Step 4 - Install ROS-Ag lawn tractor simulator

Goto <https://github.com/ros-agriculture/ros_lawn_tractor>

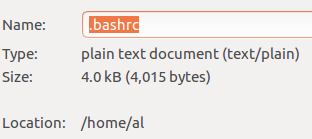
(There is also a Pure Pursuit option in the Google Drive <https://drive.google.com/open?id=1xIZZrAvjdeUaAolCRMZPGvgHo_0h7M0uH14-Q9gnoiw>

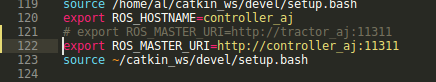
)

Follow the instructions (duplicated here):

* $ cd catkin\_ws/src
* $ git clone https://github.com/ros-agriculture/ros\_lawn\_tractor.git
* $ git clone https://github.com/bsb808/geonav\_transform.git
* cd ..
* $ rosdep update
* $ rosdep install -y --from-paths . --ignore-src --rosdistro ${ROS\_DISTRO}
* $ sudo apt-get install python-catkin-tools
* $ catkin build
* $ roslaunch lawn\_tractor\_sim lawn\_tractor\_sim.launch

Optional - If you have previously modified the ROS\_MASTER\_URI in your .bashrc file to have a separate controller tractor you will need to update .bashrc so the ROS\_MASTER\_URI is set to your laptop performing the simulation. ( Enter $ nano ~/.bashrc) or find .bashrc in /home/al and open with Sublime)



* + - * Comment out tractor\_aj
      * enable ROS\_MASTER\_URI controller for local laptop simulation
      * Save and $ source ~/.bashrc
      * 

| # if you want the laptop (aka the controller) to be the master, when testing ROS using the laptop, comment out the tractor;  # export ROS\_MASTER\_URI=http://tractor\_aj:11311  # if you are running the tractor it needs to be the master so comment out the controller/laptop;  # export ROS\_MASTER\_URI=http://controller\_aj:11311  # if you are running the simulator in Docker the master needs to be localhost; The controller/laptop and tractor should be commented out.  # Docker wants ROS\_MASTER\_URI=http://localhost:11311  export ROS\_MASTER\_URI=http://localhost:11311 |
| --- |

source /opt/ros/kinetic/setup.bash

source /home/al/catkin\_ws/devel/setup.bash

export ROS\_HOSTNAME=controller\_aj

# if you want the laptop (aka the controller) to be the master, when testing ROS using the laptop, comment out the tractor;

