Navigating Turtlebot2 with a PhantomiX Pincher robotic arm

Mohamed ALI
Danie Jianah SONIZARA

centre universitaire Condorcet

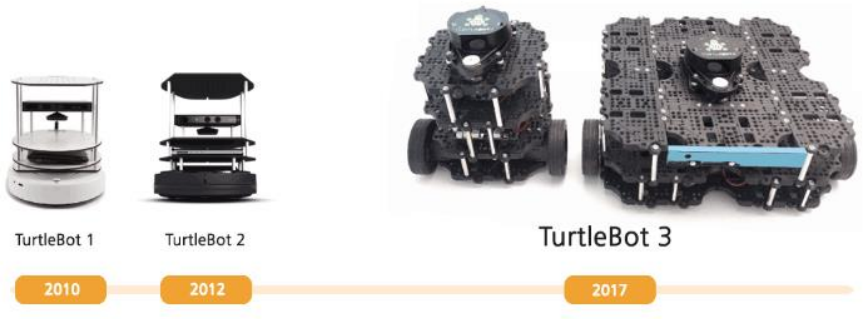
UNIVERSITÉ DE BOURGOGNE

Contents

I. Overview of the TurtleBot2

1. What is a TurtleBot2

A TurtleBot is a low-cost, personal robot kit with open-source software, which was created at Willow Garage (the company that supports ROS) by Melonee Wise and Tully Foote in November 2010. With TurtleBot, you'll be able to build a robot that can drive around your house, see in 3D, and have enough horsepower to create exciting applications.



The TurtleBot 2 is the brand new version of the TurtleBot proposed by Willow Garage and ClearPath Robotics. If the version 1 relied on a mobile technology that was more than 10 years old, the new TurtleBot 2 robot is based on Kobuki robot from Yujin Robot, a Korean company. This mobile base has been designed to provide a modern, up-to-date platform and everything has been redesigned on TurtleBot 2 to provide a much more powerful robot.

2. Robot description

TurtleBot2 consists of:

➔ Basic components

- an Yujin Kobuki mobile base with USB cable to control it from a PC,
- a 2200 mAh battery pack which provides 3 hours of operational time,
- a Kinect sensor with cable to power the Kinect sensor directly from the robot's battery,
- an Asus F201E – KX066DU laptop,
- fast charger,
- and a hardware mounting kit attaching everything together and adding future sensors. Assembling the kit is quick and easy using a single allen wrench (included in the kit).

➔ Additional component

- The Turtlebot2 is intended to use a 3D sensor in order to navigate autonomously and map.
- By default, the Kinect sensor is provided with the robot, but any other sensor can be attached.
- Docking station : battery charger intended to carry out an autonomous battery charge
- The 4S2P battery 2200 mA/h. It provides 7 hours of operational time.

3. Specifications

Maximum translational velocity	65 cm/s
Maximum rotational velocity	3.14 rad/s
Payload	5 kg (hard floor) 4 kg (carpet)
Threshold Climbing	< or =12 mm
Rug Climbing	< or =12 mm
Expected Operating Time	3/7 hours (small/large battery)
Expected Charging Time	1.5/2.6 hours (small/large battery)
Bumpers	*3 (left , center , right)
Cliff sensor	*1
Wheel drop sensor	*1 (one per wheel)

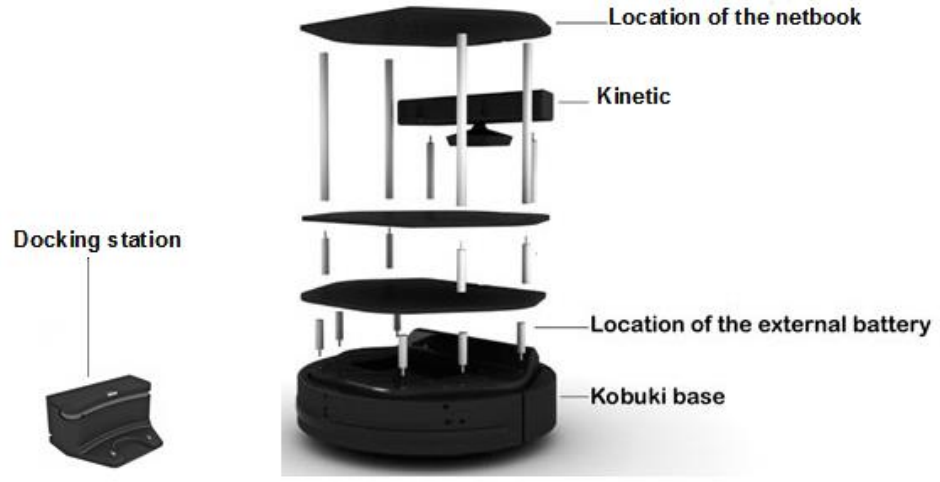


Figure 1: Components of the Turtlebot2

4. Software

We will use ROS indigo under Ubuntu 14.04 to work with and to program the Turtlebot2. If you need more information about these software click on the link below:

- ROS Indigo : http://wiki.ros.org/indigo
- Ubuntu: https://www.ubuntu.com

II. Starting with ROS

Once ROS Indigo is installed under Ubuntu, it is necessary to do the checking below to avoid errors before using the Turtlebot.

1. Terminal commands

Ctrl + Alt +T: To open a terminal

Ctrl + Shift + T: To open a new terminal in a new tab

Ctrl + L: To clear the terminal

Ctrl + C: To stop the running code, type Ctrl-C in the same terminal window.

