Appendix B
Software Infrastructure

EECS 4421, Intro. to Robotics

Course Instructor: Michael Jenkin
Teaching Assistant: Robert Codd-Downey

Report Contributors

Ramavath, Sai Varun	213506936
Patel, Henilkumar	215245707
Rego, Jared	214843874

4th Year
Electrical Engineering
and Computer Science

Contents

List of Figures

Listings

1 Refactor `block_robot` to exploit `xacro`.

1.1 Motivation

The `xacro` package operates by parsing macros defined in a .xacro file into a .urdf file that can be passed on to gazebo and rviz for simulation. It provides several options for defining robot links, joints, geometries, parameters and anything else that needs to be defined more than once which helps immensely in simplifying calculations and improving code conciseness.

1.2 Setup and Understanding `xacro`

A useful feature of `xacro` is the ability to create multiple macro files and include them in a base .xacro file that gets parsed into .urdf, making it much easier to read through an entire complex robot definition. For this exercise, the robot dimensions and inertia calculations were first defined in the .xacro file `blockrobotdiminert.xacro` (Listing 4) which was then included in `blockrobot.xacro` via the addition of an include tag at the top of the file:

```xml
<!-- IMPORT PARAMETERS FILE -->
<xacro:include filename="$(find cpmr_apb)/urdf/blockrobotdiminert.xacro" />
```

The extent of macro definitions was kept simple in this exercise in an attempt to understand and familiarize ourselves with the main functionality of the package, but its use in later exercises will be more robust.

Macro parsing takes place on issuing the following command in a bash terminal:

```
xacro -o foo.urdf blockrobot.xacro
```

which outputs the expanded macros to `foo.urdf` in the current working directory - note: omitting the output file declaration outputs the contents of the parsed file to `stdout`.

While this solution provides the desired results, it is inconvenient to manually run it every time one makes a change to the robot properties; luckily, an automated solution exists via launch file modifications. The cpmr_apb directory provided a way to find the `block_robot` urdf file located in `cpmr_apb/urdf/`, and assign it to a parameter named "robot_description". The change needs to be made in this line by making ROS process the command mentioned above:

```xml
<!-- AUTOMATE XACRO TO URDF PARSING -->
  <param name="robot_description" command="xacro '$(find cpmr_apb)/urdf/blockrobot
  .xacro' prefix:=$(arg tf_prefix)" />
```

which tells the launch file to look in the cpmr_apb project directory, find the main .xacro file and run the parsing command as described above. Since this is in the launch file(s), it runs every time a simulation is performed through gazebo or rviz. With the basic setup complete, let us take a look at how `xacro` has been implemented for this exercise.

1.3 Implementation

The `blockrobotdiminert.xacro` file initializes the dimensions of the box and cylinder links used for the `block_robot`, along with their masses. This information is then used to calculate the inertia of each link based on formulae provided in comments in the `blockrobot.urdf` file that came with the project directory.

The syntax is quite easy to follow at this stage: a `xacro` property can be initialized using the `xacro:property` tag which takes name and value as arguments. These names can be used elsewhere in the .xacro file (or across multiple .xacro files where `blockrobotdiminert.xacro` has been included) much like C constants defined in a header file. The syntax used to reference these properties is: `${foo}`. For example, Listing 3 shows how the `base_link` was defined using properties defined in Listing 4 - note: here b_ refers to a box geometry, while c_ refers to a cylinder.

```
  <!-- BASE LINK DEFINITION -->
  <link name="$(arg tf_prefix)/base_link">
    <visual>
      <geometry>...</visual>
    <inertial>
      <mass value="${b_m}" />
      <inertia ixx="${b_ixx}" iyy="${b_iyy}" izz="${b_izz}" ixy="0" ixz="0" iyz="0"
    />
    </inertial>...</link>
```

Listing 3: An example call of `xacro` properties.

```
<?xml version="1.0"?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro">
  <!-- BOX PROPERTIES -->
  <xacro:property name="b_m" value="4.5" />
  <xacro:property name="b_h" value="0.05" />
  <xacro:property name="b_w" value="0.10" />
  <xacro:property name="b_l" value="0.10" />

  <!-- CYLINDER PROPERTIES -->
  <xacro:property name="c_m" value="0.01" />
  <xacro:property name="c_h" value="0.05" />
  <xacro:property name="c_r" value="0.05" />

  <!-- XACRO DEFINITIONS BASED ON FORMULAE IN COMMENTS ABOVE -->
  <xacro:property name="b_ixx" value="${1/12 * b_m * (b_h * b_h + b_l * b_l)}" />
  <xacro:property name="b_iyy" value="${1/12 * b_m * (b_h * b_h + b_w * b_w)}" />
  <xacro:property name="b_izz" value="${1/12 * b_m * (b_w * b_w + b_l * b_l)}" />
  <xacro:property name="c_ixx" value="${1/12 * c_m * c_h * c_h + 1/4 * c_m * c_r *
    c_r}" />
  <xacro:property name="c_iyy" value="${1/12 * c_m * c_h * c_h + 1/4 * c_m * c_r *
    c_r}" />
  <xacro:property name="c_izz" value="${1/2 * c_m * c_r * c_r}" />
</robot>
```

Listing 4: `blockrobotdiminert.xacro` used to simplify inertia computations when defining the various links in `blockrobot.xacro`.

