Laboratory class #3: Simulation and control of a quadrotor using Gazebo

1. Gazebo Simulator

In the previous laboratory we used a 2D simulator called Stage. In this laboratory we introduce Gazebo, a 3D dynamic simulator that efficiently simulates heterogeneous robots in complex indoor and outdoor environments. The main features of Gazebo are: multiple physics engines, different robot models and environments, a wide variety of sensors and graphical interfaces. Figure 1 shows the gazebo client window where an environment with an unmanned ground vehicle (UGV) has been simulated.

Figure 1: Gazebo Simulator [1].

In order to run Gazebo, execute the following commands:

```
$ roscore
```

```
$ rosrun gazebo_ros gazebo
```

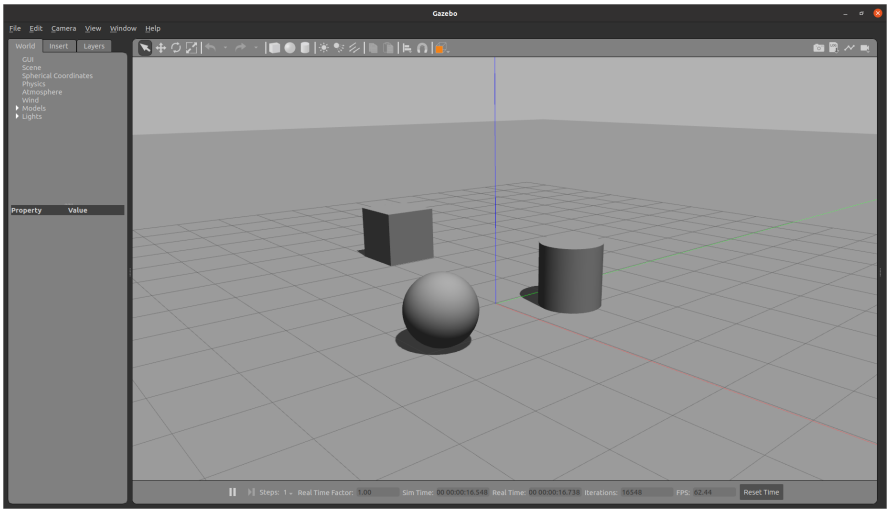

Figure 2: Empty world with a box, a cylinder and a sphere.

[1] Image obtained from https://www.clearpathrobotics.com/assets/guides/kinetic/warthog/WarthogSimulation.html

If you want to add new elements, on the top left you click the 'Insert' tab and insert new models. You can also insert geometric models like boxes, spheres or cylinders using the toolbar. Figure 2 presents an empty world of Gazebo with different geometric elements inserted.

2. URDF for Robot Modeling

Unified Robot Description Format (URDF) is an XML format for representing a robot model. In order to simulate a robot, it is necessary to describe its components in a model. The process of creating a new model covers from building a visual robot model to adding physical and collision properties. Once the URDF model describing the robot assembly is created, a node which simulates the motion and publishes the JointState and transforms must be launch. Finally a robot_state_publisher to publish the entire robot state is used. This XML file (figure 3.left) can then be visualized using ROS's RVIZ package, as shown in figure 3.right .

```
<link name="base_footprint">
  <visual>
   <geometry>
    <cylinder length="0.18" radius="0.25"/>
   </geometry>
   <material name="black"/>
  </visual>
</link>

<link name="right_wheel">
  <visual>
   <geometry>
    <cylinder length="0.04" radius="0.04"/>
   </geometry>
   <origin rpy="0 1.57075 0" xyz="0 0 0"/>
   <material name="black"/>
  </visual>
</link>

<joint name="footprint_to_right_wheel" type="fixed">
  <parent link="base_footprint"/>
  <child link="right_wheel"/>
  <origin xyz="0 -0.15 0.05"/>
</joint>
```