2. Updates

The first thing to do is to update the packages already installed on your Workstation.

```
$ sudo apt-get update
```

3. The "~/.bashrc" file

Before using a real robot, make sure that your robot is connecting to your Workstation. To do that you have to modify the "~/.bashrc" file of the Workstation to get the ROSMASTER working on the Turtlebot2 Netbook and to be able to launch some ROS commands from the Workstation.

```
$ gedit .bashrc
```

Edit the following highlighted code inside your "~/.bashrc" file. The numbers represent the IP addresses of your Turtlebot and your Workstation

```
#ROS INDIGO

ROS_DISTRO=indigo

source /opt/ros/indigo/setup.bash

#catkin

source ~/ros/indigo/catkin_ws/devel/setup.bash

export ROS_MASTER_URI=http://192.168.0.100:11311 # configuration for turtlebot X / CO-P-ROBOTXX

#check the value of the local IP assigned by the TurtleBot router.

export TURTLEBOT_IP=192.168.0.100
```

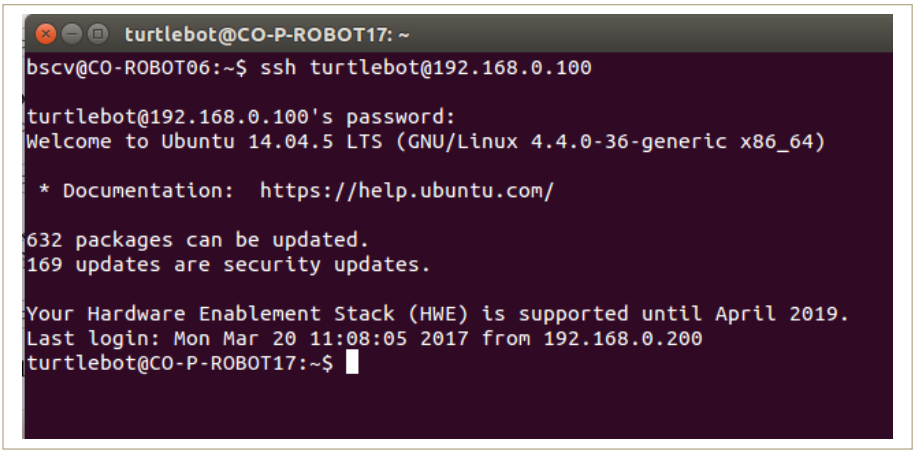
- Every time after modifying the "~/.bashrc" file, save it then close all the running terminals then open a new terminal to start new manipulations
- Be careful, when you do simulation on fake turtlebot, you have to comment the highlighted code

4. Testing the Network using "ssh"

To test your setup on, write the following code on the Workstation in a new terminal.

```
$ ssh turtlebot@192.168.0.100
```

If the following code appears, it means that your Turtlebot is successfully connected to your Workstation



5. Download the packages

5.1. Rplidar-turtlebot2 package

First we download the Rplidar-turtlebot2 package inside the source folder (ros → indigo → catkin_ws → src) of our Workstation by cloning the link from the GitHub repository.

```
$ cd ros/indigo/catkin_ws/src git clone https://github.com/roboticslab-fr/rplidar-turtlebot2.git
```

Then we build and configure the new installed package by using catkin_make and rospack profile

```
$ cd ros/indigo/catkin_ws/catkin_make
```

```
$ rospack profile
```

Finally we setup rules for USB connection by moving to the script folder of rplidar_ros package.

```
$ roscd rplidar_ros/scripts
```

And make the scripts executable

```
$ chmod +x create_udev_rules.sh
```

Launch the script provided in the rplidar_ros package and it will automatically setup the rules.

```
$ roslaunch turtlebot_le2i rplidar_minimal.launch
```

5.1. The Arbotix packages

In our Workspace:

```
$ cd ros/indigo/catkin_ws/catkin_make
```

We install the arbotix package using the command:

```
$ sudo apt-get install ros-indigo-arbotix
```

Then we catkin_make again:

```
$ cd ros/indigo/catkin_ws/catkin_make
```

5.2. The turtlebot_arm packages

In our Workspace git clone the package from https://github.com/turtlebot/turtlebot_arm :

```
$ cd ros/indigo/catkin_ws/src git https://github.com/turtlebot/turtlebot_arm.git
```

Then we catkin_make again:

```
$ cd ros/indigo/catkin_ws/catkin_make
```

III. Part 1 - MOTION CONTROL

In this first part, we will use the *Twist* commands to move the robot from a starting point A to a point B (situated in a certain distance from A), then rotate the robot in 180 degrees and make it comes back to the starting point A , all the paths at the same speed.

1. Mobile base with Low-Level Programming by sending Twist message to a real Robot[1]

First POWER ON the TurtleBot then *ssh* into the Workstation and run this launch file (you can also use your own launch file):

```
$ roslaunch turtlebot_bringup minimal.launch
```

To start with, input small values for the linear and angular speeds to avoid the robot flying across the room.

In another ssh terminal, write the following code:

```
$ rostopic pub -r 10 /cmd_vel_mux/input/navi geometry_msgs/Twist '{linear: {x: 0, y: 0, z: 0}, angular: {x: 0, y:0 ,z: 0.4}}'
```

➔ Changing the z coordinate in the angular position as shown the code above make the robot rotate

If the value is positive, you rotate the robot to the left

If it is negative the robot will rotate in the right.

➔ The linear x coordinate will change the position of the TurtleBot. The more the value is high the more your robot will move fast

A positive value makes move the robot forward.

A negative value makes it move backward.

In this example the robot moves 0.1 m forward:

[1] Reference : ROS by example volume 1 by R. Patrick Goebel (7.6 Publishing Twist Messages from a ROS Node)

```
$ rostopic pub -r 10 /cmd_vel_mux/input/navi geometry_msgs/Twist '{linear: {x: 0.1, y: 0, z: 0}, angular: {x: 0, y:0 ,z: 0}}'
```

Notice that on the real robot we use the *cmd_vel_mux/input/navi* not only the *cmd_vel*

2. Mobile base with High-Level Programming

2.1. Timed Out-and-Back in the ArbotiX Simulator[2]

Before running the code on a real robot, we have to make sure that it works in the Arbotix Simulator on your Workstation. So close all the running terminal and open a new one.

