# Abstract

This paper presents the motion planning for a differential drive robot which moves in an indoor environment with static and dynamic obstacles: in particular, the environment consist in an hospital or a medical center and the tasks assigned are collecting medicines or different types of packages from the reception and leave them to the patients, avoiding collisions with static and dynamic obstacles.

The Differential Drive Robot is equipped with a Hokuyo Scanning Laser Rangefinder which is a small, affordable, and accurate laser scanner perfect for robotic applications. It is used to map the environment and localize the robot in the map. The localization algorithm used is Monte Carlo localization approach, which works thanks to the use of a particle filter to track the pose of a robot against a known map. In order to avoid dynamic obstacles, Dynamic Window Approach (dwa local planner) and Time-Elastic Band Local Planner (teb local planner) helps to plan the right strategy of collision avoidance.

## INDEX

# Introduction

The MedBot is a service-robot that assists humans in hospital, carrying packages from one point to another with a safe path, computed at each instant according to the environment status, in order to avoid collisions with obstacles.

The main aim of the hospital transport mobile robot MedBot  is to carry out such tasks as the delivery of off-schedule meal trays, lab and pharmacy supplies, and patient records.

Unlike many existing delivery systems in the industry which operate within a rigid network of wires buried or attached to the floor, this robot is expected to be able to navigate much like a human would, including handling uncertainty and unexpected obstacles. The navigation system of MedBot takes advantage of the specific structure of a hospital environment and relies on the latest map update based on provable sensor-based motion planning algorithms. Also incorporated in the system are algorithms for handling unmodeled factors such as obstacles (people).

The hospital transport mobile robot MedBot is expected to become one answer to the current shortage of help in hospitals. Specifically, it addresses the need for assistance with such tasks as point-to-point delivery (i.e. meal trays, pharmacy and lab supplies, patient records, and mails). Given the environment's map of the modeled hospital, one technical objective of the MedBot project is to successfully handle the uncertainty present and to navigate based only with the knowledge of the global map. Also, the MedBot should require little modification to a given hospital and thus make a maximum use of on-board sensing of the natural environment.

The mobile robot considered is a differential drive robot with two standard fixed wheels and one castor wheel on front. It is equipped with a laser sensor which helps the localization of the robot in the environment and allows it to use the navigation stack which is great with dynamic obstacles. The SLAM (Simultaneous Localization and Mapping) algorithm has been used to know the global static map of the environment, in particular the study case concerned Gmapping approach to build the map.

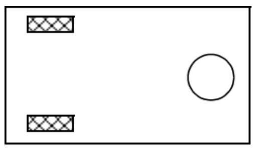
## Dynamic Obstacle Avoidance

Since the obstacles in the hallways are constantly changing, obstacle geometry and positions cannot be prestored in the MedBot memory. Obstacle avoidance must be an online operation with information about the environment continuously obtained from onboard sensors. While traveling along the prescribed path, sensor information about the MedBot's immediate surroundings is continuously analyzed. If an obstacle is detected obstructing the prescribed path, the robot will leave its path if that is necessary to avoid it. If the obstacle has moved during this pause, then MedBot will simply continue following the prescribed path as if nothing happened. If the obstacle is still present then sensor information is used to decide on the most promising direction for maneuvering around the obstacle, either right or left.

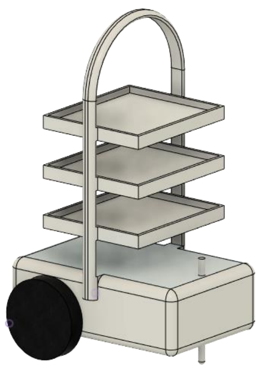
No modeling of obstacles is performed. To maneuver around an obstacle, points along the accessible (visible) part of the obstacle boundary are identified. In identifying the obstacle boundary points, only sensor information from the local map is used.

## Prototype

Typically, the differential drive robot is a mobile robot with two motorized fixed standard wheels and one castor wheel. This configuration allows the robot to move in an indoor environment with no great effort.



Our robot has been designed using Fusion 360 which is a computer-aided design application for 3D mechanical design, simulation, visualization, and documentation developed by Autodesk.



The design parameters in terms of dimension and dynamic properties of the differential drive robot are shown in the following table.

| | Value | Unit |
|---|---|---|
| Length | 600 | mm |
| Width | 400 | mm |
| Height | 1020 | mm |
| Mass | 9.8 | Kg |
| Material | Plastic | ABS |

| | Value | Unit |
|---|---|---|
| Diameter | 240 | mm |
| Width | 50 | mm |
| Mass | 2.1 | Kg |
| Wheel separation | 400 | mm |
| Material | Rubber | |

## Kinematic Costraints

Typically, each wheel must satisfy both the no sliding condition and the pure rolling condition:

- no sliding condition imposes that there is no velocity component in the direction perpendicular to the sagittal plane of the wheel;
- pure rolling condition imposes that all the traction is transferred in motion (every time instant the contact point between the ground and the wheel must have zero relative velocity).

**Standard fixed wheel** has two degrees of freedom and can traverse Front or Reverse. The center of the wheel is fixed to the robot chassis. The angle between the robot chassis and wheel sagittal plane is constant. Fixed wheels are commonly seen in most WMRs in which them are attached to motors and are used to drive and steer the robot.

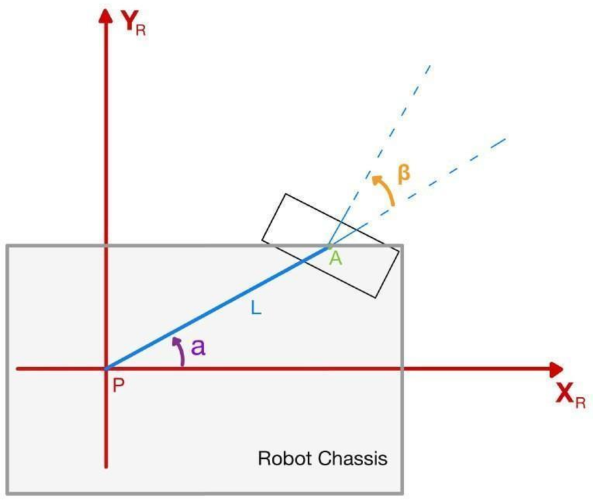


*Figure: Standard wheel*

As can be seen there are different parameters to consider:

- $\alpha$ is the angle of vector pointing A respect to X-axis of robot reference frame;
- $\beta$ is the angle between a plane perpendicular to the sagittal plane of the wheel and the vector pointing A and it is fixed;
- $L$ is the distance between the center of the wheel and the origin of the robot reference frame.

The velocity vector of the robot chassis, referred to robot reference frame, must be divided into two components (one along the X-axis and the other along the Y-axis). After that, velocity components are decomposed along perpendicular and parallel directions to the wheel plane; these components are transposed to the wheel.

