POLYTECHNIC OF BARI

DEPARTMENT OF ELECTRICAL AND INFORMATION ENGINEERING

AUTOMATION ENGINEERING MASTER'S DEGREE

Course on ROBOTICS - MOBILE ROBOTICS

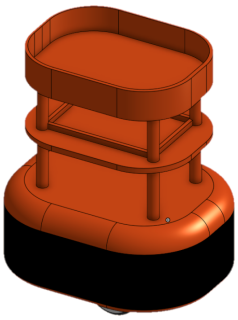
Prof. Engr. Luca **DE CICCO**

Engr. Carlo **CROCE**

Engr. Gioacchino **MANFREDI**

Project work:

Autonomous Navigation for BookBot



Students:
D'ALESSANDRO Vito Ivano
VENEZIA Antonio

ACADEMIC YEAR 2020-2021

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 of a bookstore and the task assigned to the robot is collecting books from known point of the map and leave them at the reception, avoiding collisions with obstacles.
The DDR is equipped with an Hokuyo Scanning Laser Rangefinder which is a small, affordable and accurate laser scanner that is perfect for robotic applications. It is used to to map the environment and localize the robot in the map. The localization algorithm used is Monte Carlo localization approach, which uses a particle filter to track the pose of a robot against a known map. The local planner used is the Time-Elastic-Band Local Planner (teb local planner) which is an online collision avoidance strategy for mobile robots useful to avoid dynamic obstacles too.
Video of the simulation is available on the following link:
`https://youtu.be/0HSZ3LU_FVI`
ROS package is available on the following git repository:
`https://github.com/ivano-dalessandro/bookbot`

Contents

1 Introduction

One of the ultimate goals of indoor mobile robotics research is to build robots that can safely carry out missions in populated environments: in particular, the example considered is a service-robot that assists humans in a bookstore, carrying books from one point to another with a safe path, computed at each time istant according to the environment status, in order to avoid collisions with obstacles. The mobile robot considered is a differential drive robot with two standard fixed wheels on the front and one castor wheel on the back; it is equipped by a laser which helps the localization of robot in the environment and allows to use Time-Elastic-Band Local Planner for navigation stack which works well with dynamic obstacles. SLAM (Simultaneous Localization and Mapping) algorithm has been used in order to know the global static map of the environment and helpful for the navigation stack, in particular the study case uses a **Gmapping** approach to build the map.

In summary:

In Sec.2 a 3D model of the custom robot is proposed with its main design parameters and a brief introduction to its kinematics model and constraints that are mandatory for designing the appropriate controller.

In Sec.**??** a brief introduction to ROS is presented, with a particular focus on the ROS Navigation Stack and its components

In Sec.4 simulation and results are presented.

2 The Prototype

Typically, the differential drive robot is a mobile robot with two motorized fixed standard wheels and one castor wheel and is used in indoor environments.

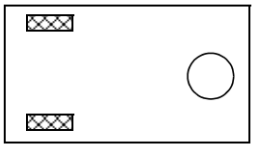


Figure 2.1: Scheme of DDR

The robot has been designed using Autodesk Inventor Pro 2022 which is a computer-aided design application for 3D mechanical design, simulation, visualization, and documentation developed by Autodesk.