- First launch in a terminal the fake Turtlebot

```
$ roslaunch rbx1_bringup fake_turtlebot.launch
```

- Then run *Rviz* to monitor the fake robot. *Rviz* will show your robot executing the maneuver You can also click the **Reset** button to clear any **Odometry** arrows from the previous section.

```
$ rosrun rviz rviz -d `rospack find rbx1_nav`/sim.rviz
```

- Finally, run the *timed_out_and_back.py* node:

```
$ rosrun rbx1_nav timed_out_and_back.py
```

The figure 2 showed what is observed on *Rviz*. The yellow arrows represent the position and orientation of the robot at various points along its trajectory as reported by the fake robot's internal odometry.

[2] Reference : ROS by example volume 1 by R. Patrick Goebel (7.8.1 Timed Out-and-Back in the ArbotiX Simulator)

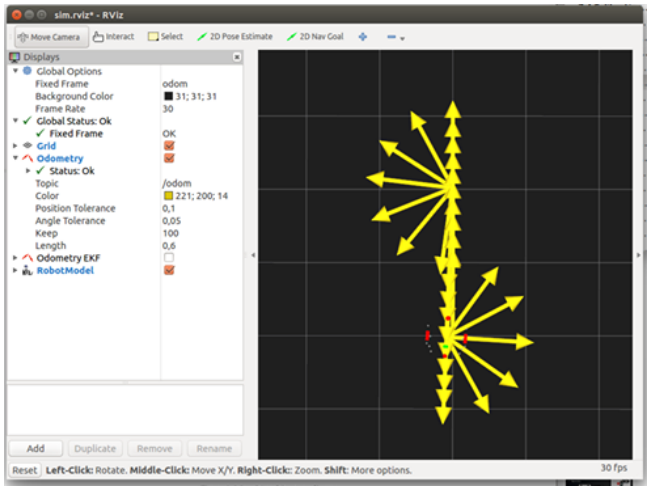


Figure 2: Time out-and-back simulation on fake Turtlebot

2.2. Timed Out-and-Back using a real robot [3]

To run the *timed_out_and_back.py* script on a real robot, first terminate any running simulation and make sure that you have a plenty of room to work in. Then, bring up your robot's startup launch file after ssh into your workstation.

- On the robot's laptop , in an ssh terminal , run :

```
$ roslaunch turtlebot_le2i remap_rpilidar_minimal.launch
```

- In another ssh terminal, we will also run the *odom_ekf.py* script so we can see the TurlteBot comb's combined odometry frame in *Rviz* :

```
$ roslaunch rbx1_bringup odom_ekf.launch
```

- Then run *Rviz* on your Workstation with the *nav_ekf* config file.

```
$ rosrun rviz rviz -d `rospack find rbx1_nav`/nav_ekf.rviz
```

- Finally, launch the odometry-based out-and-back script just like we did in simulation on the robot's laptop after logging in with ssh

```
$ rosrun rbx1_nav odom_out_and_back.py
```

The result is shown in the figure 2 on *Rviz*.

[3] Reference : ROS by example volume 1 by R. Patrick Goebel (7.8.2 Timed Out and Back using a Real Robot)

3. Navigating a Square using Twist + Odometry [4]

As we did with the odometry-based out-and-back script, we will monitor the position and orientation of the robot to make it move in a square by setting four waypoints, one at each corner. At the end of the run, we can see how close the robot gets back to the starting location and orientation. Let's start with a simulation and then try it on a real robot.

3.1. Navigating a Square in the ArbotiX Simulator

First run the *fake_turtlebot* launch file:

```
$ roslaunch rbx1_bringup fake_turtlebot.launch
```

Then run *rviz*

```
$ rosrun rviz rviz -d `rospack find rbx1_nav`/sim.rviz
```

Finally we run the *nav_square.py* script

```
$ rosrun rbx1_nav nav_square.py
```

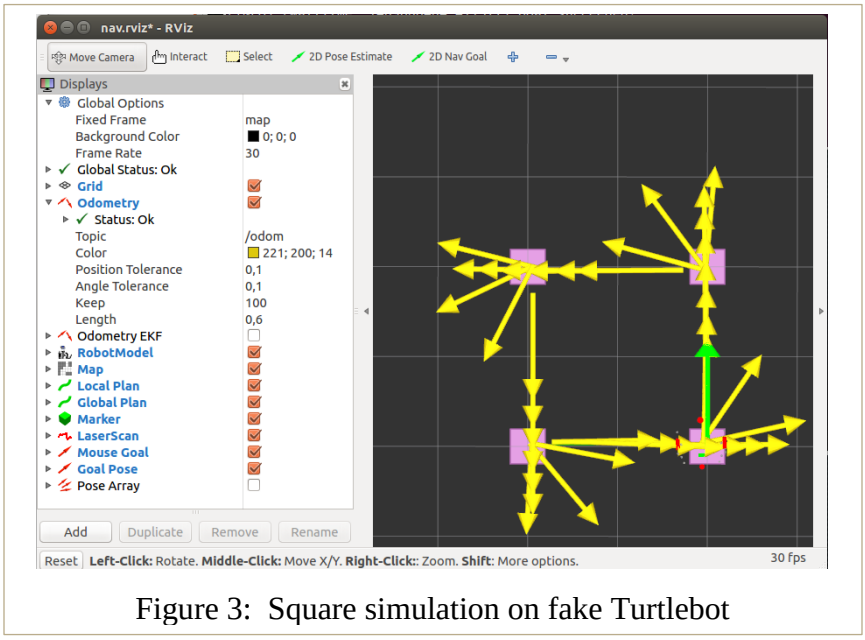


Figure 3: Square simulation on fake Turtlebot

[4] Reference : ROS by example volume 1 by R. Patrick Goebel (7.9 Navigating a Square using Odometry)

3.2. Navigating a Square using a Real Robot

First terminate any running simulated robots, then in a ssh terminal, launch the startup file for your robot. In this case, we will use a minimal launch file given by:

```
$ roslaunch turtlebot_le2i remap_rpilidar_minimal.launch
```

```
$ roslaunch rbx1_bringup odom_ekf.launch
```