# if you are running the tractor enable it as the master; Comment out the controller/laptop;

export ROS\_MASTER\_URI=http://tractor\_aj:11311

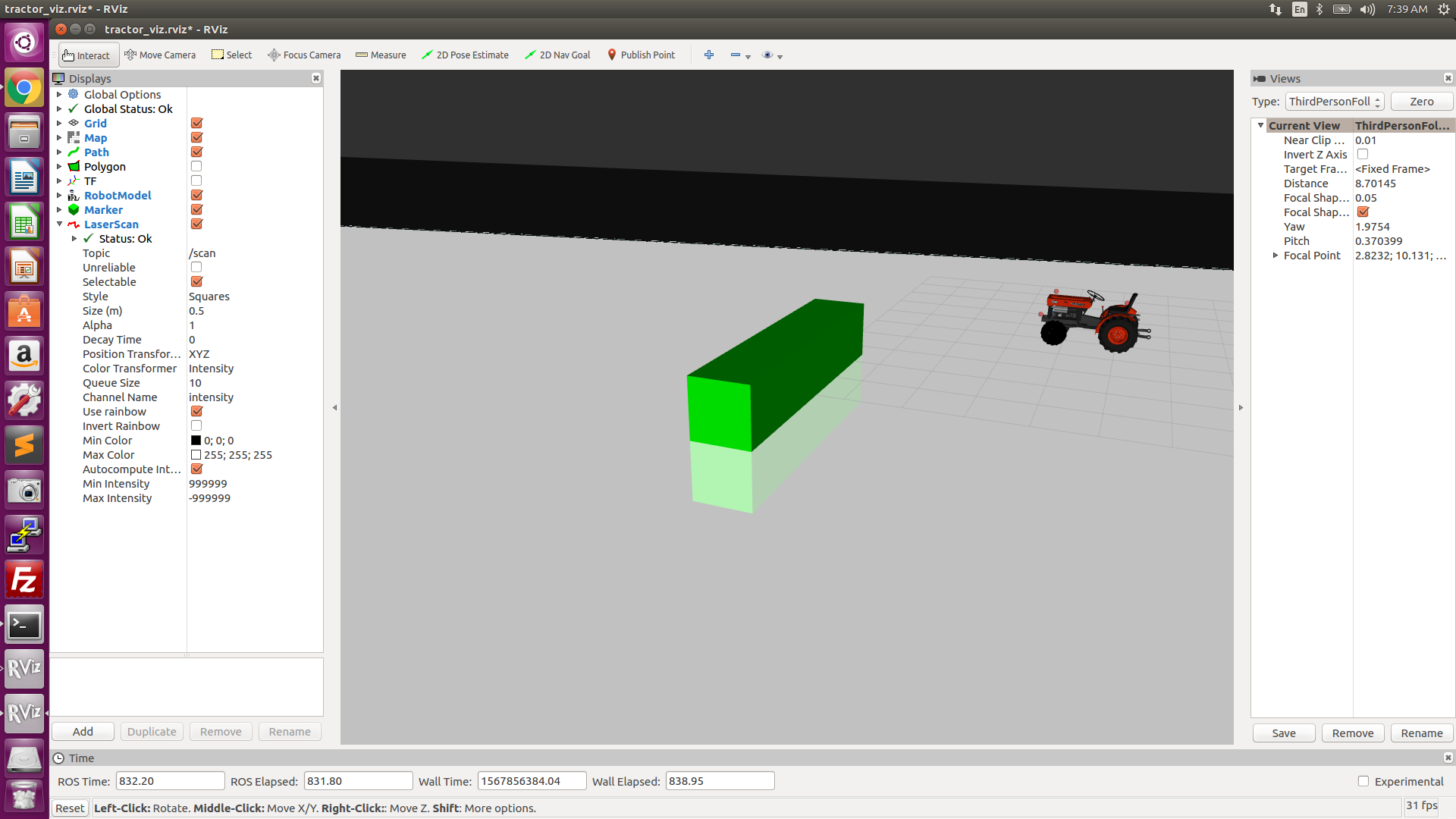
# if you are running the tractor enable it as the master; Comment out the controller/laptop;

# export ROS\_MASTER\_URI=http://controller\_aj:11311

# if you are running the simulator in Docker enable localhost as the master; The controller/laptop and tractor should be commented out.

# export ROS\_MASTER\_URI=http://localhost:11311

source ~/catkin\_ws/devel/setup.bash



* The tractor is now waiting for a 2D nav goal. You can send the vehicle on missions by using the 2D nav goal function in rviz. Click on the 2D Nav Goal button. Then click in the world where you want the tractor to end and drag the arrow in the direction you want the robot to end in. Once released you should see the tractor move to that location and heading.
* Warning: Obstacle detection is not enabled so it will not maneuver around the green block

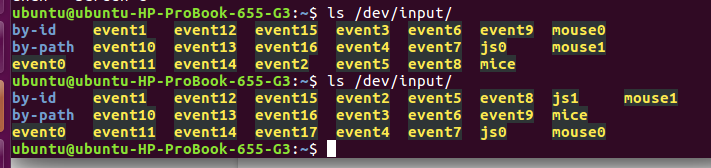
##### RViz keystroke controls

* **Left mouse button**: Click and drag to rotate around the focal point.
* **Middle mouse button**:
  + **Scrollwheel -** Roll forward and back for zoom
  + Click and drag to move the focal point in the plane
* **Right mouse button**: Click and drag to zoom in/out of the focal point. Dragging up zooms in, down zooms out.

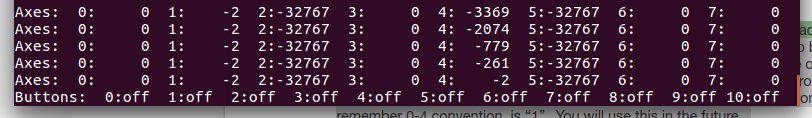
**Test drive using gamepad**

Confirm gamepad input:

* Before connecting the gampad run $ ls /dev/input/
* Now connect the gamepad and run the same command $ ls /dev/input/