After some mathematical operations, imposing no sliding condition (so imposing that velocity component of the wheel along the perpendicular direction to the wheel plane is null), the following condition is obtained:

$$\dot{x}_R \cos \alpha + \beta + \dot{y}_R \sin \alpha + \beta + \dot{\theta} l \sin \beta = 0$$

On the other hand, imposing pure rolling condition (which means to impose that velocity component along the parallel direction to the wheel plane is equal to $\dot{\varphi} r$ ) the following condition is obtained:

$$\dot{x}_R \sin \alpha + \beta - \dot{y}_R \cos \alpha + \beta - l\dot{\theta} \cos \beta - \dot{\phi} r = 0$$

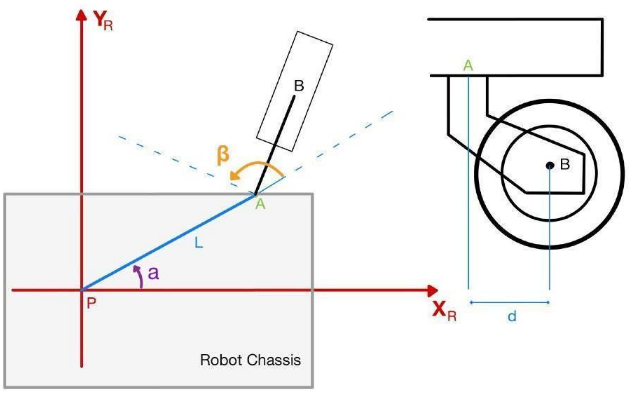
In a compact form and considering the velocity of the robot respect to the global reference frame, the kinematic constraints of a standard fixed wheel can be defined as follow:

$$\begin{bmatrix} \cos \alpha + \beta & \sin \alpha & l \sin(\beta) \end{bmatrix} R(\theta) \dot{\xi}_I = 0$$

$$\begin{bmatrix} \sin \alpha + \beta & -\cos \alpha + \beta & -l \cos(\beta) \end{bmatrix} R(\theta) \dot{\xi}_I - \dot{\phi} r = 0$$

**Castor Wheel**

The same approach used to derive the kinematic constraints of standard fixed wheel could be used for the castor wheel:



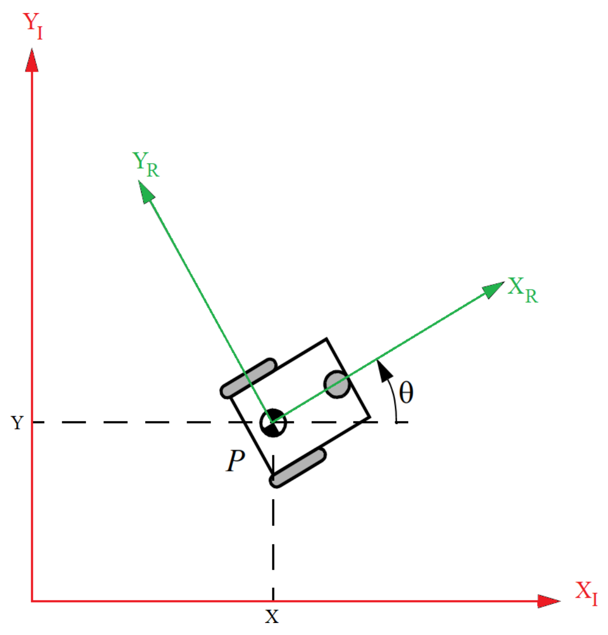
- alpha the angle of vector pointing A respect to X-axis of robot reference frame;
- $\beta$ is the angle between the sagittal plane of the wheel and the vector pointing A and it is variable respect to time;
- $L$ is the distance between the center of the wheel and the origin of the robot reference frame;
- d is the distance from the point A (which is the point in which each wheel can rotate) to the point B (the center of the wheel).

$$\cos \alpha + \beta \quad \sin \alpha l \sin(\beta) \; R(\theta) \dot{\xi} I \; + d\dot{\beta} = 0$$

$$\sin \alpha + \beta \quad - \cos \alpha + \beta \quad -l \cos(\beta) \quad R(\theta)\dot{\xi} I \; - \; \dot{\varphi} r = 0$$

Notice that the pure rolling constraint is the same of the standard wheel because the offset on the rotation axis do not influence the parallel motion in the wheel plane; on the other hand, the no sliding constraint present an extra term which allows to satisfy this constraint, controlling the steering velocity of the wheel. For these reasons, castor wheels are also known as omnidirectional wheels.

## Kinematic Model



Deriving a model for the whole robot’s motion is a bottom-up process. Each individual wheel contributes to the robot’s motion and, at the same time, imposes constraints on robot motion. Wheels are linked together, based on robot chassis geometry, and therefore their constraints are combined to form limitations on the overall motion of the robot chassis. But the forces and constraints of each wheel must be expressed with respect to a consistent reference frame.

In order to specify the position of the robot on the plane, a relationship can be established between the global reference frame of the plane and the local reference frame of the robot. To specify the position of the robot, choose a point P on the robot chassis as its position reference point. The basis defines two axes relative to P on the robot chassis and is thus the robot's local reference frame. The position of P in the global reference frame is specified by coordinates x and y, and the angular difference between the global and local reference frames is given by $\theta$. We can describe the pose of the robot as a vector with these three elements:

$$\xi_I = \begin{bmatrix} X \\ Y \\ \theta \end{bmatrix}$$

After that, all kinematic constraints given by each wheel must be referred to the robot reference frame in order to get the kinematic model of the robot. Considering kinematic constraints of standard and castor wheel discussed above the kinematic model of differential drive robot is derived:

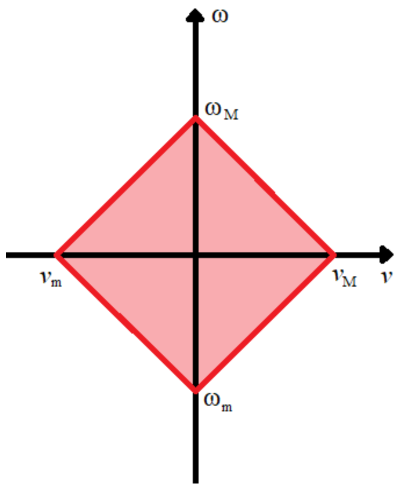
$$\xi_I = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = R^T(\theta) \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 1 \\ \frac{1}{2l} & -\frac{1}{2l} & 0 \end{bmatrix} \begin{bmatrix} r\dot{\phi}_1 \\ r\dot{\phi}_2 \\ 0 \end{bmatrix}$$

where $R^T(\theta)$ is the rotation matrix which gives us the robot reference frame orientation respect to the global reference frame (inertial frame).

## Control Inputs

The control input parameters are the magnitude of the linear velocity vector referred to the robot reference frame and the angular velocity of the robot chassis. All physical systems are exposed to saturation problems because all the input of the system cannot assume infinite values. Also, for the case of mobile robots, control input values must lay in the admissible control space.

For the differential drive robot, it's easy to show that the admissible control input is:



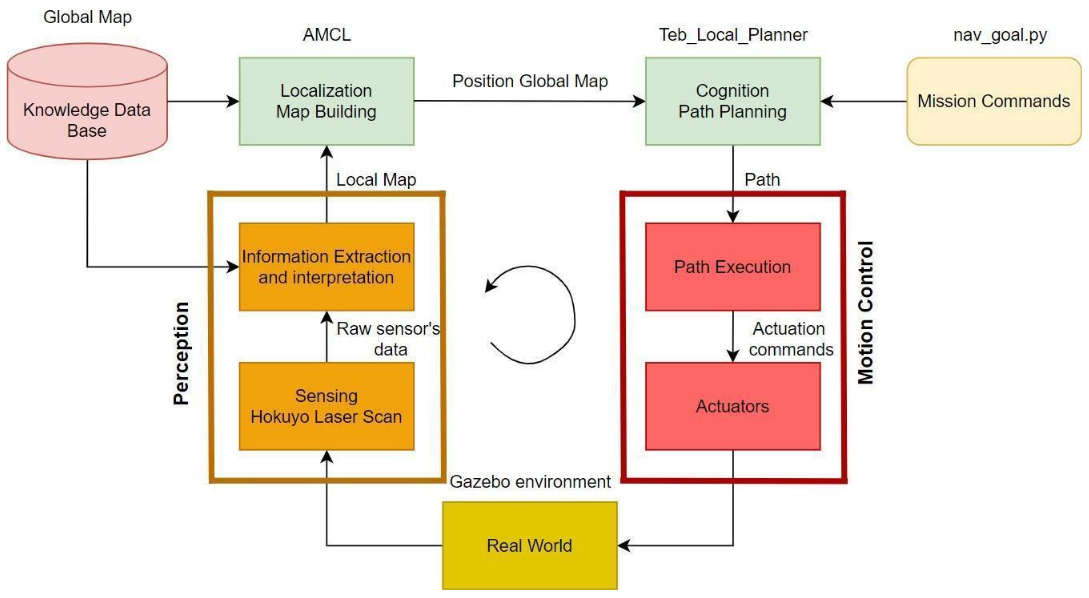
In the study case, it has been set the following control

$$\omega \in [-1.5, 1.5]\ rad/s$$

$$v \in [-0.18, 0.7]\ m/s$$

So the robot can rotate in place (as all differential drive robots) and can move both forward and backward.