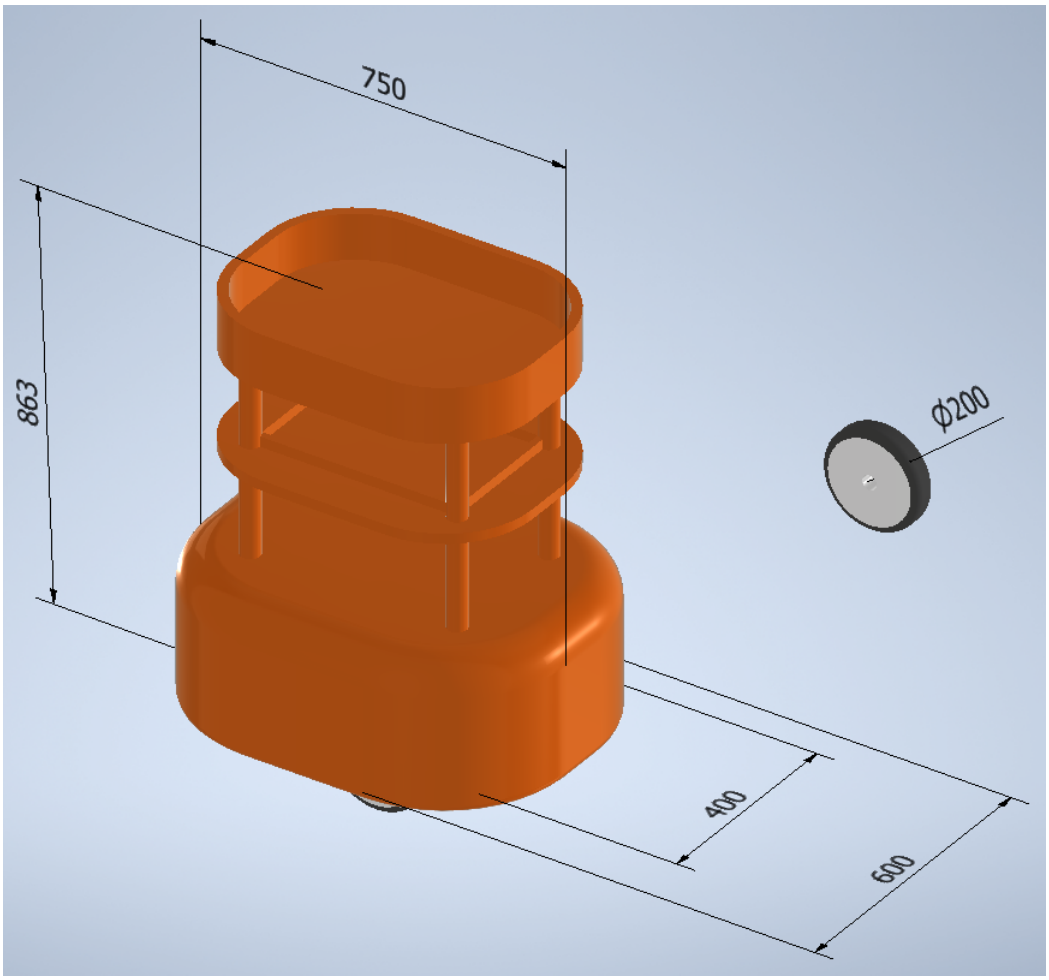


Figure 2.2: Main design parameters

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

	Value	Unit
Length	750	mm
Width	600	mm
Height	863	mm
Mass	9.8	Kg
I_{xx}	1	Kg m^2
I_{yy}	11.3	Kg m^2
I_{zz}	0.6	Kg m^2
Material	Plastic	ABS

Table 2.1: Design parameters and dynamic properties of the chassis

	Value	Unit
Diameter	200	mm
Width	50	mm
Mass	3.9	Kg
Wheel separation	400	mm
Material	Aluminium	

Table 2.2: Design parameters and dynamic properties of the wheels

2.1 Kinematics Constraints

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).

2.1.1 Standard Fixed Wheel

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 WMR's where the wheels are attached to motors and are used to drive and steer the robot.

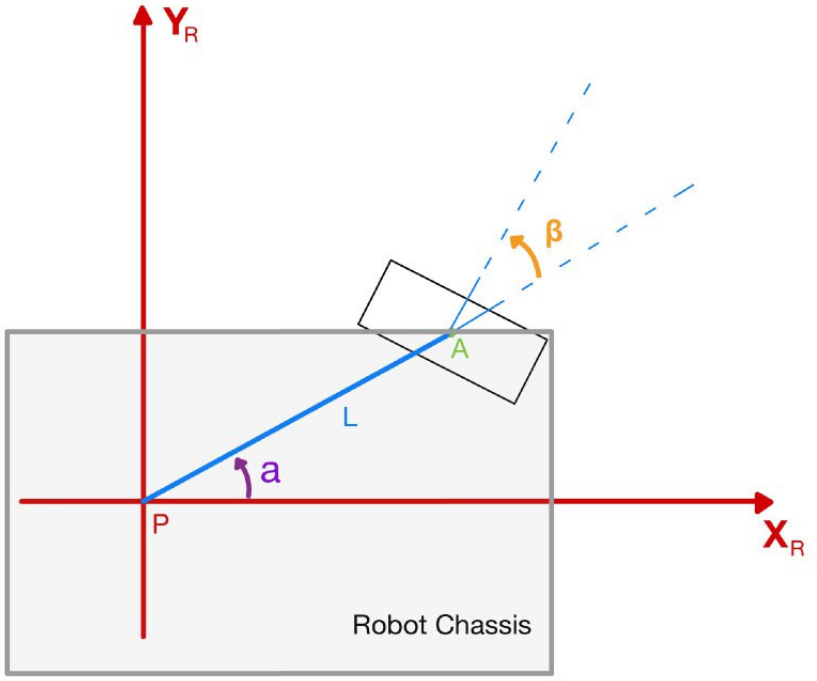


Figure 2.1: Standard Fixed Wheel

As can be seen in Fig.2.1, there are different parameters to consider:

- α is the angle of vector pointing A respect to X-axis of robot reference frame;
- β 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, refereed to robot reference frame, must be divided into two components (one along the X-axis and the other one 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:

$$\dot{x_R}\cos\alpha+\beta+\dot{y_R}\sin\alpha+\beta+\dot{\theta}l\sin\beta=0 \tag{1}$$

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 $\dot{\phi}r$) the following condition is obtained:

$$\dot{x_R}\sin\alpha+\beta-\dot{y_R}\cos\alpha+\beta-l\dot{\theta}\cos\beta-\dot{\phi}r=0 \tag{2}$$

