Instruction for Preliminary Project 2

RC car keyboard control

1. Introduction

For the term project, we are going to use a simulator in ROS which is called "GAZEBO". This simulator runs on ROS framework and communicates with ROS core. So we explained how to send a control to GAZEBO simulator and receive sensing data from it. We used a RC car and laser sensor in GAZEBO. The RC car can be controlled by sending linear velocity and steering angle. The laser sensor is given in preliminary project. The assignment's goal is implementing a control publisher on ROS framework and manually make a RC car follow a given path.

2. GAZEBO simulator

You already have a GAZEBO simulator. When you installed ROS kinetic, GAZEBO was installed together. So you can easily start the program. This section explains the way to use a gazebo and communicate with a gazebo world.

2.1. Using roslaunch to Open World Models

The roslaunch tool is the standard method for starting ROS nodes and bringing up robots in ROS. To start an empty Gazebo world similar to the rosrun command in the previous tutorial, simply run this.

```
roslaunch gazebo_ros empty_world.launch
```

You can see the gazebo world like figure 1.

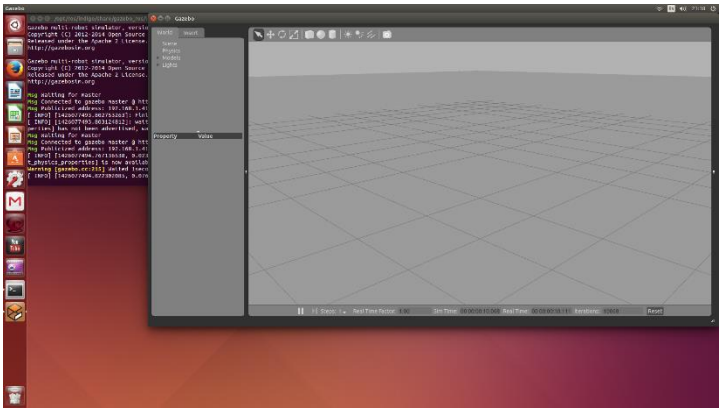

Figure 1. GAZEBO Empty world

In your project file, there is a launch file to open a world model which contains the RC car.

3. Assignment

This assignment is making nodes to control RC car by a keyboard. This will be helpful for your future project assignment. You have to download 'preproject2.zip', 'racecar-simulator.zip', 'racecar.zip' files and extract them at ~/catkin_ws/src/. These files contain a RC car model, and the model you will use is a fixed version of MIT RC car model (https://mit-racecar.github.io/).

3.1. Install Packages

You have to install some packages to simulate RC car on the GAZEBO world. You can download and install them by following commands.

```
sudo apt-get install ros-kinetic-effort-controllers
sudo apt-get install ros-kinetic-ackermann-msgs
sudo apt-get install ros-kinetic-ackermann-controller
sudo apt-get install ros-kinetic-controller-manager
sudo apt-get install ros-kinetic-joy
sudo apt-get install ros-kinetic-ros-control
sudo apt-get install ros-kinetic-ros-controllers
sudo apt-get install ros-kinetic-gazebo-ros ros-kinetic-gazebo-ros-pkgs
sudo apt-get install ros-kinetic-gazebo-ros-control
```

Some of these commands are needed to control the RC car model by a keyboard. After installation, compile your project.

```
cd ~/catkin_ws/
catkin_make
```

3.2. GAZEBO world

You can launch the GAZEBO world using a following command.

```
roslaunch preproject2 preproject2.launch
```

You will see a RC car and its sensing range. Check which node is running by using rosnode command. After that, check which topic is made by the node using rostopic command.

```
rosnode list
rostopic list
```

3.3. Velocity Topic

In the RC car model, a subscriber receives a command for the velocity and steering angle of the RC car. To control your RC car in the simulator, you can make a publisher to send a message to the RC car topic. The type of a message is <ackermann_msgs::AckermannDriveStamped>, and the name of the topic is "/vesc/high_level/ackermann_cmd_mux/input/nav_0." If you made a publisher of the topic, you can control the RC car through this topic and message.

4. Skeleton Code

- ✓ keyboard_controller.cpp

4.1. keyboard_controller.cpp

The node publishes RC car control values to the "/vesc/high_level/ackermann_cmd_mux/input/nav_0" topic. You need to write a code which publishes an adequate control of the RC car. Control varies with the keyboard input so that you can control the RC car as you want in the GAZEBO when executing the node. Following table is pair of a keyboard input and a control output.

w	Move straight forward	**a**	Turn left
s	Move backward	**d**	Turn right
q/e	Increase/decrease the speed of the car by 10%	**Etc.**	Stop the car

We defined ASCII code of keyboard inputs at the top of skeleton code like figure 4. You can see

the variable "keyboard_input" in your skeleton code. This variable has a keyboard input value. You can check which key is pressed by watching "keyboard_input" with ASCII code.

```
#define KEYCODE_W 0x77
#define KEYCODE_A 0x61
#define KEYCODE_S 0x73
#define KEYCODE_D 0x64
#define KEYCODE_Q 0x71
#define KEYCODE_E 0x65
#define KEYCODE_R 0x72
```

Figure 2. ASCII code

In the skeleton code, there exists a member variable "drive_msg_stamped" whose type is <ackermann_msgs::AckermannDriveStamped>. You need to send this argument to "/vesc/high_level/ackermann_cmd_mux/input/nav_0". The publisher is also declared as member variable, "pub_". You use this variable to publish "drive_msg_stamped". The <ackermann_msgs::AckermannDriveStamped> type has member variables: "drive_msg_stamped.drive.speed" which means the speed of the RC car and "drive_msg_stamped.drive.steering_angle" which means the steering angle of the RC car. You can set the member variables so that you can send a message as you want. For example:

```
drive_msg_stamped.drive.speed = [speed];
```

```
drive_msg_stamped.drive.steering_angle = [steering_angle];
```

If you set the steering angle positive, the RC car will turn left.

After you set the member variables, you should send the message through the publisher. For example:

```
pub_.publish(drive_msg_stamped);
```

5. Submission Format

Compress your project folder which includes all your project files. Then, upload it on eTL. The name of the compressed file should be "**IS_[Student ID]_Pre_Project2.tar.gz**". For example, if your ID number is 123456789, the file name should be 'IS_2016-123456789_Pre_Project2.tar.gz'.

Due to: 2017.10.16 23:59 KST