Probabilistic Robotics Hw1

Tahsincan Köse

8 November 2018

1 Robot Operating System (ROS)

Robot Operating System is a middle-ware that has numerous functionality such as allowing the description and configuration of robots as both compact and/or complex models (through URDF[1]), providing extremely easy-to-use inter-process communication methods *(topics and services)* and offering hardware-level handling both with the simulated and real robot. It is a completely open source framework being supported by `OpenSourceRoboticsFoundation` (OSRF)[2]. ROS is not a simulator, rather it is a very beneficial tool that can be integrated with any simulator listed below to produce much advanced applications that exploits above features.

2 Gazebo

OSRF also maintains another big-sized product, that is Gazebo simulator [3]. It is a simulation suite that allows the description of a robot and its environment through *Simulation Description Format* (SDF)[4] files and supports several ways (*plugins*) to interface with it. Actually, Plugin [5] concept plays the main role in the connection between ROS and Gazebo and it is generally the way to manage things between these two utilities.

2.1 a. Kinematics of differential drive robots

Gazebo and ROS is by far the most used combination of utilities among the Robotics community. Therefore, innumerable samples and examples can be found on the web regarding to differential drive robots of any kind. For instance, in Figure 1 there is a very simple differential drive robot with one front and two back wheels. The repository [6] contains all the necessary files, folders and

[1] `http://wiki.ros.org/urdf`
[2] `https://www.openrobotics.org/`
[3] `http://gazebosim.org/`
[4] `http://sdformat.org/`
[5] `http://gazebosim.org/tutorials?tut=ros_gzplugins`
[6] `https://github.com/eborghi10/my_ROS_mobile_robot`

resources to create the required functionality to be simulated in Gazebo. In detail, it basically registers a high-level *ROS controller*, **diff_drive_controller**[7] through a configurable YAML[8] file for the parameters specific to the robot in hand (wheel separation, wheel radius, maximum angular velocity and so on). In summary, through the usage of this specific controller, Gazebo can successfully simulate the kinematics of any validly modelled robot.

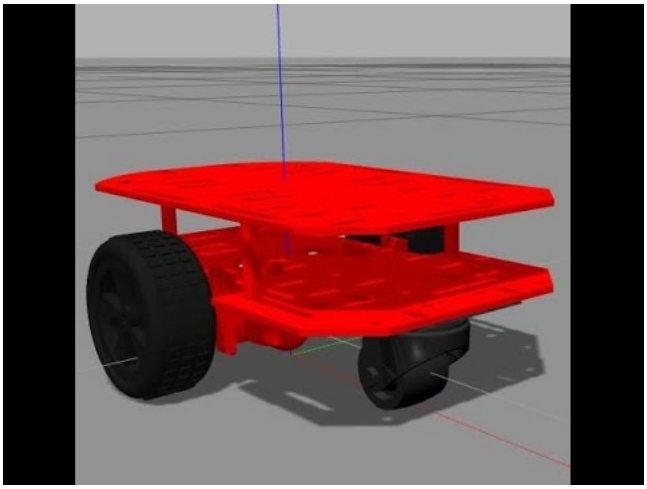


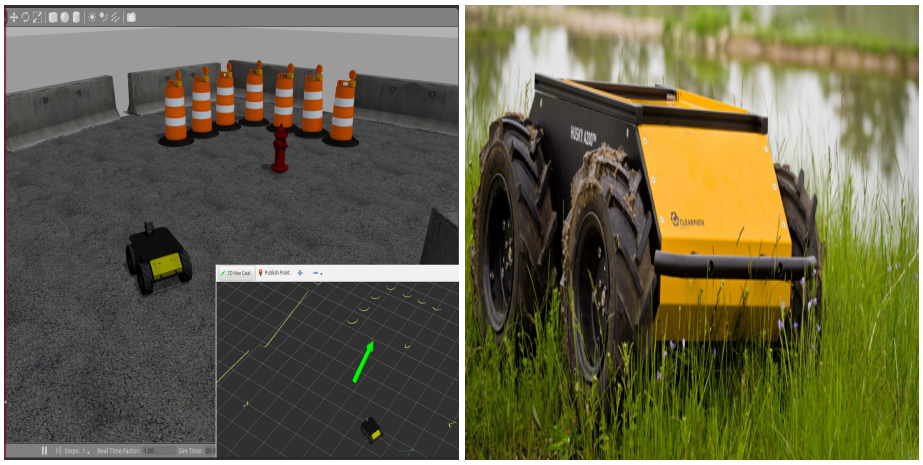
Figure 1: Wheeled robot in Gazebo

Example above illustrates the job of an individual. Since ROS and Gazebo are completely open source tools, any company, organization or institution that use it must conform to the open source licence specifications. For instance, Husky[9] robot from Clearpath robotics is such a framework that has fully open-source code[10] with respect to ROS and Gazebo. Much better is that it uses **diff_drive_controller** of ROS, too, which interfaces with Gazebo.

