Gazebo simulation with cwruBot

Wyatt Newman
January, 2015

Gazebo and Rviz are useful tools for developing robot code. A Gazebo simulation uses a physics engine to compute dynamics, including impacts from contact dynamics, friction and slipping, motor controllers, etc. To consider contact dynamics, Gazebo must be aware of modeled objects in its virtual world. One of these objects is our simulated robot, specified in the "Universal Robot Description Format" (URDF). Additional objects can be inserted into the virtual world interactively via Gazebo's menus.

Another valuable capability of Gazebo is simulation of sensors, which includes LIDAR, Kinect and color cameras. Virtual sensor data computed from perception of a virtual world is published on ROS topics, identical in style to physical measurements from the same sensor types. Thus, one can develop perceptual processing algorithms in simulation, and then apply them to the real robot system.

A simple model of Case's wheelchair-based robots is contained in the package cwru_urdf, within the file cwruBot_urdf.xacro. This URDF describes 17 physical components (including visualization, collision boundaries and inertial properties), most of which are statically attached to a frame called "base_link". A handful of these components are movable, including the 2 drive wheels, the 2 caster brackets and the 2 caster wheels. Additionally, the model contains a LIDAR unit mounted on the front, like those of Jinx, Harlie and Roberto. Gazebo simulates LIDAR sensing as virtual LIDAR rays reflect from virtual objects in the virtual world.

Both "Gazebo" and "Rviz" offer visualizations. However, there is an important underlying distinction. Gazebo is meant to take the place of reality—e.g. one of our mobile robots moving through the lab or the hallway, acquiring sensory data as it goes, and possibly colliding with physical objects. It is the intent that your code that is developed using Gazebo should be immediately applicable to the physical robot and the physical sensors.

When running a physical robot, "Gazebo" will not be present. Instead, the real robot will respond to commands, and real sensors will publish to odom and laser/scan topics. However, it is still useful to run Rviz, whether operating in a simulated or physical world.

The Rviz simulator displays an illustration of the model robot plus overlays of many other options. One of the most useful applications of rviz is to visualize sensor data. Rviz does not distinguish among sources of data—whether generated as virtual sensors with synthetic data, or real sensors with real data. Rviz is only aware of data arriving via ROS topics. When this data includes reference coordinate frames, then Rviz can overlay visualization of the data on visualization of the robot model.

An example is shown below in Fig 1. In this case, Gazebo and Rviz are both running, using the cwruBot robot model, including a simulated LIDAR sensor. The operator can choose the viewpoint independently for Gazebo and Rviz. In the example of Fig 1, these were chosen to be similar viewpoints. The Gazebo view, on the left, includes the cwruBot and several models imported from the Gazebo drop-down list: a Jersey barrier, a construction barrel, a construction cone and a dumpster.

The rviz view, on the right, does not show these objects. The rviz view is only aware of information transmitted via ROS topics. Since the simulated LIDAR data is published, Rviz is able to overlay this

data in a consistent coordinate frame, shown as red spheres. The LIDAR data alone may seem unintuitive, but viewed side-by-side with Gazebo, we can see that some of the LIDAR points are due to the dumpster, some due to the construction barrel, and some due to the construction cone. However, when running the real robot, a remote operator would *only* have the Rviz view, since only sensor data (not an omniscient presence) would be available. Perception algorithms must try to make sense of the available data. Fortunately, the available LIDAR data is typically adequate for avoiding collisions.

