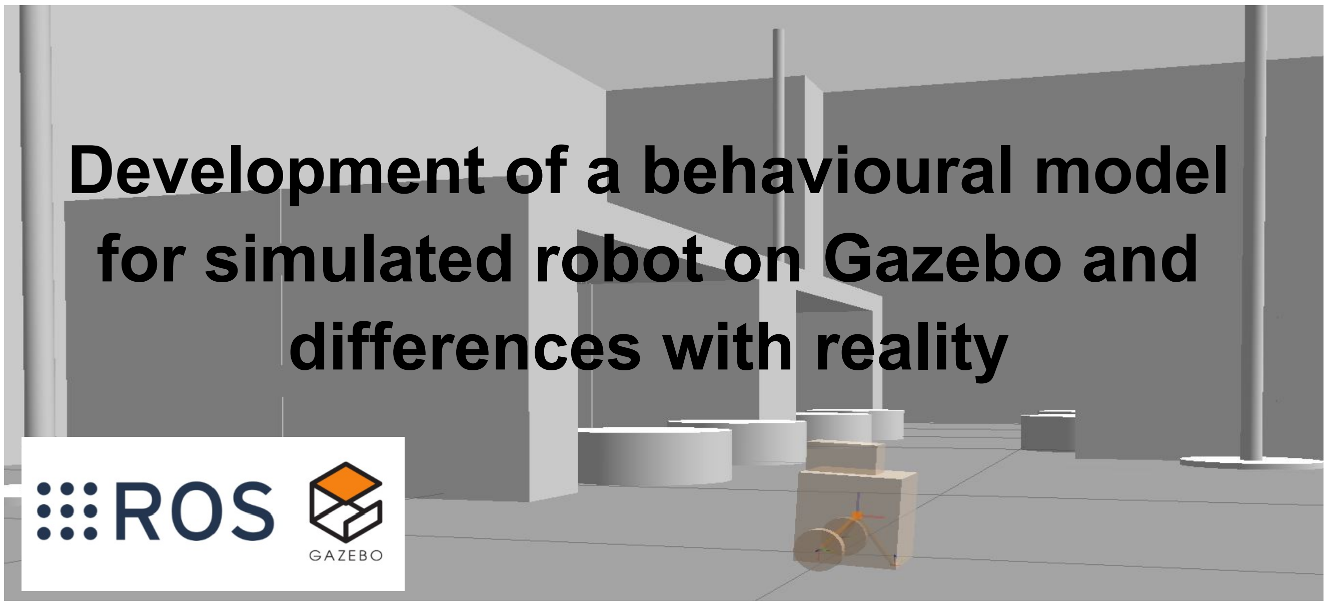
Internship at the University of Helsinki
Computer Science Department



Development of a behavioural model for simulated robot on Gazebo and differences with reality

Lucas MARTIN
IUT Montpellier-Sète
Département GEII

Academic Tutor: Ms Artaud-Gillet
Internship Tutor: M Fagerholm

18 June, 2019

Special Thanks

Before to start this report, we want to thank all the persons who allowed me to do this internship:

Mr VALENZA, IUT director , and Mr GILLET, director of the GEII Department, who accept to let us do this internship in Finland.

The IUT International Relationship and ERASMUS for the administrative cooperation and for the monthly grant respectively.

Mrs GILLET and Mrs LEGRAIS for their constant support and follow-up during all the internship.

CHRISTOPHE François, FAGERHOLM Fabian and MIKKONEN Tommi for their supervision, help and knowledge through all this internship.

KINNUNEN Petri who helped us to configure ROS on the Turtlebot3 raspberry Pi and helped us for the communication between our laptop and the Turtlebot.

All other members of the Computing Department and administration for welcome and help.

All members of UbiKampus to let us work in their workspace and the Empirical Software Engineering Research Group to let us work and test our robot in the Software Factory.

To finnish, my two friends and colleagues, LYOT Nicolas and SANCHEZ Joris, with whom I shared this international experience.

Introduction

To conclude our University Technological Diploma in Electrical Engineering and Industrial Computing, we had to do a 3 months internship to validate it. This internship can be done in a foreign country. With the UIT network, we were able to go to Finland at the University of Helsinki in Computer Science Department which is at Kumpula Kampus. We attached with a research group from the department.

During this internship, the objective of the work of my colleagues (Joris and Nicolas) and me was to identify the differences between the reality and Gazebo, a software for simulation. For that, we had to do the modelling of three different robots: Pulurobot (Joris), mBot (Nicolas) and TurtleBot3 Burger (me). After done the model, we had to implement a logic with ROS to test the robots on Gazebo and get first information about the simulation. To finish, we had to test the same logic in the real world and compare the result with results of simulation to know if the software can be used as part of research. This work had for objective to propose a first view about the precision of the software. We just had to do simplification and estimation concerning the modelling.

All this work was done autonomously with the help of Internet, due to the fact that our tutors didn't know how the using tools work. For that reason, we had to search on the different website, with tutorials videos or forums the answer than we need to. Beside we need to adapt our knowledge and what we found on internet to what our tutors needed while sharing our knowledges between us. To follow the progress of the project, we created a Trello with different tasks to had a visual of the project. What's more we did one meeting every two weeks approximately where tutors give us more details about what they want to test and where we show to them what we did.

The software which we used work on Ubuntu (Ubuntu Xenial Xerus). For that reasons we had to work on an Ubuntu virtual machine (VirtualBox) because we didn't have access to a Linux Computer with sudo rights. This generated many problems about Frame Per Seconds (FPS) or different bug while programing. Thus, may be some informations given in this report could be wrong on a true Linux's computer.

To know how Gazebo's working, we followed the instructions and the requirement of our tutors which were around 4 big axis. The first axis was the creation of the model of our robot. It was the first step of our project. The next step was the logic implementation with ROS to create a GUI which allows to control the robot. It was the biggest part of our work and the hardest difficulties of the internship. The third axis was the implementation on the real turtlebot and the last was the study of both Gazebo and real test to compare the results.

Table of contents

Presentation of the University of Helsinki

The University of Helsinki (*figure 1*) was founded in 1640 in Turku before it's transfer in Helsinki in 1829. It was founded by Christina Queen of Sweden, and it's the main and the oldest university of Finland with 36 500 students and 10 000 administrative staff. It contains 11 faculties and 11 research institutes. As Finland has two official languages Swedish and Finnish, the university give lessons in Finnish and Swedish with some other lessons in English which lead to a pretty good level in english. For that reasons, the university welcome many foreign students from Europe and the communication in many laboratories are in english.

HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

Figure 1: Logo of the University of Helsinki

Kumpula Kampus:

We worked in the Science Faculty, in Computer Science Department which is located in Kumpula Kampus. The department is lead by Sasu Tarkoma and it's located in Exactum Building (*figure 2*) with Mathematical Department. The department have an open place to work named UbiKampus where other trainees doctoral students and staff worked and share their knowledges. The computer Science department is famous to welcomed one of the more famous computer engineer: Linus Torvalds creator of Linux Kernel, who was a student of the University between 1988 and 1996.