2 Provide a hat to the top of the robot.

2.1 Motivation

The objective of this exercise is to get comfortable designing robots using the Universal Robot Description Format (URDF). The URDF model of a robot can make or break simulations which makes it crucial that a person simulating robots be able to troubleshoot such files for making quick tweaks and improvements on the simulations. The hat is a good intro on robot design since it requires a minimum understanding of relative frame definitions, yet is not a complex structure.

2.2 Implementation

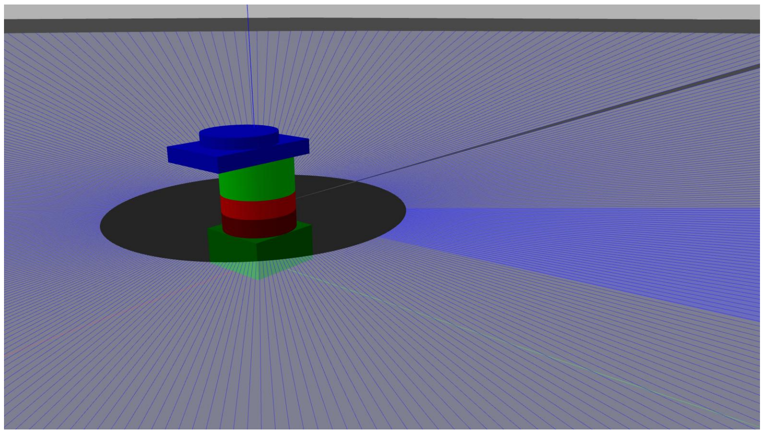

Figure 1: The `block_robot` looking much more adorned with a hat that points in the direction of forward linear motion (along the x-axis). Note: it is evident that the hat is pointing in the correct direction by observing the laser lines which are incremented at 1° starting from the back of the robot, therefore leaving a gap in the laser coverage behind the robot.

In order to add a hat to the block robot, the object is first described using xml syntax within the `blockrobot.xacro` file. The hat is made to resemble a baseball cap, with the box representing the brim of the hat and a cylinder to represent the crown of the hat. Both components of the hat are comprised of the base geometry of the shape as well as a collision shape with the same dimensions as the base geometry. Listing 5 displays the xml descriptions of these shapes in the `blockrobot.xacro` file.

```
<!-- HAT BRIM LINK DEFINITION -->
<link name="$(arg tf_prefix)/hat_link_brim">
  <visual>
    <geometry>
      <box size="0.15 0.1 0.02" />
    </geometry>
    <material name="green"/>
  </visual>
  <collision>
    <geometry>
      <box size="0.15 0.1 0.02" />
    </geometry>
  </collision>
  <inertial>
    <mass value="1e-5" />
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <inertia ixx="1e-6" ixy="0" ixz="0" iyy="1e-6" iyz="0" izz="1e-6" />
  </inertial>
</link>
<!-- HAT TOP LINK DEFINITION -->
<link name="$(arg tf_prefix)/hat_link_top">
  <visual>
    <geometry>
      <cylinder length="0.02" radius="0.05" />
    </geometry>
    <material name="green" />
  </visual>
  <collision>
    <geometry>
      <cylinder length="0.02" radius="0.05" />
    </geometry>
  </collision>
  <inertial>
    <mass value="1e-5" />
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <inertia ixx="1e-6" ixy="0" ixz="0" iyy="1e-6" iyz="0" izz="1e-6" />
  </inertial>
</link>
```

Listing 5: Hat link definition.

In Listing 5 above, the box shape is described using the `<geometry>` tag with dimensions L:0.15m, W:0.1m, H:0.02m and given the name `hat_link_brim`. The `<material>` tag is used to specify the color of the shape, which in this case is set to be blue. These components specify the visual attributes of the brim for the gazebo simulator. The collision box specifies the region in which the brim collides with other objects. In this case, the collision box is specified to have the same geometry as the brim so that a collision only occurs if the brim is in direct contact with another object.