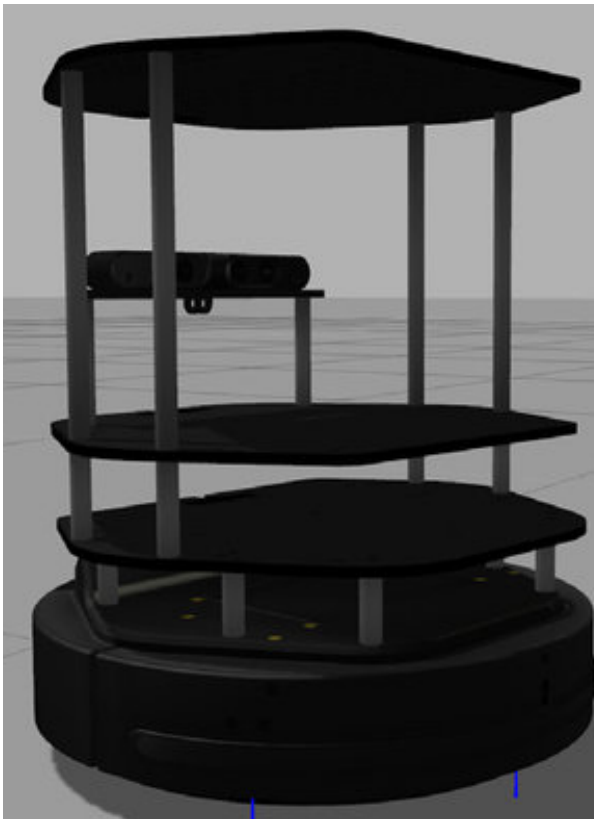

Figure 3: Visual Robot Model with URDF. XML file (left), Visual model (right).

3. Simulate a quadrotor using hector_quadrotor ROS package

In this laboratory we are going to use hector_quadrotor[2] stack which contains packages related to modeling, control and simulation of quadrotor UAV systems. It provides packages related to modeling, control and simulation of quadrotor UAV systems. The main packages are:

- **hector_quadrotor_description**: provides a generic quadrotor URDF model as well as variants with various sensors.
- **hector_quadrotor_gazebo**: contains the necessary launch files for simulation of the quadrotor model in gazebo.
- **hector_quadrotor_controller**: provides libraries and a node for quadrotor control.
- **hector_quadrotor_teleop**: contains a node that allows you to control a quadrotor using a gamepad.

In order to download and install hector_quadrotor in your local workspace, execute the following commands:

```
cd ~/catkin_ws/src
git clone https://github.com/ros-geographic-info/unique_identifier.git
git clone https://github.com/ros-geographic-info/geographic_info.git
git clone https://github.com/RAFALAMAO/hector_quadrotor_noetic.git
cd ~/catkin_ws
catkin build
```

Execute the following command to run a simulation on a empty world:

```
$ roslaunch hector_quadrotor_gazebo quadrotor_empty_world.launch
```

A Gazebo simulator window will open with a simulated quadrotor in an empty environment. The figure 4 shows the result of the execution.

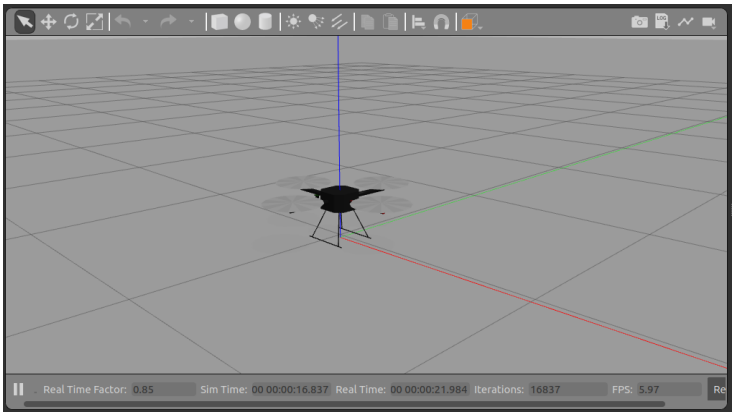

Figure 4: Hector quadrotor simulation on Gazebo.

