Step 16 - Test steering control

Assumptions

* Transmission control servo is mounted and controlled by a Teensy
* Wheel speed sensor is mounted and controlled by a Teensy

To be completed in this step:

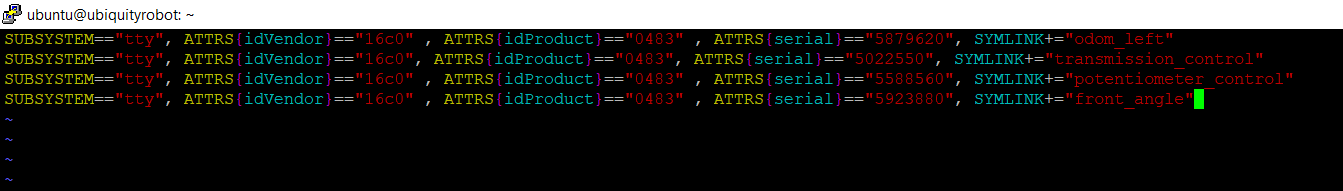
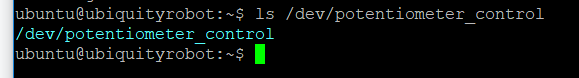
* Program both Teensy’s
* Configure PID settings

|  |  |
| --- | --- |

* Connect the motor controller power leads to the motor
* Run sample program that reads steering sensor
  + Connect USB drive
    - Make cable long enough to connect
    - Route cable
    - See which devices are currently connected $ ls /dev/ttyACM\*
      * /dev/ttyACM0 /dev/ttyACM1 /dev/ttyACM2 /dev/ttyACM3
  + Create a UDEV rule for the front\_angle Teensy

1. Create a UDEV rule for the front\_angle Teensy
   * Background: <https://arduino.stackexchange.com/questions/3680/in-linux-how-to-identify-multiple-arduinos-connected-over-usb>; <http://www.googlinux.com/clearing-dmesg-logs/index.html>
   * Logon to the RPI; Confirm the teensy is able to be seen by RPi. $ ls /dev/ttyACM\*, then connect front angle sensor, $ ls /dev/ttyACM\*
   * Unplug / turn off power to any USB devices
   * Save the dmesg log. $ dmesg > dmesg-`date +%d%m%Y`.log
   * Clear the log. $ sudo dmesg -c
   * Plug in the Teensy to be used. Remember this has a unique serial number.
   * List the dmesg $ dmesg
   * Results:

| ubuntu@ubiquityrobot:~$ dmesg  [118616.881363] usb 1-1.3.2: new full-speed USB device number 45 using dwc\_otg  [118617.014192] usb 1-1.3.2: New USB device found, idVendor=16c0, idProduct=0483  [118617.014207] usb 1-1.3.2: New USB device strings: Mfr=1, Product=2, SerialNumber=3  [118617.014215] usb 1-1.3.2: Product: USB Serial  [118617.014223] usb 1-1.3.2: Manufacturer: Teensyduino  [118617.014230] usb 1-1.3.2: SerialNumber: 5588760  [118617.015401] cdc\_acm 1-1.3.2:1.0: ttyACM0: USB ACM device |
| --- |

* Setup udev rule mapped to “steer\_motor ” because later you will want to update the setting: “upload port = /dev/steer\_motor ” in the platformio.ini file. Background reading: <https://linuxconfig.org/tutorial-on-how-to-write-basic-udev-rules-in-linux>
* Edit and update this line, then copy it because it will be easier to paste into the editor.
* SUBSYSTEM=="tty", ATTRS{idVendor}=="16c0" , ATTRS{idProduct}=="0483" , ATTRS{serial}=="5588760", SYMLINK+="steer\_motor"
* Change permissions so I can use FileZilla and Sublime to edit the udev files
  + $ sudo chmod -R 777 /etc/udev/rules.d/05-serial.rules
* $ sudo edit /etc/udev/rules.d/05-serial.rules
* 
* Move the cursor to the end of the last character in the last line
* Press INSERT; Move the cursor to the end and press enter
* Copy the new line above and paste into the editor by right mouse clicking at the beginning of the new line.
* Press Escape to exit the insert mode
* Save the file by entering “:w” then enter
* Quit using “:q” then enter
* Reboot the RPi for the changes to take effect $ sudo reboot
* $ ls /dev/steer\_motor to confirm the link has been made
* 

