Pogo Report

Iván López Broceño

December 2021

Contents

Remy's repo: https://gitlab.u-angers.fr/remyguyonneau/pogo

My repo with this documentation and some programs: https://github.com/ivrolan/my_pogo

1 Setting the robot

1.1 Configuring the Raspberry Pi

We will use Ubuntu Server 20.04 as the OS running in our Raspberry Pi.

Download an image of the OS and flash it in a bootable SD with the tool of your preference. I've used Balena Etcher but you can use other one, as the Raspberry Pi Imager.

Once the SD is flashed with our system, we can insert it in the Rpi and configure the system from an external monitor and keyboard. I had to insert this line to the `config.txt` to make it work with the monitor in the room 214.

```
hdmi_force_hotplug=1
```

1.1.1 Credentials

```
user: ubuntu
passwd: tubutubu
```

1.2 Wifi communication

We have to set up also the wifi to let us connect to the robot even when it is moving (not attached to the router via Ethernet cable). We'll follow the documentation of Remy Guyonneau: https://gitlab.u-angers.fr/remyguyonneau/pogo/-/blob/master/documentation/main.pdf - "Activation wifi - par point de access"

This should be already done and working.

After that, check the communication via ssh typing in your PC **connected to the same wifi that the robot**

I recommend setting a variable environment in your `./bashrc` on your PC.

```
export POGO_IP=192.168.1.13
```

Test the ssh connection:

```
ssh ubuntu@$POGO_IP
```

And type the password1.1.1

1.3 Installing and setting ROS environment

1.3.1 Installing ROS

As we are on Ubuntu 20.04 we are going to use ROS noetic, precisely the `ros-base` pkg.

Installation tutorial ROS Noetic on Ubuntu

Remember to create a catkin workspace creating a directory with a `src/` inside and compile it.

The SD already configured has the `pogo_ws`

1.3.2 Enabling the communication between the robot and your computer with ROS

We have to set more environment variables. To communicate both machines, they have to be connected to the same `roscore` and we have to "say" to our machines where is this `roscore` setting the environment variables `$ROS_MASTER_URI` and its own IP with `$ROS_IP`.

I've configured it as it follows:

In the Raspberry Pi - robot:

```
export ROS_MASTER_URI=http://192.168.1.13:11311/
export ROS_IP=192.168.1.13
```

In my own PC:

```
export ROS_MASTER_URI=http://192.168.1.13:11311/
export ROS_IP=$(ifconfig wlp2s0 | grep "inet " | awk '{print $2}')
# this allows us to get our own wifi IP from the wifi connection
# when we open a session in ssh
export POGO_IP=192.168.1.13
```

We can test this communication easily with:

In the robot:

```
$ roscore
```

In your PC:

```
$ rostopic list
/rosout
/rosout_agg
```

If an error pops out instead of getting the output of the topics, check that the `roscore` is running with the correct IP in the robot.

2 Support for the RPLIDAR sensor

It was designed in Solidworks. Files can be found in https://github.com/ivrolan/pogo-rplidar-case-files.git

