

INTRODUCTION TO TURTLEBOT2I

Luis D Santiago

PORTLAND STATE UNIVERSITY Portland OR

Contents

Objective	2
Before we begin	2
References used	2
Turtlebot2i first use	3
Power on	3
Connect and Start the turtlebot2i	3
Navigate the Turtlebot2i using the keyboard	3
Using Arm Demo	4
Detect the environment and Navigate (Intermediate)	4
ROS Intro	5
What is ROS?	5
Why ROS?	5
What does that mean to me?	6
Adding your own Code	6
How to install in own computer.	6
Turtlebot2i troubleshooting	7
Background	7
Update the Turtlebot2i repository	7
Robot	7
Charger will not charge the LiPo battery	8
Arm Block Demo does not work	8
Is it the battery?	9
How to diagnose a bad servo?	9
RobotModel not shown in Rviz	10
Moving the base to a location in the map.	10
How to identify a location	10
Send location to turtlebot2i	11

Objective

The goal of this paper is to introduce students to turtlebot2i and provide a high-level overview of ROS. It is not intended to be a ROS tutorial. Instead it shows how to use existing demos in a way that demonstrates some of ROS's features such as distributed computing and the benefits of bundling with launch file. By the end of this paper you should know how to use the turtlebot2i demos, have a some understanding of how ROS is used, and know how to make code changes without affecting other students.

To accomplish this goal first, I will explain how to run the turtlebot2i demos. As you use these demos please note that sometimes you will run from the robot and other times from a workstation. While this approach is not required, I chose it to emphasize the distributed computing features of ROS. Next, I will provide a high-level ROS explanation that should help you understand how the demo steps worked. Lastly, I will provide a few suggestions of how to make your own code changes, how to run them from your computer, and some troubleshooting examples.

Before we begin

I would like to make it clear that most of the content in this document is not original. Instead I compiled useful information from various sources into a single document and added my comments based on my experience using it in the lab. You will learn and practice ROS programming in class. While this is not the only way to use the turtlebot2i, the approach used to run the demos were chosen to emphasize how launch files were used for code separation and how the ROS server allows to distributing computing without much effort from the developer.

There are additional reference books that I donated to the lab for your use. Please respect the authors and do not publish or re-distribute the material in this document; it is copyrighted material.

References used

- Turtlebot2i wiki: <https://github.com/Interbotix/turtlebot2i/wiki>
- ROS Wiki: <http://wiki.ros.org/ROS/Introduction>
- A Gentle Introduction to ROS, Jason M. O'Kane, Independently published, oct 2013, ISBN= 978-1492143239, <http://www.cse.sc.edu/~jokane/agitr/>
- ROS Robotic by Example, Carol Fairchild and Thomas Harman, Packt Publishing, June 2016

Turtlebot2i first use

Assumptions:

1. Kobuki base and LiPo battery are fully charged.
2. Robot is connected to the WiFi.

This section provides step-by-step instructions on how to use turtlebot2i for the first time. Do worry if you don't understand what is going on and why we are issuing commands in two computers; that will be addressed later. For now, just complete the next three sections (from 'Power on' to 'Navigate the Turtlebot2i using the keyboard'), play, and get familiar with the robot.

Power on

1. Disconnect the chargers and remove the cables
2. Turn on the base and wait 10 seconds
3. Turn on the Joule and wait ~45 seconds (to allow the OS to boot)

Connect and Start the turtlebot2i

(The following steps are needed for nearly everything you do with turtlebot2i)

4. Log on to the Workstation (ask lab manager for password)
 - a. ssh into the robot -> `$ ssh turtlebot@tb2i-1.local`
 - b. turtlebot2i password is robot1
5. Go to turtlebot2i directory - `$ cd turtlebot2i`
6. Run goros- `$goros`
7. Run roscore- `$roscore`
8. Open another terminal/tab
9. Repeat steps 4-7 then go to next step
10. Bring up the full robot with mapping, arm pose manager, moveit and zone detection using one of the roslaunch commands below:
 - a. Start robot and create a new map (erasing data from previous sessions):
 - i. `roslaunch turtlebot2i_bringup turtlebot2i_demo1.launch new_rtabmap:=true`
 - b. To resume mapping:
 - i. `roslaunch turtlebot2i_bringup turtlebot2i_demo1.launch`
 - c. To use existing map for localization purposes only:
 - i. `roslaunch turtlebot2i_bringup turtlebot2i_demo1.launch localization:=true`