2.2 b. Linear and angular velocity

diff_drive_controller allows the stimuli of a Twist[11] that adjusts the linear and angular velocity of the robot through ROS topics[12]. Naturally, any type of limit on both the velocity and acceleration magnitudes can be specified through *config* files. Moreover, since this is a ROS controller, it can properly actuate any type of wheeled robot in any realistic simulation environment.

[7] `http://wiki.ros.org/diff_drive_controller`
[8] `http://yaml.org/`
[9] `https://www.clearpathrobotics.com/husky-unmanned-ground-vehicle-robot/`
[10] `https://github.com/husky/husky`
[11] `http://docs.ros.org/melodic/api/geometry_msgs/html/msg/Twist.html`
[12] `http://wiki.ros.org/Topics`



(a) Clearpath's Husky in Gazebo (b) Clearpath's Husky in Real World

Figure 2: Husky Robot

2.3 c. Noise Incorporation

Since these controllers are fully open source, it is easy to intervene with the plugin, e.g. wrapping control input assignment with the noise model, through code at ROS level. For those who seek a lower level of handling (probably much easier) can also manipulate the control inputs at Gazebo side, i.e. directly intervening the rendering of models through Gazebo plugins such that they conform to noise models.

2.4 d. Distance Sensor

Gazebo has a variety of sensors on the web, which have already been accurately modeled based on their experimental parameters. However, it is also possible to model non-existing sensors from scratch. SDFormat offers a <**sensor**> element that allows the specification of a <**noise**> with respect to its *type*, *mean* and *stddev*. In their website, there is a separate tutorial[13] regarding to sensors.

[13] `http://gazebosim.org/tutorials?cat=guided_i&tut=guided_i3`

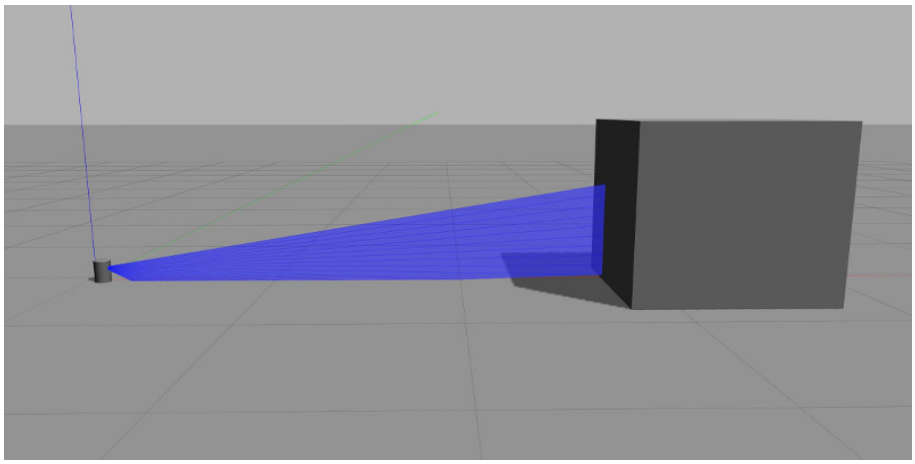
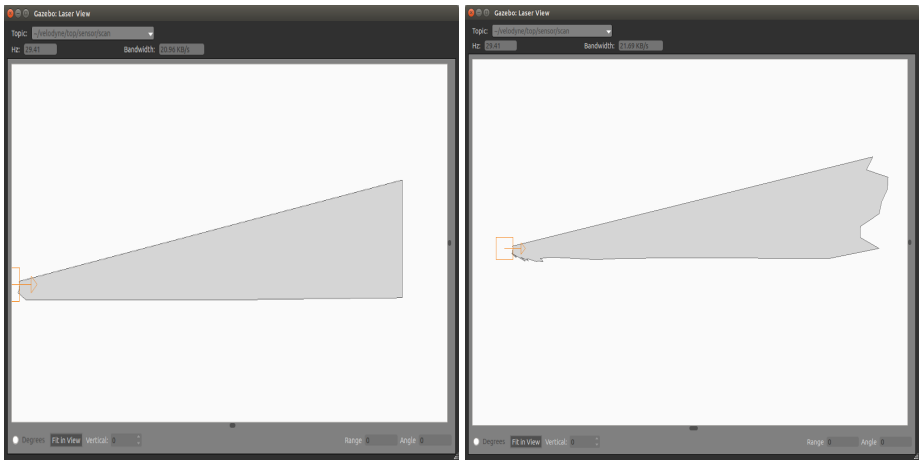


