F1/10 Autonomous Racing

Assignment 3 - CS4501/SYS4582

Madhur Behl (madhur.behl@virginia.edu)

Keyboard Control for the F1/10 Car

Due Date: April 3, 2019

Overview

This assignment is to make sure you have setup your car correctly to the point where you can send it commands manually from your keyboard, and it will execute them. This will confirm that the car runs (always a good thing) and that the wireless network for communicating with the car is also correctly setup. In addition to being a basic check, the keyboard controls will allow you to drive the F1/10 racecar around and collect data through the sensors in the following week. Once this data is acquired, you can test your perception and control algorithms on this data first, before deploying them on the car.

Introduction – The Teensy Setup, The Jetson, and the ESC/Servo

The Teensy

The RF receiver module on the car sends pulse width modulation (PWM) signals to both the servo (controls the steering) and the ESC (controls the throttle). These are via the CH1 and CH3 connection ports on the receiver. This PWM signal is a 100Hz wave with varying duty cycles. A duty cycle of 10% on both the throttle and the steering makes the car reverse at full speed as well as orient wheels in one direction. The opposite actions are observed when the duty cycle is set to 20%. A PWM signal of 15% duty cycle on both channels results in 0 throttle and 0 steer (i.e wheels oriented straight ahead). These values have been determined through testing of similar Traxxas models which have the same ESC and Servo and therefore should work for your car as well.

To bypass the RF receiver, we feed the required 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 TK1 via USB interface, configured as a serial port (using rosserial). The **Teensy itself runs a ROS node that listens for this particular command. The Teensy is already flashed with this code.** All you need to do is publish the correct messages from the nodes running on the Jetson.

The Jetson

There are two nodes that are relevant to the keyboard control on the Jetson. The `talker.py` node and the `keyboard.py` node.

The `talker.py` node takes in as input values in the **(-100 to 100)** range for both the steering and the throttle channels. This node listens to the published messages from `keyboard.py` and converts them to appropriate PWM values. The teensy node listens to the messages published by `talker.py` and generates the PWM waves with the specified duty cycle.

Assignment Description and Code Layout:

Your task in this assignment is to:

- Implement the `talker.py` node.
- Modeify the `keyboard.py` node

To begin the assignment, skeleton code is provided on the git repo. https://github.com/linklab-uva/f1tenth-course-labs

This is the same repository that you have used for previous assignments.

A new ROS package called `race` will be used for this assignment. On your virtual machines, you should already have the `beginner_tutorials` package that we have been using previously for the labs. By running `git pull` in your local `f1tenth_labs` folder, you can fetch the latest version of the repo which contains the `race` package.

```
cd ~/github/f1tenth_labs
git pull
```

We will use the `race` package for this assignment. The contents of the race package folder should be the following:

```
> race
    >> msg
        >>> drive_param.msg
        >>> drive_values.msg
    >> src
        >>> keyboard.py
        >>> talker.py
    >> CMakeLists.txt
    >> package.xml
```

The code in this package uses two custom messages - `drive_param.msg` defines a pair of float values to define the desired throttle and steering (published by `keyboard.py`). And `drive_values.msg` custom message defines an int pair that is used to transmit the commanded PWM to the Teensy

The `drive_param.msg` file declares the following message type

```
float32 velocity
float32 angle
```

The `drive_values` message declaration has the following fields

```
int16 pwm_drive
int16 pwm_angle
```

How to transfer the `race` package from your `VM` to the `Jetson`

Step 1: `ssh` into the `Jetson TK1`

Make sure the Jetson and the Ubiquiti Wifi access point are powered on. The car need to be powered on.

```
ssh ubuntu@192.168.X.1
password: ubuntu
```

X is your team's unique sub-net ID. (Same as the sticker on the Ubiquiti) 'ubuntu' is the default password. **Do NOT change this password**

Step 2: Create a `catkin_ws` folder on the Jetson

Note: This may already exist on the Jetson and you can skip this step.

In the same terminal as the ssh connector above. Follow the steps below to create a `catkin_ws` on the Jetson TK1

```
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws/src
catkin_init_workspace
cd ..
catkin_make
source ~/catkin_ws/devel/setup.bash
```

Step 3: Move the `race` package to the Jetson

Open a terminal on a host computer [i.e the virtual machine] and type:

```
scp -r ~/github/f1tenth_labs ubuntu@192.168.X.1:home/ubuntu/catkin_ws/src
```

This will transfer the contents of the `~/github/f1tenth` folder to the `home/ubuntu/catkin_ws/src` folder on the Jetson. If you try to build the packages on the Jetson right away, you may get errors since the `keyboard.py` and `talker.py` files are incomplete and will not compile.

---

keyboard.py node

The function of this node is to parse the user pressed keyboard commands and convert them into a value between **(-100,100)** for both the steering and the throttle. This node publishes on a topic called `drive_parameters` using a custom message called `drive_param`. The up and down arrows (or any other keys of your choice) should map to the velocity field of the message and the left and right arrows (or any other keys of your choice) should map to angle field of the message.