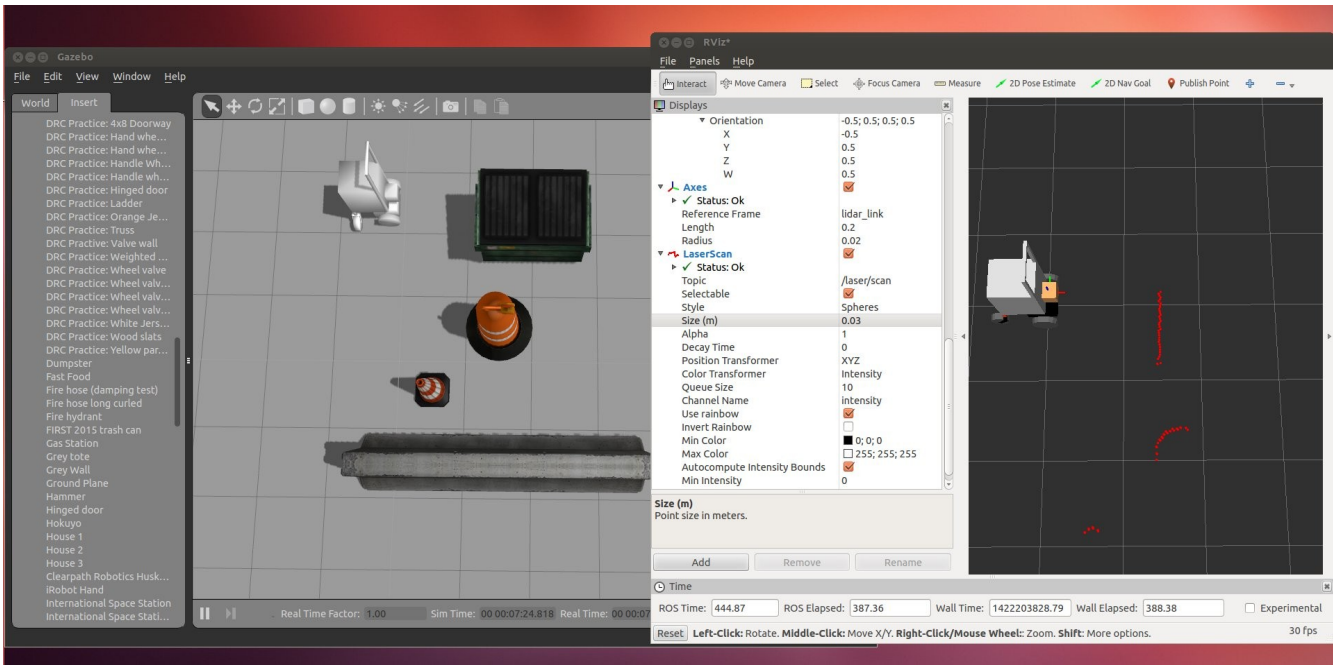

Illustration 1: Gazebo and Rviz views of cwruBot in a virtual world

The topics of interest (at present) are: cmd_vel, odom and laser/scan. The cwruBot accepts speed/spin commands via topic cmd_vel. It outputs odometry on the /odom topic and simulated LIDAR on the /laser/scan topic.

With the system running, one can interact manually with the robot using command-line publication of Twist commands. E.g.:
rostopic pub -r 10 /cmd_vel geotry_msgs/Twist '{linear: {x: 0.2, y: 0.0, z: 0.0}, angular: {x: 0.0,y: 0.0,z: 0.1}}'

Will cause the robot to move forward at 0.2m/sec and rotate “up” (counterclockwise, as viewed per Fig 1) at 0.1 rad/sec, making large, slow circles. The robot will stall if it runs into a heavy object--but it is able to push around lighter objects (e.g. construction cones). However, dynamic properties should not be trusted. At present, inertias, max wheel torque and max speeds should not be trusted. Further, friction between the wheels and the floor, weight of models, friction between models and floor, contact compliance, etc. are only very approximate. Nonetheless, this should be adequate for simulating smooth robot motion commands in cluttered environments.

Note that it is not a good idea to send manual Twist commands to the robot. Rather, the robot commands should be computed algorithmically according to some sound logic, and the commands should be sent by a robot commander node as ROS publications. Be careful, as well, to assure that the robot commander does not terminate with a non-zero Twist command, else the robot will try to keep moving with the last received command.

Running the simulator:
In a terminal, run

```
roscore
```

(for the physical robot, this would run on the robot's computer).

In another terminal run:

```
rosrun gazebo_ros gazebo
```

This should produce a flat, gray ground-plane. If this does not come up, kill this process and restart it.

(For the physical robot, the simulator is not necessary, but we do launch a few “bridge” nodes that convert back and forth between ROS messages and hardware I/O).

In another terminal, run:

```
roslaunch cwru_urdf cwruBot_urdf.launch
```

(For the physical robot, as noted above, we have an alternative launchfile that starts up necessary ROS bridge nodes).

Start Rviz:

```
rosrun rviz rviz
```

(This is useful whether using Gazebo or the physical robot. To connect to the physical robot from a desktop workstation, though, there are a few network connection commands required, including informing ROS that “roscore” is running on the remote robot).

In the Rviz menus, choose "base_link" as the “fixed frame” , and add a display of LaserScan (on topic /laser/scan) to visualize the lidar. (Same for simulated or real robot).

Run `rostopic echo odom` to see the robot achieve the commanded speeds, and to see the pose evolve as the robot moves. (Same for simulated or real robot).

The Rviz display will work equivalently whether the laser/scan or odom data is real or simulated.

Anticipated Extensions:
We expect to extend the cwruBot to include point-cloud data from a Kinect device, as well as color video (which may also derive from the Kinect device). Processing image and point-cloud data should be testable in simulation and applicable in practice.