1 Background and scope of the work

In this section I will briefly introduce the works background and the aim of the research.

Title of the work is "Implementation and comparison of methodologies for indoor self-localization of AMR", where the acronym AMR stands for Automated Mobile Robots. The referred methodologies are 2 localization algorithms (a Gmapping Particle Filter and a Feauture Based EKF), which the robot uses to retrieve its pose (in x, y, θ coordinates) and map the environment (i.e. reconstruct it) using the SLAM approach. SLAM stands for Self Localization and Mapping. To test the methods, the trajectory of Figure 2 is used, which represents a standard docking-delivery trajectory. The AMR performance is calculate by the error deviation from a theoretical trajectory and the one the robot actually performs in the simulation, hence an error estimate.

The "Montagelinie" of Figure 1 will be simulated as a rectangular sum of individual blocks representing individual stations. At the end of the Open Loop manouvre of Figure 2, the AMR stops in the middle of 2 stations.

For the simulation we use ROS Indigo on Ubuntu 14.04 and Gazebo 2.2 software, integrated with ROS.

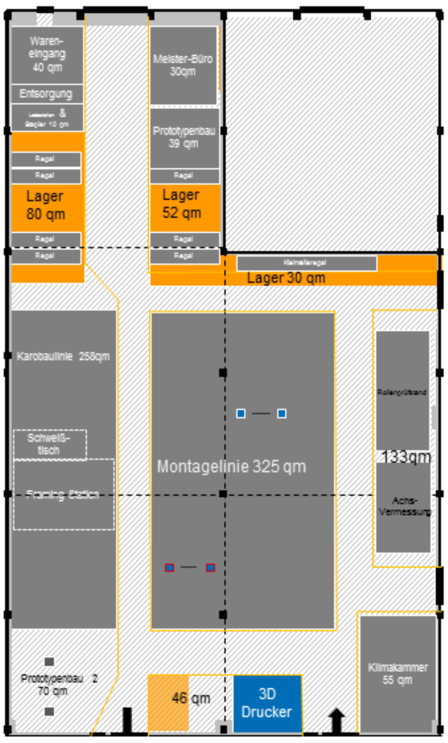


Figure 1: WZL production plant in Aachen

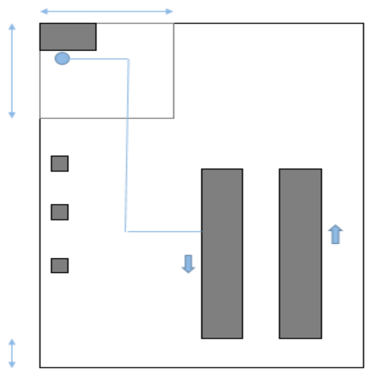


Figure 2: Open Loop trajectory

2 Updates

Before making the WZL site in Gazebo, the algorithm are tested in the *simple indoor environment*, according to the plan, found in the Prep. phase dedicated section. The first will be the Gmapping Particle Filter, which pseudocode is found in page 4. The simple indoor plant resambles the MRL Arena [1]. This small indoor environment contains an uneven structure, which is suitable to test the Gmapping or Amcl package algorithm already provided by the ROS distribution. The resulting building indoor environment is visible in Figure 4 and Figure 3. The figure shows two extra white small blocks, which I will discuss later on. They represent a WirelessTransmitter and Receiver world model, needed to begin the Global Localiztion of the algorithm.