The name of the node initialized in `keyboard.py` is called `keyboard_talker` [Yes it is indeed confusing that this node is also being called a talker]

In the `keyboard.py` node, the user input from the keyboard is implemented via a non blocking call to stdin.

The assignment requires you to fill in the missing conditions to increment/decrement the throttle/steering

e.g. this means every time the upper arrow key is pressed the throttle value increases by 0.1. Therefore to

drive the car forward, the user needs to repeatedly tap the upper arrow (or hold it pressed down for sometime) for the car to move forward. The same applies to the steering and to moving backwards.

In addition, map one of the keyboard keys as **reset** i.e. upon pressing the reset, the throttle and steering are set to zero.

To summarize, complete the `keyboard.py` code to generate `drive_param` messages that are published on the topic `drive_parameters` . To create a message you need to map the keyboard presses into float32 values between **(-100,100)** for both the `velocity` and the `angle` fields of the message. You ned to do so in increments of **0.1**.

The values that `keyboard.py` publishes will be subscribed by the `talker.py` node , which will map these values into an integer for the PWN message. Lets take a look at what needs to be done for the ~talker.py` node next.

---

talker.py node

Some skeleton code is provided but you will have to code almost the entirety of this node.

The `talker.py` node has the following functions:

- It subscribes to the topic `drive_parameters` being published by the `keyboard.py` node.
- It maps the float32 `velocity` and `angle` fields of the `drive_param` message into PWM values (mapping details included below).
- It publishes the calculated PWM values on a topic `drive_pwm` using a custom message format called `drive_values` . As you may have guess the Teensy subscribes to the `drive_pwm` topic.

You need to write code that subscribes to the `drive_parameter` topic being published by the `keyboard.py` node. You need to write a callback function which can parse the message received on this topic.

Converting `velocity` and `angle` values to PWM values

As mentioned in the Introduction section, the Teensy can generate duty cycles between 10% to 20% and these maps to the extreme ends of throttle and steering for the car.

What does this mean for our `talker.py` node ?

- The 10% duty cycle (full throttle, full steer in one direction) corresponds to a value of **6554**
- The 20% duty cycle (full reverse, full steering in the opposite direction) corresponds to a value **13108**
- The 15% duty cycle (zero throttle, zero steering) corresponds to a value of **9381**

This means that the `talker.py` node should map the floating values it receives from the `keyboard.py` node to the **integer PWM range (6554,13108)** i.e. map from **(-100,100) to (6554,13108)** where -100 maps to the PWM value 6554, and +100 maps to the PWM value 13108.

Lets take a look at an example:

Say the `velocity` field of the received `drive_param` message is 25.8. First we observe that this is a positive velocity therefore it implies throttle forward. Therefore it will map to a PWM angle between (9381,13108) [negative values will map to (6554,9381)]. If a range of 100 maps to a PWM range of (13108-9381) i.e. 3277 values Then 25.8 should map to : 9381 + ((3277/100)*25.8) = 10226.466. However, this is incorrect! Since the resulting PWM value must be an integer. Therefore we need to round off the value to 0 decimal places and the resulting PWM value will be 10226

This example is for a specific value but you need to write a code that can take in any value between the range (-100,100) and generate the corresponding PWM (integer) values in the range (6554,13108) correctly. This needs to be done for both the `velocity` and `angle` values received by this node.

- The calculate PWM values must be stored in the custom message(`drive_values`) and published via the `drive_pwm` topic. The teensy listens on this topic and generates the required waveforms.

As you can guess the PWM values calculated for the `velocity` input needs to be added to the `pwm_drive` field of the published message and the PWM values for steering needs to be added to the `pmw_angle` field of the messages being published on the topic `drive_pwm` .

Be sure to run `catkin_make` , or modify the `CMakeList.txt` file or the `package.xml` file as necessary, especially if you are defining your own custom messages.

What you need to demo and submit

You will be asked to demo controlling the F1/10 racecar with your keyboard.

- You will ssh into the Jetson first
- launch `roscore`
- run the `talker.py` node that you have created

```
rosrun race talker.py
```

- This node should wait for desired throttle and steering inputs from the user which will be sent by the `keyboard.py` node.

- We will then run the rosserial node -i.e. the Teensy node. Using the rostral command to launch the node.

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

- We will then run the `keyboard.py` node

```
rosrun race keyboard.py
```

We should see the `talker.py` node print the velocities and steering angles its receiving. 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. The range of user input in this implementation is limited to (–100,+100) which corresponds to 10% and 20% duty cycle respectively. Again, refer to the introduction section if you aren't sure why these thresholds of the duty cycle are important.

We should be able to now command the steering through the keyboard. The same type of control holds good for throttle.

`Pro Tip:` You should be mindful of not exceeding the duty cycle thresholds (-100,100) and not map to PWM values outside the range (6554, 13108) and therefore should implement a saturation condition in your code.

There are a lot of opportunities to be creative in this lab and we hope you enjoy implementing controlling the F1/10 racecar using your keyboard. Just as with the turtles, if you can control em with the keyboard, you can control em autonomously.

Extra Credit

Add a piece of code that will run on the car, and checks regularly for the state of the network. If the network's status is poor (i.e. bad connection) then it issues a STOP command.