More command options in <https://github.com/Interbotix/turtlebot2i/wiki/01:-Getting-Started>

Navigate the Turtlebot2i using the keyboard

1. Connect to the robot and start the robot (previous section)
2. Open a new terminal but DO NOT ssh into the robot
3. Go to turtlebot2i directory - `$ cd turtlebot2i`
4. Run goros- `$goros`
5. `roslaunch turtlebot_teleop keyboard_teleop.launch`
6. Follow on screen instructions and have fun

Using Arm Demo

Note: Hardware can be a little problematic and it does not always work. Check the 'Arm Troubleshooting' section if you have issues.

Here, we are just going to use the block demo to see the arm in motion.

1. Connected and Start the turtlebot2i (a previous section)
2. Open a new terminal but DO NOT ssh into the robot
3. Go to turtlebot2i directory - `$ cd turtlebot2i`
4. Run goros- `$goros`
5. Place at least one of the blocks in front of the robot.
6. Run `$ roslaunch turtlebot2i_block_manipulation block_sorting_demo.launch`
7. Follow on screen instruction and have fun (note: robot 1 had a gripper issue as of summer 2017)

Detect the environment and Navigate (Intermediate)

This section assumes you already know how and when to use x-term. It may be best for some students to skip this section until they understand ROS, turtlebot2i and its components a little better. It also assumes that you completed the previous sections, know something about the navigation stack, and how maps work.

1. You need to have a good map
 - a. See instructions for checking the robot map [here](#))
2. If you know that you already have a good map proceed to step 3
 - a. If you do not have a good map refer to the section 'Connect and Start the turtlebot2i' and in step 12 launch with create a new map (option a) use the keyboard teleop to navigate and create a map.
 - b. Once you navigated the lab stop, the launched file (ctrl-c) and it will save the map automatically
3. Start the robot, if not connected already, and for the last step use start localization: `$roslaunch turtlebot2i_bringup turtlebot2i_demo1.launch localization:=true`
4. Use the keyboard tele-op to navigate the room and find the objects. (the signs placed on the lab see [here](#)). You can use Rviz to see when the locations are detected.
5. Once all zones are detected. CLOSE keyboard teleop
6. From the workstation or the robot type any one of the following commands:
 - a. `rostopic pub /zone_destination_request std_msgs/String "charging_zone" --once`
 - b. `rostopic pub /zone_destination_request std_msgs/String "pickup_zone" --once`
 - c. `rostopic pub /zone_destination_request std_msgs/String "sorting_zone" --once`
7. Robot should move automatically near that area

ROS Intro

I recommend reading at least the first five pages of Gentle introduction to ROS, digital version available [here](#). Also, check out the ROS wiki [here](#). Just remember to not worry yet if the next two sections mean nothing to you right now; I will try to explain why they are relevant in the following sections. Also, they will be covered again in class along with other topics in greater detail. If you need more help you can also use the reference book left in the lab.

What is ROS?

From the ROS wiki: ROS is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly used functionality, message passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers.

The ROS runtime "graph" is a peer-to-peer network of processes (potentially distributed across machines) that are loosely coupled using the ROS communication infrastructure. ROS implements several different styles of communication, including synchronous RPC-style communication over services, asynchronous streaming of data over topics, and storage of data on a Parameter Server. These are explained in greater detail in our Conceptual Overview.

Why ROS?