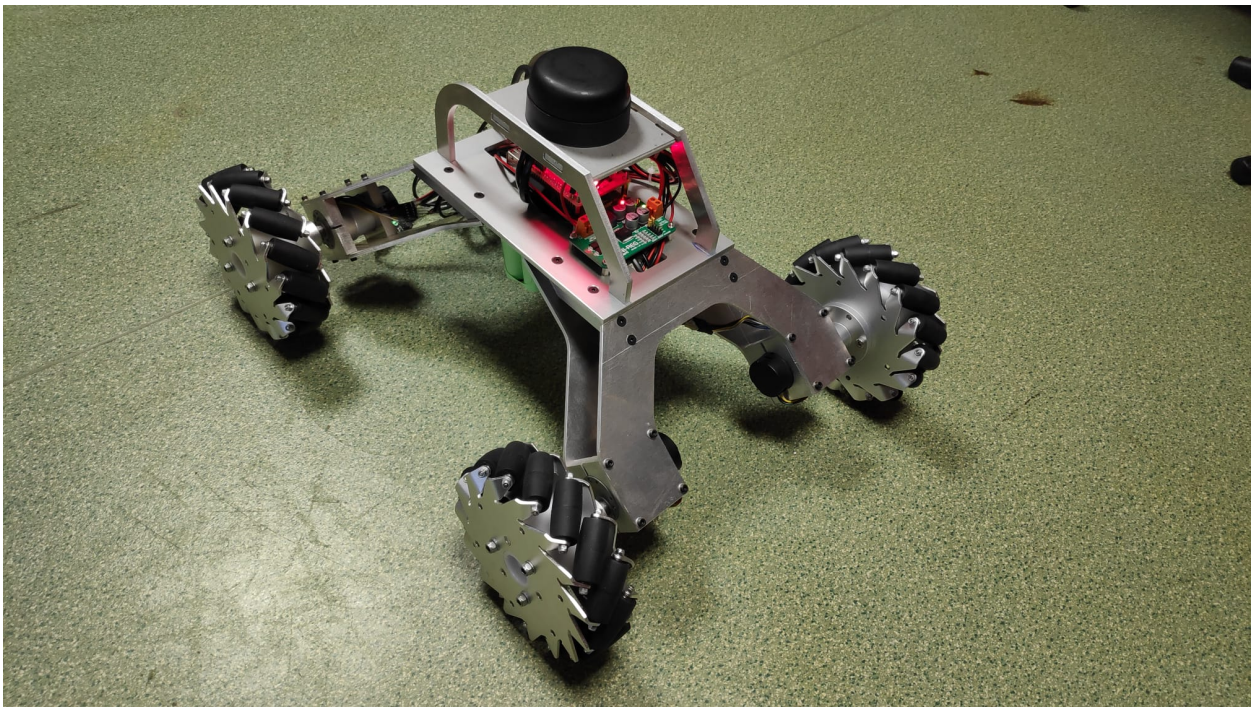

Figure 1: Full robot design.

3 ROS code

3.1 Movement

Nodes used to move the robot from the main repo of Remy and my repo (files marked with a *):

- To enable the motors and listen for jacobian velocities.
  - `pca9685_node`
  - `pid_node.py`

  They can be launched with `roslaunch pogo robot.launch`
- To transform the `/secured_cmd_vel` topic of geometry_msgs/Twist msgs to jacobian commands
  - `jacobian_node.py*`
- To control the velocities with the keyboard:
  - `my_control.py*`

  To increase/decrease the linear velocities use the key arrows. You can adjust the step of change in each tap changing the parameter inside the code. To increase/decrease the angular speed, press K or L. To stop the robot (put all velocities at 0) press the spacebar

In the robot:

```
$ roslaunch pogo robot.launch
```

In your PC:

```
$ rosrun pogo jacobian_node.py
$ rosrun pogo my_control.py
```

3.2 Sensing with the RPLIDAR

The LIDAR sensor mounted in top of the robot is a RPLIDAR A2. The code to make it work can be found at Slamtec github rplidar_ros

Clone the repo in the `src/` directory of your catkin workspace:

```
$ git clone https://github.com/Slamtec/rplidar_ros
```

WARNING !!!

Testing this package in the last version gave problems with the sensing. The system could receive data, but the max_range was 0, and no points of the laser were displayed.

This was due to the SDK update to the version 2.0. I have posted an issue on the repo, so in the future check that thread to see if the bug was solved. Until then, go back to the version 1.9 with a git checkout to a previous state.

```
$ git checkout 36684a01dcee58ada142b9199c04158b1439a84e
```

`36684a01dcee58ada142b9199c04158b1439a84e` is the hash of the commit to upgrade to the v1.9 of the SDK. You can check that with the output of

```
$ git log
commit 36684a01dcee58ada142b9199c04158b1439a84e (HEAD)
Author: kint <huasheng_zyh@163.com>
Date:   Fri Aug 24 01:04:21 2018 -0700

upgrade sdk 1.9.0
[new feature] support baudrate 57600 and 1382400, support HQ scan response
[bugfix] TCP channel doesn't work
[improvement] Print warning messages when deprecated APIs are called; imporve angular accuracy for u
```