Figure 2: Exactum building of Kumpula kampus

Software Factory:

In addition of Kumpula Kampus, we tested robots in Software Factory, which is a development laboratory where Company and students can go in to work. It's a place with many computer and robots to share knowledge and develop computing projects. The room is run by the Empirical Software Engineering Research Group, research group of our tutors.

1) Simple modelling on Gazebo

Gazebo (*figure 3*) is a software for robot simulation which works on Linux OS. As we said in the introduction, the goal of the internship is to know if Gazebo is a viable way to test robots as part of the research asked by our tutors. To do that, we had to model a room and a robot on this software. We were three trainees in this subject with three different robots to test and model in Gazebo. In this report, we will develop take the TurtleBot3 example on which we worked during this internship.

The first part of the project was to take the software in hand and model the robot after finishing the tutorials. So, as our tutors advised us, we did the Gazebo's beginner and some intermediate tutorials to start with the software before modelling our first robot. This was very helpful because we never used a modelling software before that.

Figure 3: Logo of Gazebo

1.1) How Gazebo's working?

Gazebo allows to create or picks up from other user, models, worlds or plugins. In our case, we will just talk about models and worlds. We will add plugins which allow to create a working way for robots, later with ROS.

Gazebo gets a Model Editor in which we can create a model for a robot. The model is a group of link which puts together to form a robot with Joint. Gazebo Model Editor gets the 3 different basic shapes for a link: cube, sphere, cylinder. We can mainly modify for each link inertia, poses, size of visual and collision. When we did the different links, we had to create the Joints to assemble the robot. Joints work with a Parent / Child relation. The parent one, is supposed to carry the child link. There are different types of Joint than we used for the modelling:

- Revolute: A simple joint with a rotation axis on x, y and / or z.
- Ball: A joint that allows the rotation on all directions for a ball or free wheel.
- Fix: A fix joint that allows no movement from the child link.

After the Model Editor takes on hand, we had to modelling the TurtleBot in Gazebo. For that, we had to take measure of the robot and reference of sensors.

The TurtleBot3 (*figure 4*) is just a simple robot with two wheels in front and a little ball in the back of its. First, the different floors of the TurtleBot was not relevant to model. Actually, the project is just a first test of the differences between Gazebo and reality. The floor modelling could was a waste of time for us. As we said in the presentation of the Model editor, there is 3 different shapes for a link. We had to choice which shape is the better to model each link. The most precise approximation of the robot's was a cube for the chassis and for Lidar, Cylinder for the two wheels and sphere for the ball. After we made the chassis's link, we did the wheels and ball, we had to put Joint in the end. We decided to take Revolute joint for wheels and Ball joint for the little ball.

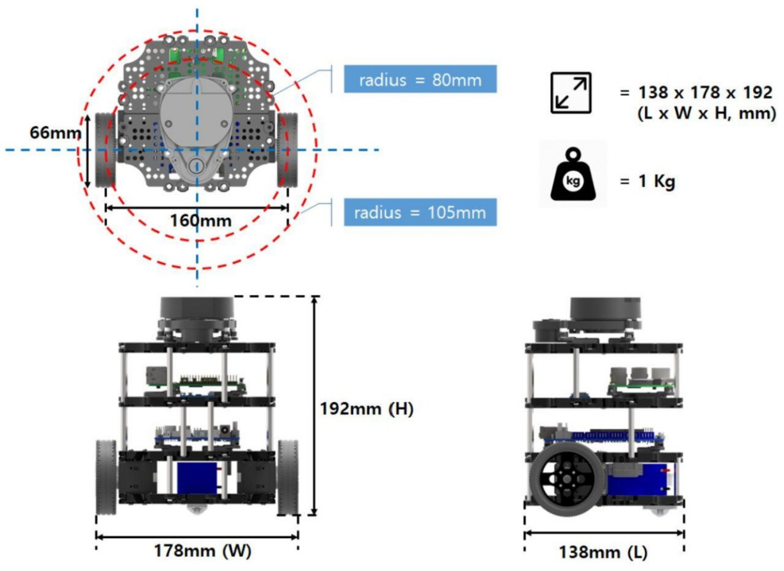


Figure 4: Measure of the turtlebot3 (source: emanual.robotis.com)

To finish the modelling, we had to create the LIDAR. LIDAR is a laser scan sensor that can measure a distance and an angle of an obstacle. LIDAR is located in the top of the robot. To create a sensor, Gazebo uses itself XML format: the Spatial Data File (SDF) format. SDF is a XML format that describes objects, environments, visualization and control for simulation, developed specially for Gazebo. It takes on board many different physical thing as lightning, gravity or dynamic. It's a complete format for robot simulation with many physics precision to have the better simulation available. The SDF of LIDAR was written manually with the help of the LIDAR's datasheet, Gazebosim tutorials and SDF format website (*figure 5*).

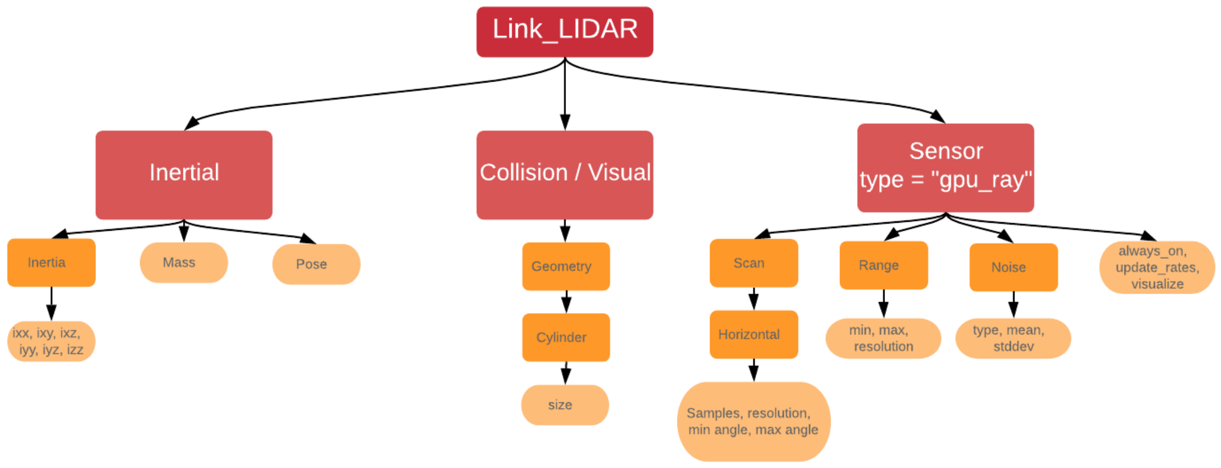


Figure 5: Architecture of our Link_LIDAR file

After writing the LIDAR’s model, we put it on the Turtlebot model done before. We just had to implement a fixed joint to don’t allow it to move. After this implementation (*figure 6 and 7*), we did a first simulation on Gazebo to see if the robot physics was good and if the sensor was ok.