It comes with debugging tools:

- Rviz (<http://wiki.ros.org/rviz>) is one of the 3D visualizers available in ROS to visualize 2D and 3D values from ROS topics and parameters. Rviz helps visualize data such as robot models, robot 3D transform data (TF), point cloud, laser and image data, and a variety of different sensor data
 - Command: `rviz rviz`.
- The `rqt_plot` program (http://wiki.ros.org/rqt_plot) is a tool for plotting scalar values that are in the form of ROS topics. We can provide a topic name in the Topic box.
 - Command: `rqt_plot rqt_plot`
- The `rqt_graph` (http://wiki.ros.org/rqt_graph) ROS GUI tool can visualize the graph of interconnection between ROS nodes.
 - Command: `rqt_graph rqt_graph`

Read "Gentle Introduction to ROS" for many more good reasons

Concurrent resource handling: Handling a hardware resource by more than two processes is always a headache. Imagine, we want to process an image from a camera for face detection and motion detection, we can either write the code as a single entity that can do both, or we can write a single threaded code for concurrency. If we want to add more than two features in threads, the application behavior will get complex and will be difficult to debug. But in ROS, we can access the devices using ROS topics from the ROS drivers. Any number of ROS nodes can subscribe to the image message from the ROS camera driver and each node can perform different functionalities. It can reduce the complexity in computation and increase the debug-ability of the entire system.

What does that mean to me?

The full meaning is different depending your goal but, regardless, it should mean faster and simpler design. I will try to relate this to lab examples and possible projects.

From the Hardware view: Let's imagine you are moving the arm or adding another hardware. You can develop or reuse drivers with basic functionality using existing methods of communication like services and messages. Your concern is to provide communication method to access your hardware capabilities without having to worry about how or where your hardware is going to be used.

From the software perspective: No need to wait for hardware to be available or learn a new API. You can write code and possibly debug it before the hardware is available (only if a model is available). Also, if you must deal with many modules or interact with hardware it may be unlikely that you will have to worry about synchronization or how it was implemented in hardware. The ROS messaging server and graph will take care of most of the complications that come with synchronization, you just need to subscribe to the messages that has the data you need and publish in a format that can be used by another node and let ROS do the rest.

From the design side: Once again let's say want to move the arm to a different location or add a different one. With ROS it is possible to change the robot's design without buying any hardware, without making actual mechanical changes, and without writing code to operate the robot. Once a new model is model is completed higher level code can be tested against the model before the hardware is changed/acquired and before drivers are completed. In my opinion, though it is the hardest part to learn and master, creating robot models is one of the most powerful tools ROS provides, enabling parallel development and reducing costs.

Adding your own Code

You may remember the peer-to-peer network process from the 'What is ROS' section above. What this means to us is that, after the drivers are launched, it doesn't matter what computer we use as long as we are using the existing nodes and messages. It will be unlikely for a student to have a need to modify the files in the robot (unless new hardware is added). Most of the changes should be made to the files located on the remote computer. Furthermore, you may even do all the changes on your computer and just connect the robot to the network to validate your changes in the actual robot.

How to install in own computer.

```
sudo apt install ros-kinetic-turtlebot* libudev-dev ros-kinetic-find-object-2d ros-kinetic-rtabmap-ros ros-kinetic-moveit ros-kinetic-octomap-ros ros-kinetic-manipulation-msgs ros-kinetic-controller-manager python-wxgtk3.0
```

Setup TurtleBot2i Source Code & Catkin Environment

```
source /opt/ros/kinetic/setup.bash
cd ~
mkdir -p ~/turtlebot2i/src
cd ~/turtlebot2i/src
```

```
git clone https://github.com/Interbotix/turtlebot2i.git .
git clone https://github.com/Interbotix/arbotix_ros.git -b turtlebot2i
git clone https://github.com/Interbotix/phantomx_pincher_arm.git
git clone https://github.com/Interbotix/ros_astra_camera -b filterlibrary
git clone https://github.com/Interbotix/ros_astra_launch
cd ~/turtlebot2i
catkin_make
```

bashrc setup

- You will need to place the following at the bottom of your ~/.bashrc file
- Edit .bashrc with your choice of editor (gedit, nano, vi)
- Be sure to replace example.hostname with your computer's hostname

```
source /opt/ros/kinetic/setup.bash
alias goros='source devel/setup.sh'
export ROS_HOSTNAME=example.hostname
export TURTLEBOT_3D_SENSOR=astra
export TURTLEBOT_3D_SENSOR2=sr300
export TURTLEBOT_BATTERY=None
export TURTLEBOT_STACKS=interbotix
export TURTLEBOT_ARM=pincher
```

dialout permission

```
sudo usermod -a -G dialout turtlebot
```

Thats all now try to connect to the robot network and launch one of the demos.

