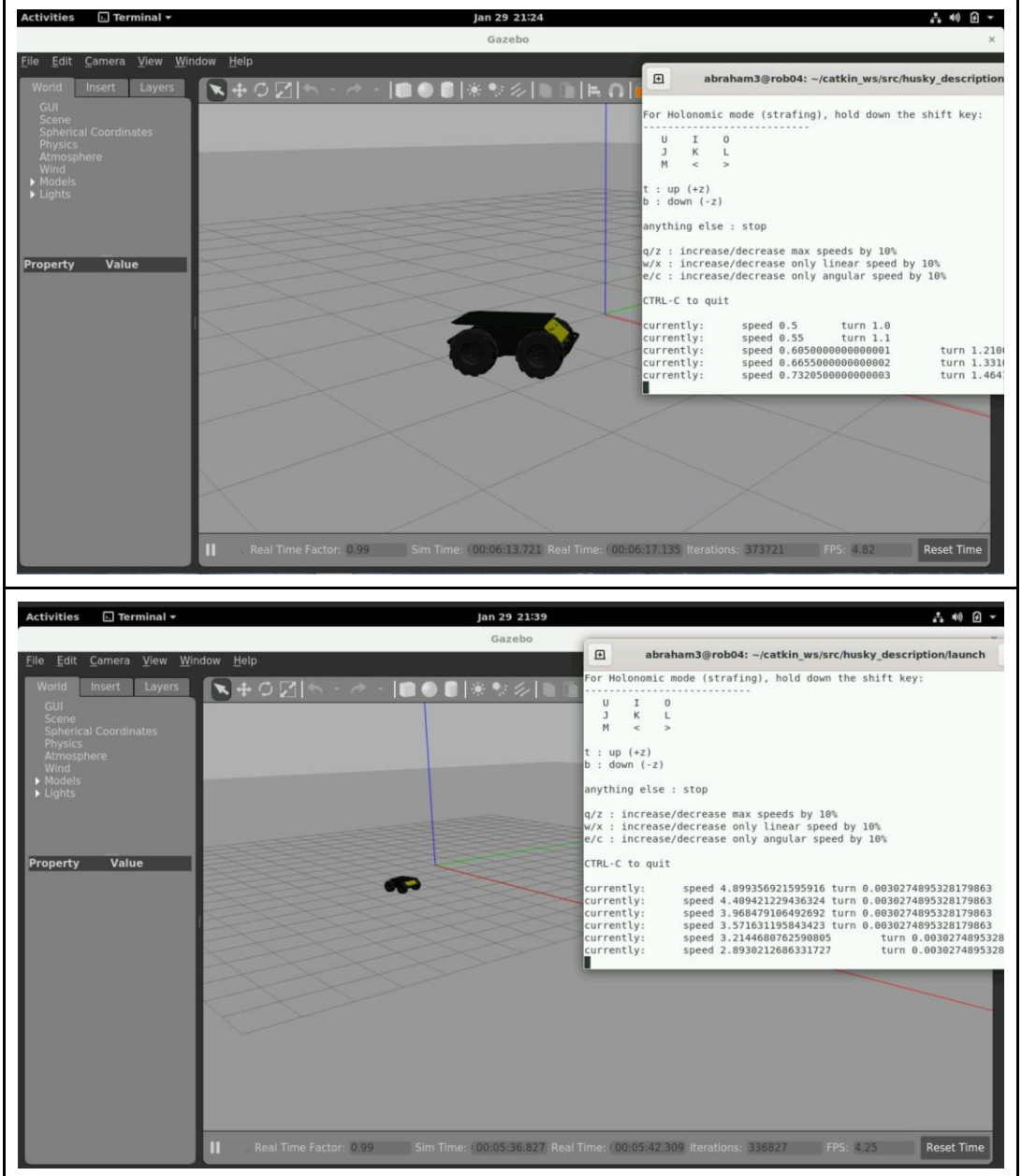


***Figure 4:*** Screenshots of the robot travelling in a circular motion (top) and linear motion (bottom).

# Question 4

The problem asked us to model a more complex world in Gazebo, so our team decided to simulate a café setting. Initially, the development of this world was done directly in Gazebo using more advanced components than the simple coke cans we saw in class. We utilized the models for the café building, café tables, cabinets, bookshelves, and finally a Clearpath Husky (as a tribute to our robot from Question 3). The café, tables, bookshelf, and cabinet were set to be static components so that the block robot could not move anything if it bumped into them.

