Running CAT Vehicle Simulation with ROS

ECE 492 CAT Vehicle Research Experience for Undergraduates

June 3, 2019

Rahul Kumar Bhadani rahulbhadani@email.arizona.edu

Contents

1 Introduction

A number of autonomous vehicle application requires a well-established framework capable of producing multi-vehicle simulation. The application of multi-vehicle simulation ranges from testing vehicle-following algorithms, traffic simulation, ego vehicle detection, perception and recognition of other vehicles in the traffic, connected vehicle systems, etc. In this tutorial, we will discuss how to run autonomous vehicle simulation using the CAT Vehicle testbed implemented in ROS/Gazebo.

2 Pre-requisite

1. Ubuntu 18.04
2. ROS Melodic
3. Gazebo 9.0
4. Packages: ros-melodic-controller-manager, ros-melodic-ros-control ros-melodic-ros-controllers, ros-melodic-gazebo-ros-control, ros-melodic-velodyne, ros-melodic-joystick-drivers, ros-melodic-novatel-span-driver

However, a number of packages are required to be built at source since they are not yet able through apt on melodic. A few of them that I have identified can be downloaded from our lab's GitHub page which we detail in later sections.

3 Downloading the CAT Vehicle Testbed

Before we proceed for the simulation, download the `catvehicle` package from GitHub repository along with another package `obstaclestopper` and few other packages to drive the vehicle (kind of) safe. If you have already done that, you are not required to re-do this step.

Steps for setting up your testbed is reproduced below from the README of the GitHub page https://github.com/jmscslgroup/catvehicle.

1. Follow the steps mentioned in the ROS wiki page http://wiki.ros.org/melodic/Installation/Ubuntu%C2%A0 on how to install ROS Melodic.
2. In addition to that we are required to install some additional ros packages

```
sudo apt-get install ros-melodic-velodyne ros-melodic-novatel-span-driver
```

3.1 Create your workspace

```
cd ~
mkdir -p catvehicle_ws/src
cd catvehicle_ws/src
catkin_init_workspace
```

3.2 Download packages

```
cd ~/catvehicle_ws/src
git clone https://github.com/jmscslgroup/catvehicle
git clone https://github.com/jmscslgroup/obstaclestopper
git clone https://github.com/jmscslgroup/control_toolbox
git clone https://github.com/jmscslgroup/sicktoolbox
git clone https://github.com/jmscslgroup/sicktoolbox_wrapper
git clone https://github.com/jmscslgroup/stepvel
cd ../
catkin_make
```

You will also require to source your setup script file inside the devel folder generated after `catkin_make` command. Follow the steps below to add the source command to your `.bashrc` file

```
echo "source ~/catvehicle_ws/devel/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

This will make your bash shell aware of presence of `catvehicle` package and where to find them.

4 Running the simulation

In this section, we will follow step-by-step instructions that will enable you to create a multi-vehicle simulation.

4.1 Initiating Gazebo world

First, we will create a virtual world in Gazebo where we will later spawn a vehicle model. Open a new terminal and type:

```
roslaunch catvehicle catvehicle_empty.launch
```

This will start gzserver. To view the world, start gzclient by typing following in a new terminal tab or window (I usually prefer a new tab. You can open a new tab in the same terminal window by typing `Ctrl+Shift+T`).

```
gzclient
```

You will find that in Gazebo window, there is nothing except an empty plane. In the next step, we will spawn a vehicle model.

4.2 Spawning vehicle models.

In order to spawn a vehicle model, we will launch `catvehicle_spawn.launch` file. By default the name of the vehicle and its associated namespace *catvehicle* and it spawned at origin. You can change the name/namespace and position of the vehicle by supplying values to option arguments in the run time. Open another terminal tab or window and type following command:

```
roslaunch catvehicle catvehicle_spawn.launch robot:=catvehicle X:=0 Y:=20 \
yaw:=1.57079632679
```

This will spawn a vehicle model with a name *catvehicle* at (x,y) co-ordinates of (0, 20) with respect to world frame with of $\pi/2$ radian.

To spawn another vehicle model type the same command with different name (supplied through `robot1` argument at a different position.

4.3 Creating a launch file to execute stepvel node from stepvel package

In a new terminal tab or window, open stepvel.launch in the launch folder of the `catvehicle` package:

```
roscd catvehicle/launch
gedit stepvel.launch
```

Then analyze stepvel.launch you just opened. It has content similar to one reproduced below:

```
<launch>

 <param name="enable_statistics" value="true" />
      <arg name="leader_robot" default="catvehicle"/>
      <arg name="vel" default="2.0"/>
      <arg name="strAng" default="0"/>

    <group ns="$(arg leader_robot)">
        <param name="constVel" value="$(arg vel)"/>
        <param name="strAngle" value="$(arg strAng)"/>
        <node pkg="stepvel" type="stepvel_node"
        name="stepvel_$(arg leader_robot)" output="screen" required="true">
        </node>
    </group>
</launch>
```

This file takes three arguments: a robot for the name of the car, vel for velocity and strAng for steering angle. You can send velocity to a car model by typing following command in a new terminal window or tab:

```
roslaunch catvehicle stepvel.launch robot:=batvehicle
```

You will see in Gazebo that vehicles have started moving. Because of the inherent dynamics of the car and disturbances due to frictions, unknown disturbances, you will find out that vehicle doesn't move with exactly 2m/s (or whatever velocity you choose). You check this by typing `rostopic echo /batvehicle/vel` in a new terminal or a tab.

Additionally, you can set parameter `constVel` and `strAngle` from `stepvel` node to change the input being sent on the topic `cmd_vel` as follows:

```
rosparam set /catvehicle/strAngle 0.15
rosparam set /catvehicle/constVel 4.15
rosparam set /batvehicle/strAngle 0.05
rosparam set /batvehicle/constVel 7.44
```