Turtlebot2i troubleshooting

Background

Each Turtlebot2i is connected, configured, and paired to one of the lab computer. Both are connected the lab's WiFi but not connected to the internet.

Update the Turtlebot2i repository.

Robot

There are two ways of accomplishing this:

- 1) Via internet using git
 - a. Remove the HDMI adapter from the Joule then connect the robot to a monitor using the lab's HDMI to VGA adapter and the wireless keyboard
 - b. Turn on base and the Joule, wait until it boots. It should auto logon but, if it doesn't, the password is 'robot1'
 - c. Change the WIFI to PSU Guest and open a browser to accept the terms
 - d. Once internet is obtained. Open a terminal
 - i. Rename or delete the existing turtlebot2i directory

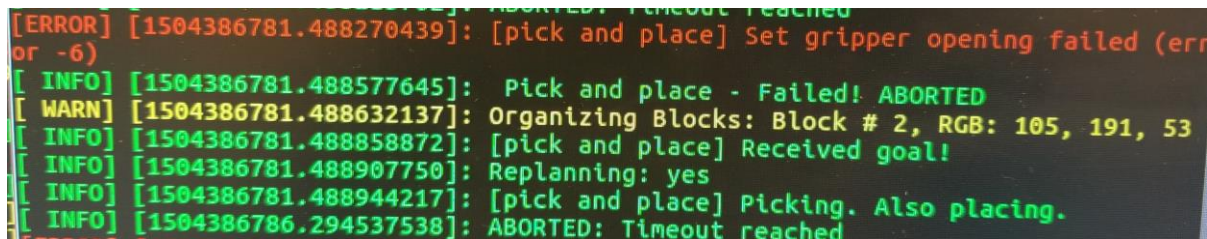
- ii. Create new dir >\$mkdir -p ~/turtlebot2i/src
 - iii. >\$cd ~/turtlebot2i/src
 - iv. Do a git clone:
 - 1. >\$git clone https://github.com/Interbotix/turtlebot2i.git .
 - 2. >\$git clone https://github.com/Interbotix/arbotix_ros.git -b turtlebot2i
 - 3. >\$git clone https://github.com/Interbotix/phantomx_pincher_arm.git
 - 4. >\$git clone https://github.com/Interbotix/ros_astra_camera -b filterlibrary
 - 5. >\$git clone https://github.com/Interbotix/ros_astra_launch
 - v. >\$ cd ~/turtlebot2i
 - vi. >\$ catkin_make
 - e. **IMPORTANT** Change the WIFI back to the lab's network (ROSnet or ROSnet-5G)
- 2) Via USB Drive
- a. Using a laptop with internet, clone all the repositories or get the zip file from the website
 - i. git clone https://github.com/Interbotix/turtlebot2i.git
 - ii. git clone https://github.com/Interbotix/arbotix_ros.git -b turtlebot2i
 - iii. git clone https://github.com/Interbotix/phantomx_pincher_arm.git
 - iv. git clone https://github.com/Interbotix/ros_astra_camera -b filterlibrary
 - v. git clone https://github.com/Interbotix/ros_astra_launch
 - b. Zip and create a tarball of all the files
 - c. Copy to a USB drive
 - d. Connect The usb drive to the
 - e. Connect the robot to a monitor

Charger will not charge the LiPo battery

The Li-Po battery needs to have more than 8.2 Volts of charge for the balance charging to work. The easiest and safest solution is to connect the battery to the base and charge the base for about 15 minutes.

Arm Block Demo does not work

The arm was flaky at best my success rate was close less than 30% on the first attempt. The most common error I ran into is the gripper not working (capture in image below). The cause of the problem was a low battery or the gripper servo failing.



```

[ERROR] [1504386781.488270439]: [pick and place] Set gripper opening failed (err
or -6)
[ INFO] [1504386781.488577645]: Pick and place - Failed! ABORTED
[ WARN] [1504386781.488632137]: Organizing Blocks: Block # 2, RGB: 105, 191, 53
[ INFO] [1504386781.488858872]: [pick and place] Received goal!
[ INFO] [1504386781.488907750]: Replanning: yes
[ INFO] [1504386781.488944217]: [pick and place] Picking. Also placing.
[ INFO] [1504386786.294537538]: ABORTED: Timeout reached
  
```

Figure 1 Arm Gripper error example

The easiest way to know if the battery is too low with the robot still powered on is by connecting the robot to the power supply. If the status light is orange, charge the robot until the status light is solid green (could take up to two hours).