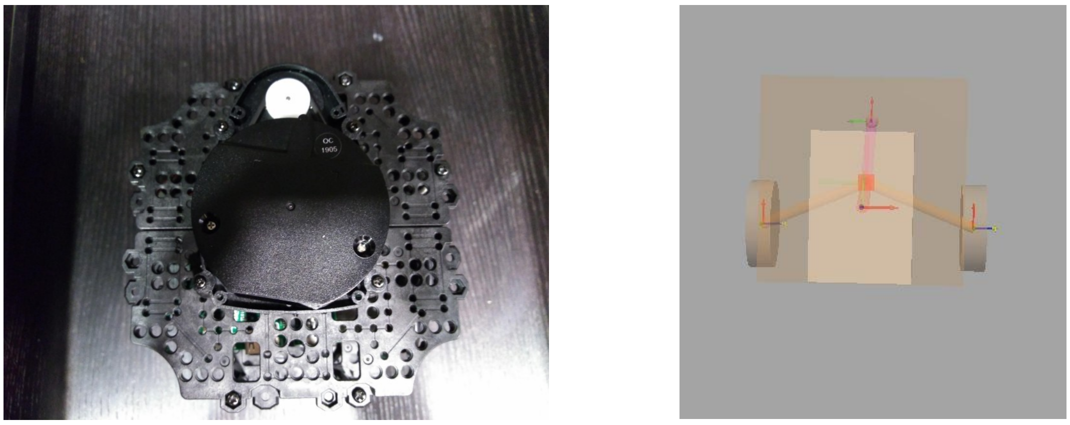


Figure 6: View of the top of the Turtlebot3 and model of the robot

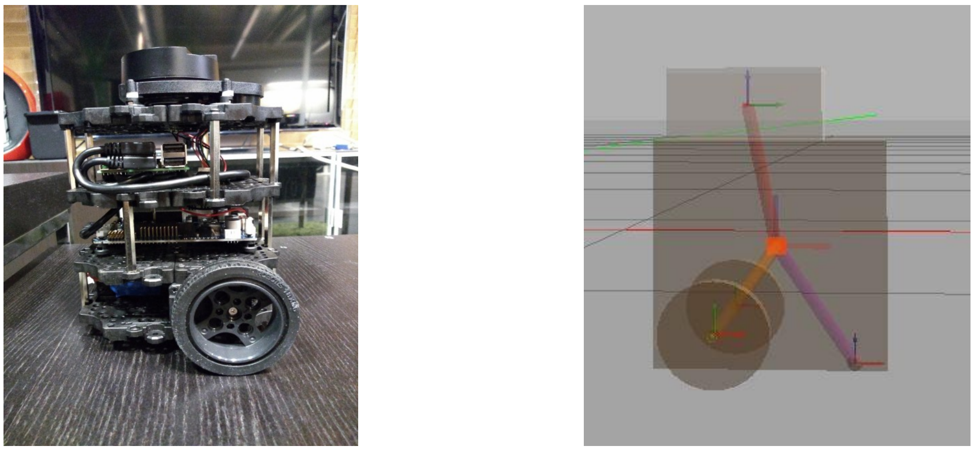


Figure 7: View of the side of both models

After solving small precision problem about placement of wheels, ball and balance, we could start the modelling of an environment. We chose to model the Software Factory room instead of UbiKapus because robots are in the Software Factory and the room is smaller and easier to model than UbiKampus. What’s more, we didn’t want to disturb other people who worked at Ubikampus.

1.2) Software Factory Model room

We modelled the Software Factory with the Building Editor tool which allows us to create a building. Building tool can import architectural plan to simplify the creation. In our case we put wall manually.

According to our tutor's suggestions, we did the measure approximately without precising measure instrument except a 20 cm ruler for some objects. The walls of the room were made with the building editor tool. For an easier modelling and interest test, we just created the leg of the chairs, which is the only part of the chair that can be in contact with the robot, while LIDAR can't see it. What's more, we only modeled chair under desk for an easier modelling and a better visual. The bottom left corner of the Software Factory room is in a semi-circle form, which was simplified by 3 right walls with an opening because Gazebo can't model circle wall as we know. We also decided to model the sofa and the shelf by a wall for an easier modelling. Concerning chairs, tables and pillars, the model was made by the editor model and we put table and pillar as static objects which can't be moved as wall.
With all of that informations, we have the next result on *figure 8.*