Figure 3: Distance sensor simulated in Gazebo



(a) Side view of ideal sensor output (b) Side view of noisy sensor output

Figure 4: Distance Scanner Sensor

In Figure 3, a working example can be inspected. Distance sensor in this example is a laser scanning sensor that sends beams through a fixed field of view. In Figure 4b, the sensory output clearly reflects the existence of a noise when it is deliberate.

2.5 e. Installation

Installation instructions for Gazebo can be followed through this link. Also for ROS, this link provides necessary instructions.

3 Carla

According to their own definition: "CARLA has been developed from the ground up to support development, training, and validation of autonomous urban driving systems. In addition to open-source code and protocols, CARLA provides open digital assets (urban layouts, buildings, vehicles) that were created for this purpose and can be used freely. The simulation platform supports flexible specification of sensor suites and environmental conditions."

Carla is an open source simulator specifically developed for differential wheel automobiles, funded by Intel. It uses Unreal Engine, hence the graphics are much more realistic than Gazebo. Indeed, the state-of-the-art simulators that are being developed currently (e.g. Carla and Microsoft's AirSim[14] employs Unreal Engine as their physics motor. Therefore the graphics are extremely realistic, those who diminish the gap between simulation and real world, which is a valid concern in case of Robotics applications. Installation instructions can be accessed through this link. Nonetheless, building Carla might be more cumbersome since it requires registration to Unreal Engine[15]. Finally, it has a bridge to ROS[16].

3.1 a. Kinematics of differential drive robots

As previously mentioned, Carla Simulator focuses on the development of autonomous automobiles in an urban environment. Hence, their kinematic is simulated as realistic as possible conforming to the given control inputs.

3.2 b. Linear and angular velocity

Unfortunately, there isn't a ready-made API for linear and angular velocity to be directly commanded. Nonetheless, current implementation provides 5 different type of control input: Steer, throttle, brake, hand-brake and reverse. One workaround might be again using ROS-Carla bridge and provide a plain /cmd_vel topic. When a command is published to this topic, Carla side should process this command and translate to its own type of control inputs.

3.3 c. Noise Incorporation

Carla is not as skillful as Gazebo regarding to noise introduction, too. In one of the sample files[17], it is introduced with a very (perhaps, too much) simple

[14] https://github.com/Microsoft/AirSim

[15] https://www.unrealengine.com/en-US/what-is-unreal-engine-4

[16] https://github.com/carla-simulator/carla/tree/master/carla_ros_bridge

[17] https://github.com/carla-simulator/carla/blob/master/Deprecated/PythonClient/client_example.py

Figure 5: Carla Simulator

method. What they have done is simply sampling a zero-mean random variable and adding it to the ideal control input. Therefore, a more realistic model requires further effort.

3.4 d. Distance Sensor

Currently, only a LiDAR based distance sensor[18] is available. Since, this is an open source simulator, introducing new sensors is not an impossible task, however not trivial either. In terms of the context of this course, it might be too involving to develop such facilities. Having assumed a LiDAR would suffice, the problem of noise introduction reveals itself. Currently, there isn't any noise logic in Carla source codes. However, the ROS bridge can play a crucial role in here to implement that feature.

4 Virtual Robotics Experimental Platform (V-REP)

V-REP is a open-source simulation suite developed by the Coppelia Robotics[19] company. Its graphics resemble to those of cartoons and its API is much more user-friendly with respect to Gazebo. For instance, adding and configuring sensors can be done in run-time through its GUI. It can also interface with ROS[20].