[2] http://wiki.ros.org/hector_quadrotor
https://github.com/RAFALAMAO/hector-quadrotor-noetic
https://www.researchgate.net/publication/262247993_Comprehensive_Simulation_of_Quadrotor_UAVs_Using_ROS_and_Gazebo

The topics published by the simulator can be shown on a terminal by executing:

```
$ rostopic list
```

These topics allow you to interact with the drone and perform movements. The previous launch file "quadrotor_empty_world.launch" contains the following ros node executions:

```
<launch>
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
       <arg name="paused" value="$(arg paused)"/>
       <arg name="use_sim_time" value="$(arg use_sim_time)"/>
       <arg name="gui" value="$(arg gui)"/>
       <arg name="headless" value="$(arg headless)"/>
       <arg name="debug" value="$(arg debug)"/>
  </include>
  <include file="$(find hector_quadrotor_gazebo)/launch/spawn_quadrotor.launch" />
</launch>
```

empty_world.launch : This file contains the environment information to be loaded into the simulator.

spawn_quadrotor.launch: this file contains the information necessary to launch the robot model and controller. The contents of this file are:

```
<arg name="model" default="$(find hector_quadrotor_description)/urdf/quadrotor.gazebo.xacro"/>
<arg name="tf_prefix" default="$(optenv ROS_NAMESPACE)"/>
<arg name="controller_definition" default="$(find hector_quadrotor_controller)/launch/controller.launch"/>
```

URDF model (quadrotor.gazebo.xacro):

The file hector_quadrotor_description)/urdf/quadrotor.gazebo.xacro contains the URDF model of the quadrotor. Open the file containing the information of the URDF model and understand its structure and the information it contains in order to perform the following exercise.

Exercise 1.

Identify the mass of the quadrotor, where the sonar sensor is described and the topic where ros publishes the measurements of this sensor. Indicate also the type of data published by this topic and the values published by the sonar sensor when the quadrotor is standing on the ground. In order to see the measurement, it will be necessary to display the content of the topic by executing the command "rostopic echo".

mass= 1.477

y publica en el sonar_height y devuelve sensor_msg/ Range

sensor described from line 32 till the end, file quadrotor_base.urdf.xacro

Controller (controller.launch):

The file hector_quadrotor_controller)/launch/controller.launch contains the controller used for the simulated quadrotor. As you can see in the launch file, the parameters of the controllers are in "hector_quadrotor_controller/params/controller.yaml".

In order to understand the control performed, figures 5 and 6 show a high-level comparison showing the block diagram of the controller shown in the theory class and the controller used by the hector_quadrotor simulator.