Once you have the package compiled in the right version you can test it on the robot (as it will take data from the USB it has to be executed on the robot):

```
$ roslaunch rplidar_ros rplidar.launch
```

And then in your pc:

```
$ rviz
```

And add the LaserPoint topic visualizer and set properly the frame to see the points published in the topic `/scan`. Something like this

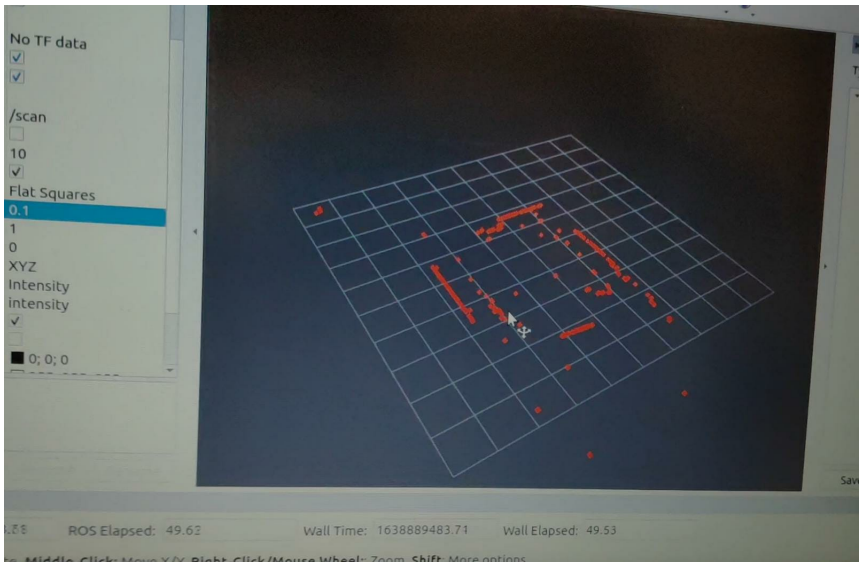

Figure 2: Example of Lidar visualization

3.3 SLAM with hector_slam package

Now that we have the lidar sensing working we can use it to create a map of the surroundings of the robot at the same time that we locate the robot in that map. Additionally to the lidar data, we also need the relation of positions between the proyection of the center of the robot in the floor and the lidar sensor.

3.3.1 Setting up the TF of the robot

The transforms (TF) state the relation between coordinate frames. All of those TF have to link the frames creating a tree (each frame has only one parentframe).

In this case of use, the tree of transforms would be like:

$map \rightarrow base\_link \rightarrow laser$

To do that, a `tf_broadcaster` is created, that links `base_link` with the `laser` frame. This program can be found at https://github.com/ivrolan/pogo_setup_tf.git