1. Make sure battery is charged
2. In a new terminal type the command `>$ rviz rviz`
3. File> Open Config
4. Navigate to `~/turtlebot2i/src/turtlebot2i/turtlebot2i_bringup/rviz/turtlebot2i.rviz`
5. From the Planning Request/ Planning Group select `pincher_gripper`



-
- Motion Planning
- Context Planning Manipulation Scene Objects Stored Scenes Stored States S
- Commands
- Plan
- Execute
- Plan and Execute
- Stop
- Failed
- Query
- Select Start State:
- <current>
- Update
- Select Goal State:
- Options
- Planning Time (s): 5.00
- Planning Attempts: 10.0
- Velocity Scaling: 1.00
- Acceleration Scaling: 1.00
- ☐ Allow Repanning

- Under Query click Select Goal State
- Select <Random Valid>
- Click Update

12. Then click Plan and Execute
13. Check if the result is Success or Failed

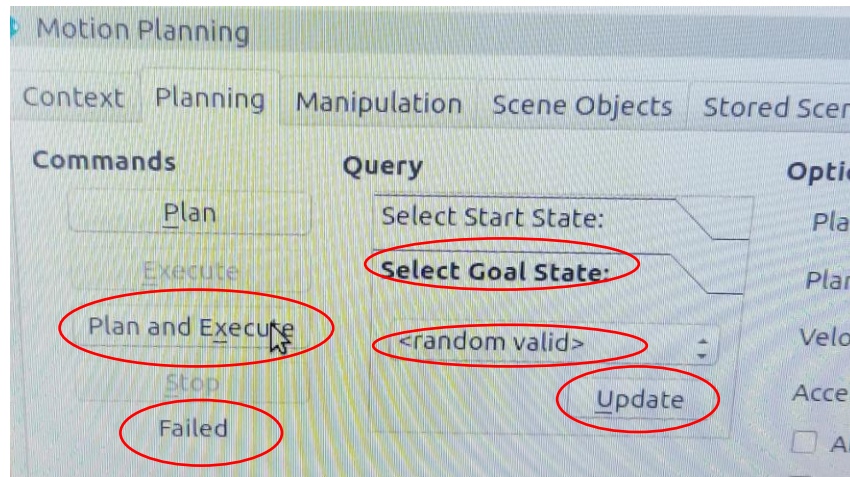


Figure 4 Steps 8 to 12

RobotModel not shown in Rviz

This happens often and I do not have a perfect solution for it but the following works in most cases.

1. Go to File> Open Config
2. Navigate to `~/turtlebot2i/src/turtlebot2i/turtlebot2i_bringup/rviz/turtlebot2i.rviz`

Moving the base to a location in the map.

How to identify a location

1. Have the robot running
2. Connect the robot
3. Open a terminal window (either workstation on robot)
4. `>rostopic echo /clicked_point`
5. `>$ cd turtlebot2i`
6. `>$goros`
7. `>$ rviz rviz`
8. In RVIZ on the top bar select the "add"