## Control Scheme (loop control)



The most important feature of conventional mobile robotics is perception: mobile robots can travel across much of earth's surfaces, but they cannot perceive the world nearly as well as humans so they must be equipped with sensors in order to sense the environment where they are moving. Indeed, perception is more than sensing: it is also the interpretation of sensed data in meaningful ways. Using both interpretation of sensor's data and a global map built previously with a SLAM algorithm, it is possible to know with a certain probability (given by covariance matrices) the pose of the robot in the environment (localization problem). Robot pose is used to find a free-obstacles path in order to avoid collision with obstacles. Obviously, the path planner needs to know the goal position that has to be reached. The path found gives the actuation commands which must be sent to the motors that actuates the robot's wheels.

# Robot Operating System – ROS

ROS is an open-source, meta-operating system for fixed or mobile robots. It provides the ser- vices expected from an operating system, including hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers.

# URDF

The Unified Robotic Description Format (URDF) is an XML file format used in ROS to describe all elements of a robot through specific tags. Even if URDFs are a useful and standardized format in ROS, they are lacking many features, especially in working with Gazebo environments. Thus, the Simulation Description Format (SDF) has been introduced, offering a complete tool to efficiently describe everything from the world level down to the robot level.

In the study case, the robot model has been realized with Fusion 360, and then the conversion to the URDF was make feasible using a tool in python, that allows us to directly convert through Fusion 360 file to URDF.

```
<?xml version="1.0" ?>
<robot name="robot_description" xmlns:xacro="http://www.ros.org/wiki/xacro">
  <xacro:property name="cameraSize" value="0.05"/>
  <xacro:property name="cameraMass" value="0.1"/>
  <xacro:include filename="$(find medbot)/urdf/provanew.gazebo" />
  <xacro:include filename="$(find medbot)/urdf/materials.xacro" />
  <!--Dummy base link-->
  <link name="chassis">
  </link>
  <joint name="base_link_to_base" type="fixed">
    <parent link="chassis"/>
    <child link="base_link" />
    <origin rpy="0.0 0 0" xyz="0.15 0 0"/>
 </joint>
  <!--Base Link-->
  <link name='base_link'>
    <inertial>
      <mass value="9.8129" />
      <origin xyz="-0 0 0" rpy="0 0 0"/>
      <inertia
        ixx="1.0087"     ixy="-6.05459e-06"     ixz="-0.0112309"     iyy="11.3554"     iyz="-0.000118317"
izz="0.684012"/>
    </inertial>
    <collision name='collision'>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://medbot/meshes/base_link.stl" scale="0.001 0.001 0.001"/>
      </geometry>
      <surface>
        <friction>
          <ode>
            <mu>0</mu>
            <mu2>0</mu2>
            <slip1>1.0</slip1>
            <slip2>1.0</slip2>
          </ode>
        </friction>
      </surface>
    </collision>
    <visual name='base_link_visual'>
      <origin xyz="-0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://medbot/meshes/base_link.stl" scale="0.001 0.001 0.001"/>
      </geometry>
    </visual>
  </link>
  <!--Left Standard Fixed Wheel-->
  <link name="left_wheel">
    <collision name="collision">
      <origin xyz="0.15 -0.201 -0.12" rpy="-0 0 0" />
      <geometry>
        <mesh filename="package://medbot/meshes/left_wheel_1.stl" scale="0.001 0.001 0.001"/>
      </geometry>
    </collision>
    <visual name="left_wheel_visual">
      <origin xyz="0.15 -0.201 -0.12" rpy="0 0 0" />
      <geometry>
      <mesh filename="package://medbot/meshes/left_wheel_1.stl" scale="0.001 0.001 0.001"/>
      </geometry>
    </visual>
    <inertial>
      <origin xyz="2.7755575615628914e-17 0.024999999999999994 0.0" rpy="0 0 0"/>
      <mass value="2.1036104408437253" />
      <inertia
        ixx="0.008011" iyy="0.015146" izz="0.008011" ixy="0.0" iyz="-0.0" ixz="0.0"/>
    </inertial>
  </link>
  <joint type="continuous" name="left_wheel_hinge">
    <origin xyz="-0.15 0.201 0.12" rpy="0 0 0"/>
    <parent link="base_link"/>
    <child link="left_wheel"/>
    <axis xyz="0.0 1.0 0.0"/>
    <limit effort="10000" velocity="1000"/>
    <joint_properties damping="1.0" friction="1.0"/>
  </joint>
  <!--Right Standard Fixed Wheel-->
  <link name="right_wheel">
    <collision name="collision">
      <origin xyz="0.15 0.201 -0.12" rpy="0 -0 0" />
      <geometry>
        <mesh filename="package://medbot/meshes/right_wheel_1.stl" scale="0.001 0.001 0.001"/>
      </geometry>
    </collision>
    <visual name="right_wheel_visual">
      <origin xyz="0.15 0.201 -0.12" rpy="0 -0 0" />
      <geometry>
        <mesh filename="package://medbot/meshes/right_wheel_1.stl" scale="0.001 0.001 0.001"/>
      </geometry>
    </visual>
    <inertial>
      <origin xyz="0.0 -0.025000000000000005 0.0" rpy="0 0 0"/>
      <mass value="2.1036104408437253" />
      <inertia
        ixx="0.008011" iyy="0.015146" izz="0.008011" ixy="0.0" iyz="-0.0" ixz="0.0"/>
    </inertial>
  </link>
  <joint type="continuous" name="right_wheel_hinge">
    <origin xyz="-0.15 -0.201 0.12" rpy="0 0 0"/>
    <parent link="base_link"/>
    <child link="right_wheel"/>
    <axis xyz="0.0 1.0 0.0"/>
    <limit effort="10000" velocity="1000"/>
    <joint_properties damping="1.0" friction="1.0"/>
  </joint>
 </robot>
```

Now the robot can be refined by adding sensors and cameras according to the project's goals. As already said, a laser-based sensor already implemented in ROS called *Hokuyo Laser* has been added so that the robot can sense the environment, build its map and detect obstacles as described in the following sections. Here the code:

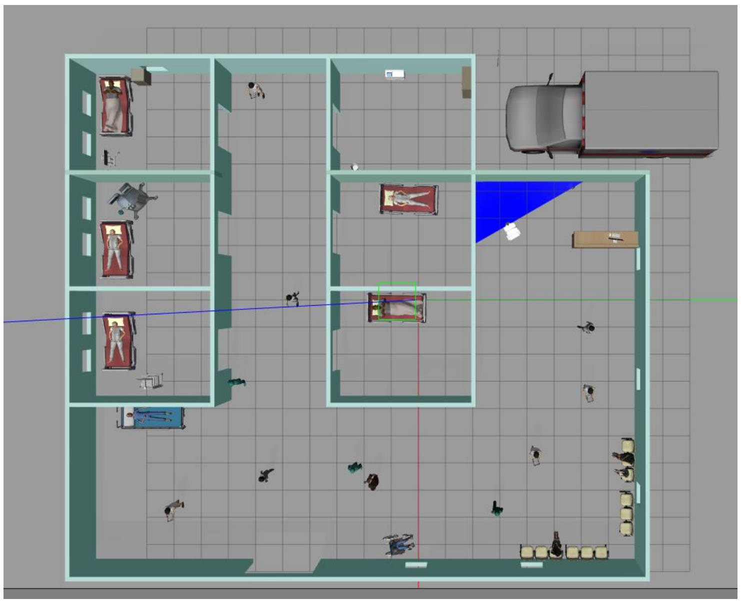
```
<!-- Hokuyo Laser Scan -->
 <gazebo reference="hokuyo">
  <sensor type="gpu_ray" name="head_hokuyo_sensor">
   <pose>0 0 0 0 0 0</pose>
   <visualize>true</visualize>
   <update_rate>40</update_rate>
   <ray>
    <scan>
     <horizontal>
      <samples>720</samples>
      <resolution>1</resolution>
      <min_angle>-1.570796</min_angle>
      <max_angle>1.570796</max_angle>
     </horizontal>
    </scan>
    <range>
     <min>0.10</min>
     <max>30.0</max>
     <resolution>0.01</resolution>
    </range>
    <noise>
     <type>gaussian</type>
     <!-- Noise parameters based on published spec for Hokuyo laser
        achieving "+-30mm" accuracy at range < 10m.  A mean of 0.0m and
        stddev of 0.01m will put 99.7% of samples within 0.03m of the true
        reading. -->
     <mean>0.0</mean>
     <stddev>0.01</stddev>
    </noise>
   </ray>
   <plugin name="gazebo_ros_head_hokuyo_controller" filename="libgazebo_ros_gpu_laser.so">
    <topicName>/mybot/laser/scan</topicName>
    <frameName>hokuyo</frameName>
   </plugin>
  </sensor>
</gazebo>
```

In addition, a camera can be added, already implemented in ROS, that can be useful for debugging during simulations and build easily and quickly the map of the environment.

```
<!--Camera-->
<gazebo reference="camera">
  <material>Gazebo/Green</material>
  <sensor type="camera" name="camera1">
    <update_rate>30.0</update_rate>
    <camera name="head">
      <horizontal_fov>1.3962634</horizontal_fov>
      <image>
        <width>800</width>
        <height>800</height>
        <format>R8G8B8</format>
      </image>
      <clip>
        <near>0.02</near>
        <far>300</far>
      </clip>
    </camera>
    <plugin name="camera_controller" filename="libgazebo_ros_camera.so">
      <alwaysOn>true</alwaysOn>
      <updateRate>0.0</updateRate>
      <cameraName>mybot/camera1</cameraName>
      <imageTopicName>image_raw</imageTopicName>
      <cameraInfoTopicName>camera_info</cameraInfoTopicName>
      <frameName>camera</frameName>
      <hackBaseline>0.07</hackBaseline>
      <distortionK1>0.0</distortionK1>
      <distortionK2>0.0</distortionK2>
      <distortionK3>0.0</distortionK3>
      <distortionT1>0.0</distortionT1>
      <distortionT2>0.0</distortionT2>
    </plugin>
  </sensor>
</gazebo>
```

# Gazebo Environment

In the study case, we recreated an indoor environment:



We decided to create from scratch a new world in Gazebo. First, we built the planimetry, taking inspiration from a real Hospital planimetry. Our planimetry is composed of 5 hospital wards, a large hall where patients can wait their turn and a hallway in which the robot mostly would operate. The place in which the robot is spawned is near a charging station, in a room at the end of the hallway.

The next step was introducing into this empty world, objects and models widely used in the hospital field, such as beds, medical devices, and generic furniture as well. (i.e., chairs for accommodate people that are waiting their turn, desks, warning signals etc..).

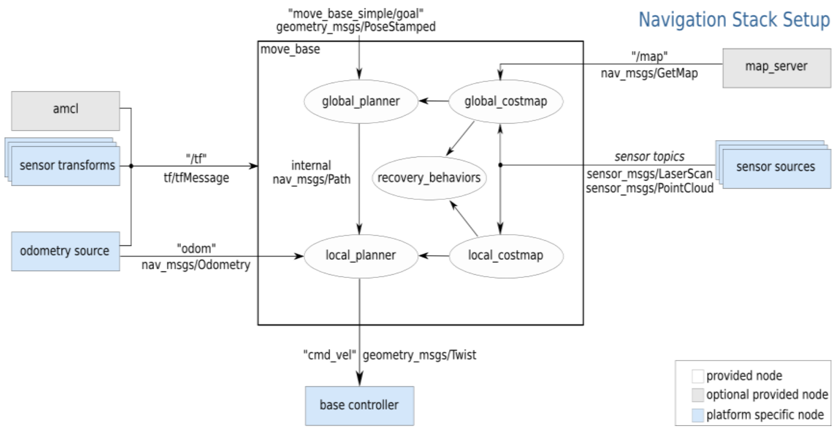
Special attention has been given to the layout of the rooms. In fact, if correctly settled, it allows the robot to perform in a better way during its tasks.

# Navigation Stack

The Navigation Stack takes information from odometry, and sensor streams and gives as output velocity commands which will be sent to a mobile base. There are three main hardware requirements that restrict its usage:

- it can be used for both differential drive and holonomic wheeled robots and assumes that the mobile base is controlled by sending desired velocity commands to achieve in the form of x velocity, y velocity and angular velocity;
- a planar laser, appropriately mounted on the mobile base, is required (this laser is used to mapping the building and for localization);
- best performance of the navigation stack on square or circular robots.

To use the navigation stack, it must be configured as shown



First of all, the navigation stack requires that the system publish information about the relationships between links coordinate frames of the robot (it is done using "TF" package) that will be used to manipulate information from sensors, to avoid obstacles in the world. Moreover, the navigation stack assumes that control inputs of the robot are sent by a base controller using a **geometry_msgs/Twist** message, so it will send linear velocity referred to the robot reference frame and angular velocity of the robot chassis; after that, it converts them into motor commands to send to the mobile base.
In the study case, the navigation stack requires a map of the environment in which the robot will move, so it is necessary to build the map using a SLAM algorithm, as discussed in the following section.

# Base Controller

A low-level controller is mandatory, in order to convert control inputs in term of linear and angular velocity into wheel's angular velocities $\dot{\varphi}_1, \dot{\varphi}_2$. In fact, by inverting the kinematics model described

a simple base controller can be defined. A differential drive controller plugin comes with ROS and it can be found in the **libgazebo_ros_diff_drive.so** file and can be included in the used package by customizing different parameters based on the robot considered, such as:

- controller frequency;
- left/right wheel's joint reference;
- wheel separation and diameter;
- maximum torque applied;
- odometry and robot reference frame.

```
<gazebo>
  <plugin name="differential_drive_controller" filename="libgazebo_ros_diff_drive.so">
    <legacyMode>false</legacyMode>
    <alwaysOn>true</alwaysOn>
    <updateRate>20</updateRate>
    <leftJoint>left_wheel_hinge</leftJoint>
    <rightJoint>right_wheel_hinge</rightJoint>
    <wheelSeparation>0.45</wheelSeparation>
    <wheelDiameter>0.24</wheelDiameter>
    <torque>10000</torque>
    <commandTopic>cmd_vel</commandTopic>
    <odometryTopic>odom</odometryTopic>
    <odometryFrame>odom</odometryFrame>
    <robotBaseFrame>chassis</robotBaseFrame>
    <rosDebugLevel>na</rosDebugLevel>
    <publishWheelTF>false</publishWheelTF>
    <publishOdomTF>true</publishOdomTF>
    <publishWheelJointState>false</publishWheelJointState>
    <wheelAcceleration>0</wheelAcceleration>
    <wheelTorque>10</wheelTorque>
    <publishTf>1</publishTf>
    <odometrySource>world</odometrySource>
  </plugin>
 </gazebo>
```