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

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

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

2.1.2 Castor Wheel

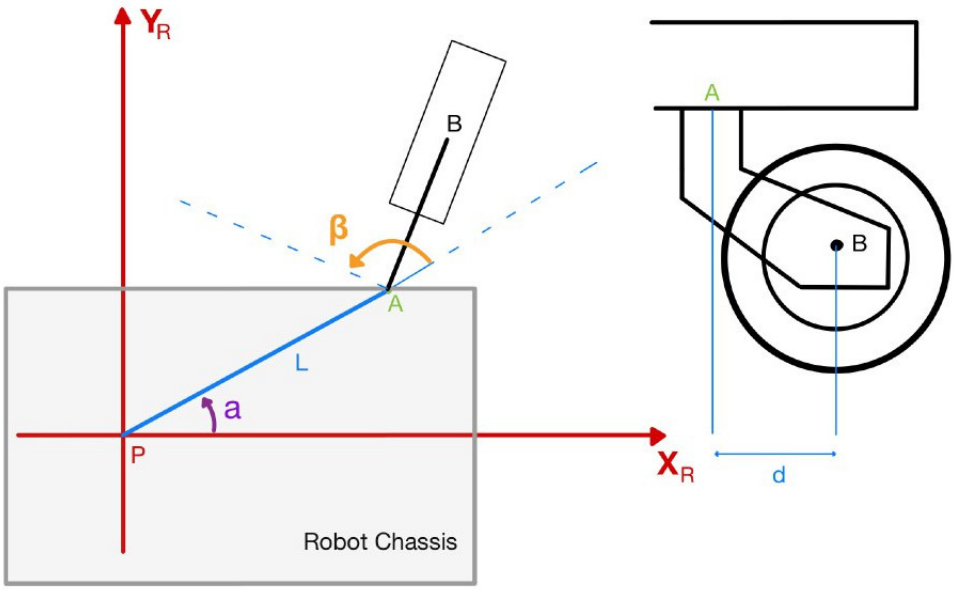


Figure 2.2: Castor Wheel

As can be seen in Fig.2.2, there are different parameters to consider:

- α is the angle of vector pointing A respect to X-axis of robot reference frame;
- β 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 respect to which wheel can rotate) to the point B (which is the center of the wheel).

The same approach used to derive the kinematic constraints of standard fixed wheel is used to derive the kinematic constraints of castor wheel, which are:

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

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

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

2.2 Kinematics 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 combine to form constraints 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 [1].

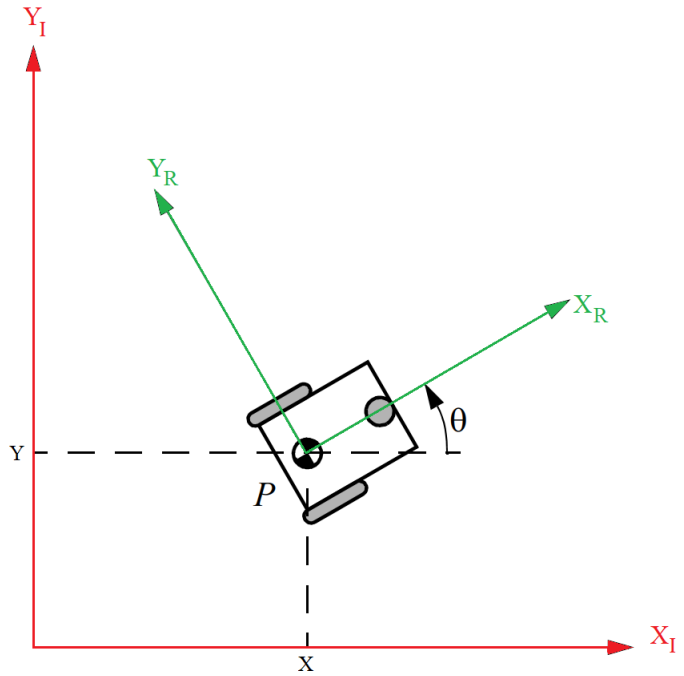


Figure 2.1: Robot and Global reference frame

In order to specify the position of the robot on the plane, it can be established a relationship between the global reference frame of the plane and the local reference frame of the robot, as shown in Fig.2.1. 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 θ. 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:

$$\xi_I = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = R^T(\theta) \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 1 \\ \frac{1}{2l} & -\frac{1}{2l} & 0 \end{bmatrix} \begin{bmatrix} r\dot{\phi}_1 \\ r\dot{\phi}_2 \\ 0 \end{bmatrix} \tag{3}$$

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

2.3 Admissible control input range

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 subject to saturation problem because all the input of the system cannot assume infinite values. Also for the case of mobile robot, control input values must lie in the admissible control space.
For the differential drive robot, it's easy to show that the admissible control input is:

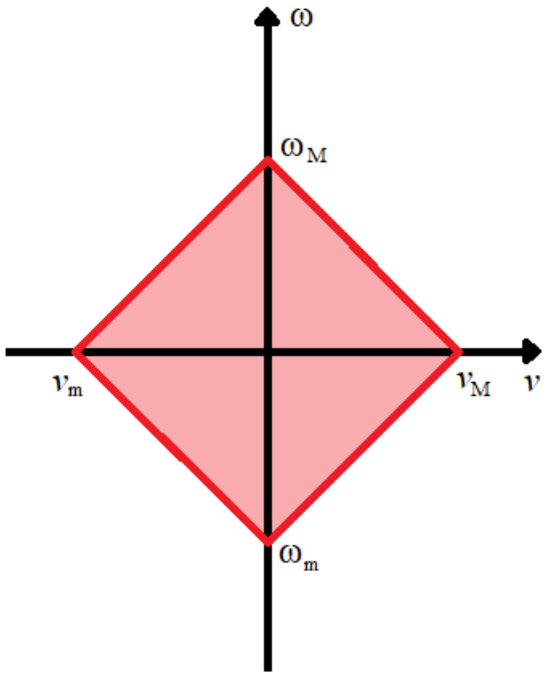


Figure 2.1: Admissible control input range

In the study case, it has been set the following control inputs range:

$$\omega \in [-2\ ,2]\ \ rad/s$$

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

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

2.4 Reference control scheme

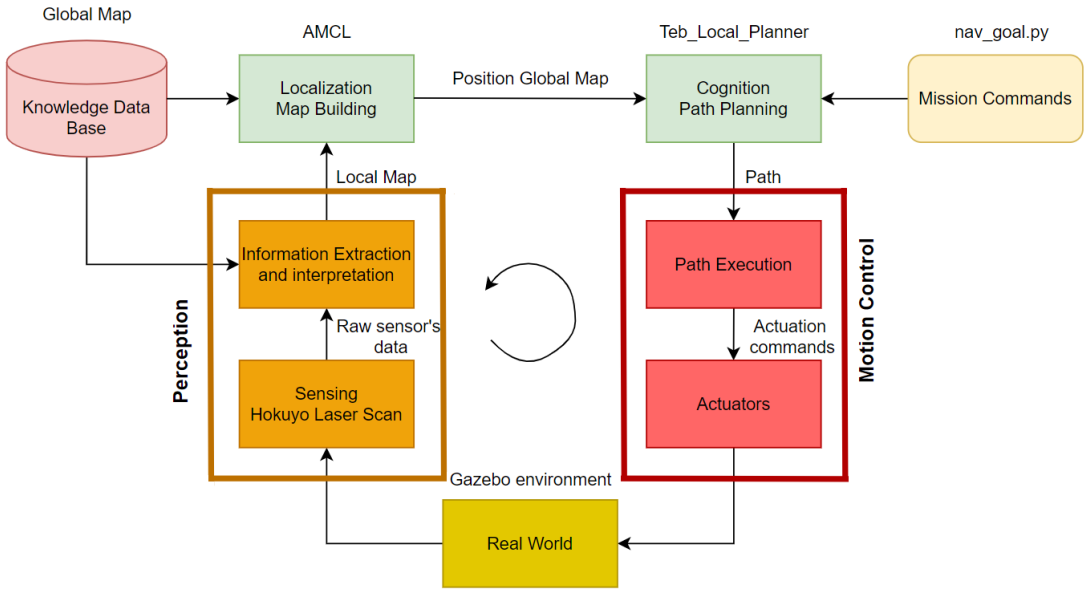


Figure 2.1: Reference control scheme for mobile robot

The most important features of conventional mobile robotics is perception: mobile robots can travel across much of earth's man-made surfaces, but they cannot perceive the world nearly as well as humans so they must be equipped by sensors in order to sense the environment where they move in. 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 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 of the robot with obstacles. Obviously, the path planner needs to know the goal position that the robot will reach. The path found gives the actuation commands which must be sent to the motors that actuates robot's wheels.
Fig.2.1 shows the high-level control scheme, refreshed at each time step.

3 Robot Operating System

ROS is an open-source, meta-operating system for fixed or mobile robot. It provides the services 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.

3.1 URDF model of custom robot

The Unified Robotic Description Format (URDF) is an XML file format used in ROS to describe all elements of a robot trough specific tags. Even if URDFs are a useful and standardized format in ROS, they are lacking many features, especially in working with Gazebo environment. Thus, the Simulation Description Format (SDF) has been introduced, offering a complete tool to describe efficiently everything from the world level down to the robot level. In the study case, the URDF model of the "BookBot" has been obtained directly from its CAD file, using a tool that allows to export robots, designed in the *OnShape* CAD software, to descriptions format like URDF or SDF. All references useful to install this tool can be found at the following link: `https://onshape-to-robot.readthedocs.io/en/latest/`.

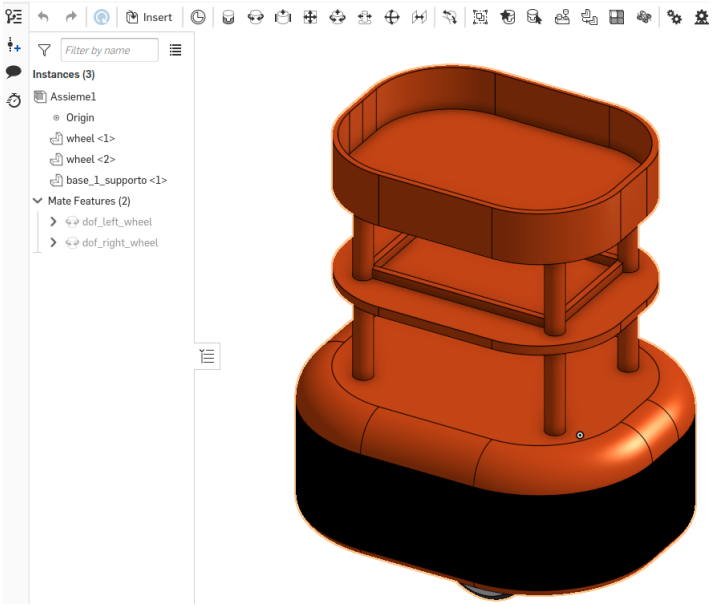


Figure 3.1: OnShape Online CAD software

At this point, the generation of the URDF file is straightforward, resumed in the following steps:

- create a new directory;
- create the configuration file $config.json$;
- launch $onshape - to - robot\ mybot$ command in a shell.