Similar to the construction of the brim, the top of the hat is specified to be a cylinder with radius 0.05m and a height of 0.02m and is given a blue color as well. Once again, the `<geometry>` tag is used to specify the shape and the `<material>` tag, to specify the color.

```
  <!-- HAT BRIM JOINT DEFINITION -->
  <joint name="hat_joint_brim" type="fixed">
    <origin xyz="0.025 0.0 0.035" rpy="0.0 0.0 0.0" />
    <parent link="$(arg tf_prefix)/camera_link" />
    <child link="$(arg tf_prefix)/hat_link_brim" />
    <axis xyz="0 0 1" />
```

```
</joint>
<!-- HAT TOP JOINT DEFINITION -->
<joint name="hat_joint_top" type="fixed">
  <origin xyz="0.0 0.0 0.015"/>
  <parent link="$(arg tf_prefix)/hat_link_brim" />
  <child link="$(arg tf_prefix)/hat_link_top" />
  <axis xyz="0 0 1" />
</joint>
```

Listing 6: Complete hat joint definition.

With the shapes of the hat defined, the next step is to connect them together. To do this a fixed joint is used to connect the overall structure to the top of the robot (to the top of the `laser_link`). Listing 6 shows the additions to the `blockrobot.xacro` file used to achieve this.

Here, the joint is specified with a name and a type. In this case, the joint is called the `hat_joint` and is a fixed type. The origin specifies where the object (child object) is placed relative to the center of another object in the world (parent object) in xyz coordinates. For example, the box `hat_link_brim` that was specified in the previous figure is the child object that is placed 0.025m in the positive x direction and 0.025m in the positive y direction in relation to the center of the `camera_link`. These offset values were chosen so that the box is placed directly on the camera module. Since the camera module has a height of 0.05m, the box needs to be placed with an offset of 0.025m in the z direction relative to the center of the laser so that it is placed directly on top of the laser. Similarly, the second joint specifies where the crown of the hat is placed, which in this case is offset by a distance of 0.045m in the z direction, which accounts for the distance from the center of the laser to the top (0.025m) as well as the thickness of the box (0.02m). The axis tag specifies the axis along which the objects are allowed to rotate, which in this case is defined as the z-axis to be consistent with the rest of the `block_robot` definition.

3 Download and drive a URDF description.

3.1 Motivation

The ability to download and simulate URDF files created by other people allows for quick and easy collaboration on large projects.

3.2 Implementation

In order to test Gazebo with a pre-made robot, we decided to use the `turtlebot3` burger robot model from ROBOTIS. We chose this robot as it came with a variety of functions as well as other options from the `turtlebot3` family (such as the waffle). In order to import the robot files to the workspace, the first step was to clone the git repository of the robot into the catkin workspace. Specifically, the following command was run in the terminal to clone the files from github into the /src folder.

```
$ git clone https://github.com/ROBOTIS-GIT/turtlebot3.git
```

. In order to move the robot around in an empty world, the teleop has to be run first. This was done using the following terminal command:

```
$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

This command uses the parameters set in the launch file to specify the type of turtlebot3 model used (in this case it was the turtlebot3 burger) as well as the files for keyboard operation.

Finally, the model was launched in an empty world with the following command:

```
$ roslaunch turtlebot3_gazebo turtlebot3_empty_world.launch
```

3.3 Driving the Turtlebot

The following video demonstrates the `turtlebot3` being drive around in an empty world.

`https://www.youtube.com/watch?v=t8swMgmduLw&feature=youtu.be`

4 Add a visual light camera to `block_robot`.

4.1 Motivation

A robot is defined as a device that can: (1) perform actions (act on its environment), (2) perceive/understand the world sufficiently well to perform these actions (sensing), (3) interact with the environment (planning, representation) [1]. Given this, a visual light camera has a wide range of applications for robots; a mobile robot's ability to navigate in an environment efficiently depends heavily on its ability to detect obstructions and determine an optimal path to its objective; a robotic manipulator must have the ability to detect lines and edges to properly categorize items of interest, and so on.

4.2 Setup and Understanding Gazebo Plugins

Gazebo is a simulator that takes in URDF descriptions of robots and simulates them in a world environment of choice (the empty_world for this exercise). The models can then be controlled/interfaced with using Gazebo sensor plugins that add a vast amount of customizability and functionality to a given robot. For example, the `block_robot` was provided to us with the planar movement plugin that allows it to move around the Gazebo world in 2D, and a ranging laser plug-in that would allow the robot to detect objects in its environment.

4.3 Implementation

The camera implementation was done by following one of the official Gazebo tutorials, and for this reason most of the parameters were kept the same. Note: some code in this section includes snippets of the next exercise. The tutorial can be accessed at:

`http://gazebosim.org/tutorials?tut=ros_gzplugins`

The camera setup starts by defining the link on the robot that will define the camera's "housing", which in our case is a replica of the `laser_link` to maintain a uniform robot profile. The process for achieving this is virtually the same as that of the `hat_link_top` definition seen in Section 2.2 of this report. Listing 7 shows the exact parameters used to define the camera on the `block_robot`, complete with an inertial definition.