After developing the world manually, we decided to create a python script similar to `populate.py` to automate the creation and population of our custom world. The source code for this script can be found in `q4_build_cafe.py` on GitHub (see Appendix). Figures 5 and 6 encapsulate multiple screenshots of the block robot interacting with our new world.

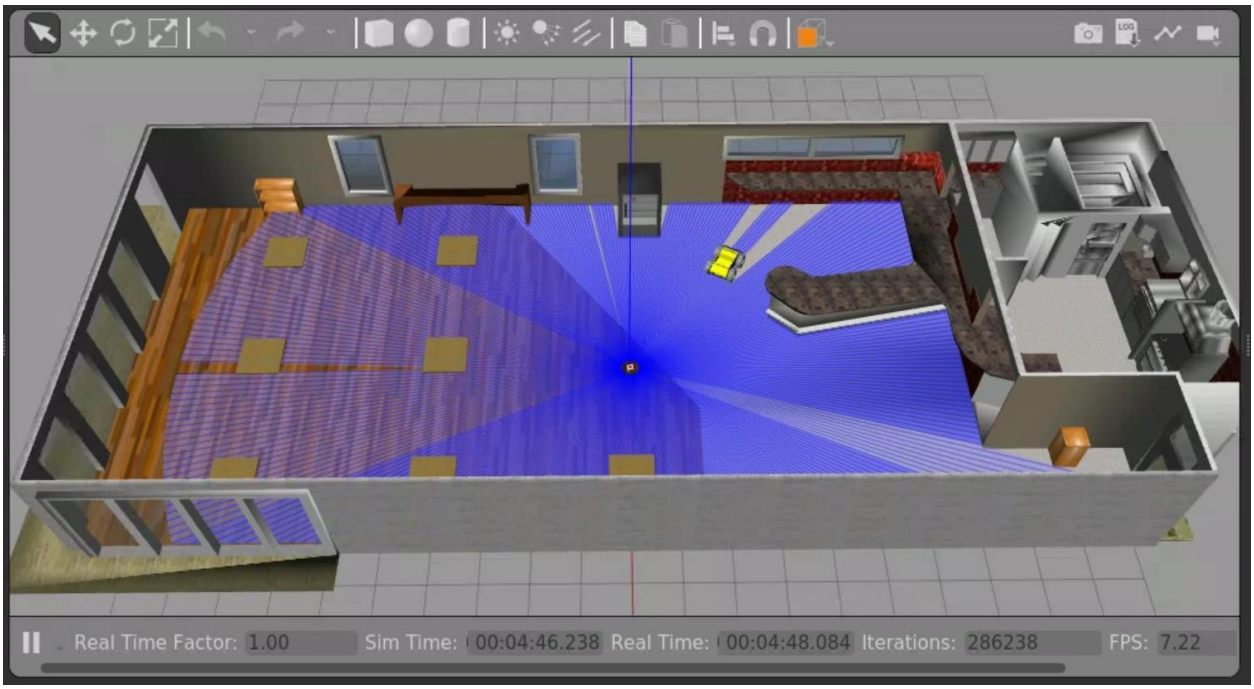


***Figure 5:*** Overview of the custom world of a café setting.

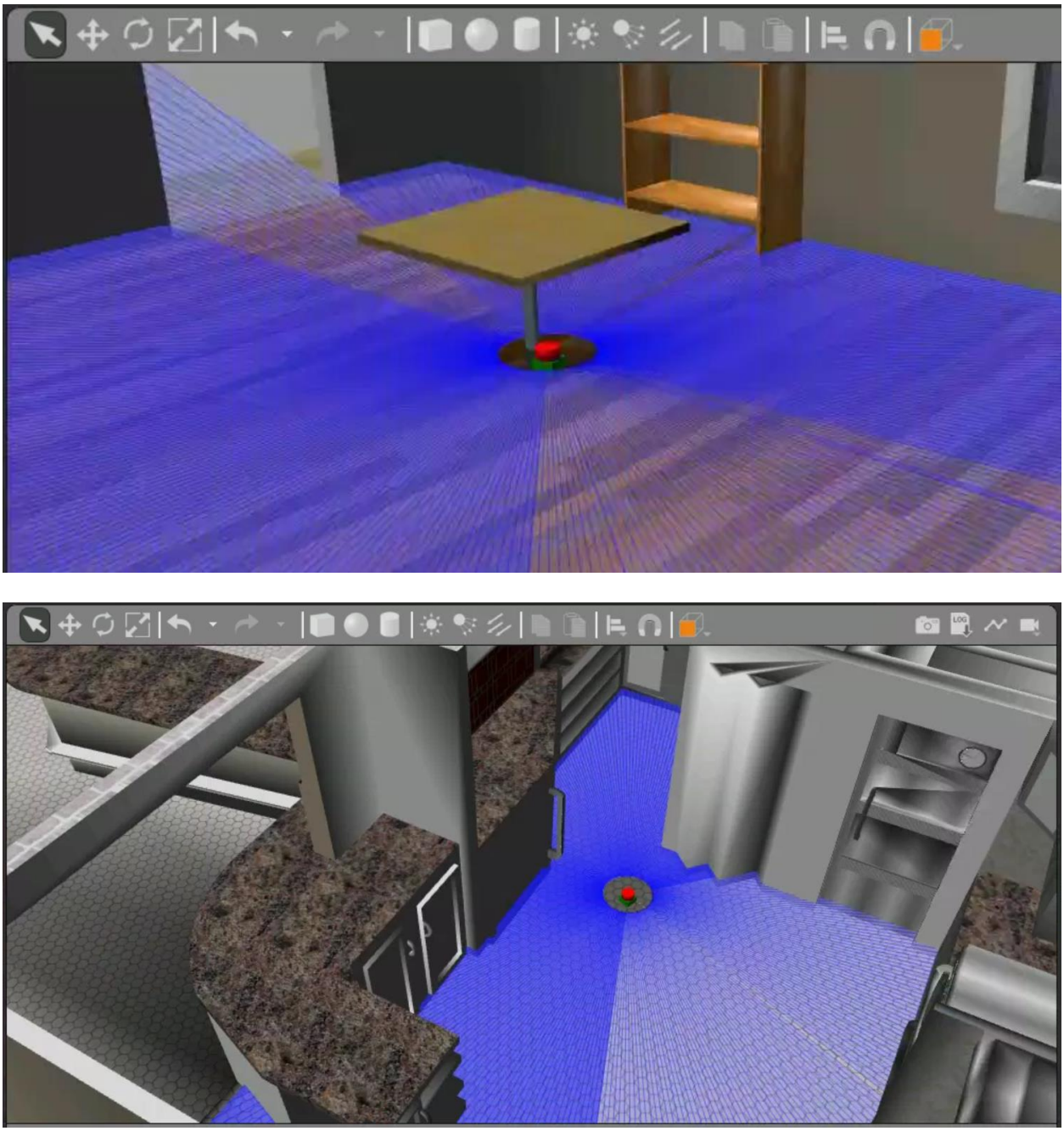

***Figure 6:*** Additional screenshots of the robot interacting with the custom café world.

# Question 5

For this question, our team had to add in a camera to the original blockrobot provided to us, on top of the existing LIDAR sensor. This involved adding in the following 3 elements: a link, a joint, and the Gazebo sensor. The URDF for this question can be found in `q5_blockrobot.urdf` on Github (see Appendix).