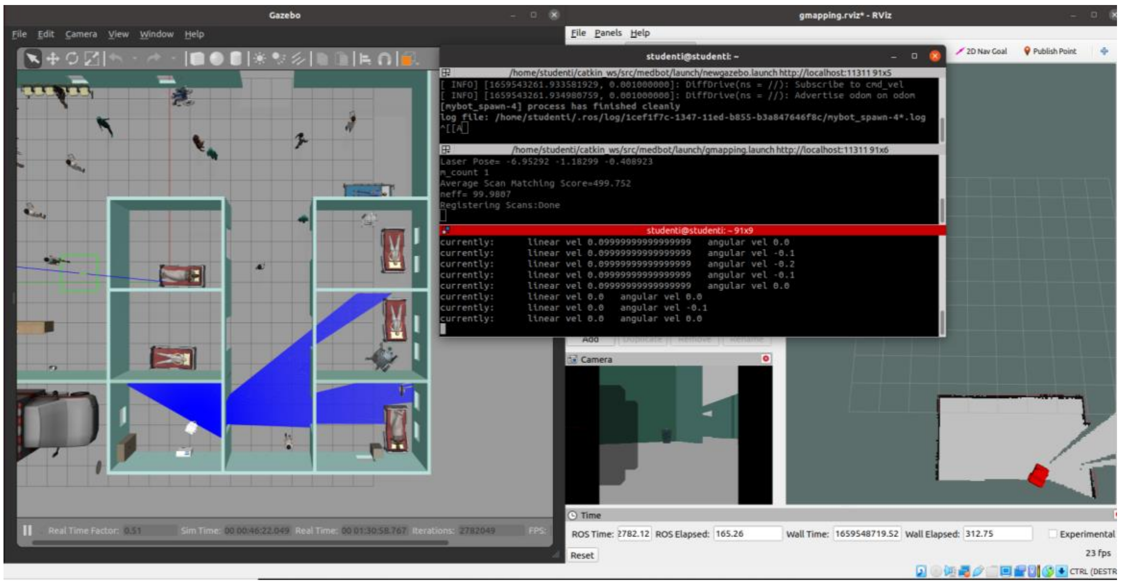
# Slam Mapping

After realizing the structure of the hospital, it was necessary to map the environment. This was feasible using the hokuyo laser scan placed on the base of the robot. To get the best mapping, this process needs a long time to be done. Correctly mapping of the environment requires slow movements of the robot and no fast rotations around the z axis. During this process, the linear velocity of our robot has been limited in a range of [-1, 1] m/s, while the rotation velocity in a range of [-0.2, 0.2] rad/s. For gaining a high accuracy and high level of details in our map, we have driven the robot also in narrow passages. During this phase the robot has covered a path that allowed him to scan the entire hospital, detecting static objects and perimeter walls of the hospital wards.

The package used to sense the indoor environment is **gmapping** which provides laser-based SLAM (Simultaneous Localization and Mapping). Using slam_gmapping, a 2D occupancy grid map can be created, from laser and pose data collected by the robot.

In order to move the robot into the environment, a keyboard teleoperation node has been used, which allows to move the robot using the following keys:

- w: increment linear velocity forward;
- x: increment linear velocity backward;
- d: increment angular velocity in the clockwise direction;
- a: increment angular velocity in the counterclockwise direction;
- s: stop the robot.



```
/home/studenti/catkin_ws/src/medbot/launch/newgazebo.launch http://localhost:11311 91x5
[ INFO] [1659543261.933581929, 0.001000000]: DiffDrive(ns = //): Subscribe to cmd_vel
[ INFO] [1659543261.934980759, 0.001000000]: DiffDrive(ns = //): Advertise odom on odom
[mybot_spawn-4] process has finished cleanly
log file: /home/studenti/.ros/log/1cef1f7c-1347-11ed-b855-b3a847646f8c/mybot_spawn-4*.log
^[[A
```

```
/home/studenti/catkin_ws/src/medbot/launch/gmapping.launch http://localhost:11311 91x6
Laser Pose= -6.95292 -1.18299 -0.408923
m_count 1
Average Scan Matching Score=499.752
neff= 99.9807
Registering Scans:Done
```

```
studenti@studenti: ~ 91x9
currently:      linear vel 0.09999999999999999   angular vel 0.0
currently:      linear vel 0.09999999999999999   angular vel -0.1
currently:      linear vel 0.09999999999999999   angular vel -0.2
currently:      linear vel 0.09999999999999999   angular vel -0.1
currently:      linear vel 0.09999999999999999   angular vel 0.0
currently:      linear vel 0.0   angular vel 0.0
currently:      linear vel 0.0   angular vel -0.1
currently:      linear vel 0.0   angular vel 0.0
```

ROS Time: 2782.12 ROS Elapsed: 165.26 Wall Time: 1659548719.52 Wall Elapsed: 312.75

*Figure: Keyboard teleoperations*

The 2D occupancy grid map generated by **gmapping** node and how it is seen in Rviz: gray areas correspond to the presence of obstacles while white one corresponds to free obstacle places.

Once the map is built, gmapping node automatically create a .yaml file in which all the map parameters are defined:

```
<!--Launch slam_gmapping node-->
<node pkg="gmapping" type="slam_gmapping" name="slam_gmapping" output="screen">
  <param name="base_frame" value="$(arg base_frame)"/>
  <param name="odom_frame" value="$(arg odom_frame)"/>
  <param name="map_update_interval" value="0.5"/>
  <param name="maxUrange" value="6.0"/>
  <param name="maxRange" value="8.0"/>
  <param name="sigma" value="0.05"/>
  <param name="kernelSize" value="1"/>
  <param name="lstep" value="0.05"/>
  <param name="astep" value="0.05"/>
  <param name="iterations" value="5"/>
  <param name="lsigma" value="0.075"/>
  <param name="ogain" value="3.0"/>
  <param name="lskip" value="0"/>
  <param name="minimumScore" value="100"/>
  <param name="srr" value="0.01"/>
  <param name="srt" value="0.02"/>
  <param name="str" value="0.01"/>
  <param name="stt" value="0.02"/>
  <param name="linearUpdate" value="0.5"/>
  <param name="angularUpdate" value="0.436"/>
  <param name="temporalUpdate" value="-1.0"/>
  <param name="resampleThreshold" value="0.5"/>
  <param name="particles" value="100"/>

  <param name="xmin" value="-100.0"/>
  <param name="ymin" value="-100.0"/>
  <param name="xmax" value="100.0"/>
  <param name="ymax" value="100.0"/>

  <param name="delta" value="0.05"/>
  <param name="llsamplerange" value="0.01"/>
  <param name="llsamplestep" value="0.01"/>
  <param name="lasamplerange" value="0.005"/>
  <param name="lasamplestep" value="0.005"/>
  <remap from="scan" to="$(arg scan_topic)"/>
</node>
```

## AMCL

AMCL is a probabilistic localization system for a robot moving in a 2D map. It implements the adaptive (or KLD-sampling) Monte Carlo localization approach, which uses a particle filter to track the pose of a robot against a known map. AMCL takes in a laser-based map, laser scans, and transform messages, and outputs pose estimates. On startup, amcl initializes its particle filter according to the parameters provided.