```
<!-- CAMERA LINK DEFINITION -->
<link name="$(arg tf_prefix)/camera_link">
  <visual>
    <geometry>
      <cylinder length="${c_h}" radius="${c_r}" />
    </geometry>
    <material name="red"/>
  </visual>
  <collision>
```

[1] *Computational Principles of Mobile Robotics* slide 3, Michael Jenkin

```
      <geometry>
        <cylinder length="${c_h}" radius="${c_r}" />
      </geometry>
    </collision>
    <inertial>
      <mass value="${c_m}" />
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <inertia ixx="${c_ixx}" iyy="${c_iyy}" izz="${c_izz}" ixy="0" ixz="0" iyz="0"
    />
    </inertial>
  </link>
```

Listing 7: Camera link definition.

Next we define the camera joint that joins it with the `laser_link` using the joint examples that were provided with the cpmr_apb project directory; the origin offsets are calculated using the same logic from the hat definition. The `axis` in Listing 9 refers to the axis of rotation; in the interest of keeping with the ”Hello World”-esque nature of this assignment, this has been defined only around the z axis (vertical dimension). The final parameter of interest here is, like the `hat_link`, the offset in the z-direction indicating that the camera is stacked on top of the laser link.

```
  <!-- CAMERA JOINT DEFINITION -->
  <joint name="camera_joint" type="fixed">
    <origin xyz="0.0 0.0 0.05" rpy="0.0 0.0 0.0" />
    <parent link="$(arg tf_prefix)/laser_link" />
    <child link="$(arg tf_prefix)/camera_link" />
    <axis xyz="0 0 1" />
  </joint>
```

Listing 8: Camera joint definition.

As a side note, Listing 7 again showcases the power of the `xacro` package since defining a second link identical to another is a simple matter of calling in the relevant dimension and, subsequently, inertial macros.

The camera setup continues by defining a gazebo reference that will identify this instantiation of the camera and will determine its name definition in the `tf` transformation tree (more on this in the next exercise). This differs from the actual name given to the camera which is used for topic and node references in ROS (camera name is *vislight_cam* in this exercise). The entire camera definition can be found in Listing 13, but as mentioned, many of the parameters have been kept the same as the tutorial.

```
  <!-- GAZEBO CAMERA PLUGIN -->
  <gazebo reference="$(arg tf_prefix)/camera_link">
    <sensor type="camera" name="vislight_cam">
      <update_rate>30.0</update_rate>...</gazebo>
```

Listing 9: Camera Gazebo reference and plugin definition.

4.4 Viewing the Output

The first point to note is that Gazebo provides robot simulation tools, as mentioned above, but does not feature any plug-in visualization abilities. Any 3D visualizations such as camera feeds, laser scans and so on must be done in RVIZ, which logs sensor information and provides tools to view the sensor information.

The next point to note is that the camera node publishes *topics* (that other nodes can subscribe to) under the camera name `vislight_cam` as discussed above. The camera feed is published by `vislight_cam` to the topic `image_raw`; the RVIZ node subscribes the this topic and processes the information from the feed. Therefore, in order to see this information, RVIZ needs to be launched using the provided launch file in the cpmr_apb directory.

Once the RVIZ GUI has launched, the camera can be added by using the *Add* button in the lower left side of the window, selecting Camera, and the topic of interest under the camera tab. The camera feed should show up in the lower left side of the window. The robot can now be driven around the world (via a teleop keyboard or `rostopic pub`) and the camera will provide a way to see what the robot sees. The output of the `block_robot`'s camera feed is shown in Fig. 2 with some reference objects placed.

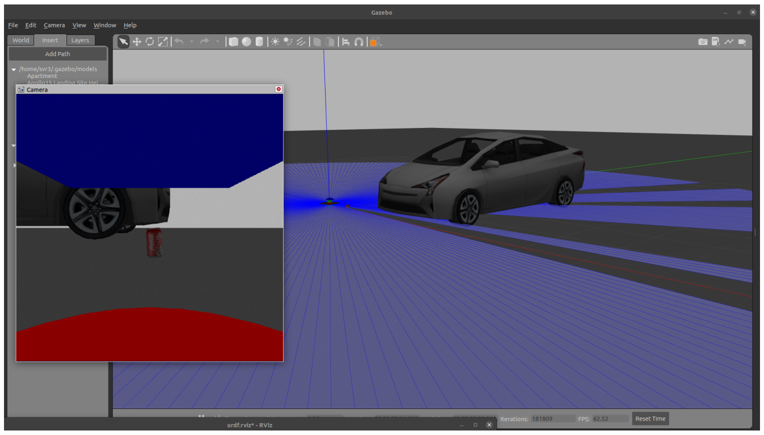

Figure 2: The `block_robot's` camera view (looking forward, as indicated by the hat), looking at the empty world with some random objects inserted for reference.

5 Assigning a namespace to block_robot.

5.1 Motivation

Namespaces are a powerful feature in ROS that allow for easy to debug multi-robot simulations.

5.2 Setup and Understanding Namespaces

By default, topics are published at the top-level of the name hierarchy (the '/' global namespace), and subscribers also belong to this namespace. Having one robot in this namespace does not create any problems since there is only one instance of important robot control mechanisms and sensor information (such as `odom, cmd_vel, vislight_cam/image_raw` and so on) that needs to be communicated in the network and therefore can be identified easily. With two or more robots, there are now multiple instances of these topics that need to be published and subscribed to which makes it crucial that ROS can distinguish between the robots and communicate with the desired robot.

Namespaces in ROS provide a way to "push" topics and nodes down into unique categories that can be communicated with individually, therefore solving problems such as moving both robots at the same time in one publish to the `cmd_vel` topic.

5.3 Implementation

Changing the namespace hierarchy in ROS has two main steps: (1) define links in the `blockrobot.xacro` with a parameter that would allow to dynamically change their prefix, and (2) modifying the launch files to pass in the prefix on model compilation. The first step is necessary to ensure that `tf` can correctly publish link transformations (read: link status updates) to the topics with subscribers that depend on this information; like RVIZ. Listing 10 shows the required parameter definition in the `blockrobot.xacro` and an example use case of the parameter.