1. Update PlatformIO environment ini

* Open FileZilla, Update the platformio.ini file in /home/ubuntu; Add section for front\_angle

| ;  [env:front\_angle]  platform = teensy  board = teensy31  framework = arduino  upload\_port = /dev/front\_angle |
| --- |

* “board = teensy35” is used for a Teensy 3.5

1. Update bash scripts to help PlatformIO compiling
   * Background: There are two bash scripts that have been used for compiling, compile\_a.sh and compile\_b.sh. These scripts need to be updated because you have a new microcontroller connected to the USB hub.
   * Connect all the USB cables and run $ grep 16c0 /sys/bus/usb/devices/\*/idVendor to list all the Teensy devices connected.
   * For testing, run the scripts without the platformio or roslaunch commands enabled. I had issues getting permissions correct.
   * To resolve permission issues I ran:
     + $ sudo **chown** root:plugdev /sys/bus/usb/drivers/usb/{**bind**,unbind}
     + $ sudo **chmod** g+w /sys/bus/usb/drivers/usb/{**bind**,unbind}
     + See: <https://community.platformio.org/t/multiple-boards-upload-port-works-but-crashes-2nd-board/9029/8>

| ubuntu@ubiquityrobot:~$ grep 16c0 /sys/bus/usb/devices/\*/idVendor  /sys/bus/usb/devices/1-1.3.1.1/idVendor:16c0 (joystick)  /sys/bus/usb/devices/1-1.3.1.2/idVendor:16c0 (left speed)  /sys/bus/usb/devices/1-1.3.1.3/idVendor:16c0 (transmission)  /sys/bus/usb/devices/1-1.3.1.4/idVendor:16c0 (front\_angle)  /sys/bus/usb/devices/1-1.3.2/idVendor:16c0 (steer\_motor) |
| --- |

| **File: compile\_a.sh**  #!/bin/bash  # this script to "turn off" the USB ports and to start the compile process  #/sys/bus/usb/devices/1-1.3.1.1/idVendor:16c0 (joystick)  #/sys/bus/usb/devices/1-1.3.1.2/idVendor:16c0 (left speed)  #/sys/bus/usb/devices/1-1.3.1.3/idVendor:16c0 (transmission)  #/sys/bus/usb/devices/1-1.3.1.4/idVendor:16c0 (front\_angle)  #/sys/bus/usb/devices/1-1.3.2/idVendor:16c0 (steer\_motor)  #  #  # Comment out the device you will be programming  #  echo -n "1-1.3.1.1" > /sys/bus/usb/drivers/usb/unbind  echo -n "1-1.3.1.2" > /sys/bus/usb/drivers/usb/unbind  echo -n "1-1.3.1.3" > /sys/bus/usb/drivers/usb/unbind  echo -n "1-1.3.1.4" > /sys/bus/usb/drivers/usb/unbind  #echo -n "1-1.3.2" > /sys/bus/usb/drivers/usb/unbind # (steer\_motor)  #  # Un-comment the line for the platformio enviromnent you want to compile  #  #sudo platformio run -t upload -e front\_angle  sudo platformio run -t upload -e steer\_motor |
| --- |

| **File: compile\_b.sh**  #!/bin/bash  # this script to "turn on" the USB ports after the compile process is complete  #/sys/bus/usb/devices/1-1.3.1.1/idVendor:16c0 (joystick)  #/sys/bus/usb/devices/1-1.3.1.2/idVendor:16c0 (left speed)  #/sys/bus/usb/devices/1-1.3.1.3/idVendor:16c0 (transmission)  #/sys/bus/usb/devices/1-1.3.1.4/idVendor:16c0 (front\_angle)  #/sys/bus/usb/devices/1-1.3.2/idVendor:16c0 (steer\_motor)  #  echo -n "1-1.3.1.1" > /sys/bus/usb/drivers/usb/bind  echo -n "1-1.3.1.2" > /sys/bus/usb/drivers/usb/bind  echo -n "1-1.3.1.3" > /sys/bus/usb/drivers/usb/bind  echo -n "1-1.3.1.4" > /sys/bus/usb/drivers/usb/bind  #echo -n "1-1.3.2" > /sys/bus/usb/drivers/usb/bind  #  cd /home/ubuntu/catkin\_ws/src  #roslaunch teensy\_launch.launch |
| --- |

To test:

* $ grep 16c0 /sys/bus/usb/devices/\*/idVendor
* $ ls /dev/ttyACM\*
* $ bash compile\_a.sh then $ ls /dev/ttyACM\*
* $ bash compile\_b.sh then $ ls /dev/ttyACM\*
* If you have seen how they appear, disappear and reappear, you can now take out the comment for the PlatformIO statement in compile\_a.sh