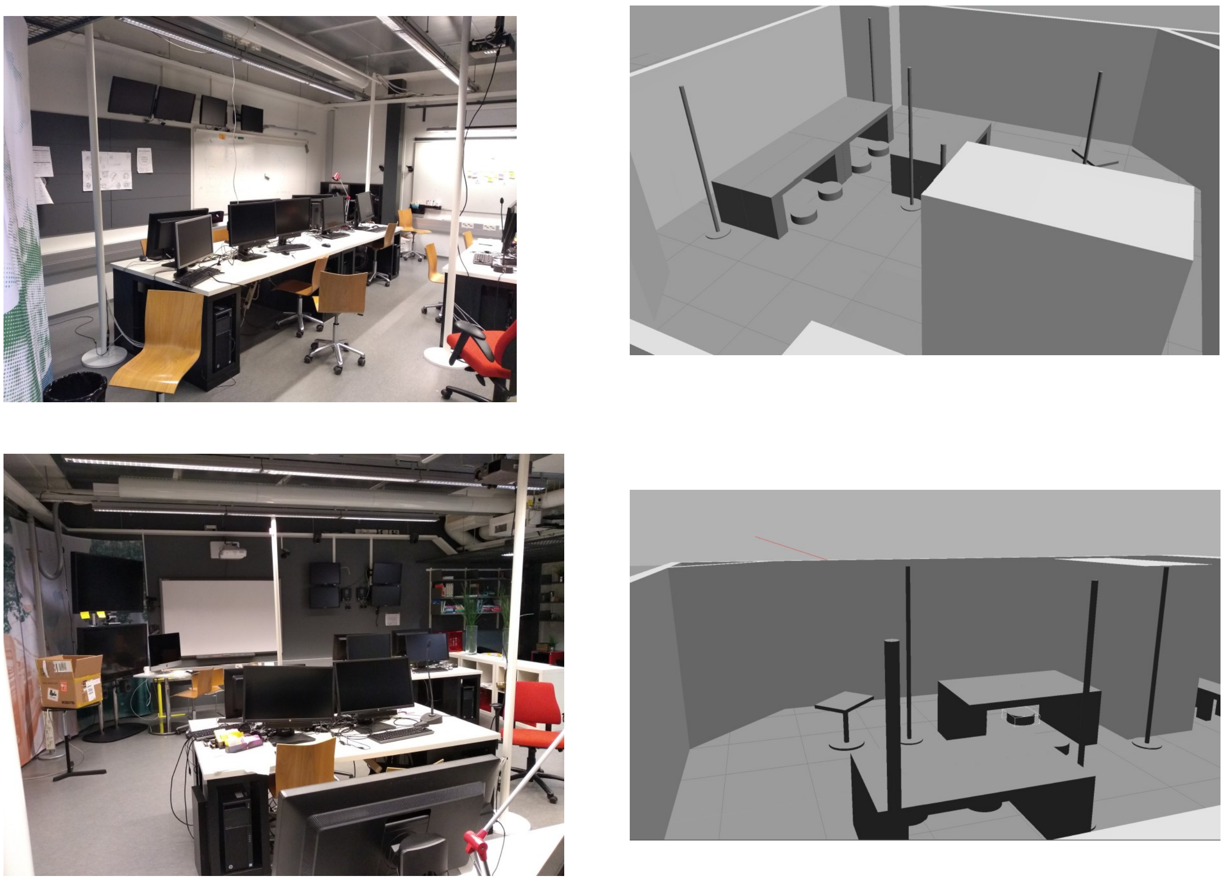


Figure 8: Different point of view of the real and model of the Software Factor

This room will be used by our robot. With this model (*figure 9*), some tests have to be necessary to see how Gazebo's working:

- Calculate, for a given speed in parameter, robot speed in simulation and real.
- What happen if the robot go into a chair.
- What happen when it runs into the base of pillar.

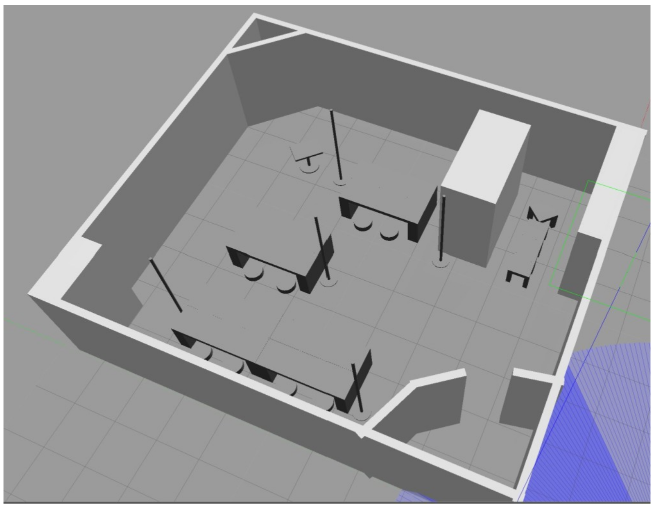


Figure 9: Global view of the modelling

1.3) Encountered difficulties

In this part, not many difficulties were encountered. If we follow the gazebosim tutorials, all should be ok. The only problem encountered is the repetitives crashes at when we started to use the software. The crashes stopped later when we launched gazebo with sudo command. What's more, the software in our virtual machine got many problems about Frame Per Seconds (FPS). Actually, we got approximately between 5 and 10 FPS in normal mode and near to 2 FPS during simulation. Without that part, the software is easy to take on hand. It is also easy to understand model and building editor. Even the SDF description of LIDAR is easy to write. There are many examples on internet as the Gazebosim tutorials or the sdformat website which explain with schematic block this format. In addition, XML is a computing languages which work with tag, so it's easy to read a program which came from another person. It is also important to launch the simulation when the building is done, because if some part of the building don't have the is_static set to true, part of the build which spawn overlap on a same place, will blow up or move at each start of simulation. It is also important for some object to enable the gravity to not get flying object.

2) Implement the Logic with ROS

The second step of our internship was to implement a logic on our robot with Robot Operating System (ROS). ROS (*figure 10*) is a collection of tools, libraries and conventions which have to aim to simplify the task to create behavior for robots and share this expertise. It works on Linux system and tolerate various languages for client as C++ and Python. What's more, ROS also has a lot of tools to create and build software for robots. This software are available on ROS official website (*ros.org*), Github or by the shelf with the `apt-get` command.

```
$ sudo apt-get install ros-distro-package_name
```

ROS.org

Figure 10: Logo of ros.org

For this project, we had the choice between Ros Kinetic, Lunar and Melodic. Kinetic Kame (*figure 11*) is the more documented and more viable version for our robots, have enough packages available for our application and it works with Gazebo and Ubuntu 16.04 Xenial Xerus. To learn how ROS works and to take it on hand, we did all the beginner tutorial of ROS. After tutorials were made, we started to implement the logic with ROS packages than we found on ROS website. The aim of the logic was a General User Interface (GUI) in which we can control robot with keyboard or virtual joystick, build a map with LIDAR sensor, avoid obstacle, navigation from A point to B point and systematic or random area covering movement. Based with roscpp and rospy libraries, we used some tools and packages.

Figure 11: ROS Kinetic Kame logo

2.1) URDF file

When we started to use ROS, we noted than it uses Universal Robotic Description Format (URDF) instead of SDF. URDF is a XML file format used by ROS to describe all elements of a robot such as SDF format. Despite the two formats look alike, some differences exist between both. The main difference is the description. URDF format can only describe a single robot when SDF format can describe a world, several robots and many physics notion. What's more, some SDF arguments are not present in URDF. Finally there is no pose for the different shape given that there is no description of a world in URDF. So to place correctly each shape, their positions are given by put poses on each joint. To conclude, the SDF format is easier to use with the help of Gazebo and the translation from SDF to URDF can be a real difficulty especially to put the correct position to each shape.

As we said before, the URDF format gets some differences with SDF format. For instance, detail relative at the surface element aren't in URDF and detail relative to a world as the wind. What's more the visual and collision element are less precise in URDF than SDF. The basic element as geometry in visual and collision, inertia or mass are same for both format. To counterbalance the loss of the precision, a gazebo element can be attached to a URDF link in which we can set gravity or avoid the self collision mainly. The *figure 12* will show how our URDF is working.