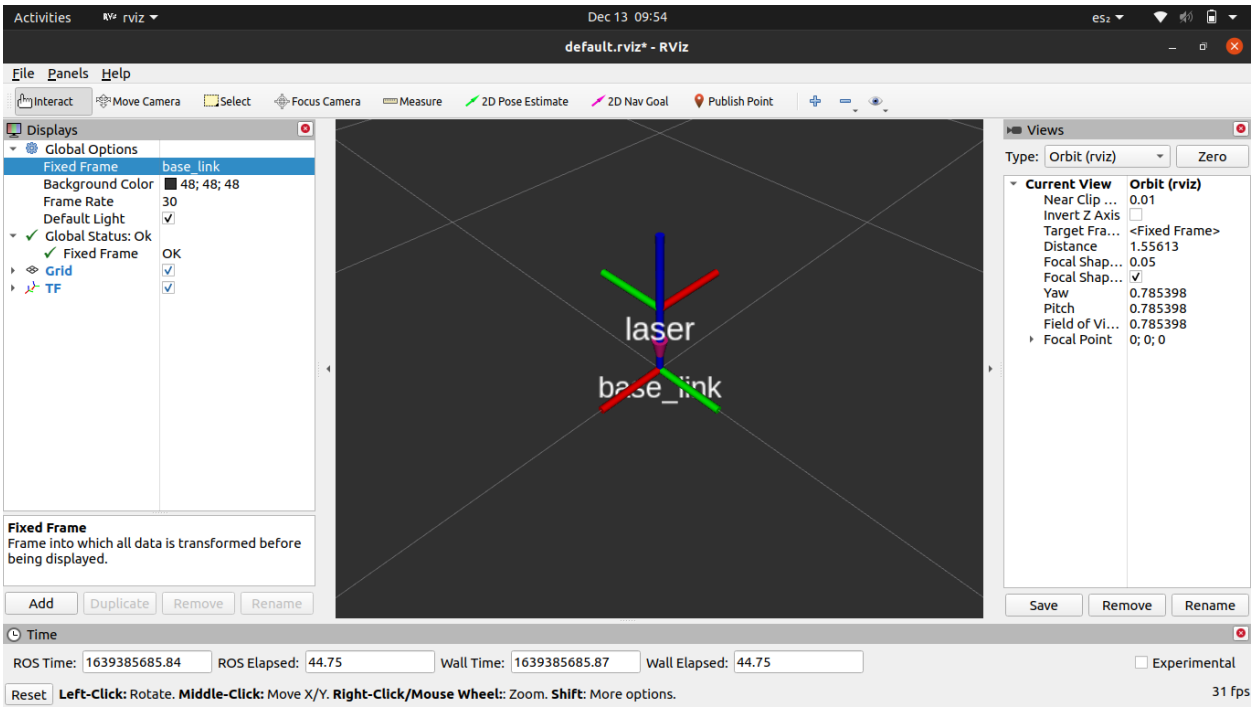

Figure 3: How the tf should be linked thanks to tf_broadcaster

3.3.2 The hector_slam package

The repo can be found at https://github.com/tu-darmstadt-ros-pkg/hector_slam.

Changes:

- In `hector_slam_launch/launch/tutorial.launch` change the line

```
<param name="/use_sim_time" value="true"/>
```

by this one

```
<param name="/use_sim_time" value="false"/>
```

- Also in `hector_mapping/launch/mapping_default.launch` we have to change:

```
<arg name="tf_map_scanmatch_transform_frame_name" default="scanmatcher_frame"/>
<arg name="base_frame" default="base_footprint"/>
<arg name="odom_frame" default="nav"/>
```

By:

```
<arg name="tf_map_scanmatch_transform_frame_name" default="laser"/>
<arg name="base_frame" default="base_link"/>
```

```
    <!-- If we don't want to use odom we have to change it name to the one in "base_frame"-
->
    <arg name="odom_frame" default="base_link"/>
```

Given all of that well configured we can test it like:

- Launching the control of the movement with:
  - ROBOT: `roslaunch robot robot.launch`
  - PC: `rosrun pogo jacobian_node.py`
  - PC: `rosrun pogo my_control.py`
- Launching the rplidar sensor
  - ROBOT: `roslaunch rplidar_ros rplidar.launch`
- Running the hector slam
  - PC: `roslaunch hector_slam_launch tutorial.launch`

And we can move the robot frame around a map created on the fly in rviz. For more information about hector slam see https://wiki.ros.org/hector_slam.

3.4 Navigation

3.4.1 With move_base

Now the goal is to use this map and localization provided by `hector_slam` to navigate. We could use move_base but I didn't succed to make it work with the map provided in real time by `hector_slam`, `move_base` was always using its own plugins to generate the maps, although the path created by its planner if a goal was selected in rviz and the commands to follow it seemed correct.

3.4.2 With hector_navigation

We tried also the hector_navigation pkg

Change the `hector_navigation/hector_exploration_node/config/costmap.yaml` file to:

```
    global_costmap:

  map_type: costmap
  track_unknown_space: true
  unknown_cost_value: 255
  obstacle_range: 2.5
  raytrace_range: 3.0
  footprint: [[0.2, 0.26],
              [0.2, -0.26],
              [-0.2, -0.26],
              [-0.2, 0.26]]
  inflation_radius: 0.2
  #transform_tolerance: 0.5
```

```
inscribed_radius: 0.3
circumscribed_radius: 0.32

global_frame: map
robot_base_frame: base_link
update_frequency: 0.5
publish_frequency: 0.1
static_map: true
rolling_window: false

#Investigate what this actually does
cost_scaling_factor: 10.0
```

Now you can run with the hector slam running:

```
$ roslaunch hector_exploration_node exploration_planner.launch
```

To get goal poses and paths to follow to explore the area. To follow this path we can use the `simple_exploration_controll` included in the `hector_navigation` pkg.