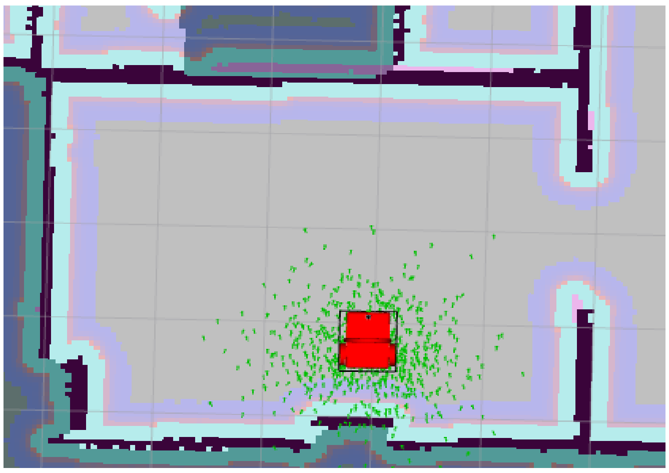


*Figure : AMCL on the start-up phase*

An estimation of robot pose is defined as vectors whose origins are the estimated position of the robot whereas directions are their estimated orientation. The smaller the area occupied by vectors; the more accurate estimation of robot pose is given. Moreover, to reduce the area, an initial pose of the robot should be provided allowing the algorithm to localize the robot in the map properly.

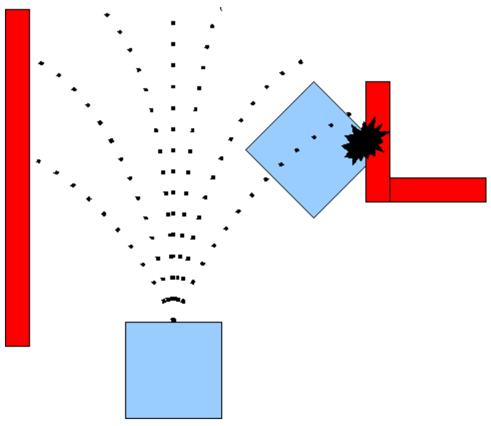
## BASE LOCAL PLANNER

The base local planner provides a controller that drives a mobile robot in the plane: using a map, the planner creates a kinematic trajectory for the robot to reach from a start point the goal location. On the way, the planner creates, at least locally, around the robot, a value function, represented as a grid map. This value function encodes the costs of traversing through the grid cells. In fact, in order to score trajectories efficiently, a map grid is used. For each control cycle, a grid is created around the robot (the size of the local cost map), and the global path is mapped into this area. This means certain of the grid cells will be marked with distance 0 to a path point, and distance 0 to the goal. A propagation algorithm then efficiently marks all other cells with their manhattan distance to the closest of the points marked with zero. This map grid is then used for the scoring of trajectories.

## DWA (Dynamic – Window - Approach)

At the beginning, the environment in which the Medbot could move has been set with static obstacles and the so-called *Dynamic Window Approach* (DWA) is well suited for the navigation purpose: given a global plan to follow and a cost map, this local planner produces velocity commands to send to the mobile base. It is already implemented in a ROS package called *dwa_local_planner* package. The basic idea of the DWA algorithm can be summed as:

- discretely sample in the robot’s control space ($\dot{x}$, $\dot{y}$, $\dot{\theta}$);
- For each sampled velocity, perform forward simulation from the robot’s current state in order to predict what would happen if the sampled velocity were applied for some (short) period of time;
- evaluate (score) each trajectory resulting from the forward simulation, using a metric that incorporates characteristics such as: proximity to obstacles, proximity to the goal, proximity to the global path, and speed. Discard illegal trajectories (those that collide with obstacles);
- pick the highest-scoring trajectory and send the associated velocity to the mobile base;
- clean and repeat.

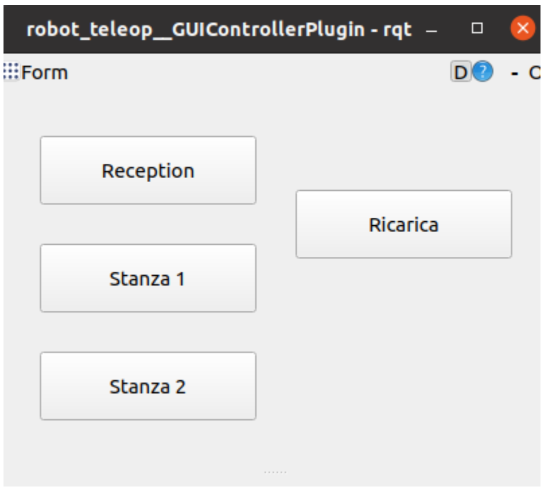


## TEB (Time – Elastic – Local - Planner)

This base local planner, implemented in the **teb_local_planner** package, allows an online optimal local trajectory planner for navigation and control of mobile robots as a plugin for the ROS navigation package. The initial trajectory generated by a global planner is optimized at runtime w.r.t. minimizing the trajectory execution time (time-optimal objective), separation from obstacles and compliance with kinodynamic constraints such as satisfying maximum velocities and accelerations. The robot used can be either holonomic or non-holonomic. The optimal trajectory is efficiently obtained by solving a sparse scalarized multi-objective optimization problem. The user can provide weights to the optimization problem in order to specify the behavior in case of conflicting objectives. Since local planners such as the Timed-Elastic-Band get often stuck in a locally optimal trajectory as they are unable to transit across obstacles, an extension is implemented: a subset of admissible trajectories of distinctive topologies is optimized in parallel. The local planner can switch to the current globally optimal trajectory among the candidate set.

## Move Base Controller

The main task of the robot is to move in the environment avoiding fixed and mobile obstacles. To carry out these tasks, a graphic interface has been created that allows to indicate the position in the hospital that the robot has to reach. Each button corresponds to a room or zone in the environment. The interface is a node that writes the goal on a topic. A new node called "listnav.py" is subscribed to this topic, reads the environment, and transforms it into coordinates [x, y, z, r, p, y] to be written to move_base. We set the 2D target position using MoveBaseAction which is a SimpleActionServer implemented by the move_base node, an action server with a single target policy, which accepts geometric_msgs / PoseStamped message type targets. The Simple-ActionClient interface is used to communicate with this node. Thus, the so-called move_base node tries to achieve a desired pose by combining the global and local motion planner to perform a navigational task that includes obstacle avoidance.