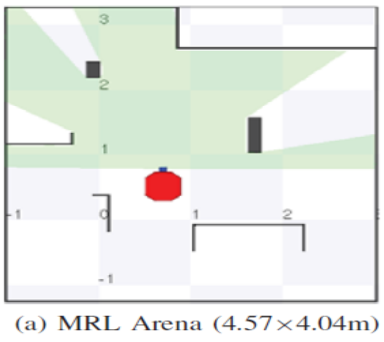


Figure 3: The 2D model arena, used to reproduce the simple model arena

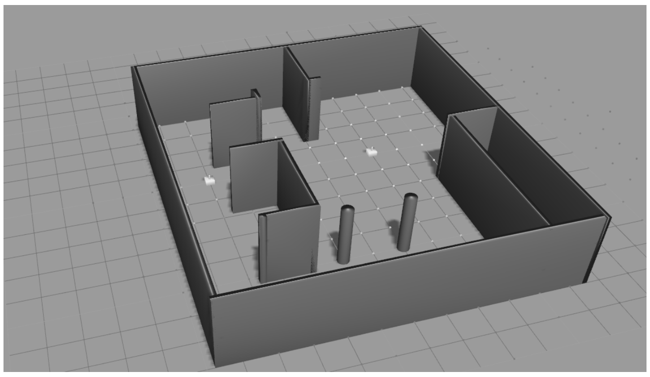


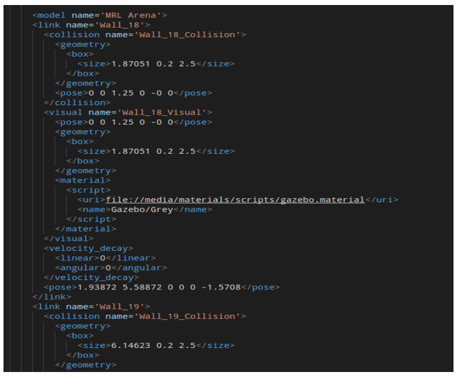
Figure 4: Realization of the model in Gazebo

Each world defined in the Gazebo simulator consists of several components, whose characteristics are visible in Figure 5 and Figure 6. The world model is written in a XML fashion and includes collision, visual and inertial property. In this example the link for Wall18 is one of the component of the entire indoor arena, whose collision property defines the wall's dimension (and the obstructed space it covers in the world), the visual the visibility in the GUI, and the inertial the physical properties. Most of the times collision and visual coincide, like in this case. The only relevant *parameters* for this static arena are the poses of the walls. By adding more "pieces", the entire indoor world has been created.

The WirelessTransmitter (I will explain why this model has been added in the following part) model contains a further tag, the *sensor* tag, which includes the plugin that fires the sensor to transmit data over the Gazebo simulator. It is actually not visible here the process that is is hidden behind, but by adding the $< pluginname : filename : .so >$ of line 22, the sensor is able to publish its data on a ROS topic, which can be used to whatever node needs it.

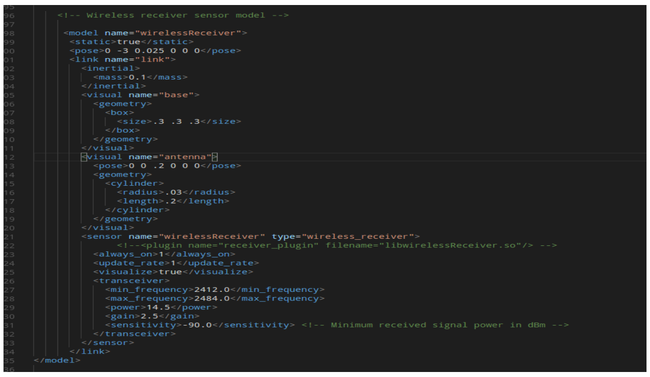
Additionally, two pillars have been added, in order to add some random objects in the environment.

Concluding for this environment model, it has components and contains also paramenters, that can be tuned to change the model behavior. For the sensors in this case, they are visible in the code lines of Figure 6.



```
<model name='MRL Arena'>
<link name='Wall_18'>
  <collision name='Wall_18_Collision'>
    <geometry>
      <box>
        <size>1.87051 0.2 2.5</size>
      </box>
    </geometry>
    <pose>0 0 1.25 0 -0 0</pose>
  </collision>
  <visual name='Wall_18_Visual'>
    <pose>0 0 1.25 0 -0 0</pose>
    <geometry>
      <box>
        <size>1.87051 0.2 2.5</size>
      </box>
    </geometry>
    <material>
      <script>
        <uri>file://media/materials/scripts/gazebo.material</uri>
        <name>Gazebo/Grey</name>
      </script>
    </material>
  </visual>
  <velocity_decay>
    <linear>0</linear>
    <angular>0</angular>
  </velocity_decay>
  <pose>1.93872 5.58872 0 0 0 -1.5708</pose>
</link>
<link name='Wall_19'>
  <collision name='Wall_19_Collision'>
    <geometry>
      <box>
        <size>6.14623 0.2 2.5</size>
      </box>
    </geometry>
```