1. Test compiling and running blink.cpp
2. Update launch file - steer\_test\_launch.launch
3. Test program
   * If USB ports are still unbound, run $ sudo platformio run -t upload -e steer\_motor
   * Once you have a complete compile and before launching the programs
     + Check power to motor
     + Ensure the work area is clear; Double check the steering rods and there are no obstacles (e.g. wires) in the way
     + Remind yourself where to disconnect power should you need to do so
     + Run $ bash compile\_b.sh

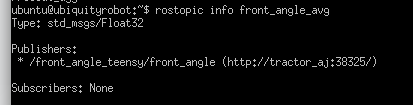
Problem:

* Need to confirm current steer angle is being published and read by test\_steer
* This is a good reminder there should be a watchdog process that if the steer angle stops being published then all systems should stop
  + Make sure /front\_angle\_avg is publishing data
    - Run
    - $ cd /home/ubuntu/catkin\_ws/src
    - $ roslaunch steer\_test\_launch.launch
    - Notes:
      * Angle is reading 49186 - the wheels are turned all the way to the right - since the extreme right is 49419 the angle sensor appears to be working
      * On Ubuntu laptop $ rosrun rqt\_plot /front\_angle\_avg

Useful video explaining setting up publisher/subscriber: <https://www.youtube.com/watch?v=2Cmdu6mkxD0>

$ rostopic info {topic}

$ rostopic info front\_angle\_avg



$ rosmsg show {type}

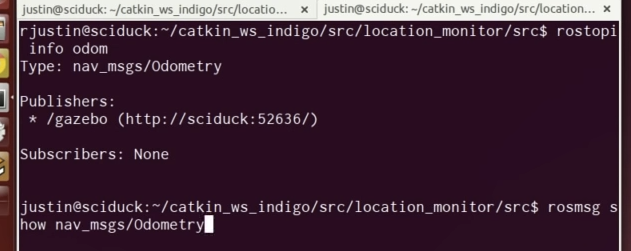
$ rosmsg show std\_msgs/Float32

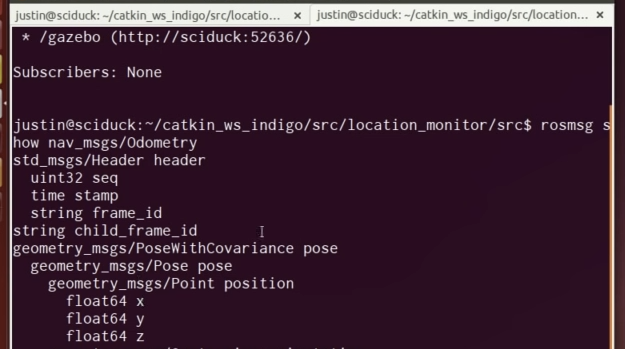
ubuntu@ubiquityrobot:~$ rosmsg show std\_msgs/Float32