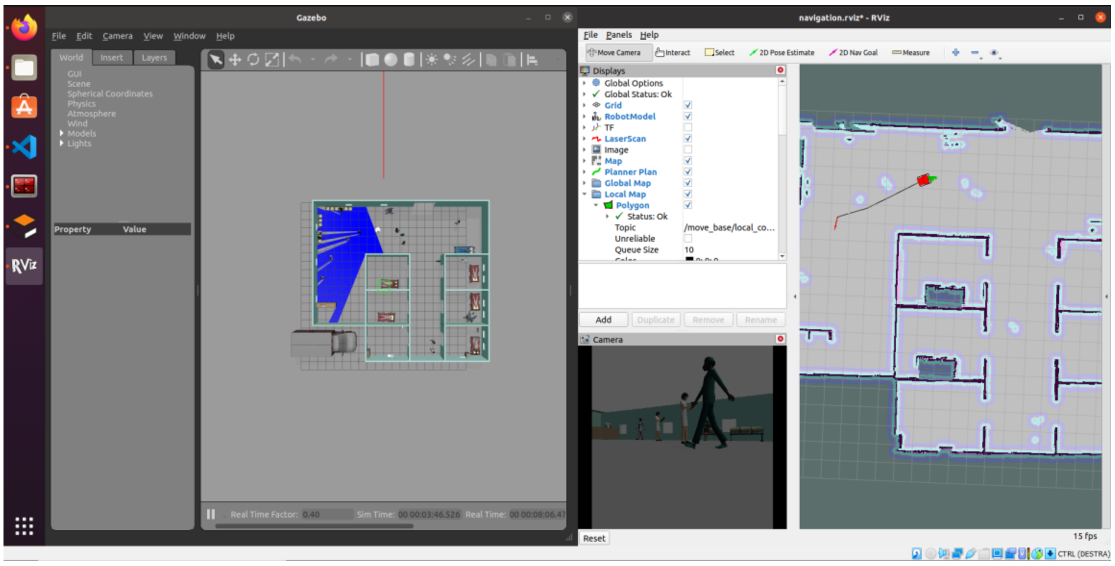


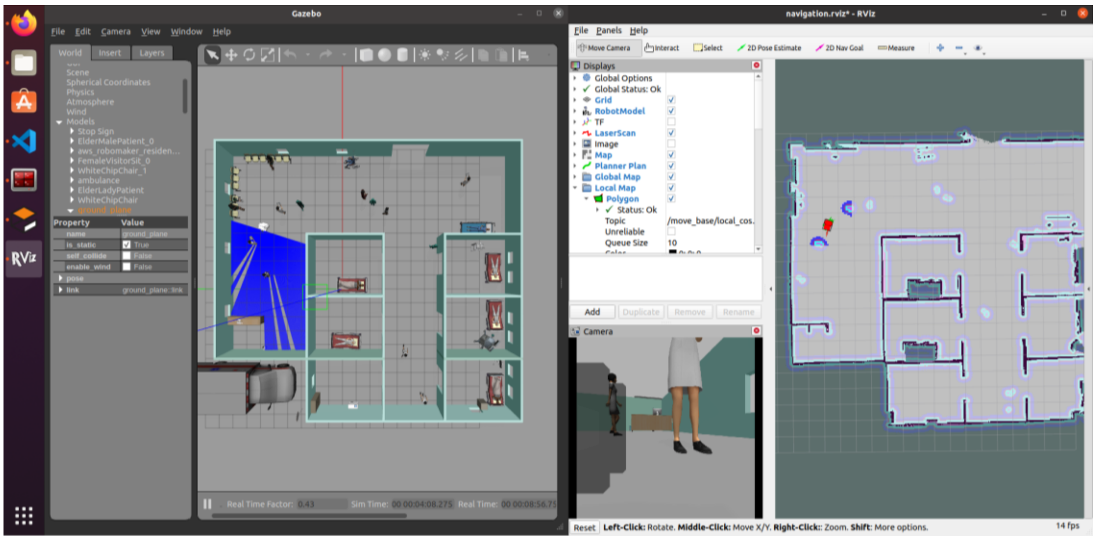
# Simulation and Results

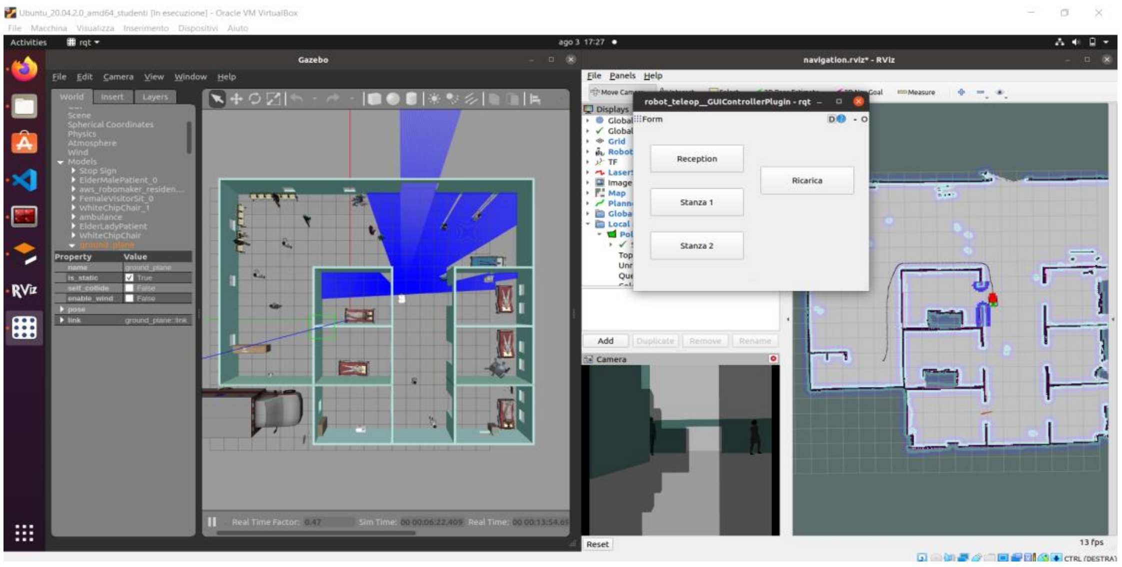
In order to simulate the behaviour of the robot in the hospital and verify if it accomplish the task, the following command must be typed on different shell windows:

- launch the gazebo world –> **$ roslaunch medbot gazebo.launch**;
- launch the move base launch file –> **$ roslaunch medbot amcl.launch;**
- launch the graphical control interface -> **$ rqt --standalone robot_teleop.rqt_plugin.GUIControllerPlugin**
- run the navigation goal node –> **$ rosrun medbot listnav.py.**

Simulation shows that the robot moves into the hospital avoiding both fixed and mobile obstacles and accomplish the task assigned: in particular, when it sense the vicinity of actors using the laser scan, it stops or change the path planned by the global and local planner. Different frames of the simulation are shown in the following figures:



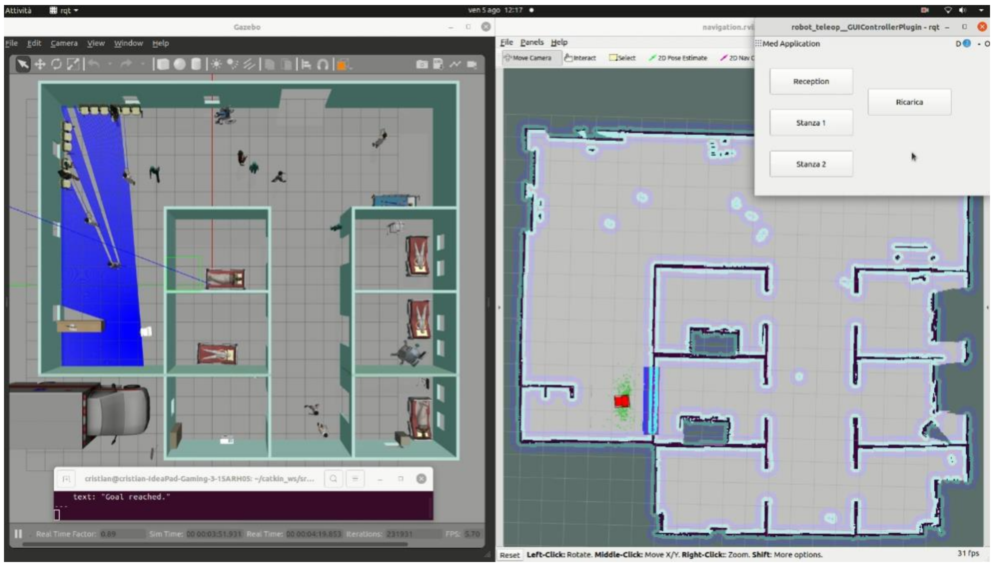




.

Below are the final poses achieved by the robot in the various environments.