Then run rviz in a normal terminal before running the nav.py file

```
$ rosrun rviz rviz -d `rospack find rbx1_nav`/nav_ekf.rviz
```

```
$ rosrun rbx1_nav nav_square.py
```

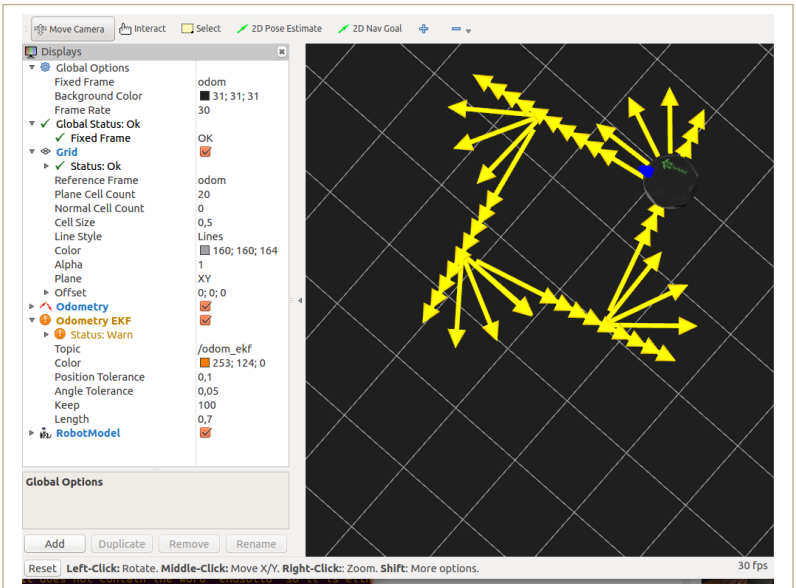


Figure 4: Navigating a square on the Turtlebot2

4. Navigation with Path Planning[5]

4.1. Testing move_base in the Arbotix simulator

Because the move_base node requires a map of the environment, we will use a blank square map to test the move_base which is included inside the *rbx1_nav* package. We will first launch the fake turtlebot.

[5] Ref: ROS By Example - Vol. 1 - Chapters 8.1…8.3 - Path Planning using move_base

```
$ roslaunch rbx1_bringup fake_turtlebot.launch
```

Then launch the blank map before running rviz to visualize the result.

```
$ roslaunch rbx1_nav fake_ move_base_blank_map.launch
```

```
$ rosrun rviz rviz -d `rospack find rbx1_nav`/nav.rviz
```

To test the robot we will move it 1.0 meter forward as we did before with the twist message then make the robot back to its initial point.

```
$ rostopic pub /move_base_simple/goal geometry_msgs/PoseStamped \ '{ header: { frame_id: "map" }, pose: { position: { x: 1.0, y: 0, z: 0 }, orientation: { x: 0, y: 0, z: 0, w: 1 } } }'
```

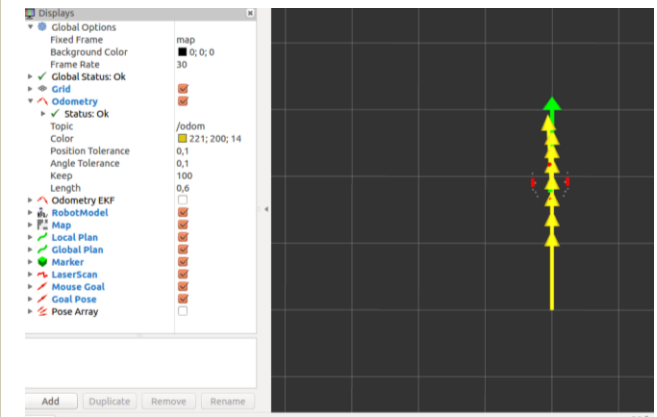


Figure 5: Testing move_base on fake Turtlebot

```
$ rostopic pub /move_base_simple/goal geometry_msgs/PoseStamped \ '{ header: { frame_id: "map" }, pose: { position: { x: 0, y: 0, z: 0 }, orientation: { x: 0, y: 0, z: 0, w: 1 } } }'
```

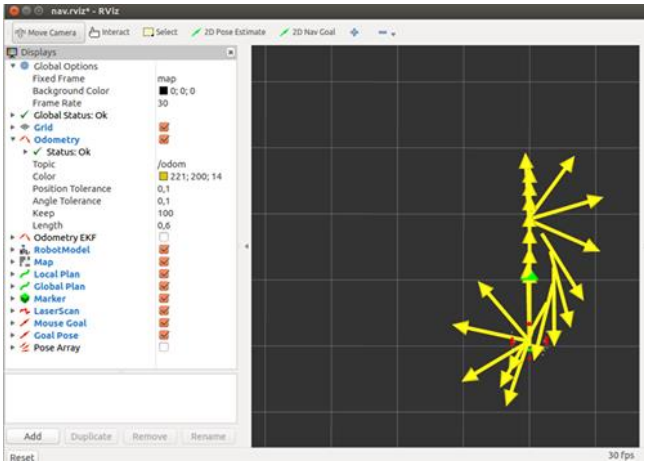


Figure 6: Testing move_base on fake Turtlebot

We can notice a thin green line which indicates the global path planned for the robot from the starting position to the goal. We can fully see that green line if we check the boxes beside the appropriate displays in the **Display** panel on the left of *RViz.*

To view the global and the local path more clearly, turn off the display for **Odometry, Goal pose** and **Mouse Pose** then re-run the *move_base* commands above.