The link code is shown below:

```
<gazebo reference="camera_link">
    <material>Gazebo/Blue</material>
</gazebo>
<link name="camera_link">
  <visual>
    <geometry>
      <sphere  radius="0.025" />
    </geometry>
    <material name="blue"/>
  </visual>
  <collision>
    <geometry>
      <sphere radius="0.025" />
    </geometry>
  </collision>
  <inertial>
    <mass value="0.01" />
    <inertia ixx="8.33e-6" iyy="8.33e-6" izz="1.25e-7" ixy="0" ixz="0" iyz="0"/>
    </inertial>
  </link>
```

Additionally, a joint needed to be added that would connect the camera sensor to the LIDAR sensor, which is done within the following joint code:

```
  <joint name="camera_joint" type="fixed">
    <origin xyz="0.0 0.0 0.025" rpy="0.0 0.0 0.0" />
    <parent link="laser_link" />
    <child link="camera_link" />
    <axis xyz="0 0 1" />
  </joint>
```

Finally, the logic for the sensor itself needed to be added in for simulation purposes in Gazebo, that logic was completed in the final major snippet of code. Note that additional logic was added to this code to allow for the ability to publish a topic from the camera's view so that the camera perspective could be demonstrated within Gazebo.

```
<gazebo reference="camera_link">
  <static>true</static>
  <sensor type="camera" name="camera">
        <camera name="head">
          <horizontal_fov>1.05</horizontal_fov>
          <image>
            <width>320</width>
            <height>320</height>
          </image>
          <clip>
            <near>0.1</near>
            <far>100</far>
          </clip>
        </camera>
        <plugin name="camera_controller" filename="libgazebo_ros_camera.so">
        <always_on>1</always_on>
        <update_rate>30</update_rate>
        <visualize>true</visualize>
        <cameraName>blockrobot/camera</cameraName>
        <cameraInfoTopicName>camera_topic</cameraInfoTopicName>
        <frameName>camera</frameName>
        </plugin>
      </sensor>
</gazebo>
```

Figure 7 shows what the augmented block robot looks like with a blue camera mounted on the top of the LIDAR sensor. We also created a simple world for the robot to look at, in order to demonstrate the mounted camera perspective, as observed in Figure 8. The world can be seen from our view in the larger window, and the smaller window on the left demonstrates the camera's perspective which we can see by using a `rosrun` command on the `image_raw` topic.