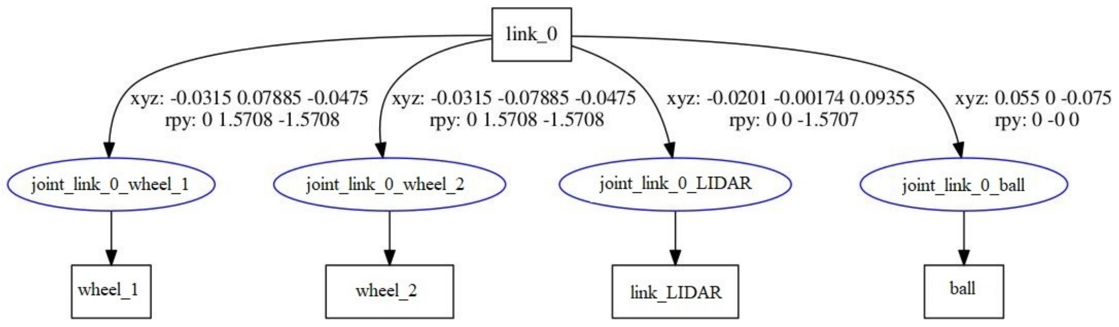


Figure 12: Graphiz of our URDF file

2.2) Tools used

We used 6 different ROS tools and commands to complete successfully this aim:

- `roscore`: `roscore` is a collection of nodes and programs that are pre-requisite of a ROS-based system. We have to run roscore to allow the communication between ROS node. The machine have to be the master as part of a ssh communication to work correctly.
  ```
  $ roscore
  ```

- `roslaunch`: `roslaunch` is a tool for easily launching multiple ROS nodes. It takes charge of .launch file, part of a package, which are write on XML. What's more, if `roscore` is off, `roslaunch` will start `roscore`.
  ```
  $ roslaunch package_name launch_file_name.launch
  ```

- `rosrun` : `rosrun` allows to run an executable file write in Python or C++ which has to be in a ROS package.
  ```
  $ rosrun package_name executable_file
  ```

- `rostopic`: `rostopic` contains command to display, debug and publish in the different topics which are named buses. In a topic nodes can exchange messages between them. In this project, we used 3 of the 9 available commands of rostopic
  `$ rostopic list` (display a list of current topics)
  `$ rostopic pub /topic_name topic_type data` (publish data to a topic)
  `$ rostopic echo /topic_name` (Display messages published to a topic)

- `catkin`: `catkin` is the official ROS build system which combines CMake and Python scripts. It is used as workspace and builder for the project:
  ```
  $ catkin_make
  ```

- `rviz`: `rviz` is a 3D visualization tool for ROS. It allows to show nodes and topics which are on and allows to show the content of topics and nodes on a map for example. It can be automatically launched in a .launch file or manually with rosrun
  ```
  $ rosrun rviz rviz
  ```

2.3) Packages used

To do this GUI, we needed to use some packages and their dependancies. We will explain only the main packages and topics which we used.

- `gazebo_ros_pkgs` : As its name show, it provides the interface between gazebo and ros. It is necessary to use it to simulate a robot using ROS in Gazebo.

- `robot::robot_state_publisher` : This package allows to publish the robot state to `tf` (a package that lets the user keep track of multiple coordinate frames over time). When the states are publish in, their are available by all other components or nodes using `tf`.

- `robot_model::joint_state_publisher` : This package will read the `robot_state_publisher` and allows you to publish `JointState` messages with name, position, velocity and effort for each joint. As for `robot_state_publisher` it's an obligatory package to publish the robot in rviz.

- `key_teleop`: `key_teleop` allows to run a python script to use keyboard of a computer to control the robot. The robot can go forward and downward but can also rotate with right and left fleched keys.
  ```
  $ rosrun key_teleop key_teleop.py key_vel:=cmd_vel
  ```

- `slam_gmapping::gmapping`: The `gmapping` package of `slam_gmapping` is used in our project to build a 2D map with LIDAR. It allows to locate the robot in the map while modelling the latter with its `slam_gmapping` node. As many of these nodes, we load the node in a .launch file with its parameter.

- `navigation::move_base` : `move_base` package is used to create a simple action which consist to go from the place of a mobile robot, to a navigation goal. It's a simple movement from A to B while avoiding obstacle.

- `costmap_2d`: This package provides the creation of a 2D costmap with data from sensor. The created costmap is used by the `move_base` package.

- `frontier_exploration` : The `frontier_exploration` package extend `move_base` and send messages to it. It allows to set a polygon in the costmap. The robot will explore this polygon until it finds the frontier of this one or until all the space will be mapped.

2.4) Our ROS working

To accomplish our aim to create a simple GUI, we decided to use RViz. RViz will load topics used and show the result of each topic on an interface. In this packages load, RViz will load gmapping and show the result of the map. To control the robot, there are two ways with key_teleop:

- The first and easier way is to use the following command which allows to use the keyboard to control the robot
  ```
  $ rosrun key_teleop key_teleop.py key_vel:=cmd_vel
  ```

- The second is to load the `key_teleop` topic and use it with the specific RViz part which is used with the mouse.

To do the point A to B movement, the move_base package will be used. With RViz, we just have to put a "2D Nav goal" arrow to the map. After the arrow was put, the robots will go from his position to the base of the arrow. The last package use is the frontier_exploration package. After put a polygon with "Publish point", the robot will explore the polygon until it founds frontier or discover all the polygon. This packages will be used for the random or covering area. The final communication graph is show on *figure 13* and the Rviz final result on *figure 14*.

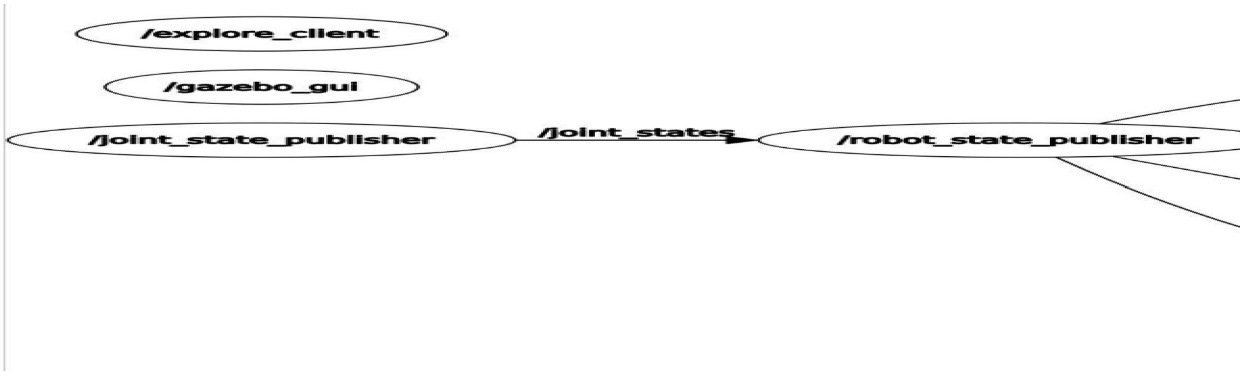
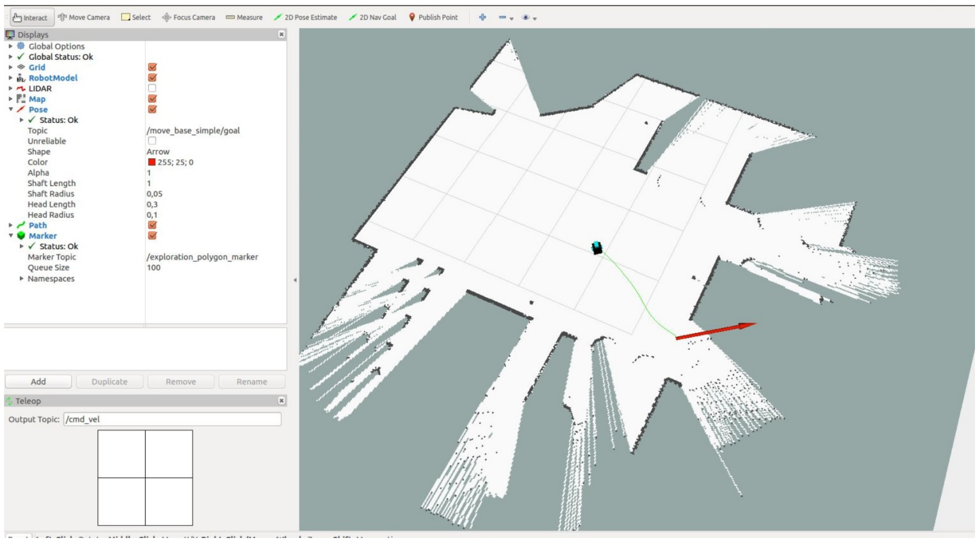


