Indoor real-time navigation for robot vehicles

Projet GSE5 2021-2022

Submitted by: SUN Yeting

YU Bicong

Tutor: Sébastien BILAVARN

Date : 23/01/2021

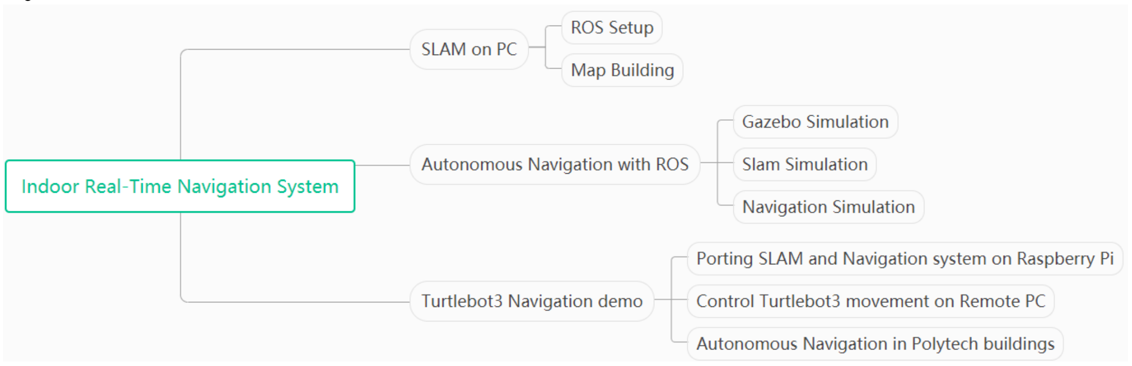
Introduction

TurtleBot3's core technology is SLAM, Navigation, and Manipulation, making it suitable for home service robots. TurtleBot Burger can run SLAM(simultaneous localization and mapping) algorithms to build a map and can drive around our room. Also, it can be controlled remotely from a laptop. The goal of the project is to implement a real-time navigation system for a robot vehicle and to set up a complete demonstrator based on an existing robot (Turlebot 3 Burger) operating under the ROS framework (Robot Operating System).

Specification

We implement the navigation system on a laptop running Ubuntu / ROS. The navigation system is ported to the main robot-embedded computer. A portable autonomous laser scanner "Lidar" is used to generate a map of the indoor navigation environment (Polytech buildings). We implement autonomous navigation with ROS. Finally, we implement real-time autonomous navigation in Polytech buildings.

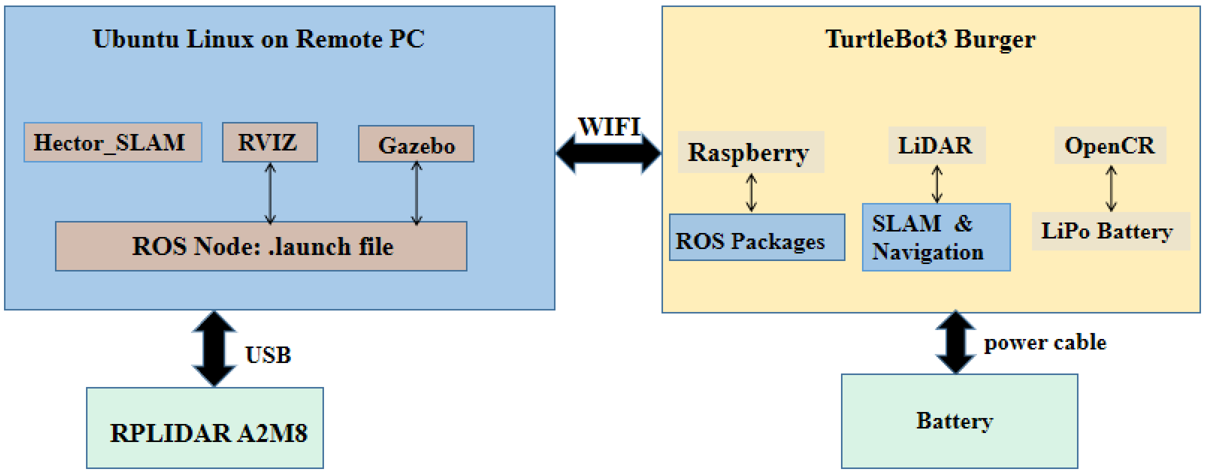
System overview


Indoor Real-Time Navigation System
SLAM on PC
ROS Setup
Map Building
Autonomous Navigation with ROS
Gazebo Simulation
Slam Simulation
Navigation Simulation
Turtlebot3 Navigation demo
Porting SLAM and Navigation system on Raspberry Pi
Control Turtlebot3 movement on Remote PC
Autonomous Navigation in Polytech buildings

High-Level Development

This project can be broken down into two main parts – software and hardware, each of which was used in order to enable both SLAM and autonomous navigation. The hardware used in this project is Lidar and Turtlebot3. We test everything on the Ubuntu 18.04 system. The software we use is RVIZ and Gazebo, which perform operations in the ROS workspace.