```
<!-- NAMESPACE DEFINITION TF_PREFIX -->
<xacro:arg name="tf_prefix" default="br" />
  ...
<!-- BASE LINK DEFINITION -->
<link name="$(arg tf_prefix)/base_link">
```

Listing 10: Defining a robot namespace in the robot description file

Listing 11 shows how the `gazebo.launch` file has been modified to pass the prefix into the robot model description file.

```
<!-- AUTOMATE XACRO TO URDF PARSING -->
<group ns="$(arg tf_prefix)">
  <param name="robot_description" command="xacro '$(find cpmr_apb)/urdf/blockrobot
  .xacro' prefix:=$(arg tf_prefix)" />
```

```
    <node name="spawn" pkg="gazebo_ros" type="spawn_model" args="-robot_namespace br
     -unpause -param robot_description -urdf -model br_spawn" />...</group>
```
Listing 11: Defining a robot namespace in the Gazebo launch file.

The `<group>` tag has been used in this case to eliminate the need to define namespaces individually for the "spawn" and "rsp" nodes that spawn the robot model, and publish information pertaining to the current robot state, respectively.

5.4 Visualizing Namespaces

The namespaces defined in the previous step can be visualized in various ways, but this section will provide the rqt_graph which shows a descriptive flow chart of all nodes and topics and how they are connected. Fig. 3 shows two robots that have been instantiated for this example - under different namespaces - and a move command has been published to their respective `cmd_vel` topics. It can be seen that each robot gets a different command, as opposed to one move command, and that each one has a camera definition that can be loaded into RVIZ. The command used to obtain this graph is:

```
$ rosrun rqt_graph rqt_graph
```

Unfortunately, it was not clear to me how to implement these namespaces in the RVIZ launch file, but further research is being done on this. For the requirements of the exercise however, the namespaces do in fact work as intended to create seperate categories for the `odom, cmd_vel, scan` topics.

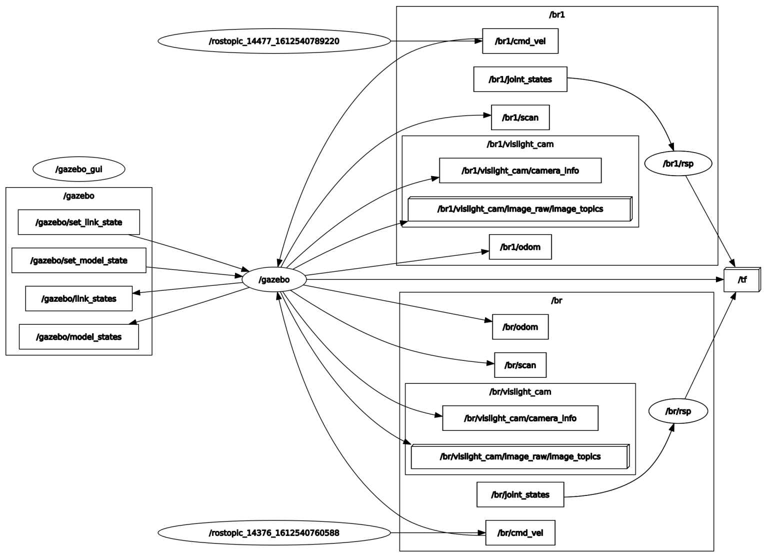

Figure 3: RQT graphs of two robots spawned under different namespaces.

6 Appendix

```xml
<?xml version="1.0"?>
<!--
  This is a very simple robot designed to help explore how ROS and Gazebo
  can interact to simulate a very simple (point) robot.