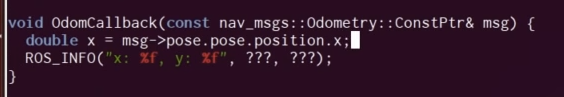
float32 data

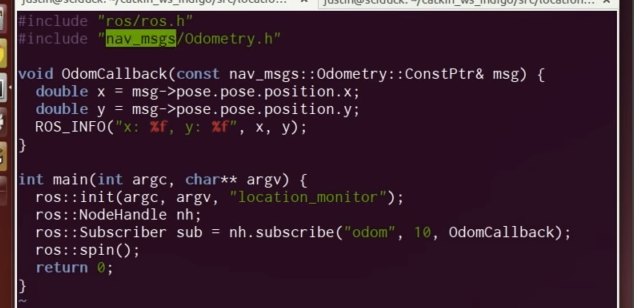
ubuntu@ubiquityrobot:~$

$ rosnode info /steer\_motor\_teensy/steer\_motor







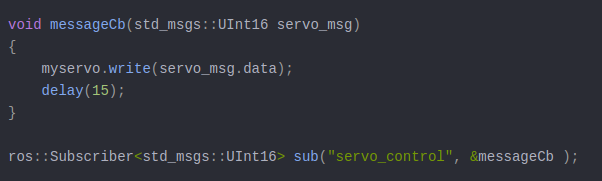


(const nav\_msgs::Odometry::ConstPtr& msg) {

|  | Yellow (pin 2) ->DIR (direction, High or Low)  RED (pin 3) ->PWM (speed, 0-255)  // Set the rotation direction.  digitalWrite(2, HIGH);  // Turn the motor on at half speed analogWrite(3, 127); |
| --- | --- |
|  |  |
|  | Connect all the wires, but for initial testing do not connect the “A” power from the controller to the motor. I don’t want to run the motor yet, until I’m sure the software is OK. |
|  | * Run test\_steer.cpp. I have hardcoded a steet\_angle\_target of 1042. There is a buffer of 20 positions (1022 - 1062) will produce a ROS\_INFO message and the wheels should be straight. * Move the wheels left or right and the LEDs should light up indicating the motor controller being engaged to bring the wheels straight * (HOW DO I CONFIRM THESE ARE NOT BACKWARDS - i.e LEFT is really the left I want and vice versa.) |

Background references:

* <https://www.cytron.io/p-30amp-5v-30v-dc-motor-driver> (OEM website)
* <https://docs.google.com/document/d/178uDa3dmoG0ZX859rWUOS2Xyafkd8hSsSET5-ZLXMYQ/view> (OEM ref for user manual)
* <https://www.instructables.com/lesson/Reversing-a-Motor/> (sample Arduino program)

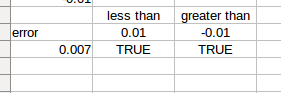


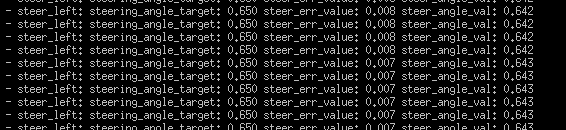
ros::Subscriber <std\_msgs::Float32> sub("/front\_angle\_avg", get\_front\_angle);

9/2/19

Problem:

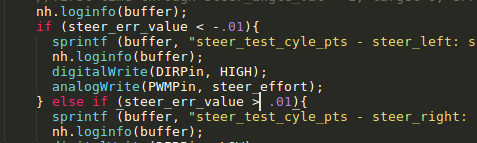
Steering program is not stopping when the error is small



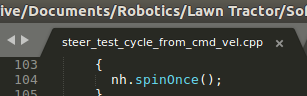


I think I need a compound if statement

9/3/2019

* Changing the signs
* 

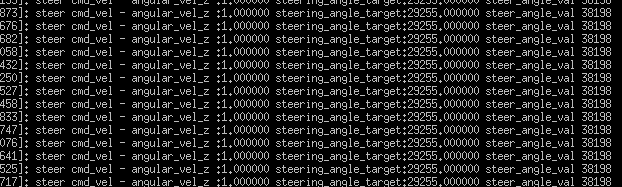
9/8/19

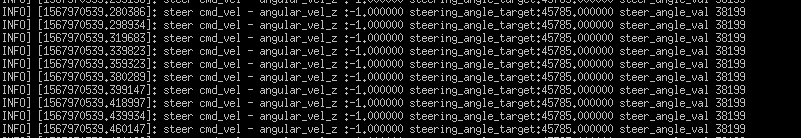
Writing program 

To read cmd\_vel and move steering based on it.

Testing process:

* Plug in joystick on laptop
* Start simple\_drive to publish to cmd\_vel
* On RPi, start the steer\_motor controller
* On laptop, $ roslaunch simple\_drive drive\_teleop.launch joy\_dev:=/deinput/js0