The purpose of this section is to design the overall system architecture and then discuss the hardware and software design and how they both contribute to SLAM, as well as autonomous navigation.



The ROS function package of TurtleBot3 includes “turtlebot3”, a collection of message files “turtlebot3_msgs”, a simulation function package “turtlebot3 _simulations”. Among them, the “turtlebot3” function package includes the TurtleBot3 robot model, SLAM and navigation function package, remote control “teleop_key” function package, and driving-related “bring up” function package.

Software design

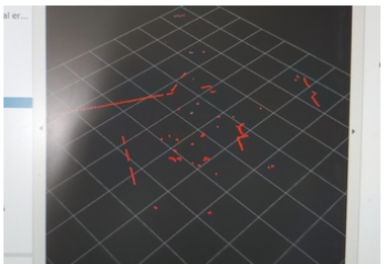
1. SLAM on PC

First, we need to set up and install ROS. We installed ROS(version Melodic). Then installed the package needed(tutlebot3, etc). After compiling all function packages (catkin_make_isolated), test them with instructions such as “roscore”.

Secondly, we want to get ROS / RP Lidar demo. In the LiDAR part, to use Lidar, we installed the package “rplidar_ros”. Then configure the .bashrc file, we add the following commands at the end of the file:

```
source /opt/ros/melodic/setup.bash
source ~/catkin_ws/devel/setup.bash
```

Then we connect the Lidar and PC with a USB cable. To detect the lidar connect normally and display the connected port, we use: ls -l /dev | grep ttyUSB. We change the authority 666 so that all users can read and write: sudo chmod 666 /dev/ttyUSB0. And we launched the node: roslaunch rplidar_ros rplidar.launch. After launching, the lidar starts to scan the environment.

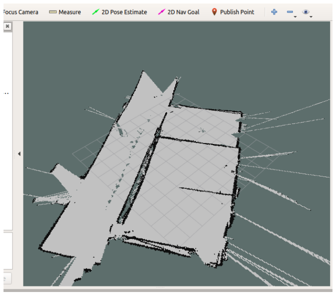


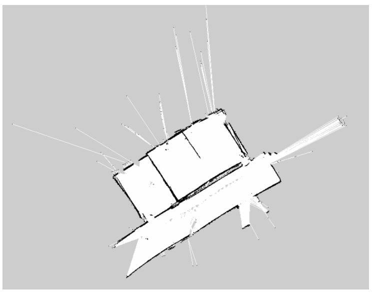
To see the topic list, we use: rostopic list, and see Lidar's data by using rostopic echo /scan. After seeing the data of lidar in the terminal, we launch RVIZ by running the command: rviz. Then we change the Fixed Frame to laser and add LaserScan, set the LaserScan topic to /scan. So that we can see some scan points in the rviz windows. It means everything is working correctly.

2. Use RP Lidar to build a map of Polytech buildings

To build the map of Polytech, we use Hector_SLAM Package. After configuring the package Hector-SLAM. We use roslaunch rplidar_ros rplidar.launch to boot the lidar, and use roslaunch hector_slam_launch tutorial.launch to launch the nodes as well as RVIZ. Then we can move the lidar slowly, and observe the map.

To save the map generated, we use the map_sever in the package navigation. We use rosrun map_server map_saver -f my_map to save the map and name it "my_map". After many scans, we choose the best one. Here is the my_map.yaml we generated:



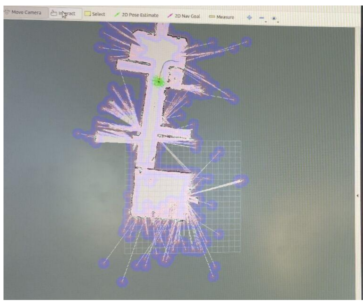


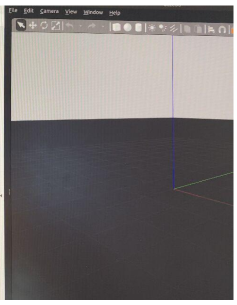
3. Autonomous navigation with ROS

In this part, we use the previously generated Polytech map to achieve simulated autonomous navigation. We opened a turtlebot3_empty_world by Gazebo, then opened the Polytech map we saved by using:

```
roslaunch turtlebot3_gazebo turtlebot3_empty_world.launch
roslaunch turtlebot3_navigation turtlebot_navigation.launch map_file:=$HOME/map.yaml
```