[18] https://carla.readthedocs.io/en/latest/cameras_and_sensors/#ray-cast-based-lidar

[19] http://www.coppeliarobotics.com/

[20] http://wiki.ros.org/vrep_ros_bridge

4.1 a. Kinematics of differential drive robots

Since, there exists a bridge between ROS and V-REP through **vrep_ros_bridge** package, it is possible to implement a probabilistic Kinematics model at ROS level. If it is desired to have a *true*[21] simulation, then it must be implemented at V-REP level. Since V-REP is an open source simulator, model rendering can be altered such that effectively simulates a noisy real world case.

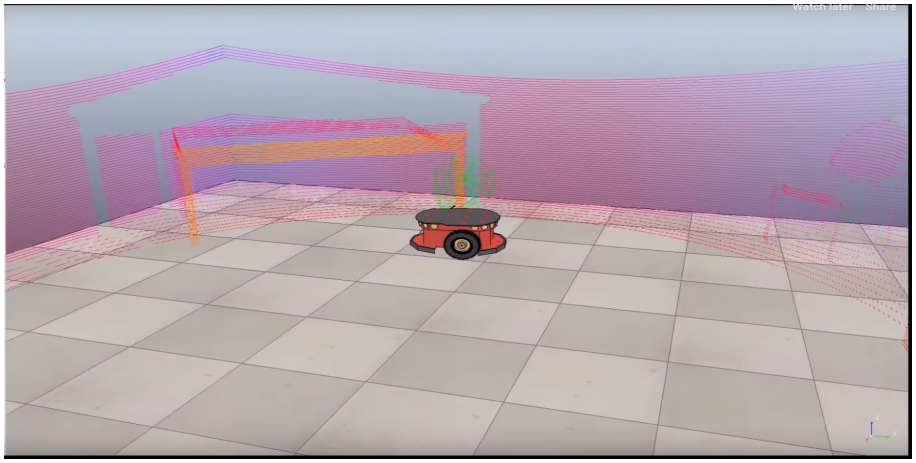


Figure 6: V-REP Simulator

4.2 b. Linear and angular velocity

According to its documentation, it is possible to *get()* and *set()* the velocity of each joint. However, each joint can have only one velocity value. Therefore, I understand that it lacks the ability to simulate a Twist through its own API. Moreover, an assignment[22] given in Georgia Tech also focuses on this problem, too. What the students are expected is to simulate this twist behavior through wheel commands. Twist command is simply given at ROS level *(client)* and V-REP*(server)* level translates it to wheel commands. I think, this is a valid yet cumbersome approach.

4.3 c. Noise Incorporation

In terms of noise, the API unfortunately does not provide any function. Therefore, this ability must be implemented at ROS level.

[21] In the sense, command is given ideally, yet the simulator introduces its own source of noise

[22] `http://dream.georgiatech-metz.fr/sites/default/files/CS7630-Homework-05.pdf`

4.4 d. Distance Sensor

V-REP provides ready-made proximity sensors[23] for this purpose. For those who seek advanced behavior can also employ a number of sensors listed in this video. The main concept of V-REP is to provide a very concise API for basic, out-of-box functionality and leave the rest to the developers through plugins. For example, a question[24] asked on the simulation of noisy sensor output is answered in a way that proves this notion.

4.5 e. Installation

Since this is a product belonging to a commercial company, different versions exist. For educational purposes, they provide a non-limited licence so that all features can be used. In order to install the simulator in Ubuntu, download the corresponding zip file from the link. Then the debian package can be built and installed as any other package.

5 Conclusion

In conclusion, I desire to definitely continue with Gazebo. Since the community which actively uses it is much bigger than the other alternatives, all the requirements that we respect in the context of this course have heavily undergone the cycle of improve/test/release with numerous times, hence best implemented in Gazebo without a debate. Moreover, there are a variety of ready-made examples for noisy kinematics and noisy sensors in the web developed either by individuals or interest groups which can be used as ground truths without any inconveniency. Also, I'm using Gazebo for nearly 2 years, so I will not waste any time on learning the simulator itself. In other simulators there would be a learning curve that might not be convenient for the limited time span that this course offers.

[23] http://www.coppeliarobotics.com/helpFiles/en/proximitySensors.htm
[24] http://www.forum.coppeliarobotics.com/viewtopic.php?t=6587