With all the previously mentioned running (except the `my_control.py` node):

```
$ rosrun hector_exploration_controller simple_exploration_controller
```

In my case, I didn't success using this node: the robot was kept turning in circles, without following the path given by the `hector_exploration_planner`.

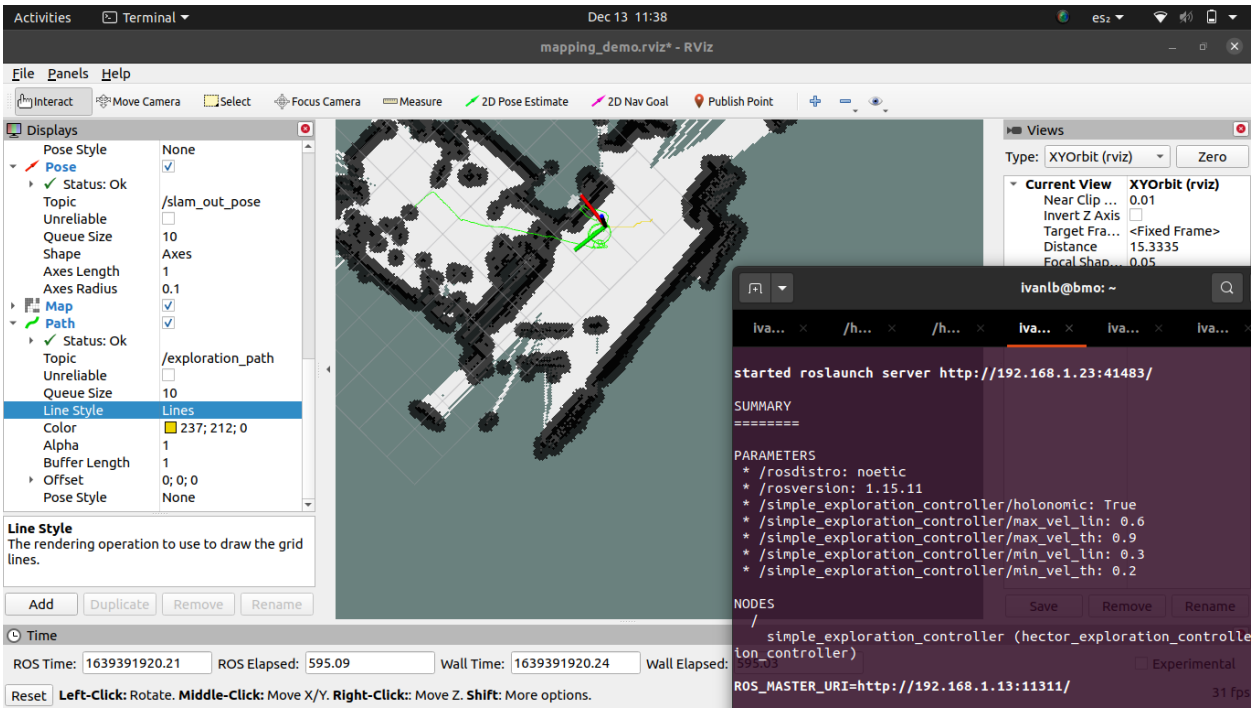

Figure 4: The robot doesn't follow the yellow line (the path given)

4 Further details - Contact information

This report was done by Iván López Broceño, student of Software Robotics Engineering at Universidad Rey Juan Carlos, Spain, during his Erasmus mobility at the Polytech Angers, France.

Do not hesitate to contact me if more information, details or help are needed. I am willing to help as much as I can.

- **Universidad Rey Juan Carlos email:** i.lopezb.2019@alumnos.urjc.es
- **University of Angers email:** ivan.lopezbroceno@etud.univ-angers.fr
- **Github:** https://github.com/ivrolan