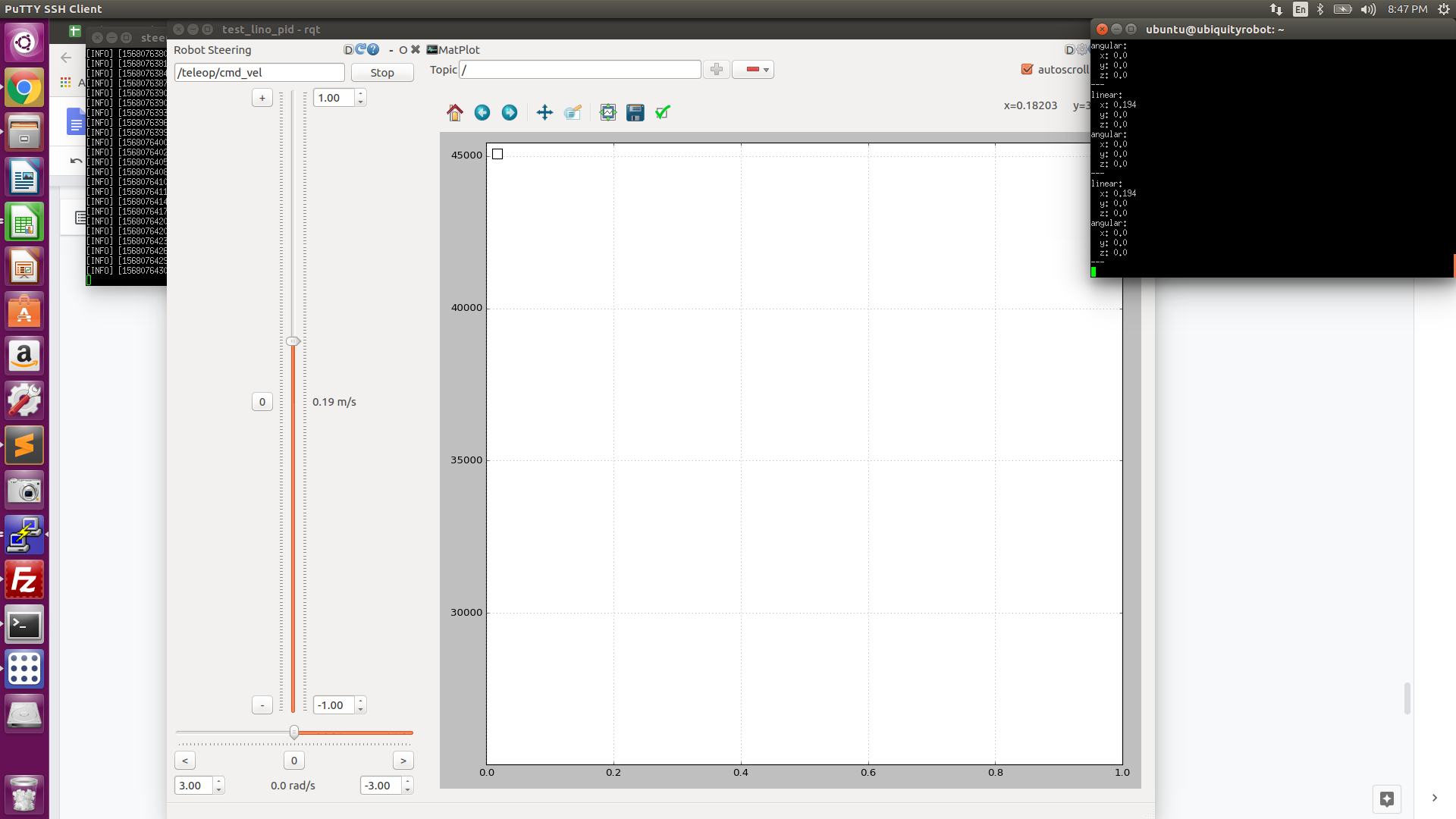


9/9/2019

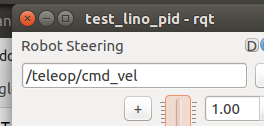
* Make changes to steer\_control\_v1.cpp to add publishing target position
* Use FileZilla to move the new source code to /src
* $ ls /dev/ttyACM\*
* $ bash compile\_a.sh
  + If the compile is not clean $ sudo platformio run -t upload -e steer\_motor
* Once compile is clean…. $ bash compile\_b.sh
  + If you just need to launch the program:
    - $ cd /home/ubuntu/catkin\_ws/src
    - $ roslaunch steer\_test\_launch.launch
* On laptop, Ctrl+Alt+T, $ rqt

|  |
| --- |

* Choose Plugins -> Robot Tools -> Robot Steering



In the rqt topic put /teleop/cmd\_vel in the topic field



On the RPi echo the topic and move the slider in rqt to see the topic being published