In the rviz window, we use 2D Nav Goal to set the destination and direction. Then the turtlebot3 in rviz will automatically navigate to the target location. It will avoid obstacles on the map and plan the best route. A demo: https://youtu.be/C8kooJ65Fe0





Hardware design

1. Build Turlebot 3 Burger

This part is teamwork, we are responsible for configuring OpenCR. We assembled part of turtlebot3, then configured the OpenCR by following the e-Manuel. (The step by step instructions can be found in the appendix) Then we test the configuration of OpenCR by “Press and hold PUSH SW 1 for a few seconds to command the robot to move 30 centimeters (about 12 inches) forward” and “Press and hold PUSH SW 2 for a few seconds to command the robot to rotate 180 degrees in place”. And it works. We could see the demonstration at the site:

https://youtu.be/CJ8-lHX4I2Q

2. Porting SLAM and navigation system on Raspberry Pi

First, Install Ubuntu MATE for Raspberry Pi 3. Because the version of raspberry is 3B+, we burn the image file of the version of ROS kinetic. Connect the HDMI cable of the monitor to the HDMI port of Raspberry Pi. Connect input devices to the USB port of Raspberry Pi. Insert the microSD card. Connect the power (either with USB or OpenCR) to turn on the Raspberry Pi.

Then, we configure the Raspberry. The time needs to be synchronized. If the time is not synchronized, TF conversion errors are prone to occur. we need to add the server in the “ntp.conf”: $ add ntp.unice.fr

Then we set the master and slave, connect to the WiFi network(HotSpot) that is connected with the remote PC and Raspberry. We tried connecting the raspberry to the school's LAN, but it didn't work. So in the end we used our own phone hotspot. We make sure that both TurtleBot and Remote PC are connected to the same network segment under the same network segment. 172.20.10.6 is our master address, and 172.20.10.3 is Rasp’s address. (find by ifconfig)

ROS_MASTER runs on the remote PC, the PC configuration is as follows:

```
export ROS_MASTER_URI=http://172.20.10.6:11311
export ROS_HOSTNAME=172.20.10.6
```

We connect to the Raspberry Pi with its IP address: ssh pi@172.20.10.3. So now we can control turtlebot3 on the remote computer.

Tests and results

Final demo showing Real-Time autonomous navigation

Run roscore from PC. Open a new terminal from the PC and connect to Raspberry Pi with its IP address. (ssh pi@172.20.10.3) The default password is **turtlebot**. We have to run the navigation node in order to bring up basic packages to start TurtleBot3 applications: roslaunch turtlebot3_bringup turtlebot3_robot.launch

On one hand, We simply tested the w-a-s-d key to controlling the movement of the robot, and the operation was successful. A demo: https://youtu.be/CJ8-lHX4I2Q

```
export TURTLEBOT3_MODEL=burger
roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

On the other hand, we want to make the robot walk along the route of the map of Polytech's reception. For this step, we use LiDAR to generate a map of the reception of Polytech. On the remote computer terminal, we run the following command:

```
roslaunch turtlebot3_navigation turtlebot3_navigation.launch map_file:=$HOME/map.yaml
```

In the final step, we place the robot in a real environment at the reception and let the robot navigate autonomously through RVIZ. According to tests, the robot can set route navigation according to its own position and destination.

A demo: https://youtu.be/v8mUfoLE1SI

Conclusion

During the course of this project, we were able to successfully implement the construction of the ROS environment, use Lidar to build the map of the Polytech building, use the map to implement the simulated navigation on Rviz and Gazebo, and then build the TurtleBot3 system, finish the OpenCR setting in the teamwork. By establishing a connection between the remote end and the Raspberry end, we could operate on a remote computer to control the robot. Finally, the robot can automatically navigate along optimized routes in our generated maps while avoiding obstacles

In this project, we still have many implementations that need to be improved. For example, the quality of the generated map. When using Lidar to generate maps, we found that some details of the map are not very accurate, such as the size of the room, the overlap of the walls. Loop closure is the hardest part; when closing a loop, we could drive another 5-10 meters to get plenty of overlap between the start and end of the loop.

References

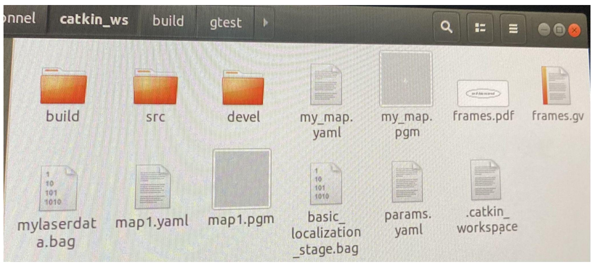
https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/

https://automaticaddison.com/how-to-build-an-indoor-map-using-ros-and-lidar-based-slam/

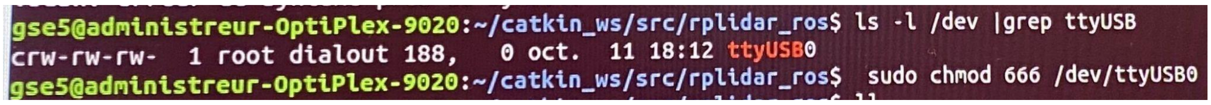
http://wiki.ros.org/ROS/Tutorials

Appendices (code)

Configuration of ROS environment



The connection between RPLidar and Ubuntu



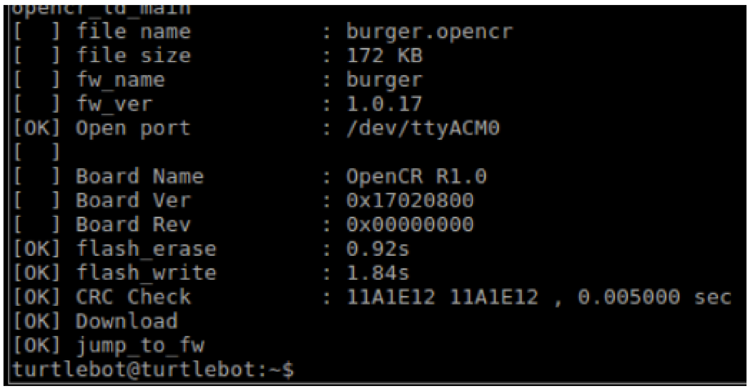
```
gse5@administreur-OptiPlex-9020:~/catkin_ws/src/rplidar_ros$ ls -l /dev |grep ttyUSB
crw-rw-rw-  1 root dialout 188,   0 oct.  11 18:12 ttyUSB0
gse5@administreur-OptiPlex-9020:~/catkin_ws/src/rplidar_ros$  sudo chmod 666 /dev/ttyUSB0
```

Configured the OpenCR

1. Connect the OpenCR to the Rasbperry Pi using the micro USB cable.
2. Install required packages on the Raspberry Pi to upload the OpenCR firmware.
   $ sudo dpkg --add-architecture armhf
   $ sudo apt-get update
   $ sudo apt-get install libc6:armhf
3. Depending on the platform, use either burger or waffle for the OPENCR_MODEL name.
   $ export OPENCR_PORT=/dev/ttyACM0
   $ export OPENCR_MODEL=burger
   $ rm -rf ./opencr_update.tar.bz2
4. Download the firmware and loader, then extract the file.
   $ wget https://github.com/ROBOTIS-GIT/OpenCR-Binaries/raw/master/turtlebot3/ROS1/latest/opencr_update.tar.bz2
   $ tar -xvf opencr_update.tar.bz2
5. Upload firmware to the OpenCR.

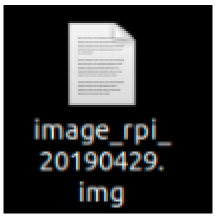
$ cd ./opencr_update
$ ./update.sh $OPENCR_PORT $OPENCR_MODEL.opencr

6. A successful firmware upload for TurtleBot3 Burger will look like below.



```
opencr_ld_main
[  ] file name          : burger.opencr
[  ] file size          : 172 KB
[  ] fw_name            : burger
[  ] fw_ver             : 1.0.17
[OK] Open port          : /dev/ttyACM0
[  ]
[  ] Board Name         : OpenCR R1.0
[  ] Board Ver          : 0x17020800
[  ] Board Rev          : 0x00000000
[OK] flash_erase        : 0.92s
[OK] flash_write        : 1.84s
[OK] CRC Check          : 11A1E12 11A1E12 , 0.005000 sec
[OK] Download
[OK] jump_to_fw
turtlebot@turtlebot:~$
```

Installation of Ubuntu MATE for Raspberry Pi 3



TIME Setting

$ sudo apt-get install ntp
$ sudo apt-get install ntpdate
$ sudo nano /etc/ntp.conf
add ntp.unice.fr