Figure 13: RQT node graph



Global Options
Global Status: Ok
Grid ✓
RobotModel ✓
LIDAR ☐
Map ✓
Pose ✓
Status: Ok
Topic /move_base_simple/goal
Unreliable ☐
Shape Arrow
Color 🟥 255; 25; 0
Alpha 1
Shaft Length 1
Shaft Radius 0,05
Head Length 0,3
Head Radius 0,1
Path ✓
Marker ✓
Status: Ok
Marker Topic /exploration_polygon_marker
Queue Size 100
Namespaces

Output Topic: /cmd_vel

Figure 14: Result of our RViz GUI

2.5) Encountered Difficulties

This part was the hardest part of the project. We didn’t know how ROS is working before this project. Even if tutorials are good, some notions are blurred and became cleaner during the implementation. The first big problem encountered was the translation between SDF and URDF. This part was very difficult for us because many parameters and attributes have a different way to be write in both format and the position of the robot is also different. As said before, SDF works with universal poses of a world meanwhile URDF works with specific pose in which the origin is the center of the base link. Poses of the following links are made with joint. They are the one of the center of the shape and have to be in function of the center of the base link. What’s more, some link or joint didn't work for unknown reasons. Indeed there is not a “ball” joint as SDF and revolute or prismatic joints will teleport the robot in the origin of the world when we ran the simulation on Gazebo. The child link of the joint will also disappear in that case. Except this point, each format have same constructions and we can copy / past some part with just some point to change as the way of write a little part.

For ROS apart the blurred notions and some little problems on the load of packages, internet was a good way to improve our logic and debug our project. The biggest part of ROS packages are used by a lot of person and they are well documented. What’s more, a lot of forums exist, especially the ROS forum (*answer.ros.org*), in which ROS packages creators are really active and help people. We had hurt with just a little problem with RViz which didn’t see the robot’s model, even with `joint_state_publisher` and `robot_state_publisher` node if the simulation is off. We don’t know if it’s normal or if it’s possible to fix that with a param ou rosparam attribute.

The last problem encountered in this part was a Gazebo / ROS problems. In fact, in the translation from SDF to URDF, the inertia part and the weight have been modified. In the firsts Gazebo’s models, the weight of the robot was not respected. We wanted to change that in the URDF file. Well, when we put the real approximate weight (~1kg for whole of robot), the robot physics changed. When we launched `key_teleop`, it is all the robot which rotated and not only the wheels. This problems is solved when we put a bigger weight (~100kg) on the robot. It is may be come from the inertia part. Inertia ask a lot of physics notion than we don’t have. So, the value put in this part are general values for this size of robots found on internet.

3) Reality implementation

When we finished with ROS and simulation part, we decided to test the robot and learn how can we will use it. For that part Petri, a doctoral student, helped us to setup the communication between our laptop and the turtlebot. The aim of this part was to take on hand the robot and start to notice the differences between what packages we took for simulation and try to put them in the reality or change existing well-known ROS Turtlebot3 projects to concord with our simulation packages. In the end we have to get two projects with same packages and same parameters with only a different ROS master (Turtlebot3 for reality, Gazebo for simulation).

3.1) Presentation of robot and setup

For this project, we used the Turtlebot3 burger model. This robot is famous to be one of the robot which are officially compatible with ROS and to be used as example in many cases to take on hand ROS. It is composed of 4 floors (*figure 15*):

- The first one is the robot and wheel floor. In this floor, a Li-Po battery is placed for the Dynamixels motors. At the bottom of this floor, a little ball is placed.

- The second floor is composed of an OpenCR. OpenCR (Open source Control module for ROS) is an open source hardware and software developed for ROS embedded systems. It is mainly used to do the communication between the Raspberry PI (3rd floor) and the Dynamixels motors (1st floor) .

- The third floor is composed of a Raspberry PI 3 B+ which is the brain of the turtlebot. It is connected to the OpenCR for the communication with the motors but also with the LIDAR. The communication between LIDAR and Raspberry is make with a UART to USB converter. We can also connect mouse and keyboard on USB ports and screen on HDMI port of the raspberry to control it with a monitor.

- The last floor is only composed of a RPLIDAR A1 sensor of SLAMTECH which allows a 360° scan for the mapping and navigation.

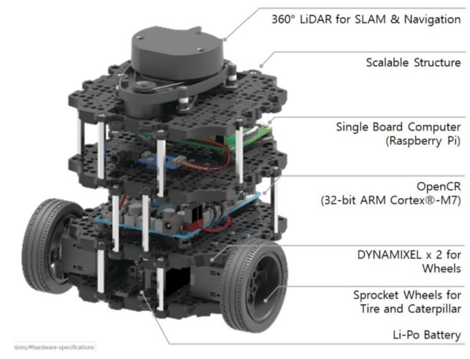

360° LiDAR for SLAM & Navigation
Scalable Structure
Single Board Computer (Raspberry Pi)
OpenCR (32-bit ARM Cortex®-M7)
DYNAMIXEL x 2 for Wheels
Sprocket Wheels for Tire and Caterpillar
Li-Po Battery
tions/#hardware-specifications

Figure 15: Components of the Turtlebot3 (source: emanualis.robotis.com)

A raspbian version exists especially for turtlebot3. This version contains ROS Kinetic already installed and some packages and nodes pre-installed. A usable package exist which allows to run a core with some useful nodes as `/cmd_vel` while starting the LIDAR. We will use this package to communicate with our laptop and to use some topics from which it can load. After this version is installed, we had to setup OpenCR. While we just have to setup the raspberry once, the OpenCR setup have to be make at each start. The setup consist to open a port of the raspberry to allow the communication between the raspberry and the OpenCR, in our case `ttyACM0`. After open this port, we have to remove the `opencr_update.tar.bz2` and download it again to obtain a newer version. A documentation is available in appendix to know how run our project. After this done, the OpenCR is up and we can launch the motor.