After that, URDF and STL file describing the custom robot should be available and can be used in the ROS package.

```
<robot name="onshape">
<link name="base_link"></link>
<joint name="base_link_to_base" type="fixed">
<parent link="base_link"/>
<child link="base_1_supporto" />
<origin rpy="0.0 0 0" xyz="0 0 0"/>
</joint>

<link name="base_1_supporto">

<visual>
<origin xyz="-0.131403 -0.0951909 -0.14874" rpy="-1.5708 6.7993e-17 1.5737e-17" />
<geometry>
<mesh filename="package://base_1_supporto.stl"/>
</geometry>
<material name="base_1_supporto_material">
<color rgba="0.811765 0.309804 0 1.0"/>
</material>
</visual>

<collision>
<origin xyz="-0.131403 -0.0951909 -0.14874" rpy="-1.5708 6.7993e-17 1.5737e-17" />
<geometry>
<mesh filename="package://base_1_supporto.stl"/>
</geometry>
<material name="base_1_supporto_material">
<color rgba="0.811765 0.309804 0 1.0"/>
</material>
</collision>

<inertial>
<origin xyz="-0.1279 -0.0950739 0.242548" rpy="0 0 0"/>
<mass value="9.8129" />
<inertia ixx="1.0087" ixy="-6.05459e-06"  ixz="-0.0112309" iyy="114" ...
    iyz="-0.000118317" izz="0.684012" />
</inertial>

</link>
...
```

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

Listing 1: Laser description in the URDF file

```
<link name="hokuyo">
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <geometry>
        <box size="0.1 0.1 0.1"/>
      </geometry>
    </collision>

    <visual>
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <geometry>
        <mesh filename="package://diff_drive_bot/meshes/hokuyo.dae"/>
      </geometry>
    </visual>

```

```
16      <inertial>
17        <mass value="1e-5" />
18        <origin xyz="0 0 0" rpy="0 0 0"/>
19        <inertia ixx="1e-6" ixy="0" ixz="0" iyy="1e-6" iyz="0" izz="1e-6" />
20      </inertial>
21    </link>
```