  The robot consists of a block 10cm x 10cm x 5cm high with the density of copper
    (9.0 g/cc)
        Given the size of the block it has a mass of 4.5 kg

        Diagonal elements of the moments of inertia are
          Ixx = 1/12 * M * (h^2+d^2) = 1/12 * 4.5 * (0.05^2+0.1^2) = 1.0e-3
          Iyy = 1/12 * M * (h^2+w^2) = 1/12 * 4.5 * (0.05^2+0.1^2) = 1.0e-3
          Izz = 1/12 * M * (w^2+d^2) = 1/12 * 4.5 = (0.1^2+0.1^2) = 7.5e-3
  Mounted on top of this is a cylinder r=5cm h=5cm which simulates a LIDAR with one
    sample every degree

  This cylinder is (almost) massless (0.01 kg)

        Diagonal elements of the moments of inertia are
          Ixx = 1/12 * M * h^2 + 1/4 * M * r^2 = 1/12 * 0.01 * 0.05^2 + 1/4 * 0.01 *
     0.05^2 = 2.0833x10^-6 + 6.25x10^-6 = 8.3383x10^-6
          Iyy = 1/12 * M * h^2 + 1/4 * M * r^2 = 1/12 * 0.01 * 0.05^2 + 1/4 * 0.01 *
     0.05^2 = 2.0833x10^-6 + 6.25x10^-6 = 8.3383x10^-6
          Izz = 1/2 * M * r^2 = 1/2 * 0.01 * 0.01 * 0.05^2 = 1.25x 10^-7

  The robot is equipped with one LIDAR with 360 samples (one per degree) from -180
    to +179

  Version 1.0.
  Copyright (c) Michael Jenkin and Gregory Dudek.

  -->

<robot name="br_spawn" xmlns:xacro="http://www.ros.org/wiki/xacro">
  <!-- IMPORT PARAMETERS FILE -->
  <xacro:include filename="$(find cpmr_apb)/urdf/blockrobotdiminert.xacro" />
  <!-- NAMESPACE DEFINITION TF_PREFIX -->
  <xacro:arg name="tf_prefix" default="br" />
  <material name="red">
    <color rgba="1 0 0 1"/>
  </material>
  <material name="green">
    <color rgba="0 1 0 1"/>
  </material>
  <!-- GAZEBO MATERIAL DEFINITIONS -->
  <gazebo reference="$(arg tf_prefix)/base_link">
    <material>Gazebo/Green</material>
  </gazebo>
  <gazebo reference="$(arg tf_prefix)/laser_link">
    <material>Gazebo/Red</material>
  </gazebo>
  <gazebo reference="$(arg tf_prefix)/camera_link">
    <material>Gazebo/Green</material>
  </gazebo>
  <gazebo reference="$(arg tf_prefix)/hat_link_brim">
```