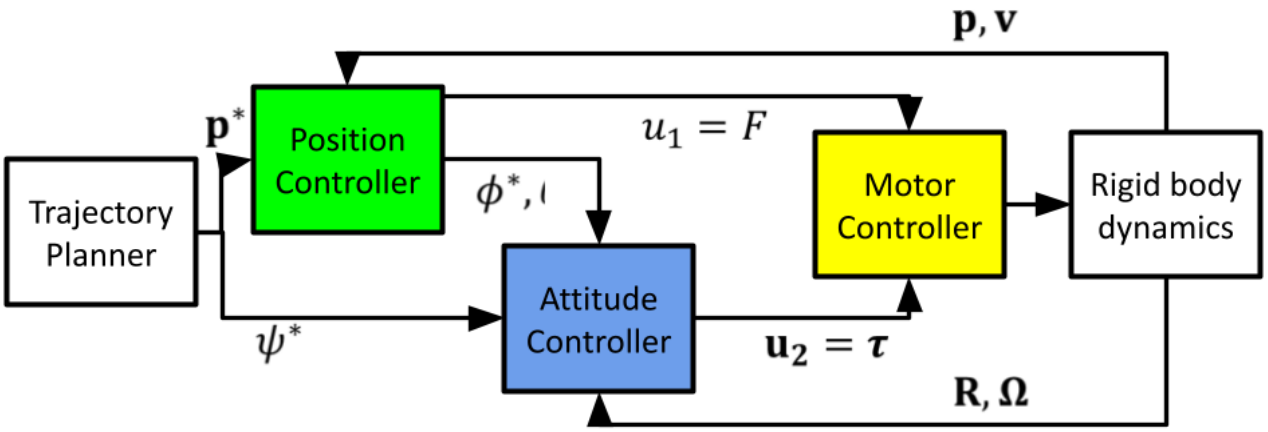

Figure 5: Quadrotor control: Diagram presented in theory class.

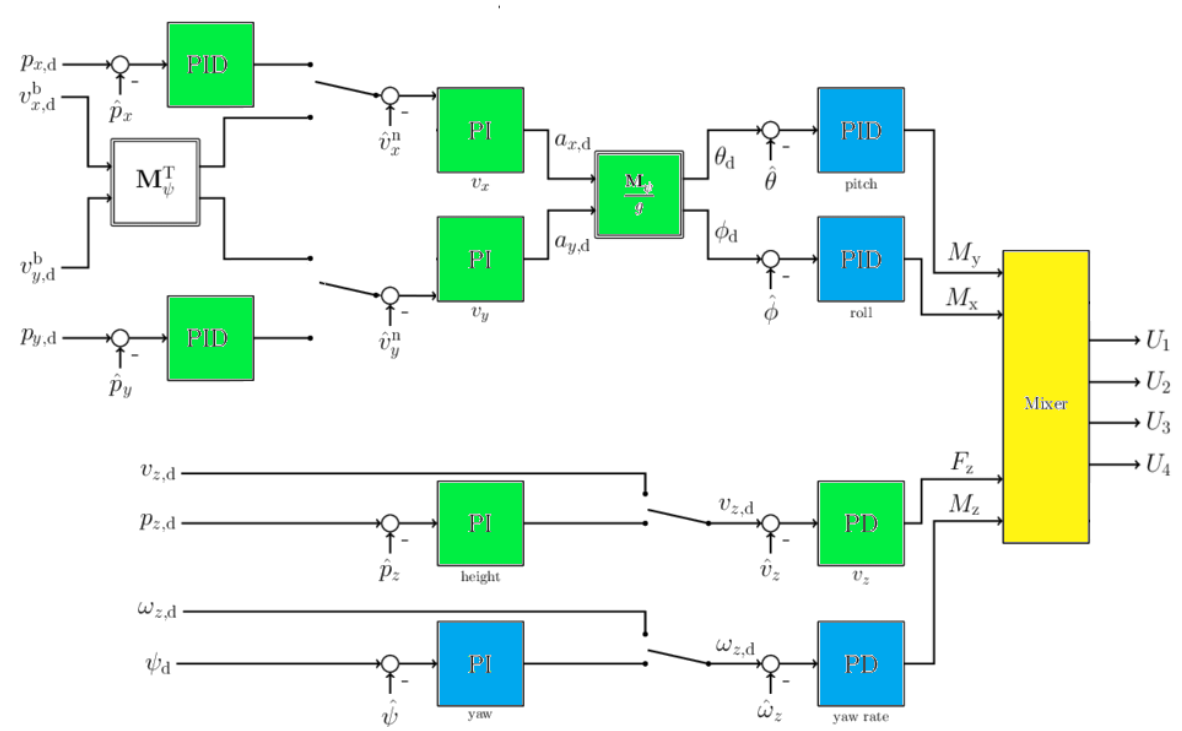

Figure 6: Hector Quadrotor control: Diagram used in laboratory session.