Listing 2: Laser settings in the gazebo file

```
1  <gazebo reference="hokuyo">
2      <sensor type="gpu_ray" name="head_hokuyo_sensor">
3        <pose>0 0 0 0 0 0</pose>
4        <visualize>true</visualize>
5        <update_rate>40</update_rate>
6        <ray>
7          <scan>
8            <horizontal>
9              <samples>720</samples>
10             <resolution>1</resolution>
11             <min_angle>-1.570796</min_angle>
12             <max_angle>1.570796</max_angle>
13           </horizontal>
14         </scan>
15         <range>
16           <min>0.10</min>
17           <max>30.0</max>
18           <resolution>0.01</resolution>
19         </range>
20         <noise>
21           <type>gaussian</type>
22           <!-- Noise parameters based on published spec for Hokuyo laser
23                achieving "+-30mm" accuracy at range < 10m.  A mean of 0.0m and
24                stddev of 0.01m will put 99.7% of samples within 0.03m of the ...
                      true
25                reading. -->
26           <mean>0.0</mean>
27           <stddev>0.01</stddev>
28         </noise>
29       </ray>
30       <plugin name="gazebo_ros_head_hokuyo_controller" ...
              filename="libgazebo_ros_gpu_laser.so">
31         <topicName>/mybot/laser/scan</topicName>
32         <frameName>hokuyo</frameName>
33       </plugin>
34     </sensor>
35   </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.

3.2 Gazebo environment

In the study case, an indoor environment has been considered: in particular, a bookstore. It presents narrow passages, actors moving along known trajectories which can be considered as dynamic obstacles the robot must avoid. At the first time, the 2D map of the bookstore isn't known so it's necessary to build a map of the environment using one of the SLAM algorithms defined in ROS.

Figure 3.1: Gazebo environment

Fig. 3.1 shows on the right the Gazebo environment and on the left the 2D map loaded into Rviz. As can be seen, the robot is equipped with a planar laser scan (in particular an Hokuyo laser scan) with a range of [0 , 180°] used to map the environment and detect the position of dynamic obstacles and a camera which is used to make the mapping of the environment easier (so it isn't considered as a sensor).

3.3 Navigation Stack

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

- 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 robot.

In order to use the navigation stack, it must be configured as shown in Fig.3.1.

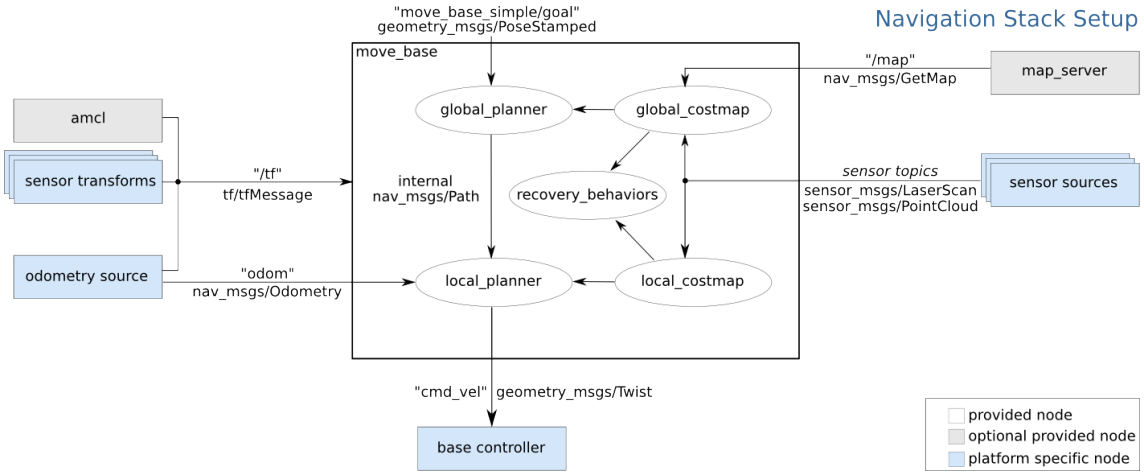


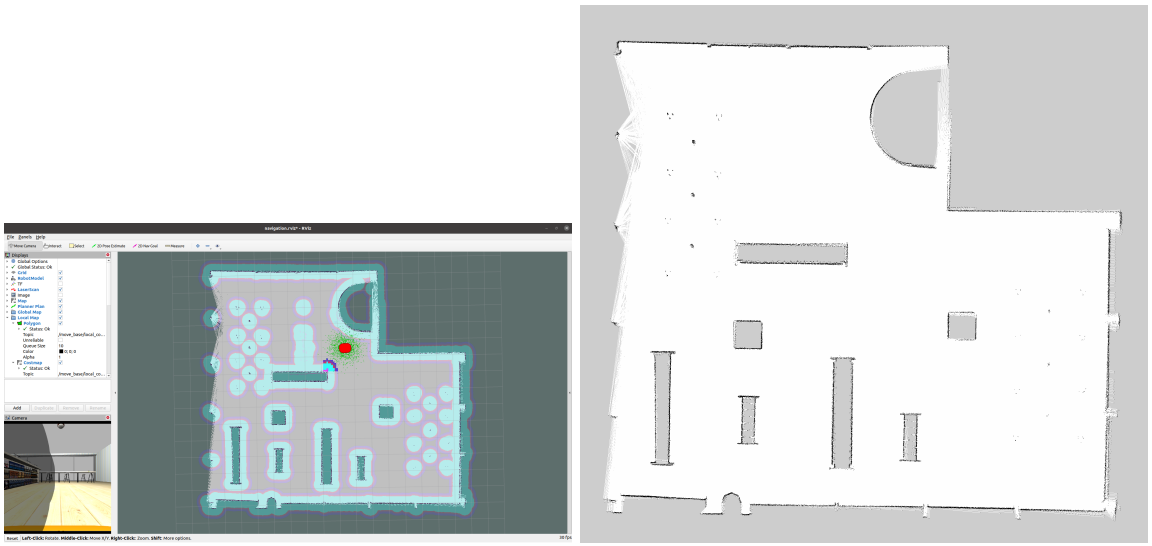
Figure 3.1: Diagram of robot configuration for navigation stack

First of all, navigation stack requires that the system publish informations about the relationships between links' coordinate frames of the robot (it is done using "TF" package) that will be used to manipulate informations 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.

3.4 SLAM Gmapping

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.



(a) Rviz environment (b) 2D map

Figure 3.1: Mapping the environment

Fig.3.1 shows the 2D occupancy grid map generated by **gmapping** node and how it is seen in Rviz: gray areas correspond to the presence of obstacle while white one correspond to free obstacle places.
When the map is built, gmapping node automatically create a .yaml file in which all the map parameters are defined:

Listing 3: Yaml file of the map built