3.2) Communication between Turtlebot3 and remote PC

To allow the communication between the turtlebot and our laptop, we had to add some lines in the end of the .bashrc file of the computer and Turtlebot. The first line which is the same for Turtlebot and laptop, create a variable named `ROS_MASTER_URI` which, if we put the IP address of remote PC, allows to our laptop to be the ROS master and allows the turtlebot to connect to the computer.

```
export ROS_MASTER_URI=http://IP_OF_REMOTE_PC:11311
```

The second line is different for turtlebot and our computer. This line will create a variable `ROS_HOSTNAME` which contains the IP of the device. For the turtlebot, we will have the IP of the turtlebot, and for the computer, the same `IP_OF_REMOTE_PC` as last line.

```
export ROS_HOSTNAME=IP_OF_DEVICE
```

After this was done we just have to source the .bashrc file and connect the remote PC to the raspberry with SSH command:

```
$ source ~.bashrc
$ ssh pi@IP_OF_TURTLEBOT
```

Now, to use ROS on two different devices, we just have to launch a ROS master terminal with roscore on the remote PC. After the roscore was launched we launch on the turtlebot one of the .launch file from turtlebot3_bringup package, basically installed on raspbian for turtlebot. When we launched the turtlebot3_robot.launch file, the topics are available in the PC and we can write on them. RViz and all other packages will be launch on the laptop (*figure 16*).

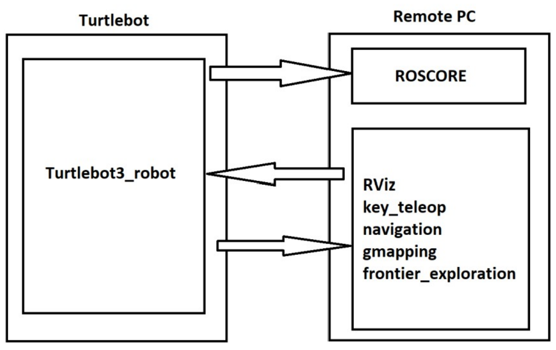


Figure 16: Simplified Block Diagram of the communication between PC and Turtlebot3

3.3) Resolution of connection problems:

A problem exists when we use a VirtualBox virtual machine. The IP address could be with 127.X.X.X instead of 192.168.X.X if the network configuration of VirtualBox is in NAT. The only way fix that we found is to start VirtualBox with a NAT network on. Once we are on the Desktop of the virtual machine, we went on the configuration, in network part and switch NAT to Bridge Access. After that, when we did ifconfig to have the IP address, a 192.168 is open (wlan).

What's more, for some unknown reasons, it is impossible to connect a PC to the Turtlebot if the Turtlebot clock isn't synchronize with the remote PC clock. To fix that problem, we have to install `ntpdate` on each devices and launch it.

```
$ sudo apt-get install ntpdate
$ ntpdate ntp.ubuntu.com
```

3.4) Encountered Difficulties:

In this part, not many difficulties were encountered except the connection problems explained before. We just use existing packages and projects which work with turtlebot which came from ROS and *emanual.robotis* website. A problem subsist in our project. For some reasons than we didn't find yet, `move_base` package didn't work. We were able to use `move_base` packages only one time by using `turtlebot3_frontier` packages. When we launch a move_base node, an error message is display on the terminal which says than `move_base` couldn't get the local map. We tried to change some parameters but at the moment, we weren't able to launch the node again.

4) Reality and simulation study

To realize a good comparison between reality and Gazebo, we chose exactly the same parameters for each nodes used. To do this, we put the real parameters to the simulation. In fact, this parameters are basic parameters for our robot. What's more, for the security of the robot, it is not good for to put approximate value that came from simulation. With the work did in last part, we learnt than packages used by ROS and *emanual.robotis* are the same as we used in our implementation with ROS. With all these parameters, if Gazebo do a simulation of the world the two results should be the same at the exception of the simulated model and the simulation room which are simplifications.

When we did the test, we tried to do approximately the same thing for both simulation and reality. For that, we tested every movement (`key_teleop`, `2d_navigation_goal…`). Despite the same parameters in reality and in simulation, there still some differences and particularities for both case.

4.1) Differences

We can't calculate exactly the speed in simulation but we noticed that the robot is really slow in comparison of the reality if we used `key_teleop`. But if we use the RViz teleop, the robot is clearly speeder in simulation. We think it's because when we run `key_teleop`, the forward movement put to `/cmd_vel` in x axis is 0.8. But, the teleop of RViz probably put 1.0 or just use a node which pass through the 1.0 limit of `/cmd_vel`. What's more, even if we had problems of FPS on our virtual machine, we saw than the movement on simulation are less fluid than the reality one. In addition to the speed, the redirect of the robot when it is at the arrival is slower than reality.

For the sensor part, some differences were listed too. First, if there isn't an obstacle at the 6 meters of the ray range which is the max range put in parameters, the real robot will put a "wall" on the map until it will explore the zone (*figure 17*). We probably can't know if it consider this point as a real wall because when the robot become closer to this point, they disappear (the robot replace the point by a blank as it should be).

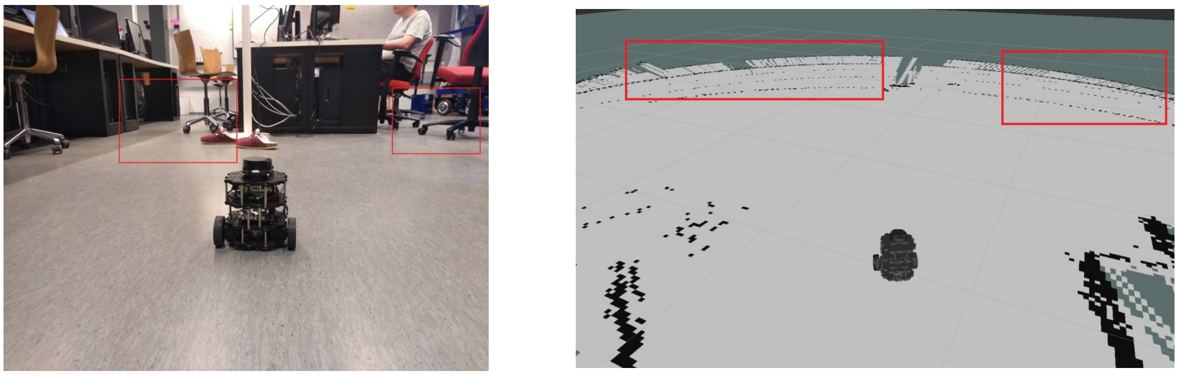


Figure 17: Example of this pixel Cloud at the max range

It is something than the simulation don't have because simulation will just clear the zone. The sensor is also a perfect sensor in simulation. In other words, the LIDAR will launch exactly the 360 ray at same with exactly 1° between each laser. It works exactly as the builder says in datasheet. In reality, some laser bring back wrong information about position of an object or are just not be take in consideration by gmapping. These errors are just little errors and don't limit the functioning of gmapping.