Figure 5: The simple indoor URDF file



```
96      <!-- Wireless receiver sensor model -->
97
98       <model name="wirelessReceiver">
99        <static>true</static>
00        <pose>0 -3 0.025 0 0 0</pose>
01        <link name="link">
02          <inertial>
03            <mass>0.1</mass>
04          </inertial>
05          <visual name="base">
06            <geometry>
07              <box>
08                <size>.3 .3 .3</size>
09              </box>
10            </geometry>
11          </visual>
12          <visual name="antenna">
13            <pose>0 0 .2 0 0 0</pose>
14            <geometry>
15              <cylinder>
16                <radius>.03</radius>
17                <length>.2</length>
18              </cylinder>
19            </geometry>
20          </visual>
21          <sensor name="wirelessReceiver" type="wireless_receiver">
22                <!--<plugin name="receiver_plugin" filename="libwirelessReceiver.so"/> -->
23            <always_on>1</always_on>
24            <update_rate>1</update_rate>
25            <visualize>true</visualize>
26            <transceiver>
27              <min_frequency>2412.0</min_frequency>
28              <max_frequency>2484.0</max_frequency>
29              <power>14.5</power>
30              <gain>2.5</gain>
31              <sensitivity>-90.0</sensitivity> <!-- Minimum received signal power in dBm -->
32            </transceiver>
33          </sensor>
34        </link>
35  </model>
```

Figure 6: Details of the Wireless sensor model within the same URDF indoor model file

Ones the small indoor model has been created, the pre-installed mapping algorithms provide by ROS, *Gmapping* and *Amcl* have to be tested. It is always good practice to download and build the packages from source when developers want to modify the files and adjust the internal parameters. The husky robot package together with the slam-gmapping one has been cloned from source, as seen in Figure 7

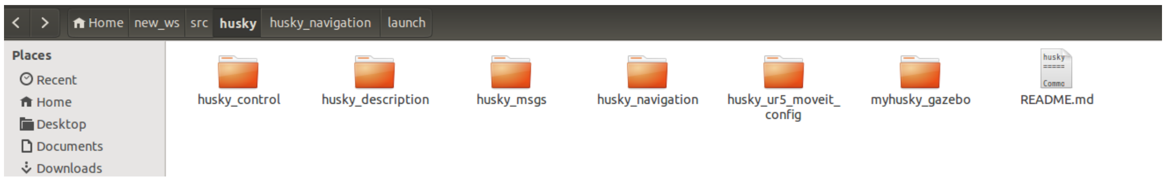


Figure 7: The husky package structure

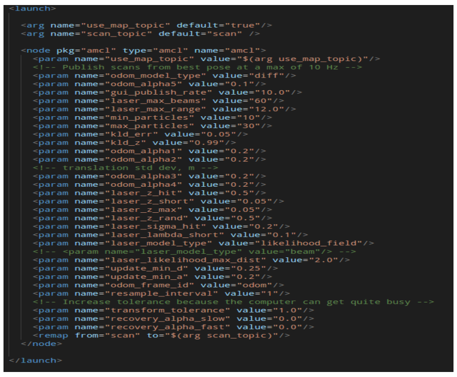
The husky robot package consists of the following subpackages, and tuneable parameters:

- husky_description: contains meshes and urdf files that assemble the husky robot. Inside the urdf folder (not visible in this picture) all the husky robot properties are built using a special type of file called the *Xacro* file, which is a more compact version of the URDF file. A proper and detailed documentation can be found under [2]. As like for the world model of Figure 5, an URDF file contains the robot components as link tags, and additionally joint tags define how the components have to be linked together. Most of the parameters that we find here are:
  - Height, lenght and width of the single robot components
  - Inertial values for the 6 degree of freedom
  - Sensors main parameters, if present. This is actually of importance for our case, hence the laser LIDAR Sick Navi contained in the Gazebo standard sensors for husky, have the following parameters:
    - Update rate: frequency in Hz for the sensor update
    - Min and Max angle: self explaining in *rad*. NOTE: this has been modified to have a 360 deg angle all around the robot, as prescribed in the paper of [3], which is the first method to be implemented.
    - Min and Max range: the smallest and the longest distance that the sensor can measure, expres in *meters*
- husky_navigation: here is were the *Gmapping* and *amcl* algorithms are fired, together with their parameters. The single parameters sets are seen in Figure 8, the way to be used will be explained later on while talking on the algorithm implementation. Additinally, inside the navigation folder, the map folder contains the stored map after the *Gmapping* runs, in a .png a .config file. For the small indoor the map has been created. It is visible in Figure 9 that there are no particular distorsions and the filter works well.
- myhusky_gazebo: this is a personal created package, containing the self-made MRL Arena.world file and the launch files needed to fire the simulation over the Gazebo node.