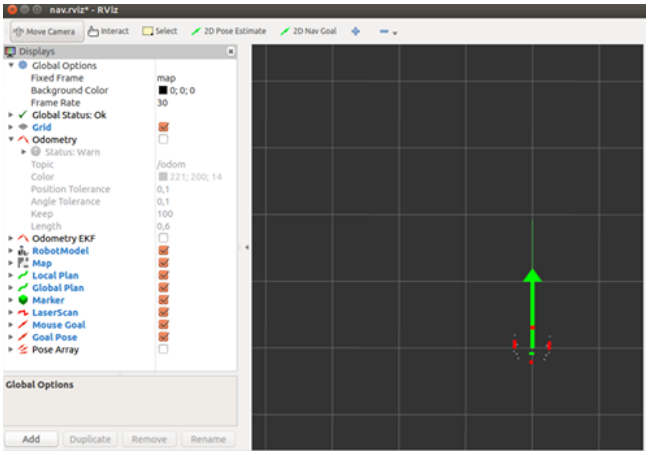
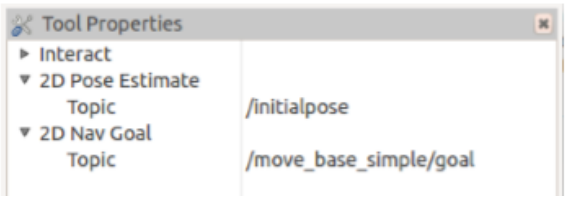


Figure 7: Testing move_base on fake Turtlebot

(RViz without Odometry)

4.2. Navigating using the mouse in *RViz*

We can also specify a goal pose using the mouse in RViz. If you launched RViz with the nav.rviz file as described above, you should be good to go. However, just to check, if you don't see the Tool Properties window on the right side of the RViz screen, click on the Panels menu and select Tool Properties. A pop-up window should appear that looks like this:



Under the 2D Nav Goal category, the topic should be listed as /move_base_simple/goal. If not, type that topic name into the field now.

If you had to make any changes, click on the **File** menu in *RViz* and choose **Save Config**. When you are finished, you can close the **Tool Properties** window by clicking the little x in the upper right corner. With these preliminaries out of the way, we can now use the mouse to move the robot. Click the **Reset** button to clear any leftover odometry arrows. Next, click on the **2D Nav Goal** button near the top of the RViz screen. Then click and hold the mouse somewhere on the grid where you'd like the robot to end up. If you move the mouse slightly while holding down the button, a big green arrow will appear. Rotate the arrow to indicate the goal orientation of the robot. Now release the mouse button and move_base should guide the robot to the goal.

4.3. Navigating a square using move_base on a fake turtlebot

We are now ready to move our robot in a square using move_base. To make sur we are starting with a clean slate, close all the running terminal and re-open a new one. Then launch the fake turtlebot file:

```
$ roslaunch rbx1_bringup fake_turtlebot.launch
```

In another terminal launch the blank map:

```
$ roslaunch rbx1_nav fake_move_base_blank_map.launch
```

Then make sure you have *RV*iz up with the *na.rviz* configuration file:

```
$ rosrun rviz rviz -d `rospack find rbx1_nav`/nav.rviz
```

Finally, run the move_base_square.py script:

```
$ rosrun rbx1_nav move_base_square.py
```

When the scripts completes, the view in *RViz* should look something like the following:

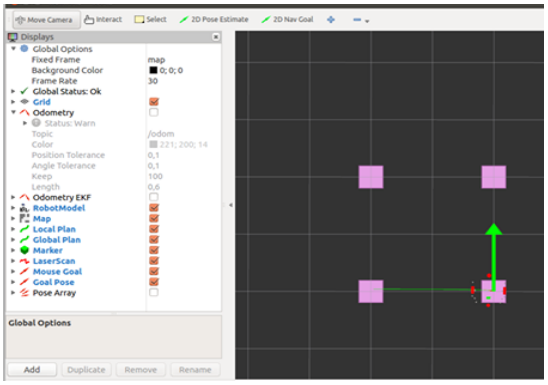


Figure 8-1: Navigating a square on a fake turtlebot (Odometry arrow off)

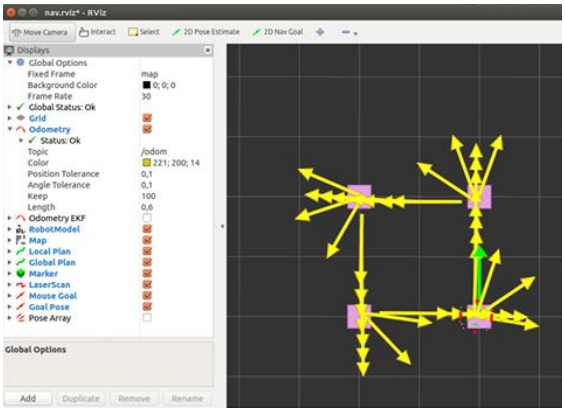


Figure 8-2: Navigating a square on a fake turtlebot (Odometry arrow on)

4.4. Avoiding simulated obstacles[6]

One of the more powerful features of move_base is the ability to avoid obstacles while still getting to the goal. To illustrate the process, will will load a new map with a pair of obstacles in the way of the robot. We will then run the move_base_square.py script again to see if the base local planner can find a path around the obstacles and still guide the robot to the four goal locations.

First terminate any running terminal. Then run the following commands:

```
$ roslaunch rbx1_bringup fake_turtlebot.launch
```

followed by:

```
$ rosparam delete /move_base
```

This command clears all existing move_base parameters which is less drastic than killing and restarting roscore.

Then we launch the new map with obstacles:

```
$ roslaunch rbx1_nav fake_move_base_map_with_obstacles.launch
```

And we run *RViz* with the *nav_obstacles.rviz* config file:

```
$ rosrun rviz rviz -d `rospack find rbx1_nav`/nav_obstacles.rviz
```

Finally run the *move_base_square.py* script:

[6] Ref: ROS By Example - Vol. 1 - Chapter 8.2.4 Avoiding Simulated Obstacles

```
$ rosrun rbx1_nav move_base_square.py
```

The view on *RViz* should look like something like this:

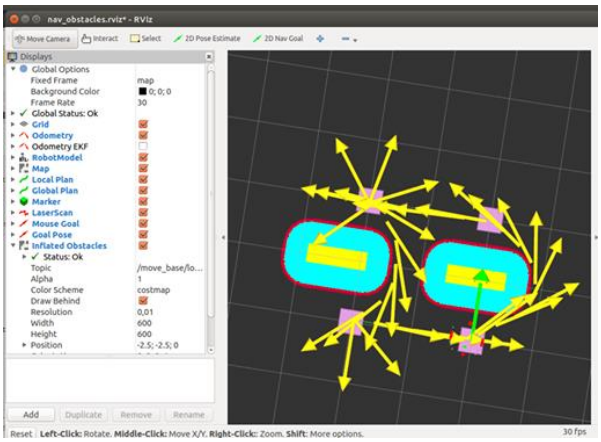


Figure 9: Avoiding simulated obstacles

4.5. Running move_base on a real robot

a. Testing move base without obstacles

After testing the move_base on a fake turtlebot, we are now ready to use our robot. Launch the minimal.launch from the turtlebot_le2i package in a ssh terminal:

```
$ roslaunch turtlebot_le2i remap_rpilidar_minimal.launch
```

In another ssh terminal run the *odom_ekf.py* script to be able to see the Turtlebot's combined odometry frame in RViz.

```
$ roslaunch rbx1_bringup odom_ekf.launch
```

Next, in an ssh terminal, launch the *move_base* node with a blank map different than the one we use in the simulation

```
$ roslaunch rbx1_nav tb_move_base_blank_map.launch
```

On your workstation run *RViz :*

```
$ rosrun rviz rviz -d `rospack find rbx1_nav`/nav.rviz
```

Finally, run the *move_base_square.py* script in an ssh terminal

```
$ rosrun rbx1_nav move_base_square.py
```

The following image shows the result of running a square without obstacles on the turtlebot.

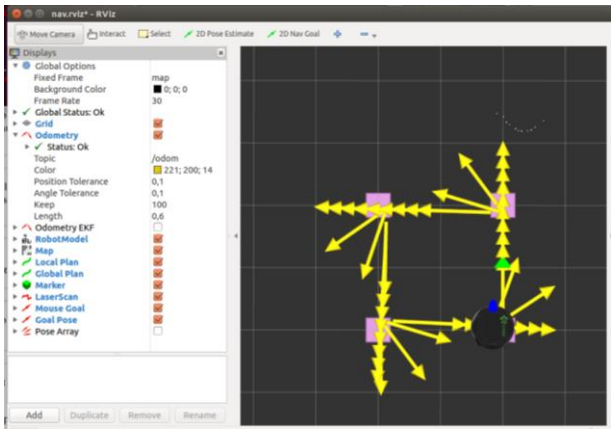


Figure 10: Running a square without obstacles on the turtlebot

b. *Testing move_ base avoiding obstacles*

First in a ssh terminal launch the following code:

```
$ roslaunch turtlebot_le2i remap_rpilidar_minimal.launch
```

As we did for the simulation, run the blank map on a ssh terminal:

```
$ roslaunch rbx1_nav fake_move_base_map_with_obstacles.launch
```

Then run rviz:

```
$ rosrun rviz rviz -d `rospack find rbx1_nav`/nav_obstacles.rviz
```

Finally, run the *move_base_square* script

```
$ rosrun rbx1_nav move_base_square.py
```

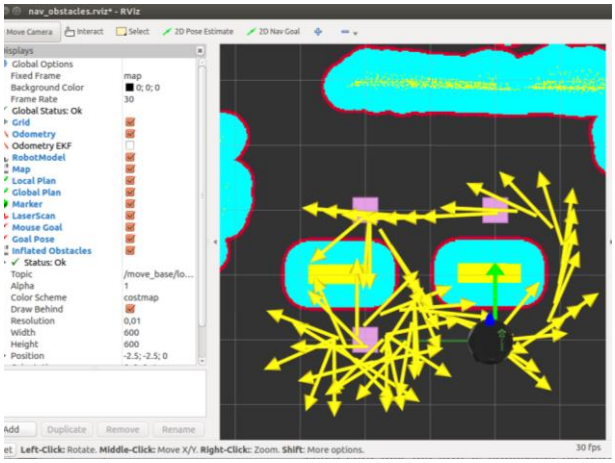


Figure 11: Running a square with obstacles on the turtlebot

IV. Part 2 – Planer Laser Rangefinder

1. Overview of the RPLIDAR A1

In this part we implement a laser on the turtlebot2. We will use the RPLIDAR A1 which is a low cost 360 degree 2D laser scanner (LIDAR) solution developed by SLAMTEC.

The system can perform 360degree scan within 6meter range. The produced 2D point cloud data can be used in mapping, localization and object/environment modeling.

RPLIDAR A1's scanning frequency reached 5.5 Hz when sampling 360 points each round. And it can be configured up to 10 Hz maximum.

RPLIDAR A1 can work excellently in all kinds of indoor environment and outdoor environment without sunlight.

RPLIDAR A1 contains a range scanner system and a motor system. After power on each sub-system, RPLIDAR A1 start rotating and scanning clockwise. User can get range scan data through the communication interface (Serial port/USB).

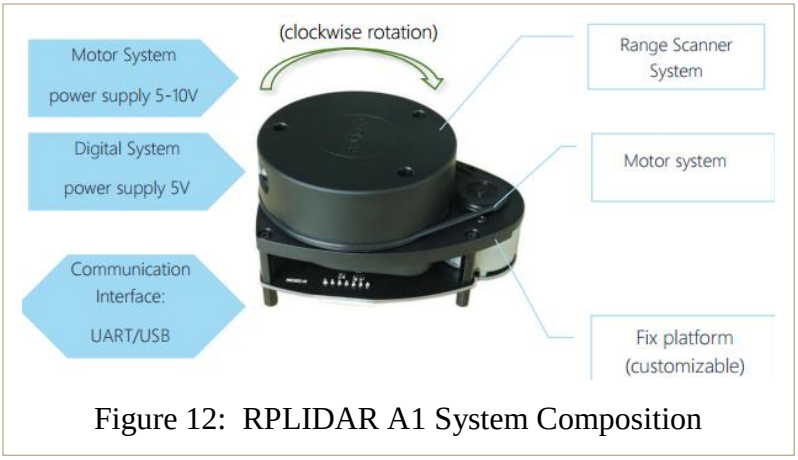


Figure 12: RPLIDAR A1 System Composition

In our case, the LIDAR was already installed on the turtlebot2. But if you want more information about the installation, please check the links below:

- https://github.com/roboticslab-fr/rplidar-turtlebot2
- http://wiki.ros.org/rplidar
- http://bucket.download.slamtec.com/e680b4e2d99c4349c019553820904f28c7e6ec32/LM108_SLAMTEC_rplidarkit_usermaunal_A1M8_v1.0_en.pdf
- http://bucket.download.slamtec.com/e9e096e9d9f30205d665260abe2cfb0c2dd62efa/LD108_SLAMTEC_rplidar_datasheet_A1M8_v1.0_en.pdf

2. Make the Lidar work

Once the rplidar-turtlebot2 package is correctly installed on the Workstation (check "Download the packages"), we can test the LIDAR by running the following launch file in an ssh terminal:

```
$ roslaunch turtlebot_le2i rplidar_minimal.launch
```

Then launch on the Workstation:

```
$ roslaunch turtlebot_le2i view_robot_rplidar.launch
```

V. Part 3 – Navigation and localization

Now that we understand how to use *move_base*, we can replace the blank map with a real map of the robot's environment. A map in ROS is simply a bitmap image representing an occupancy grid where white pixels indicate free space, black pixels represent obstacles, and gray pixels stand for "unknown".

In this part we will see how to draw a map using the rplidar.

1. Collecting and recording Scan data[7]

Log into the robot's laptop by ssh the terminal, run:

```
$ roslaunch turtlebot_le2i rplidar_minimal.launch
```

Then launch the *gmapping_demo.launch* file:

```
$  roslaunch rbx1_nav gmapping_demo.launch
```

Then bring up *Rviz* with the included gmapping configuration file:

```
$ rosrun rviz rviz -d `rospack find rbx1_nav`/gmapping.rviz
```

Next launch the teleop node for either the keyboard or joystick depending on your choice

```
$ roslaunch rbx1_nav keyboard_teleop.launch
```

Or

```
$  roslaunch turtlebot_teleop logitech.launch --screen
```

For more information about the joystick check the following link:

http://wiki.ros.org/turtlebot_teleop/Tutorials/indigo/Joystick%20Teleop

The final step is to start recording the data to a bag file. You can create the file anywhere you like, but there is a folder called bag_files in the rbx1_nav package for this purpose if you want to use it:

```
$  roscd rbx1_nav/bag_files
```

Now start the recording process:

```
$  rosbag record -O moha2_map /scan /tf
```

where *moha2_map* can be any filename you like

2. Creating the map

When you are finished driving the robot, type *Ctrl-C* in the rosbag terminal window to stop the recording process. Then save the current map as follows:

```
$  roscd rbx1_nav/maps
```

[7] Ref: ROS By Example - Vol. 1 - Chapter 8.4 - Map Building using the gmapping Package

```
$  rosrun map_server map_saver -f moha2_map
```

where " *moha2_map* " can be any name you like. This will save the generated map into the current directory under the name you specified on the command line. If you look at the contents of the *rbx1_nav/maps* directory, you will see two new files: *moha2_map.pgm* which is the map image and *moha2_map.yaml* that describes the dimensions of the map. It is this latter file that you will point to in subsequent launch files when you want to use the map for navigation. To view the new map, you can use any image viewer program to bring up the *.pgm* file created above. For example, to use the Ubuntu eog viewer ("eye of Gnome") run the command:

```
$  roscd rbx1_nav/maps
$ eog moha2_map.pgm
```

You can zoom the map using your scroll wheel or the +/- buttons.

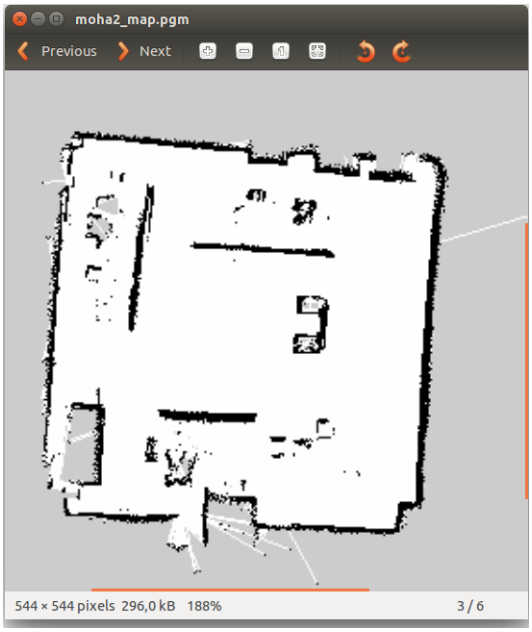


Figure 14: Example of a map

3. Navigation and Localization using a map[8]

First close any running terminal then launch:

```
$ roslaunch turtlebot_le2i view_robot_rplidar.launch
```

Now launch the tb_demo_amcl.launch file with your map as an argument:

[8] Ref: ROS By Example - Vol. 1 - Chapter 8.5 Navigation and Localization using a Map and amcl

```
$ roslaunch rbx1_nav tb_demo_amcl.launch map:= moha2_map yaml
```

Finally, bring up RViz with the included navigation test configuration file:

```
$ rosrun rviz rviz -d `rospack find rbx1_nav`/nav_test.rviz
```

Once we have our own map we can replace all the blank map from " Navigation with Path Planning" and make the robot move inside following all the steps.

VI. Part4 – Robotic arm

For the last part of the project, we will use the PhantomX Pincher Arm with the turtlebot2. To assemble the arm, refer to our technical survey about the Phantomix Pincher Robotic Arm.

1. Ros interfacing

The goal of this pat is to make our robotic arm move. To do so, we have to download the package an Arbotix package and a complete package of turtlebot arm, named turtlebot_arm that we can found on GitHub.