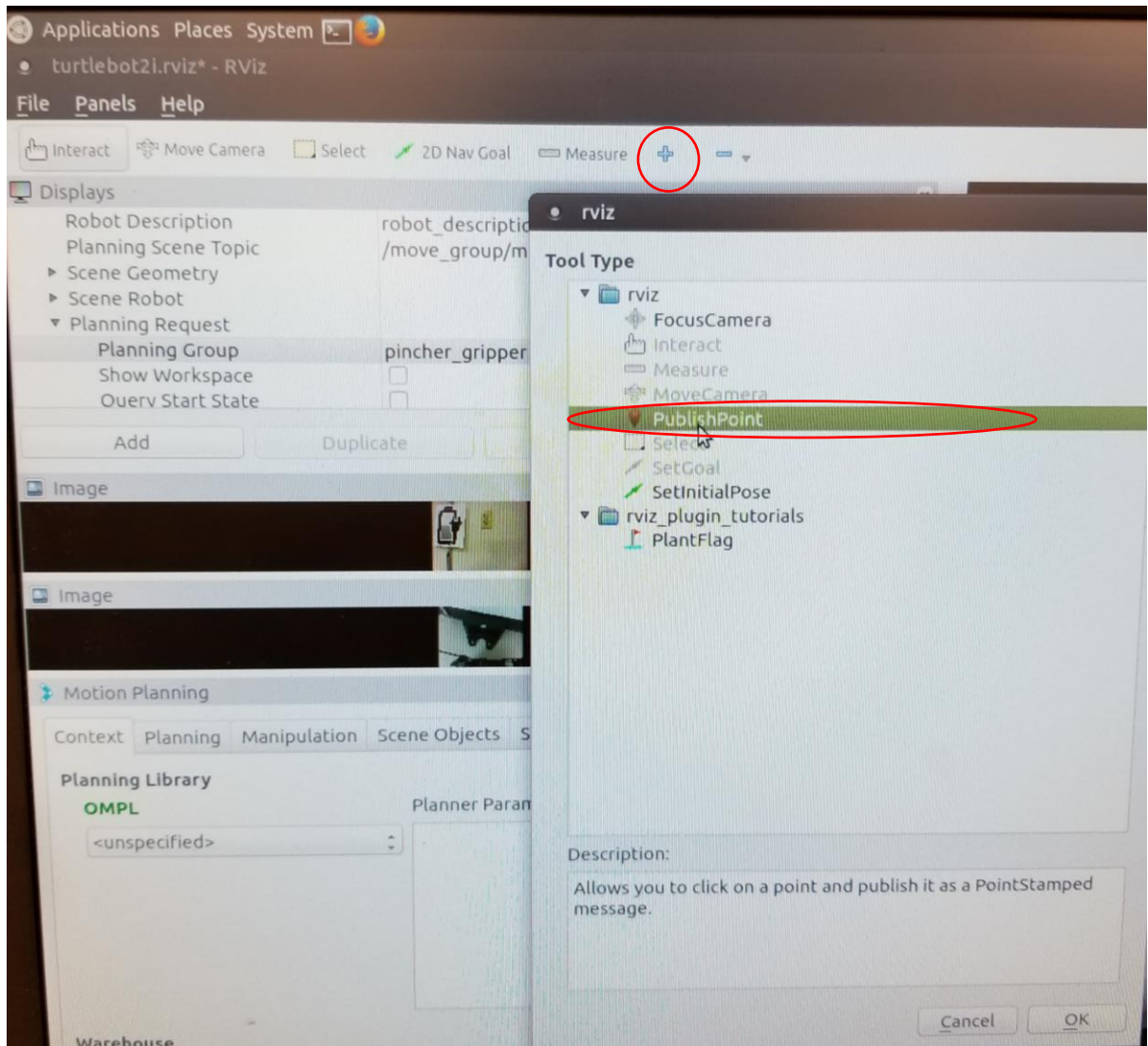


Figure 5 Steps 8 and 9 adding Publish point

9. Add Publish point
10. Using the new button click on it, then click somewhere on the map

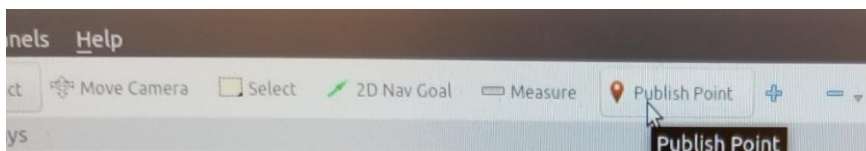


Figure 6 Selecting publish point

11. Go to the terminal window where you did the command `>$rostopic echo /clicked_point`
12. Record the value

Send location to turtlebot2i

1. Open a new terminal
2. `>$ cd turtlebot2i`
3. `>$goros`

4. `>$rostopic pub /move_base_simple/goal geometry_msgs/PoseStamped '{header: {stamp: now, frame_id: "map"}, pose: {position: {x: 1.0, y: 0.0, z: 0.0}, orientation: {w: 1.0}}}'`
5. Base should move; if it doesn't, make sure keyboard_teleop is not running