So far, so good, and the husky model is now ready to be prepared and implement the approach described on [3]. To do so, the localization and navigation algorithm has been summarized with the main actions to take in the following pseudocode:

We want to take advantage of the amcl node of ROS while making our personal algorithm implementation. To do so, we have to create our own node, communicating with the amcl one and getting the data that we need to perform the "personal algorithm" implementation. The second thing, is to modify the parameters of the amcl on the base of the actual performance during the simulation. The values of such parameters needs to be found empirically, even if for some of them rules exists. Additinally, the *move base*, and inside of it the *local* and *global costmap* parameters shoudl also be tuned.

The number of parameters for the amcl, move base, local and global costmap is huge, so I am not going to describe them all. It is possible to define a detailed explanation of them at this reference



```xml
<launch>

  <arg name="use_map_topic" default="true"/>
  <arg name="scan_topic" default="scan" />

  <node pkg="amcl" type="amcl" name="amcl">
    <param name="use_map_topic" value="$(arg use_map_topic)"/>
    <!-- Publish scans from best pose at a max of 10 Hz -->
    <param name="odom_model_type" value="diff"/>
    <param name="odom_alpha5" value="0.1"/>
    <param name="gui_publish_rate" value="10.0"/>
    <param name="laser_max_beams" value="60"/>
    <param name="laser_max_range" value="12.0"/>
    <param name="min_particles" value="10"/>
    <param name="max_particles" value="30"/>
    <param name="kld_err" value="0.05"/>
    <param name="kld_z" value="0.99"/>
    <param name="odom_alpha1" value="0.2"/>
    <param name="odom_alpha2" value="0.2"/>
    <!-- translation std dev, m -->
    <param name="odom_alpha3" value="0.2"/>
    <param name="odom_alpha4" value="0.2"/>
    <param name="laser_z_hit" value="0.5"/>
    <param name="laser_z_short" value="0.05"/>
    <param name="laser_z_max" value="0.05"/>
    <param name="laser_z_rand" value="0.5"/>
    <param name="laser_sigma_hit" value="0.2"/>
    <param name="laser_lambda_short" value="0.1"/>
    <param name="laser_model_type" value="likelihood_field"/>
    <!-- <param name="laser_model_type" value="beam"/> -->
    <param name="laser_likelihood_max_dist" value="2.0"/>
    <param name="update_min_d" value="0.25"/>
    <param name="update_min_a" value="0.2"/>
    <param name="odom_frame_id" value="odom"/>
    <param name="resample_interval" value="1"/>
    <!-- Increase tolerance because the computer can get quite busy -->
    <param name="transform_tolerance" value="1.0"/>
    <param name="recovery_alpha_slow" value="0.0"/>
    <param name="recovery_alpha_fast" value="0.0"/>
    <remap from="scan" to="$(arg scan_topic)"/>
  </node>

</launch>
```

Figure 8: The *amcl* tunable parameters

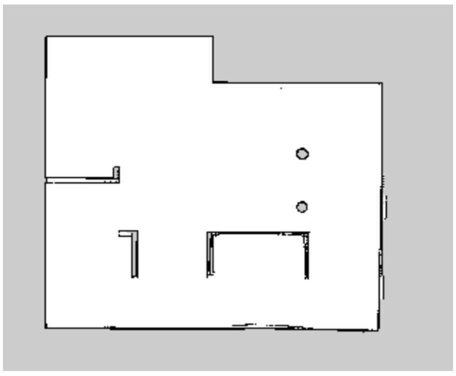


Figure 9: Result of the Gmapping for the simple indoor environment

Algorithm 1 Kirsch, Rohig algorithm

```
1: $St - 1 = St$
2: for $i = 1$ to $N$ do    ▷ With N the number of particles in the filter set by maxparticle parameter
3:     Spread particles in the anchorbox with equations 1) and 2) of [3]    ▷ This step is called Global Localization
4:     $xt[n] = p(xt|xt - 1, ut)$    ▷ Motion update - sample the particles from the motion update of the robot and move forward to estimate the error model functions
5:     $wt[n] = p(dnanoLOC|si) * p(dlaser|si)$    ▷ Measurement update - si are the particles set with i the i-th index
6:     $St = St + < xt, wt >$    ▷ add the state and weight to the total state space
7:     Perform resampling
8: end for
9: Return St
```