```
1 image: bookstore_maps.pgm
2 resolution: 0.010000
3 origin: [-20.000000, -20.000000, 0.000000]
4 negate: 0
5 occupied_thresh: 0.65
6 free_thresh: 0.196
```

3.5 AMCL

AMCL is a probabilistic localization system for a robot moving in 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 [2]. 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.

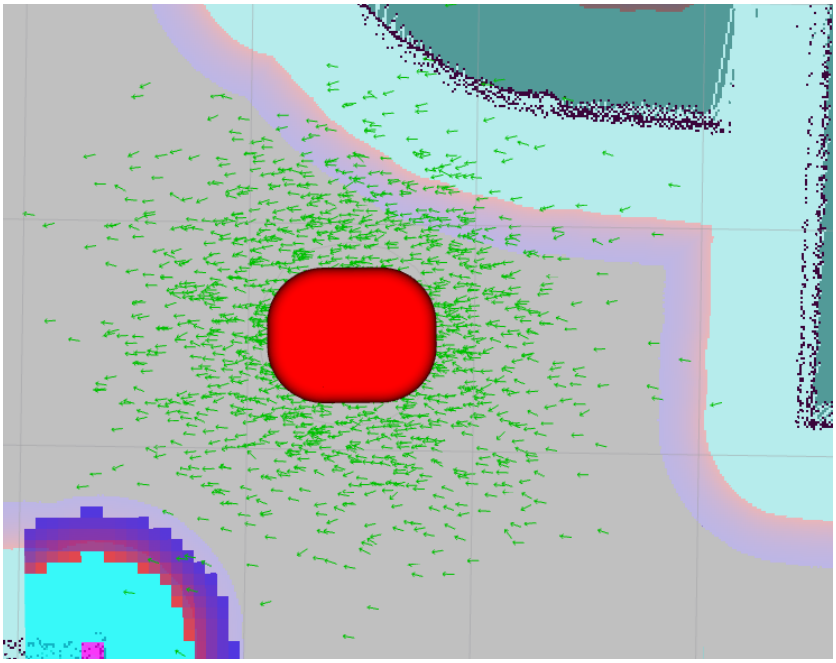


Figure 3.1: AMCL localization approach

As can be seen in Fig.3.1, an estimation of robot pose is defined as vectors whose origins are the estimate position of the robot whereas directions are their estimate orientation. The smaller is the area occupied by vectors, the more accurate estimation of robot pose is given. Moreover, in order to reduce the area, an initial pose of the robot should be provided allowing the algorithm to localize the robot in the map properly.

3.6 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 get from a start to a goal location. Along 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 costmap), and the global path is mapped onto 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 in the scoring of trajectories.

3.6.1 Dynamic-Window-Approach

At the beginning, the environment in which the bookbot can move has been set with static obstacles and the so call *Dynamic Window Approach* (DWA) is well suited for the navigation purpose: given a global plan to follow and a costmap, this local planner produces velocity commands to send to a 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 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;
- rinse and repeat.

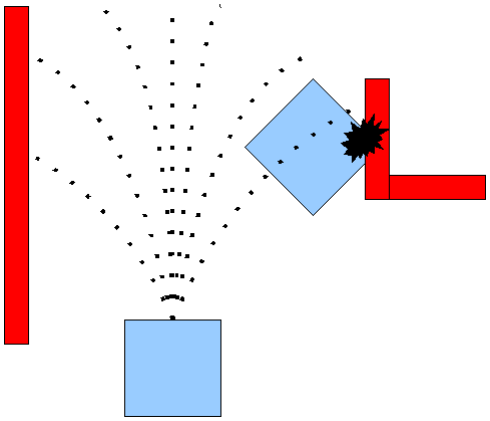


Figure 3.1: DWA local planner

Even if this base local planner performs very well with static obstacles, it encounters several problems in avoiding dynamic obstacles such as people that moves on known or randomize trajectories in the bookstore. For these reasons, a different type of base local planner has been considered: teb local planner.

3.6.2 Time-Elastic-Band Local Planner

This base local planner, implemented in the **teb_local_planner** package, implements 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 during 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 and 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 is able to switch to the current globally optimal trajectory among the candidate set. In this way, the obstacles avoidance can be done in the presence of moving obstacles, as required.

3.7 Move Base Waypoints & Books Spawning

The main task of the robot is to move in the indoor environment avoiding fixed and moving obstacles, collecting book from known points (near tables) and take them to the bookstore's reception. In order to accomplish these tasks, a node named "nav_goal.py" is defined .Different waypoints has been defined as 2D goal position using the **MoveBaseAction** which is a SimpleActionServer implemented by the move_base node, an action server with a single goal policy, taking in goals of **geometry_msgs/PoseStamped** message type. To communicate with this node, the SimpleActionClient interface is used. The move_base node tries to achieve a desired pose by combining a global and a local motion planners to accomplish a navigation task which includes obstacle avoidance.
Once the robot has achieved the goal pose, a book is spawned on one of the levels of the robot in order to simulate a person that lean a book on the robot. The spawn of the book is done using two services of **gazebo_msgs**:

- "GetModelState" which allows to get the pose of the robot when it reaches one of the waypoints defined;
- "SpawnModel" which allows to spawn a model on Gazebo defining the pose in which the model will be spawned.

3.8 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 $\dot{\phi}_1, \dot{\phi}_2$. In fact, by inverting the kinematics model described in Sec.2.2, 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.

Listing 4: Differential Drive Controller Plugin