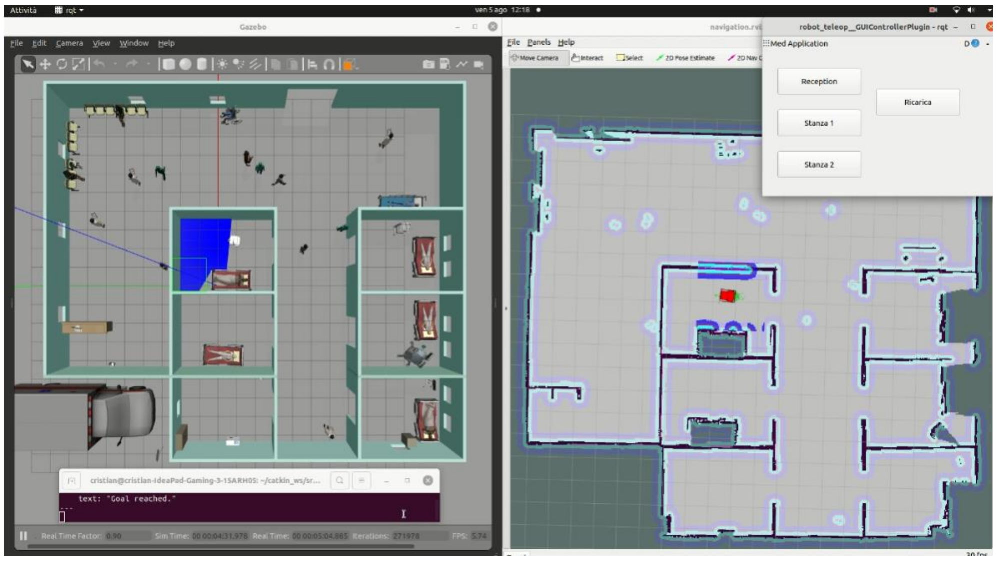
*Reception:*


Gazebo
navigation.rviz
robot_teleop__GUIControllerPlugin - rqt
Med Application
Reception
Stanza 1
Stanza 2
Ricarica
Move Camera
Interact
Select
2D Pose Estimate
Reset Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click: Zoom. Shift: More options.
31 fps

cristian@cristian-IdeaPad-Gaming-3-15ARH05: ~/catkin_ws/sr...

```
text: "Goal reached."
---
```

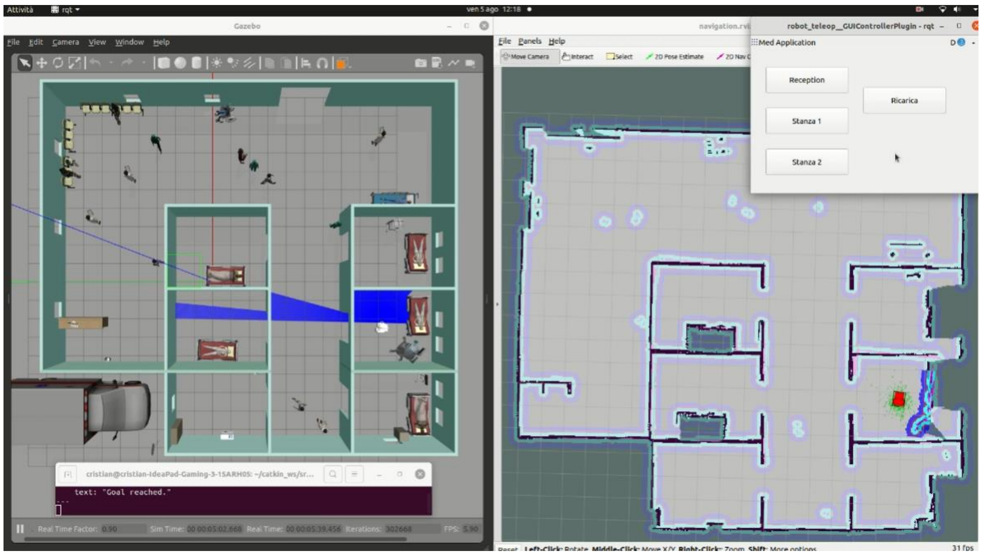
*Room 1:*



cristian@cristian-IdeaPad-Gaming-3-15ARH05: ~/catkin_ws/sr...

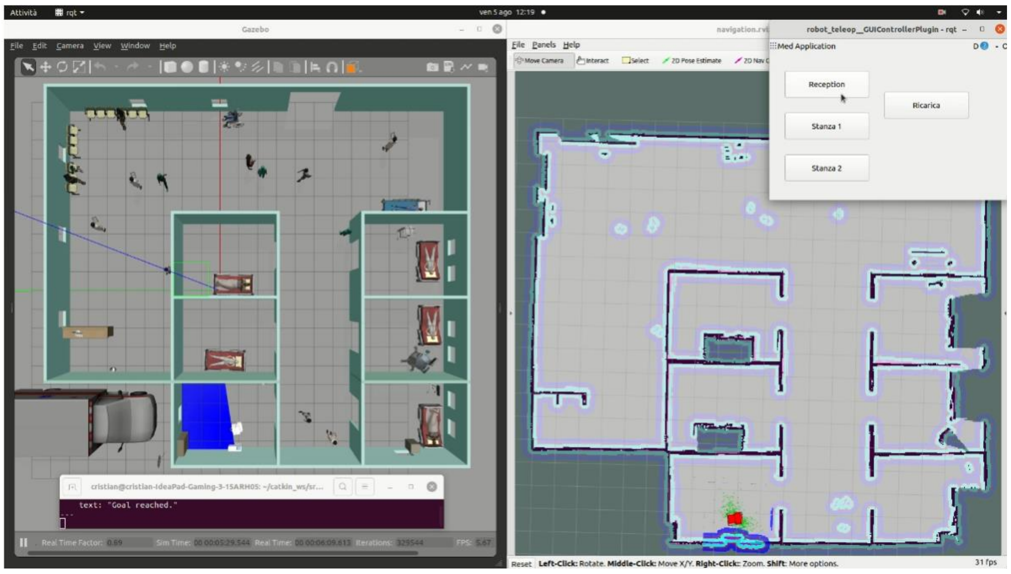
```
text: "Goal reached."
---
```

*Room 2:*



```
text: "Goal reached."
---
```

*Charging Room:*


Gazebo
navigation.rviz
robot_teleop__GUIControllerPlugin - rqt
Med Application
Reception
Ricarica
Stanza 1
Stanza 2
Move Camera
Interact
Select
2D Pose Estimate
2D Nav Goal
cristian@cristian-IdeaPad-Gaming-3-15ARH05: ~/catkin_ws/sr...
Reset Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click: Zoom. Shift: More options.
31 fps

```
text: "Goal reached."
---
```

One of the objectives of this study is to analyze the performance of the ROS local planners with a differential drive robot. After conducting the tests and analyzing the results, it can be concluded that both TEB and DWA local planners are an effective path planning strategy for a differential drive robotic system.
Both topics allow navigation in a dynamic environment with moving objects.
When considering to the obstacle avoidance capability and the higher reactivity, the TEB local planner can be distinguished as the best option. Therefore, TEB local planner can be considered as a better option for differential drive robot.

# Conclusion and Future Works

In this study we managed to move a robot in an autonomous way.

We started by designing our robot in a CAD environment with fusion 360, creating a robot with a shape suited to its objective: transporting objects from one department to another within a hospital.

We then implemented our "hospital" in gazebo from scratch, making an environment as realistic as possible. We then took care of making it navigate autonomously in a static environment.. At the end the robot is able to detect obstacles and drive past them by avoiding a collision

For navigation, we implemented two algorithms: DWA and TEB. As can be seen from the chapter "Simulation and results", navigation in a dynamic environment works with both algorithms. But TEB performs better in a highly populated environment unlike DWA.

We have also created a graphical interface that allows the robot to be controlled by choosing the environment it needs to reach.

For future work, navigation can be improved with better parameter settings. Certainly also a better design of the robot allows better navigation.

The experiment can be improved by adding an external motion capturing setup to measure the goal-reaching tolerances. Furthermore, the performances of local planners with different global planning technologies can also be studied to make a better comparison.

Another possible Improvement could be find in ROS. We were working on an entirely simulated environment, everything was run on a laptop, but what I sure is that ROS is a battery-hungry application. In the future, ROS could be run on a single-board computer like, which will handle all the applications needed to drive the robot autonomously.