Tutorial 4: The Robot Operating System: Keyboard Control

The PennArc Team

Introduction

In this tutorial we are going to cover the details regarding the low-level motor control/interface setup. The following sections will detail the Lab exercise and the last section contains the transcript of the accompanying video tutorial.

The Teensy, the Jetson and the ESC..

The RF reciever module on the traxxas sends PWM signals to the servo(controls the steer) and the ESC(controls the throttle). This PWM signal is a 100KHz wave with varying duty cycles. A duty cycle of 10% on the throttle and the steering makes the car reverse at full speed while orienting the wheels fully to one direction. The opposite actions are observed when the duty cycle was set to 20%. A signal of 15% duty cycle on both channels results in 0 throttle and 0 steer(wheels oriented straight ahead).
To bypass the RF reciever, we feed the rquired signals directly to the ESC and the servo. We generate these signals using the Teensy Microcontroller. The Teensy gets the commands for the desired duty cycle values from the Jetson via USB interface. The Teensy runs a ROS node that listens for this particular command.
There are two nodes that are relevant to the keyboard control on the Jetson. The **teleop_traxxas** node and the **talkerim**. The teleop traxxas node takes in user input in the (-5 to 5) range for the steer and the throttle channel. The **talkerim** node listens to the published messages from **teleop_traxxas** and converts this to particular PWM values. The teensy node listens to the published by **talkerim** and generates the 100KHz square waves with the specified duty cycle. In this particular implementation, a 16bit register on the teensy is chosen for the generating the square wave. As a result, a value of 9830 corresponds to (9830/65535)% duty cycle. This is an implementation choice and can be altered.

The Task Description & the Code Layout

Your task in this exercise is to

- Implement the talkerim node
- Implement the teleop_traxxas node

If you want to change the way the mapping from stdin(keyboard) to duty cycle is performed, feel free to alter the teensy code. A couple of things that might help you if you alter the teensy ..code is mentioned in [1]. The given code base is written in C++ but if you are more comfortable with Python there is a similar code base provided for python. Note that the way python is compiled adn run on ROS is different from the way C++ is built. So familiarize yourself with the python related ROS build instructions before diving into the code.

You will be provided with the skeleton package '**beginner_tutorials**'. You can choose to either make a new package or you could just take the new files from this package and include them in your existing package. You will have to update your **CMakeLists.txt** accordingly. Make sure you can perform a catkin_make once you include the nodes and custom messages in the CMakeLists.txt.

The code uses custom messages - a pair of float/int values to define the desired/commanded throttle and accelaration. You might want to go through the ROS tutorials regarding custom messages[2] for reference.

The **teleop_traxxas** node: In teleop_traxxas node, user input from keyboard is implemented via a non blocking call to **getchar()**. This was done by using the termios header defintions. Your job is to implement the parsing once you have the character entered by the user. Once you parse this character, you must then accordingly update the desired throttle/steering.

The **talkerim** node: You will have to code almost the entirety of this code - write callback functions[5] that listen to the topic published by the teleop_traxxas node. Write methods to convert these values to the required PWM duty cycle values. Send these in the custom messge("driveCmd_Timed") via the "chatter" topic. The teensy listens on this topic and generates the required waveforms.

The code that is to be completed in the **teleop_traxxas** node is the part where you have to parse the input keyboard key(example arrow keys) and change the desired throttle or steering accordingly. There are many different ways to implement

this functionality and its interesting to see the way different people find different approaches more/less natural than an other.

Remember to include the nodes in the CMakeLists of your package(see Ref. [4], [3]). If you are using your own conventions or protocols your implementation will naturally be different - For example, you could choose to not use the **talkerim** node entirely and process the stdin in the **teleop_traxxas** node itself. These are design choices and the user is free to implement whatever he/she thinks is best.

The Transcript: Video Tutorial 5

In this tutorial we are going to go over your keyboard control lab exercise.

Oh, and just as an aside, I want to mention this terminal multiplexer I use. It might be beneficial for those of you who haven't heard of it. On the screen right now, I have 4 terminal panes through which I have 'ssh'd' into the car(ubuntu@teg...). It gets rather cumbersome to operate a bunch of separate terminal windows. To avoid the clutter of so many different windows, I prefer using either tmux or terminator. The one you are seeing in this video is the terminator application. The terminals become a lot easier to manage and view.

Anyway, back to the tutorial - Im going to walk you through your Lab Exercise wherein you are asked to control the car with your keyboard. This video will make full sense to you only if you have read the accompanying description of the lab exercise. Ill talk you through the different nodes that you need to run and also the importance of each of them. Lets start by going into our workspace and sourcing the environment.
Once we got our environment setup, we begin by launching

```
$ roscore
```

We now run the **talkerim** node using

```
$ rosrun beginner_tutorials talkerim
```

This node is waiting for desired throttle and steering from the user via the **teleop_traxxas** node.

We next start the rosserial node - i.e, the Teensy. use the rosserial command to launch the node.

```
$ rosrun rosserial_python serial_node.py /dev/ttyACM0
```

We run the **teleop_traxxas** node using

```
$ rosrun beginner_tutorials teleop_traxxas
```

The 0 throttle and the 0 steer refer to the 15% duty cycle that was mentioned in the write up - where the car is oriented straight and is stationary. Notice also the corresponding value of **9830** that is transmitted by the talkerim node to the teensy. Refer to the lab handout to see how and why we get a 15% duty cycle with 9830. The range of user input in this implementation is limited to $(-5, +5)$ which corresponds to 10% and 20% duty cycle respectively. Again, refer to the lab handout if you aren't sure why these thresholds of the duty cycle are important.
We will now command the steering through the keyboard. The same type of control holds good for throttle.

You can see that I have chosen the incremental steps to be of the order of 1% duty cycle(for the steering, the throttle changes by 0.1%). This is something you can take a call upon in your implementation. Also, I have made my control to be such that the value of a particular command does not go back to 0 if the key is unpressed. These are just design choices and you could implement it in whatever style you prefer. Remember, you also have the option of doing this in Python. Refer to the documentation for related links.

There are a lot of opportunities to be creative in this lab and we hope you guys enjoy.

References

[1] Rosserial setup and tutorials

[2] ROS Tutorials - Custom Messages

[3] CMakeLists

[4] Building a Package

[5] Simple Subscriber and Publisher ROS