[4]. In [4] there is a detailed explanation on how to tune them, and the approach really depends on the specific environment, as already cited. In Figure 10 the overall vision of the nodes and topics communication is visible. This figure is actually taken from [4], but the actual personal one is pretty similar, just instead of a real robot, here called /volksbot, we use the informations from Gazebo to write on topics such as the *odometry*. Our "self-made" node wants from the amcl (node are the ellipses) informations on the global position of the robot in the environment. However, to perform a good global localization, the robot has to partially move, hence it has to publish data on the move base node, via move_base/action_topics. If the move base receives a new goal, it plans a path from the current pose of the robot to the goal pose. This global plan will be calculated using the Dijkstra algorithm. The planning of the global path is based on the information of the global costmap, which forces the planned path away from obstacles. The global plan will only change, if the local path planner notifies that an obstacle is in front of the robot or the global map changes (just by using the move base in conjunction with *GMapping*). The local path planner determines the velocity commands to execute the navigation, based on its local costmap, which also detects obstacles in the environment.

The output it is written on the map topic, that progressively is built by the filter itself.

Answering to question 3) the deliverable is an algorithm that implement the pseudocode above, and that can be validated using the validation procedure.

The validation answers the question number 4): first the algorithm has to be validated in close loop, to avoid map distorsions. Then an open loop manouvre is performed, that according to [5] is the typical path for indoor AGV, and recognized to be one of the toughest paths for the industrial reality.

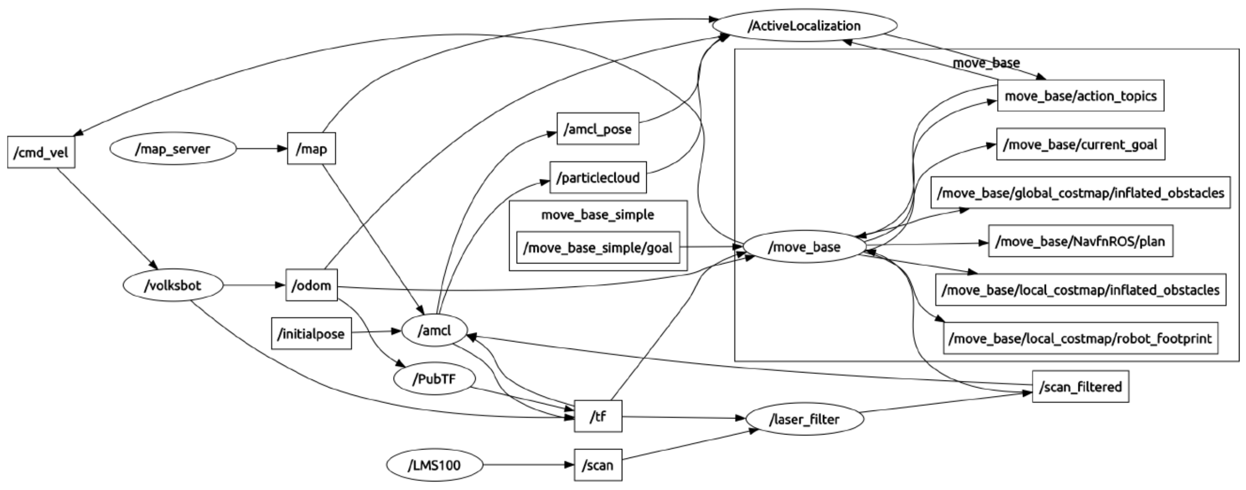


Figure 10: An example of an active localization node

3 Questions

The questions are not specific but of general way, since you are probably not dealing with the same things:

1. The first big question is more on a your opinion on the way I tackle the problem, so, if you had to do the same, how would you have done it?

2. Do you thing the creation of an additional node "The Active Localization" implementing the pseudocode in page 4 is a good strategy? The Amcl node already works, I can see the robot moving, the particles being spread, however the src code of the filter is not visible and I dont know a way of controlling it.

   We will discuss more thing during the meeting, I am actually curious to know your opinion.

4 Reference

[1] J. Santos, D. Portugal, R. Rocha. An evaluation of 2D SLAM Techniques Available in Robot Operating System.

[2] http://wiki.ros.org/urdf/Tutorials/Using Xacro to Clean Up a URDF File

[3] Kirsch, C. and C.Rhig (2011). Global localization and position tracking of an automated guided vehicle.

[4] Sebastian Gangl, Robotic exploration for mapping and change detection. Pdf is available under Dropbox/Abschlussarbeit/PEM/Literature/LiteratureResearch/Thesis

[5] Investigating Simultaneous Localization and Mapping for AGV systems, A. Palsson, M. Smedberg