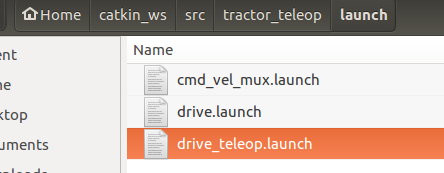
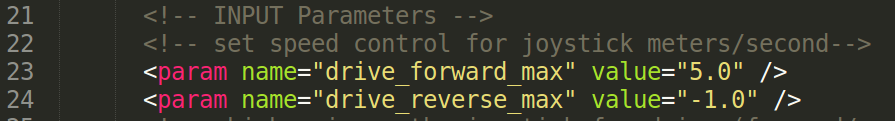


* For me the new device is “js1” # I’m told, but have not confirmed, if js0 is present that it is the accelerometer on the laptop.
* Try $ sudo jstest /dev/input/js1 or $ sudo jstest /dev/input/js0
* You should get output similar to the following. I like adjusting the window width so “Buttons” appears in column 1 of the window or wide enough so the screen does not move when the values change.



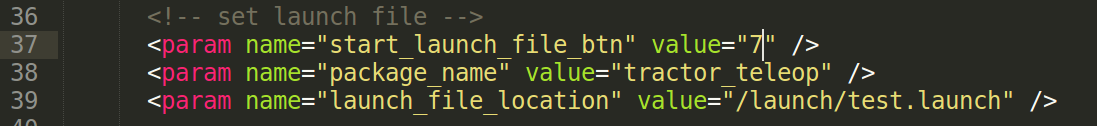
* When you use the joystick buttons you should see the values change.
* Later we will be sending /cmd\_vel twist messages. That node has a “deadman” button built in which means to send a command you will be pressing two buttons simultaneously (i.e. the deadman button and a command like the throttle or steering). Often the default setup is button 4 is the deadman. Test the various buttons and confirm if the left, top button on the front of the controller triggers “4” on and off.
* $ ctrl + C to stop the process

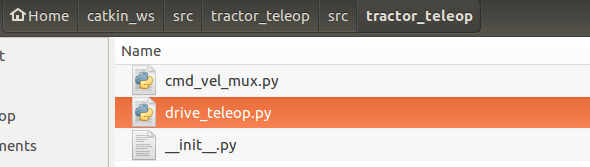
On Laptop, where joystick will be connected, install package to transfer data from /joy to /cmd\_vel:

* We will be using [https://github.com/da(https://github.com/danielsnider/simple\_drive/blob/master/src/simple\_drive/drive\_teleop.py)nielsnider/simple\_drive](https://github.com/danielsnider/simple_drive), but this has been modified somewhat so we will use the forked version that is located at <https://github.com/ros-agriculture/tractor_teleop>
* Because we need to change the code we need to load this into our local catkin\_ws
* $ cd catkin\_ws/src
* $ git clone https://github.com/ros-agriculture/tractor\_teleop.git
* cd ..
* In drive\_teleop.launch
* 
* Make sure “<param name="autorepeat\_rate" value="20" />” after the line “<param name="joy\_dev" value="$(arg joy\_dev)" />” is there.
* Also update the Max speed setting for forward and reverse. I would suggest 1.5 for both which is 1.5 meters per seconds.
  + 

| <launch>  <env name="ROSCONSOLE\_FORMAT" value="[${severity}][${thread}][${node}/${function}:${line}]: ${message}"/>  <!-- Joystick Device Argument -->  <arg name="joy\_dev" default="/dev/input/js0"/>  <!-- Joy Node -->  <node pkg="joy" type="joy\_node" name="joy\_drive" clear\_params="true" output="screen" respawn="true" >  <param name="joy\_dev" value="$(arg joy\_dev)" />  <param name="autorepeat\_rate" value="20" />  <param name="deadzone" value="0.12" />  <!-- OUTPUT TOPICS -->  <remap from="joy" to="joy" />  </node>  <!-- Teleoperation Node -->  <node pkg="simple\_drive" type="drive\_teleop" name="drive\_teleop" clear\_params="true" output="screen" respawn="true">  <param name="servo\_pan\_speed" type="int" value="5"/>  <param name="servo\_pan\_max" type="int" value="160"/>  <param name="servo\_pan\_min" type="int" value="0"/>  <!-- INPUT TOPICS -->  <remap from="joy" to="joy" />  <!-- OUTPUT TOPICS -->  <remap from="teleop/cmd\_vel" to="teleop/cmd\_vel" />  <remap from="servo\_pos" to="servo\_pos" />  <remap from="move\_base/cancel" to="move\_base/cancel" />  </node>  </launch> |
| --- |