```
<!--Differential drive controller-->
  <gazebo>
    <plugin name="differential_drive_controller" ...
        filename="libgazebo_ros_diff_drive.so">
      <legacyMode>false</legacyMode>
      <alwaysOn>true</alwaysOn>
      <updateRate>10</updateRate>
      <leftJoint>left_wheel_hinge</leftJoint>
      <rightJoint>right_wheel_hinge</rightJoint>
      <wheelSeparation>0.4</wheelSeparation>
      <wheelDiameter>0.2</wheelDiameter>
      <commandTopic>cmd_vel</commandTopic>
      <odometryTopic>odom</odometryTopic>
      <odometryFrame>odom</odometryFrame>
      <robotBaseFrame>chassis</robotBaseFrame>
      <rosDebugLevel>na</rosDebugLevel>
```

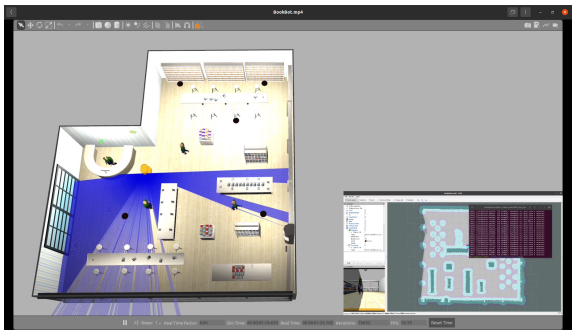
```
16          <publishWheelTF>false</publishWheelTF>
17          <publishOdomTF>true</publishOdomTF>
18          <publishWheelJointState>false</publishWheelJointState>
19          <wheelAcceleration>0</wheelAcceleration>
20          <wheelTorque>5</wheelTorque>
21          <publishTf>true</publishTf>
22          <odometrySource>world</odometrySource>
23        </plugin>
24    </gazebo>
```

4 Simulation and Results

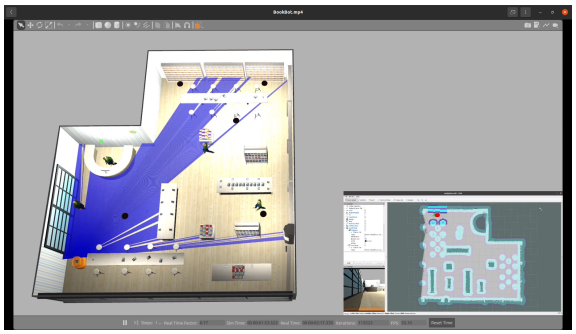
In order to simulate the behaviour of the robot in the bookstore and verify if it accomplish the task, the following command must be typed on different shell windows:

- launch the gazebo world –> $ **roslaunch bookbot gazebo.launch**;
- launch the move base launch file –> $ **roslaunch bookbot amcl_move_base.launch**;
- run the navigation goal node –> $ **rosrun bookbot nav_goal.py**.

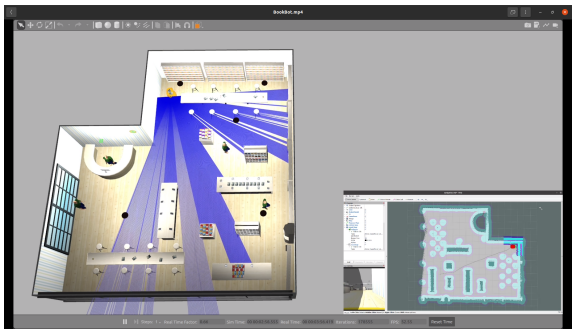
Simulation shows that the robot moves into the bookstore avoiding both fixed and mobile obstacles and accomplish the task assigned: in particular, when it sense the vicinity of actors using the laser scan, it stops or change the path planned by the global and local planner. Different frames of the simulation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



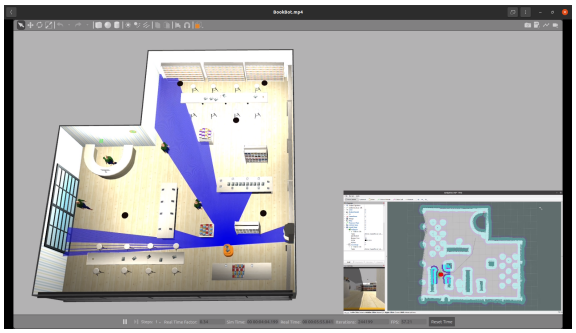
(a) Initial Pose



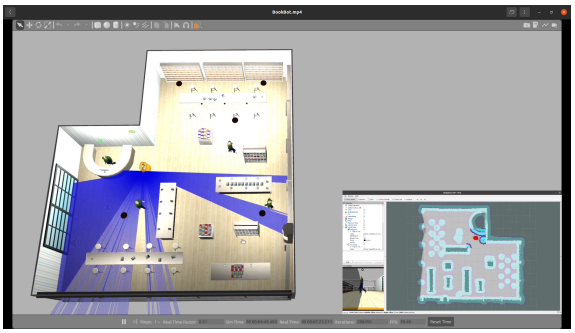
(b) First waypoint



(c) Second waypoint



(d) Third Waypoint



(e) Final Pose

Figure 4.1: Simulation of BookBot

References

[1] Davide SCARAMUZZA Roland SIEGWART Illah R. NOURBAKHSH. *Introduction to Autonomous Mobile Robots - 2nd ed.* Massachusetts Institute of Technology, 2011.

[2] ROS. *Ros Wiki.* URL: `http://wiki.ros.org`.

List of Figures

List of Tables