```
    <material> Gazebo/Blue</material>
  </gazebo>
  <gazebo reference="$(arg tf_prefix)/hat_link_top">
    <material> Gazebo/Blue</material>
  </gazebo>
  <!-- BASE_FOOTPRINT LINK DEFINITIONS -->
  <link name="$(arg tf_prefix)/base_footprint">
  </link>
  <!-- BASE LINK DEFINITION -->
  <link name="$(arg tf_prefix)/base_link">
    <visual>
      <geometry>
        <box size="${b_l} ${b_w} ${b_h}" />
      </geometry>
      <material name="green"/>
    </visual>
    <collision>
      <geometry>
        <box size="${b_l} ${b_w} ${b_h}" />
      </geometry>
    </collision>
    <inertial>
      <mass value="${b_m}" />
      <inertia ixx="${b_ixx}" iyy="${b_iyy}" izz="${b_izz}" ixy="0" ixz="0" iyz="0"
    />
    </inertial>
  </link>
  <!-- LASER LINK DEFINITION -->
  <link name="$(arg tf_prefix)/laser_link">
    <visual>
      <geometry>
        <cylinder length="${c_h}" radius="${c_r}" />
      </geometry>
      <material name="red"/>
    </visual>
    <collision>
      <geometry>
        <cylinder length="${c_h}" radius="${c_r}" />
      </geometry>
    </collision>
    <inertial>
      <mass value="${c_m}" />
      <inertia ixx="${c_ixx}" iyy="${c_iyy}" izz="${c_izz}" ixy="0" ixz="0" iyz="0"
    />
    </inertial>
  </link>
  <!-- CAMERA LINK DEFINITION -->
  <link name="$(arg tf_prefix)/camera_link">
    <visual>
      <geometry>
        <cylinder length="${c_h}" radius="${c_r}" />
      </geometry>
      <material name="red"/>
    </visual>
    <collision>
      <geometry>
        <cylinder length="${c_h}" radius="${c_r}" />
      </geometry>
    </collision>
    <inertial>
      <mass value="${c_m}" />
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <inertia ixx="${c_ixx}" iyy="${c_iyy}" izz="${c_izz}" ixy="0" ixz="0" iyz="0"
    />
    </inertial>
  </link>
  <!-- HAT BRIM LINK DEFINITION -->
  <link name="$(arg tf_prefix)/hat_link_brim">
    <visual>
      <geometry>
        <box size="0.15 0.1 0.02" />
      </geometry>
      <material name="green"/>
    </visual>
    <collision>
      <geometry>
        <box size="0.15 0.1 0.02" />
      </geometry>
    </collision>
    <inertial>
      <mass value="1e-5" />
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <inertia ixx="1e-6" ixy="0" ixz="0" iyy="1e-6" iyz="0" izz="1e-6" />
    </inertial>
  </link>
  <!-- HAT TOP LINK DEFINITION -->
  <link name="$(arg tf_prefix)/hat_link_top">
    <visual>
      <geometry>
        <cylinder length="0.02" radius="0.05" />
      </geometry>
      <material name="green" />
    </visual>
    <collision>
      <geometry>
        <cylinder length="0.02" radius="0.05" />
      </geometry>
    </collision>
    <inertial>
      <mass value="1e-5" />
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <inertia ixx="1e-6" ixy="0" ixz="0" iyy="1e-6" iyz="0" izz="1e-6" />
    </inertial>
  </link>
  <!-- JOINT DEFINITIONS -->
  <joint name="base_joint" type="fixed">
    <origin xyz="0.0 0.0 0.025" rpy="0.0 0.0 0.0" />
    <parent link="$(arg tf_prefix)/base_footprint" />
    <child link="$(arg tf_prefix)/base_link" />
    <axis xyz="0 0 1" />
  </joint>
  <!-- LASER JOINT DEFINITION -->
  <joint name="laser_joint" type="fixed">
    <origin xyz="0.0 0.0 0.05" rpy="0.0 0.0 0.0" />
    <parent link="$(arg tf_prefix)/base_link" />
    <child link="$(arg tf_prefix)/laser_link" />
    <axis xyz="0 0 1" />
  </joint>
  <!-- CAMERA JOINT DEFINITION -->
  <joint name="camera_joint" type="fixed">
    <origin xyz="0.0 0.0 0.05" rpy="0.0 0.0 0.0" />
    <parent link="$(arg tf_prefix)/laser_link" />
    <child link="$(arg tf_prefix)/camera_link" />
    <axis xyz="0 0 1" />
  </joint>
  <!-- HAT BRIM JOINT DEFINITION -->
  <joint name="hat_joint_brim" type="fixed">
    <origin xyz="0.025 0.0 0.035" rpy="0.0 0.0 0.0" />
    <parent link="$(arg tf_prefix)/camera_link" />
    <child link="$(arg tf_prefix)/hat_link_brim" />
    <axis xyz="0 0 1" />
  </joint>
  <!-- HAT TOP JOINT DEFINITION -->
  <joint name="hat_joint_top" type="fixed">
    <origin xyz="0.0 0.0 0.015"/>
    <parent link="$(arg tf_prefix)/hat_link_brim" />
    <child link="$(arg tf_prefix)/hat_link_top" />
    <axis xyz="0 0 1" />
  </joint>
  <!-- LASER GAZEBO PLUG-IN -->
  <gazebo reference="$(arg tf_prefix)/laser_link">
    <static>true</static>
    <sensor type="ray" name="head_hokuyo_sensor">
      <pose>0 0 0 0 0 0</pose>
      <visualize>true</visualize>
      <update_rate>10</update_rate>
      <ray>
        <scan>
          <horizontal>
            <samples>360</samples>
            <resolution>1</resolution>
            <min_angle>-3.1415</min_angle>
            <max_angle>3.1240</max_angle>
          </horizontal>
        </scan>
        <range>
          <min>0.20</min>
          <max>10.0</max>
          <resolution>0.01</resolution>
        </range>
      </ray>
      <plugin name="gazebo_ros_head_hokuyo_controller" filename="libgazebo_ros_laser
    .so">
        <topicName>scan</topicName>
        <frameName>laser_link</frameName>
      </plugin>
    </sensor>
  </gazebo>
  <!-- GAZEBO CAMERA PLUGIN -->
  <gazebo reference="$(arg tf_prefix)/camera_link">
    <sensor type="camera" name="vislight_cam">
```