Decide what launch file you want the “start” button to launch



* Confirm **cmd\_vel\_mux.launch** has the correct cmd\_vel destination
  + <!-- INPUT TOPICS -->
  + <remap from="teleop/cmd\_vel" to="teleop/cmd\_vel" />
* Confirm **drive\_teleop.launch** has the correct cmd\_vel destination
  + <!-- OUTPUT TOPICS -->
  + <remap from="teleop/cmd\_vel" to="teleop/cmd\_vel" />
* The original program from Daniel Snyder was written for a skid steer vehicle and mixed both joystick readings together. One of the changes with tractor\_teleop is changing how the values are adjusted. With Ackerman we do not want to mix values. Matt also removed an estop service call so this will work in simulation. If you want the estop you would add the commented code below. (see Matt for help).
* 

| #!/usr/bin/python  # python node than listens to joy node and publishes to cmd\_vel\_mux  # it will also call the relay service (e.g. you want to shut down the tractor  import rospy  import subprocess  import time  from sensor\_msgs.msg import Joy  from std\_msgs.msg import Float32  from geometry\_msgs.msg import Twist  from actionlib\_msgs.msg import GoalID  # from lawn\_tractor.srv import \* # not able to run this in simulation  class DriveTeleop:  def \_\_init\_\_(self):  self.cmd\_vel\_pub = rospy.Publisher("teleop/cmd\_vel", Twist, queue\_size=1)  self.goal\_cancel\_pub = rospy.Publisher("move\_base/cancel", GoalID, queue\_size=1)  self.joy\_sub = rospy.Subscriber("joy", Joy, self.on\_joy, queue\_size=1)  # self.service\_relay\_client = rospy.ServiceProxy("/relay\_cmd", relayCmd) # will fail in simulation  # self.service\_relay\_client\_object = relayCmdRequest() # will fail in simulation  def on\_joy(self, data):  # Drive sticks  if data.buttons[4]: # deadman 5 button  angular\_vel = data.axes[0]  linear\_vel = data.axes[1]  # Publish Twist  twist = Twist()  twist.linear.x = linear\_vel  twist.angular.z = angular\_vel  self.cmd\_vel\_pub.publish(twist)  # EStop  if data.buttons[1]: # 2 button  rospy.loginfo('Estop tractor')  # self.service\_relay\_client\_object.channel = 27  # self.service\_relay\_client\_object.state = 0  # result = self.service\_relay\_client(self.service\_relay\_client\_object)  # if result.success == True:  # self.engine\_off = 1  # self.tractor\_ready = 1  # Cancel move base goal  if data.buttons[2]: # X button - will cancel an autonomous path if one has been programmed  rospy.loginfo('Cancelling move\_base goal')  cancel\_msg = GoalID()  self.goal\_cancel\_pub.publish(cancel\_msg)  def main():  rospy.init\_node("drive\_teleop")  # rospy.wait\_for\_service("/relay\_cmd")  controller = DriveTeleop()  rospy.spin() |
| --- |

* The statements commented out are because they drive a relay board that is not implemented in the simulator.
* From ~/catkin\_ws, $ catkin build
* $ source devel/setup.bash
* $ rospack find joy # returns /opt/ros/kinetic/share/joy
* $ rospack find tractor\_teleop # returns /home/al/catkin\_ws/src/tractor\_teleop

Start joy\_input to transfer data to cmd\_vel:

* Ctrl+Alt+T, $ roslaunch tractor\_teleop cmd\_vel\_mux.launch
* Ctrl+Alt+T, $ roslaunch tractor\_teleop drive\_teleop.launch joy\_dev:=/dev/input/js0
* Note: drive\_teleop.launch has as a default: <arg name="joy\_dev" default="/dev/input/js0"/> If your joystick is on js1 you can either change the launch file or adjust the argument.
* Alternatively, the command below has both of the above commands combined
  + $ roslaunch tractor\_teleop drive.launch
* If you hold the deadman button and move the joystick there will be output on /cmd\_vel and vehicle should move.
* You should be able to open a window and run $ rostopic echo cmd\_vel and see output.
* The package is programmed with a deadman switch which is required to be pressed at the same time the left joystick is being moved. Without using both simultaneously the messages will not be sent to the vehicle. This is a safety design when operating in the real world.