* Drag the window all the way to the right

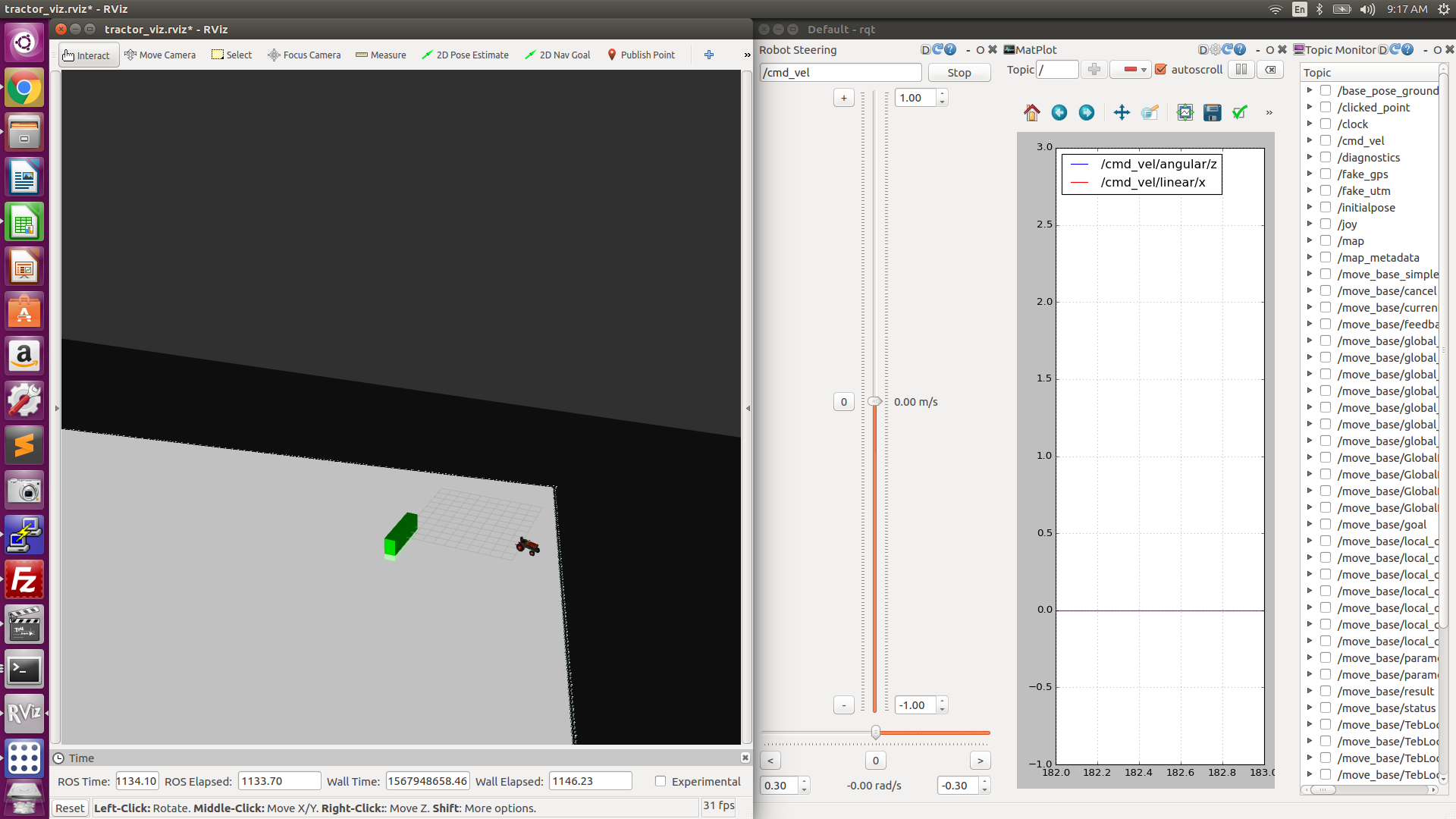
|  |
| --- |
|  |

* You can now adjust the slide bars and send cmd\_vel command using this interface
* Plugins -> Visualizations -> Plot will allow you to subscribe and visualize (i.e. plot) rostopics

|  |
| --- |

* Plugins -> Topics -> Topic Monitor lets you drag and drop the topics you are interested in

|  |
| --- |



Note

9/14/19

Debugging the target

The Output does not go negative; Found a set limits command

| Current | Target | Steer\_err\_value | cmd\_vel |
| --- | --- | --- | --- |
| 36357 | 36341 | -15 | -0.01 |
| 36357 | 36432 | 75 | -0.02 |

Testing protocol

| Cmv\_vel angular.z | Points to look for |
| --- | --- |
| 0.0 | Is the steering steady and straight; Is the steer effort zero |
| 0.03 to -0.03 | Does the steering react and move to these smaller increments |
| Move from 0 to +/- .5 | Does the steering reach the target and steer effort go to zero? What there overshoot? How long did it take to achieve the target? |
| 1 and -1 | Does the steering reach the target and steer effort go to zero? What there overshoot? How long did it take to achieve the target? |

Example output going from: 0.0 -> -0.5 -> +0.5 -> -0.7 -> +0.7 -> 0.0