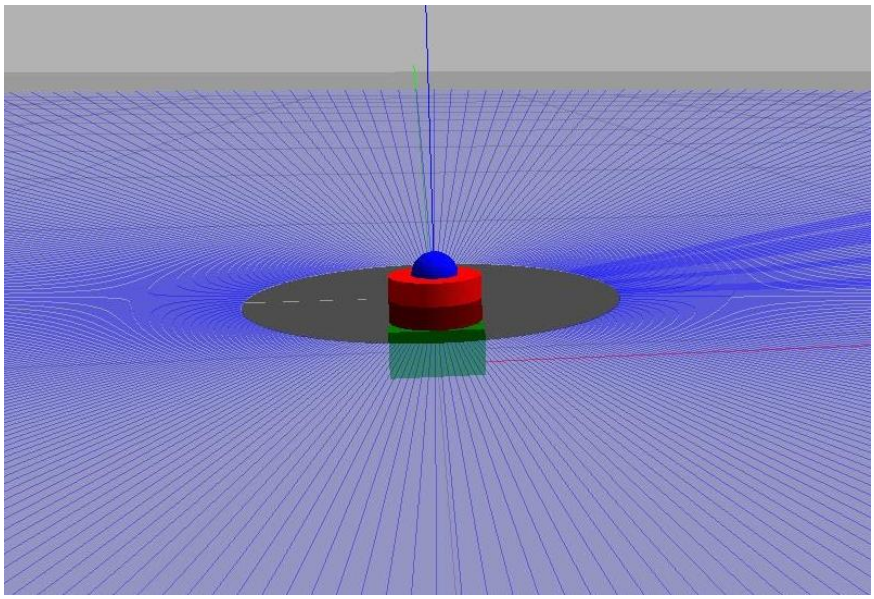

***Figure 7:*** Augmented blockrobot with blue camera mounted on the LIDAR sensor.

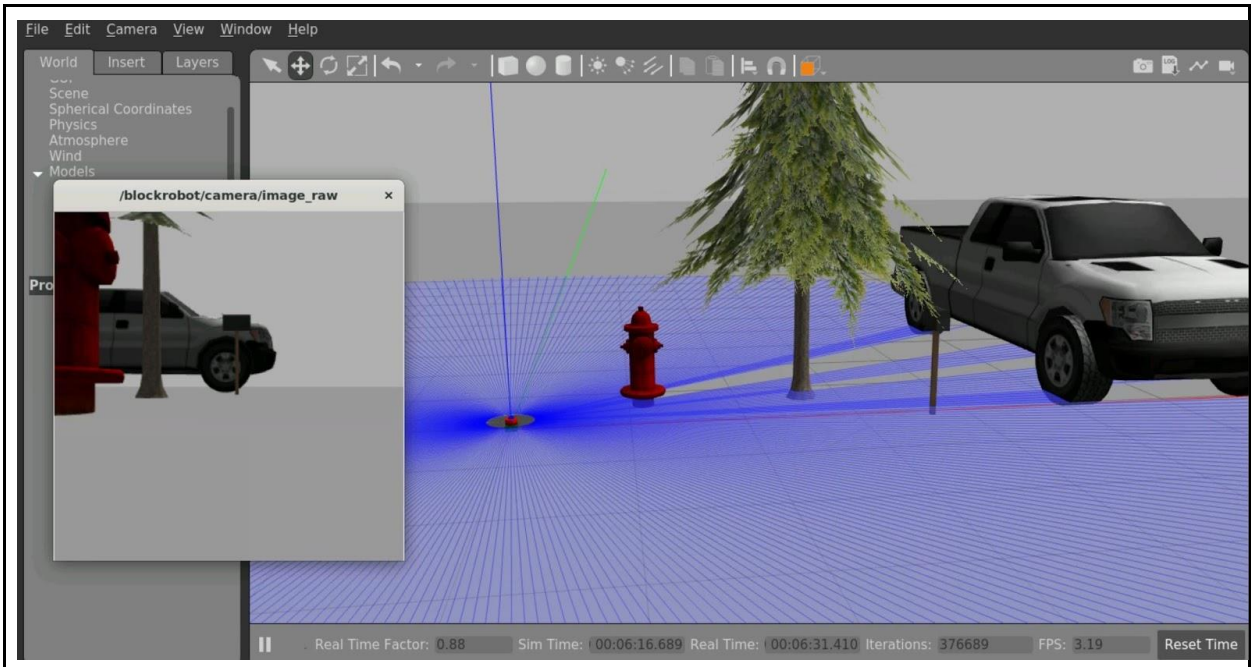


***Figure 8:*** Mounted camera perspective as seen by the blockrobot.

# Appendix

All of our source code referenced throughout this document can be found on GitHub at: https://github.com/jpaglia/eecs4421-assignment-code/

Files are found in the subfolder `/Assignment 1/` as follows:

1) Question 1 Code: `q1_blockrobot.xacro`
2) Question 2 Code: `q2_blockrobot.urdf`
3) Question 3 Code: `q3-husky-description.zip`
4) Question 4 Code: `q4_build_cafe.py`
5) Question 5 Code: `q5_blockrobot.urdf`