Additional control and visualization tools:

* Drag the rvis screen to the left so it occupies only ½ of the page

|  |
| --- |

* Ctrl+Alt+T, $ rqt

|  |
| --- |

* Choose Plugins -> Robot Tools -> Robot Steering
* Drag the window all the way to the right

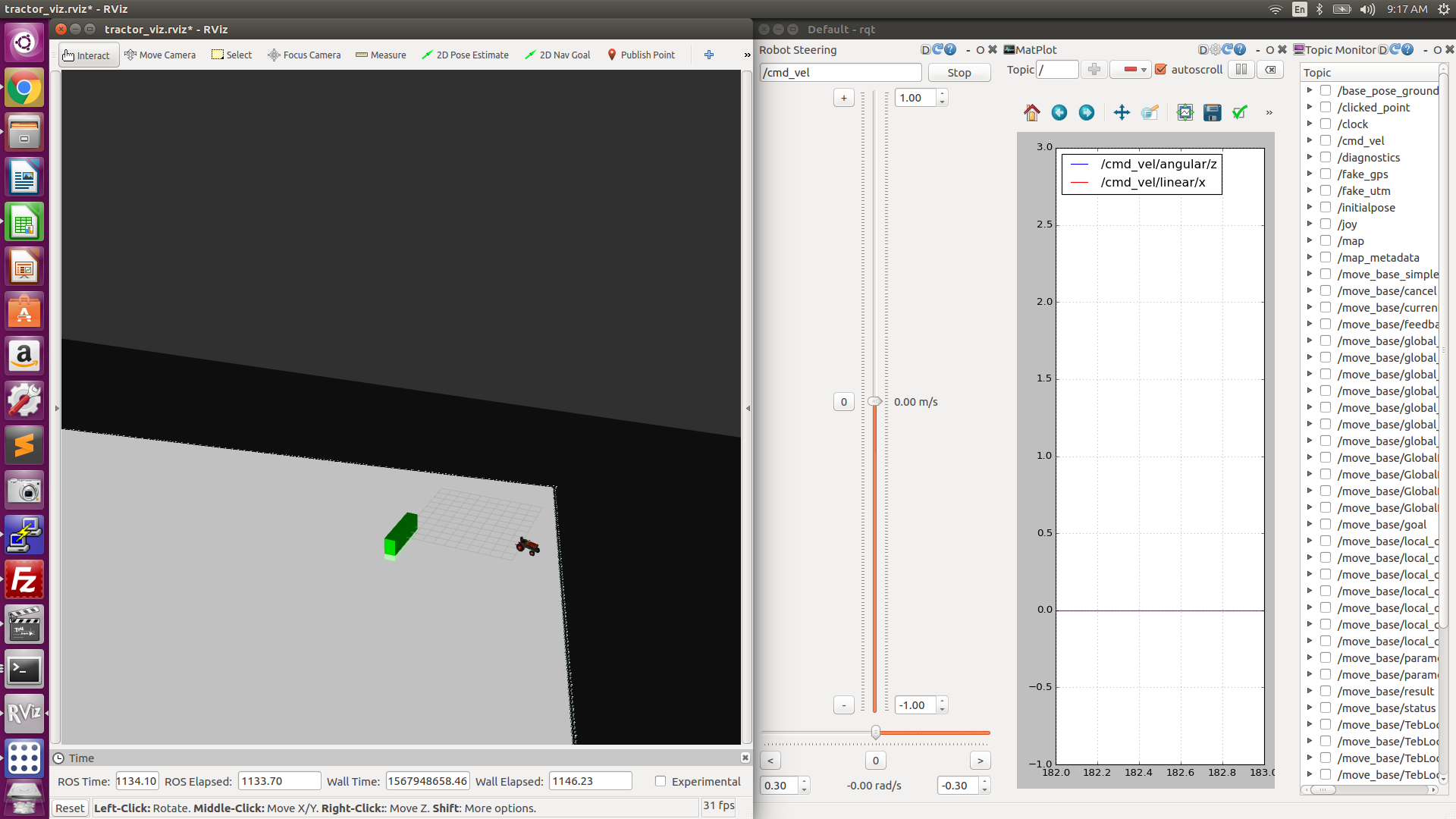
|  |
| --- |
|  |

* 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

|  |
| --- |



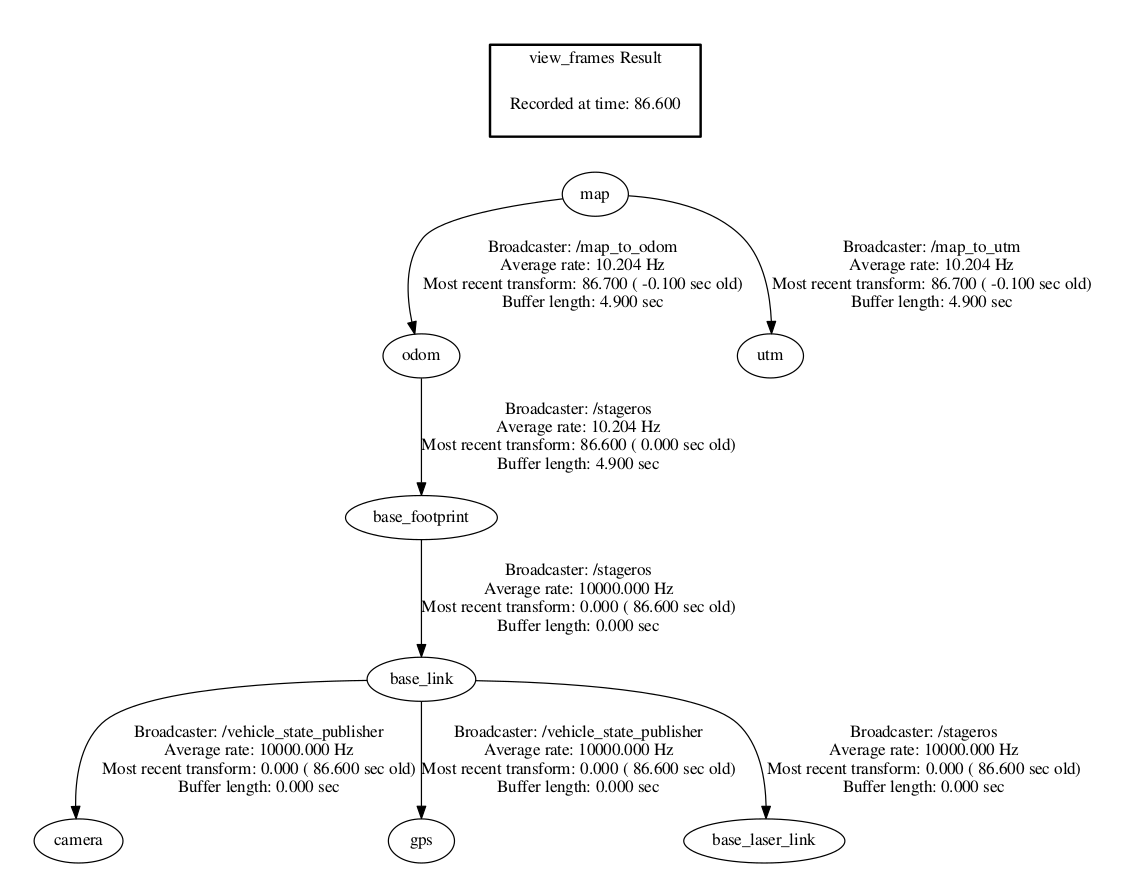
Notes about values:

* /cmd\_vel linear.x range is 1 to -1
* /cmd\_vel angular.z range is 1 to -1

Capture the tf frames

In a New window

* $ rosrun tf view\_frames && evince frames.pdf



Reference notes:

* Gamepad tutorial [link](http://wiki.ros.org/joy/Tutorials/ConfiguringALinuxJoystick) That tutorial is based on indigo. I am using Kinetic.
* Urdf file modelling an ackerman vehicle: <https://github.com/jbpassot/ackermann_vehicle/blob/master/ackermann_vehicle_description/urdf/em_3905.urdf.xacro>
* Light blog post (Nov. ‘17); Ackermann steering car robot model with simulation in Gazebo: <http://ros-developer.com/2017/11/09/ackermann-steering-car-robot-model-with-simulation-in-gazebo/>

# Control your robot with a joystick in ROS (updates yaml file to create Twist message): <http://ros-developer.com/2017/07/28/control-your-robot-with-a-joystick-in-ros/>