4.2) Similarities

The biggest part of the similarities concern the collision of the robot. In the model of the pillars as reality we put a small step at the bottom the pillars. In both cases when the robot go in the bottom of a pillar, it will be blocked by the step and cannot move on it. What's more, even if we didn't modelled whole chair, in reality the leg of the chair are too small to be get by the LIDAR. That comfort us in our choices to simplify the model of the software factory. The gmapping precision is also the same in both case except for the loss of laser data which was explain in last part.

4.3) Distinctive characteristics

In reality and simulation there is some specific characteristics in both. In simulation, the weight of the objects is wrong. When we tested in Gazebo `key_teleop`, the robot could push the chair (weight of 5kg in Gazebo) and robot get a weight of more than 100 kg to work with `key_teleop`. What's more, as we said in the ROS part, there still a problem with the inertia if a precisely solution is needed. We think it can be solve if we find the relation between simulate weight, simulate inertia and real weight. If this relation is found, it will be possible to simulate a better physics than our simulation have. The distinctive characteristics for the reality test, was in relation with the room. The software factory got some shelf with transparent or plastic colored storages. In this case, The lidar will, either not receive the laser, pass through the plastic or glasses and will take in consideration the interior of the shelf as we can see in *figure 18*.

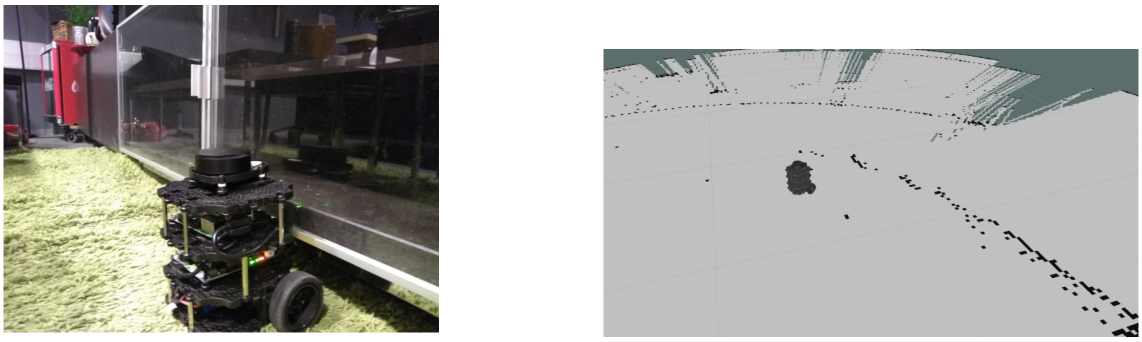


Figure 18: Results of our tests for the transparency

Work conclusion

The aim of this project is to have a first view of the simulation tool Gazebo to know if it's an exact software for future use in research field. In this goal, the software have to get precision about physics notion and have to be easy to take on hand. It also has to be easy to put precise logic in it for test part with same code as reality in order to have the same results in simulation and reality.

Positives points of Gazebo

The first positive point and not least is that Gazebo is easy to use. It gets a lot of tutorials in its website and also have an active community on forum. What's more, it uses the XML format which is a language quite easy to understand and learn, and Gazebo plugins are write in C++ language one of the most famous and using computing language. So, it is not necessary to learn a new languages to use this software. Concerning the transparence of some object explained in study part, this problem can be solve by editing texture with gazebo. Indeed, SDF format allows to change texture and transparent texture can be created for a better precision of testing. For the physics part, if we used SDF format with good knowledges in physics the software can be really precise about this notion for first tests of a robot only with inertia and some other main physics notion.

Negative point of Gazebo

The worst point than we found during the project was the precision of the URDF format. The URDF is the only file format which can be used by `gazebo_ros` package. Its precision is really bad in comparison with the SDF format because it only describes a robot and not take in count the world around the robot. Even if URDF got a gazebo element, the attribute in it are just a small part on which Gazebo is able to calculate. What's more, the simulation can be really different than the reality. The software will simulate the perfect working of the robot. It can be really different than the real test depending what is test. We said before than the physics part is complete. But, the only bad point of the physics of Gazebo is the weight. Some weight, in addition to ROS, are false and we are obliged to put wrong weight to allow the robot's working. To conclude the negative point, Gazebo required a lot of computing resources during simulation. It's really hard to use it with a virtual machine, and even with a computer on Ubuntu it mights be hard to use it without software speed reduction.

Limit of our tests

The test we did are just first test by students and have some limits. The biggest limit of our test is our hardware was not good for this project. We worked with a virtual machine which restrains the performance and the power of calculation. We think than some precision problems of the simulation may come from this as the precision of `move_base` package. This bring about other thing as we can't do the calculation of the speed to compare with reality. Our FPS problem was a really big obstacle during this internship. Another limit of the project was our knowledges about ROS mainly. We passed a lot of time to understand how ROS is working and we think we didn't use all possibility than ROS has. What's more as we said before, we don't have the knowledges in physics to test gazebo precision about a lot of physics notion than Gazebo have.

Improve the results

To improve the results of our tests for a future project, we thought about a few things to explore. First, is to put good and perfect value on the physics attributes of SDF / URDF in the concerning part. We could with this correct value improve the precision of the simulation. The first example than we didn't test is the test of the wind to see how can it works. We also want to test in the future all the potential of the SDF format. For that, we have to use Gazebo plugin for the work of the robot. ROS and URDF have to be sideline to know the capacity of Gazebo and SDF format. The last improvement is to model a better version of a room and try to change texture as transparent texture to know what will happen with the sensor.

General

In general, we think than Gazebo can be used in research for the first tests. Currently with our tests we can't say if it's a viable way to simulate something which needs a very big level of precision as aeronautics or space field. Despite of that, the software is enough precise if we need just to know how something could work before implement it in reality. In the case of Petri's work, his aim is to create a way to program a robot for children with blocks diagrams. If the blocks can be implement with ROS on a simulated robot, Gazebo can be a viable ways to test on a simulate robot to see the first problems before implement the logic in a real robot to do the finish.

All the project is available on Github of the Software Factory with a small tutorials to setup and use it:
https://github.com/TheSoftwareFactory/turtlebot3-real-and-simulation

Personal conclusion

This internship bring me a lot of professional knowledges as much as human knowledges. For the professional I had never work on a modelling and simulation software. It was very hard at the start of the internship but my tutors and friends helped me to learn Gazebo and ROS. It was really interesting to learn an unknown notion in addition to have a criticize view on it. Even if it was very hard for the implementation and the use of ROS, I really loved to discover a new way to program a robot and to learn. I wanted to know how ROS is working since my internet research for my semester 4 project (Holonomous Robot fleeing obstacle with LIDAR). Indeed I worked in autonomy. This autonomy allows me to improve my research skills and I could learn with my own way and at the speed than I wanted.

What's more, doing this internship in Finland bring me so much in the human plan. It was really good for me to see another culture, way to work and another country. I also improve my English comprehension during this internship through the necessity to communicate in English inside the laboratory.