The controller structure of the hector_quadrotor package can be found in the following files:

- hector_quadrotor_controller/src/motor_controller.cpp
- hector_quadrotor_controller/src/twist_controller.cpp
- hector_quadrotor_controller/src/pose_controller.cpp
- hector_quadrotor_controller/src/pid.cpp
- hector_quadrotor_controller/params/controller.yaml

Open the files and look at the code to understand the control module and answer the following exercises.

Exercise 2.

Write the default values of the following PID controllers: “twist controller linear.x”, “twist controller angular.z” and “pose controller yaw”.

controller.yaml, twistlinearxy, twist angularz, poselinearyaw

Exercise 3.

In the twist_controller.cpp file, what is the name of the variable that stores the desired and current linear.y velocities?. Copy the lines of code that implement the linear.z controller.

command is the desired and twist the actual

Exercise 4.

In the motor_controller.cpp file, copy the lines of code that which rotate the quadrotor in z

137-142

4. Twist_controller and Pose_controller

Let's start moving the quadrotor. We can assign velocities using the twist_controller or indicate a target position using the pose_controller.

Execute the following commands to run a simulation on a empty world and launch a graphical user interface for sending velocity commands:

```
$ roslaunch hector_quadrotor_gazebo quadrotor_empty_world.launch
$ rosrun hector_ui ui_hector_quad.py
```

Figure 7: Hector Quadrotor User Interface.

This interface publishes velocities over the topic “cmd_vel”. Try to move the quadrotor using this interface and analyze its performance in the simulator.

Exercise 5.

Another way to publish the speed is by using the "rostopic pub" command. Write the command that publishes a linear velocity at z of 0.15 in the "cmd_vel" topic. Check it out visually on the simulator

By default, the twist_controller is activated, as can be seen in the file “controller.launch” on the package hector_quadrotor_controller:

```
<launch>

<rosparam file="$(find hector_quadrotor_controller)/params/controller.yaml" />

<node     name="controller_spawner"     pkg="controller_manager"     type="spawner"
respawn="false" output="screen" args="  controller/twist --shutdown-timeout 3"/>

</launch>
```

In order to perform a control in position it is necessary to edit this file and add the **controller/pose**. In this case the controller accepts messages via the topic "command/pose" and the quadrotor will move to the goal position.

Captura mía, el controller.launch contenía todo menos el controller pose

Exercise 6.

Modify the controller.launch in order to perform a pose controller. Use "rostopic pub" on the “command/pose” topic to make the quadrotor go up 2 meters.

copiar código del launch

Exercise 7.

Create a new package, pXX_arob_lab3 (where pXX corresponds to the assigned pair number for this course (e.g. p00, p01 ...)), with a node called followTargets3D that reads from a text file the list of 3D targets (X,Y,Z) and publish them one by one to the topic /command/pose. The new goal is published when the robot is “sufficiently close” to the previous target. You can use the code you implemented in Lab2 as a starting point.

copiar código del launch

5. RViz

RViz is a 3D visualization tool for ROS that subscribes to topics and visualizes the message contents using a series of plugins. On the right window is possible to select different views while from the top menu is possible to publish user information as for example target points. RViz permits to save and load setup as RViz configuration and has many plugins to facilitate the visualization (added from the left window). In order to install and execute RViz, execute the following commands:

```
$ sudo apt install ros-noetic-rviz

$ rosrun rviz rviz
```

If you want to add a new visualization, on the bottom left you click the ‘Add’ button and then you get a display window with the possible options. It is important to select an existing frame in the global view, for example world.

Exercise 8.

Use the configuration arob_lab3.rviz to visualize the trajectory followed by the quadrotor for the different lists of targets provided in the lab. Include a screenshot of each one of them. Explain what is the cause for the different behaviors.

Caps RVIZ: 3 al 6, 2 al 5 y 1 al 4, por ponerlas en parejas