```xml
      <update_rate>30.0</update_rate>
      <camera name="head">
        <horizontal_fov>1.3962634</horizontal_fov>
        <image>
          <width>800</width>
           <height>800</height>
           <format>R8G8B8</format>
        </image>
        <clip>
          <near>0.02</near>
          <far>300</far>
        </clip>
        <noise>
          <type>gaussian</type>
          <mean>0.0</mean>
          <stddev>0.007</stddev>
        </noise>
      </camera>
      <plugin name="cam_controller" filename="libgazebo_ros_camera.so">
        <alwaysOn>true</alwaysOn>
        <updateRate>0.0</updateRate>
        <cameraName>vislight_cam</cameraName>
        <imageTopicName>image_raw</imageTopicName>
        <cameraInfoTopicName>camera_info</cameraInfoTopicName>
        <frameName>camera_link</frameName>
        <hackBaseline>0.07</hackBaseline>
        <distortionK1>0.0</distortionK1>
        <distortionK2>0.0</distortionK2>
        <distortionK3>0.0</distortionK3>
        <distortionT1>0.0</distortionT1>
        <distortionT2>0.0</distortionT2>
      </plugin>
    </sensor>
  </gazebo>
  <!-- OBJECT CONTROLLER PLUGIN -->
  <gazebo>
    <plugin name="object_controller" filename="libgazebo_ros_planar_move.so">
      <commandTopic>cmd_vel</commandTopic>
      <odometryTopic>odom</odometryTopic>
      <odometryFrame>br/odom</odometryFrame>
      <odometryRate>20.0</odometryRate>
      <robotBaseFrame>br/base_footprint</robotBaseFrame>
    </plugin>
  </gazebo>
</robot>
```

Listing 12: Entire robot description file.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro">
  <!-- BOX PROPERTIES -->
  <xacro:property name="b_m" value="4.50" />
  <xacro:property name="b_h" value="0.05" />
  <xacro:property name="b_w" value="0.10" />
  <xacro:property name="b_l" value="0.10" />
  <!-- CYLINDER PROPERTIES -->
  <xacro:property name="c_m" value="0.01" />
  <xacro:property name="c_h" value="0.05" />
  <xacro:property name="c_r" value="0.05" />
  <!-- INERTIA DEFINITIONS -->
  <xacro:property name="b_ixx" value="${1/12 * b_m * (b_h * b_h + b_l * b_l)}" />
  <xacro:property name="b_iyy" value="${1/12 * b_m * (b_h * b_h + b_w * b_w)}" />
  <xacro:property name="b_izz" value="${1/12 * b_m * (b_w * b_w + b_l * b_l)}" />
  <xacro:property name="c_ixx" value="${1/12 * c_m * c_h * c_h + 1/4 * c_m * c_r *
    c_r}" />
  <xacro:property name="c_iyy" value="${1/12 * c_m * c_h * c_h + 1/4 * c_m * c_r *
    c_r}" />
  <xacro:property name="c_izz" value="${1/2 * c_m * c_r * c_r}" />
</robot>
```

Listing 13: Entire robot description xacro macros definition file.

```xml
<launch>
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
  </include>
  <arg name="tf_prefix" default="br" />
  <arg name="tf_prefix1" default="br1" />
  <!-- AUTOMATE XACRO TO URDF PARSING -->
  <group ns="$(arg tf_prefix)">
    <param name="robot_description" command="xacro '$(find cpmr_apb)/urdf/blockrobot
    .xacro' prefix:=$(arg tf_prefix)" />
    <node name="spawn" pkg="gazebo_ros" type="spawn_model" args="-robot_namespace br
     -unpause -param robot_description -urdf -model br_spawn" />
    <node name="rsp" pkg="robot_state_publisher" type="robot_state_publisher">
    </node>
  </group>
  <group ns="$(arg tf_prefix1)">
    <param name="robot_description1" command="xacro '$(find cpmr_apb)/urdf/
    blockrobot.xacro' prefix:=$(arg tf_prefix1)" />
    <node name="spawn1" pkg="gazebo_ros" type="spawn_model" args="-robot_namespace
    br1 -unpause -param robot_description1 -urdf -model br_spawn1" />
    <node name="rsp" pkg="robot_state_publisher" type="robot_state_publisher">
    </node>
  </group>
</launch>
```

Listing 14: Entire Gazebo launch file.

```
<launch>
  <include file="$(find urdf_tutorial)/launch/display.launch">
    <arg name="gui" value="False" />
    <param name="robot_description"   command="xacro ($find cpmr_apb)/urdf/
    blockrobot.xacro" />
  </include>
</launch>
```

Listing 15: Entire RVIZ launch file.