Then, we can check that the arm is effectively connected to the computer.
Connect the PhantomX Pincher arm through USB port to and make sure that you have read, write and execute access on the USP port using this command in a ssh terminal:

```
$ sudo chmod 777 /dev/ttyUSB0
```

Make sure that the arm is connected on port *ttyUSB0* to the computer. Then, we execute the following command:

```
$ arbotix_terminal
```

You should see the following:

```
ArbotiX Terminal --- Version 0.1
Copyright 2011 Vanadium Labs LLC
>>
```

We check that our servos are all active using the `ls` command

ArbotiX Terminal --- Version 0.1

Copyright 2011 Vanadium Labs LLC

>> ls

1 2 3 4 5 ....

....

This means that all the five servos are active and recognized.
But if after using `ls` command you have:

>> ls

....

....

Put again the `ls` command, sometimes it take time to run.

In the arbotix package directory inside the ***turtlebot_arm_bringup*** folder we change the ***yaml*** file to ***new_arm.yaml*** and we write the code below:

```
port: /dev/ttyUSB0
read_rate: 15
write_rate: 25
joints: {
arm_shoulder_pan_joint: {id: 1, neutral: 205, max_angle: 180, min_angle: -60, max_speed:
90},
arm_shoulder_lift_joint: {id: 2, max_angle: 150, min_angle: -150, max_speed: 90},
arm_elbow_flex_joint: {id: 3, max_angle: 150, min_angle: -150, max_speed: 90},
arm_wrist_flex_joint: {id: 4, max_angle: 100, min_angle: -100, max_speed: 90},
gripper_joint: {id: 5, max_speed: 90},
}
controllers: {
arm_controller: {type: follow_controller, joints: [arm_shoulder_pan_joint,
arm_shoulder_lift_joint,  arm_elbow_flex_joint,
arm_wrist_flex_joint], action_name: arm_controller/follow_joint_trajectory, onboard:
False}
}
```

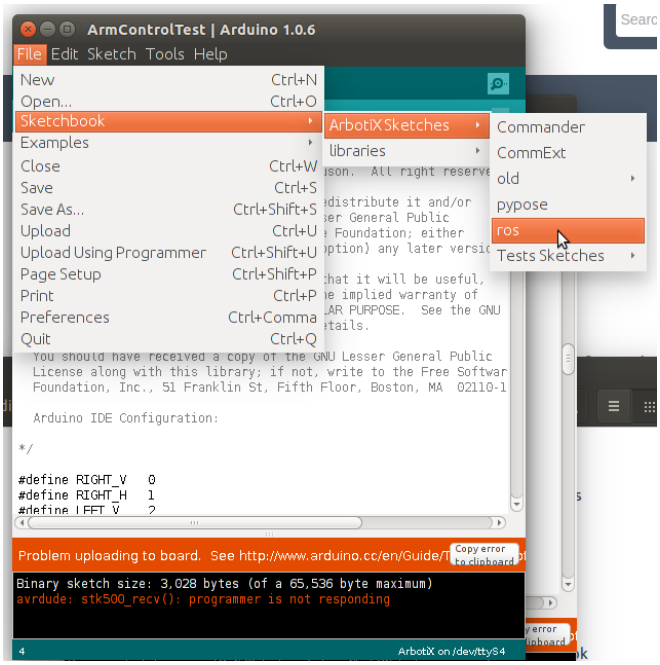
Inside the same folder, ***turtlebot_arm_bringup,*** we create a new launch file named ***new_arm.launch*** and we put the code below:

```
<launch>

<node name="arbotix" pkg="arbotix_python" type="arbotix_driver" output="screen">
<rosparam file="$(find turtlebot_arm_bringup)/config/turtlebot_arm.yaml"
command="load" />
</node>

</launch>
```

Make sure that ROS is running through Arduino at the same time. You can verify that your robotic arm work by clicking on the Tests Sketches.



2. Applications

To make the robotic arm and the turtlebot work at the same time launch the ***new_arm.launch*** launch file in an ssh terminal:

```
$ roslaunch turtlebot_arm_bringup new_arm.launch
```

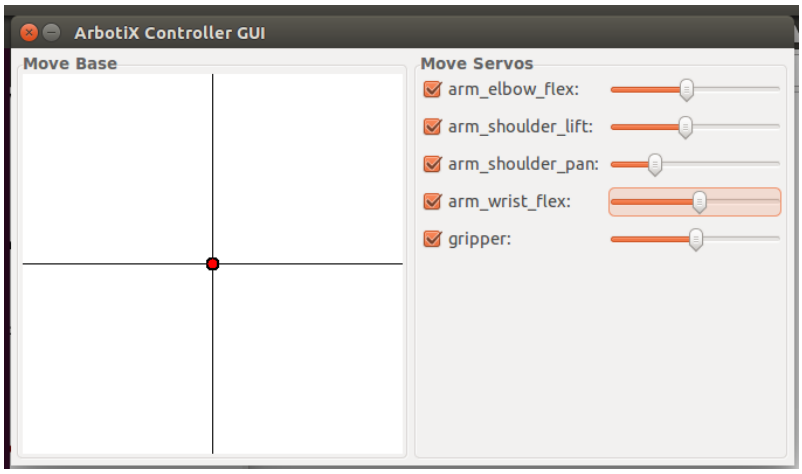
Then launch the minimal launch file for the turtlebot in another ssh termianal:

```
$ roslaunch turtlebot_le2i view_robot_rplidar.launch
```

And finally, in another terminal, run the ***gui-controller.***

```
$ arbotix_gui
```

Now you can play with your arm using the Servos move (check all the boxes) and use the red dot to move the turtlebot.



VII. References

- http://www.turtlebot.com/
- https://github.com/turtlebot/turtlebot_arm
- https://github.com/roboticslab-fr/rplidar-turtlebot2
- http://www.slamtec.com/en/Lidar/A1
- https://github.com/robopeak/rplidar_ros/wiki