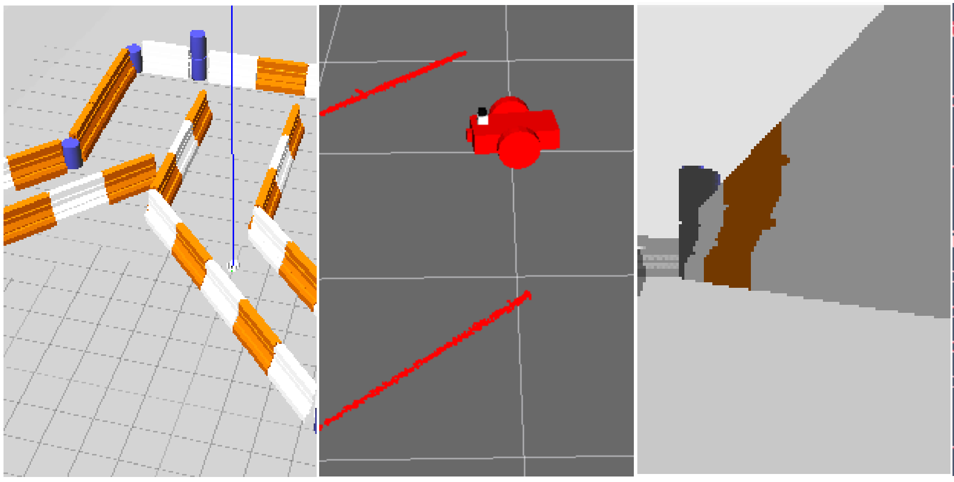
RoboND-Localization-Project

Report



```
t name="udacity_bot" xmlns:xacro="http://www.

cro:include filename="$(find udacity_bot)/ur

nk name="robot_footprint"></link>

int name="robot_footprint_joint" type="fixed"
origin xyz="0 0 0" rpy="0 0 0" />
parent link="robot_footprint"/>
child link="chassis" />
oint>

nk name='chassis'>
```

Lutong Zhang

1. Abstract

A robot model is setup in the Gazebo simulation environment, which has two wheels, a camera and a Lidar on board. AMCL node is implemented to solve the localization problem of the robot. Move_it node is employed to drive the small cart robot to the goal. The final results are listed and discussed.

2. Introduction

This report represents the Where AM I project, which simulate the localization problem of a small differential-wheels-driven robot with a camera and a lidar sensor on board in the simulation environment of Gazebo.

3. Background

The core of the Localization problem is to solve the question that where is the robot. Depending on the different situations, localization problems can be divided into three kind of localization problems: (1) local localization problem (position tracking), (2) global localization and the (3) kidnapped robot problem.

The goal of the local localization problem is to keep track of the position of the robot based on the information provided by different sorts of sensors. To get the accurate position of the robot, it needs to solve two major problems. The first one is the noise of the sensors, and the second one is the fusion of the data from different sensors. It is quite possible that the data from a noisy data yield the Gaussian Distribution. In the light of this statement, the noise of data can be measured by a mean and a standard deviation. The main goal of the global localization problem is to find the true location of the robot relative to the ground-truth map.

There are two main methods for localization: particle filter and Kalman filters. Kalman filters are good at estimating linear systems with Gaussian noise, while the particle filters are good at nonlinear problems. Kalman filters have lower computational requirements than particle filters. Kalman filters estimate current state of the system based on the sensor data, system configuration and previous states. Particle filters generate random samples and then weight each of them based on the sensor data.

4. Model Configuration

4.1 Create a Robot Model

The robot model in the ROS system is represented by a Unified Robot Description Format (URDF), which is a XML formatted file [1]. A filed named as udacity_bot.xacro was created under src/udacity_bot/, which contains the discerption of the robot model. The robot is composed of a box body and two wheels, as well as two sensors on the box-shaped body as shown in the figure below.

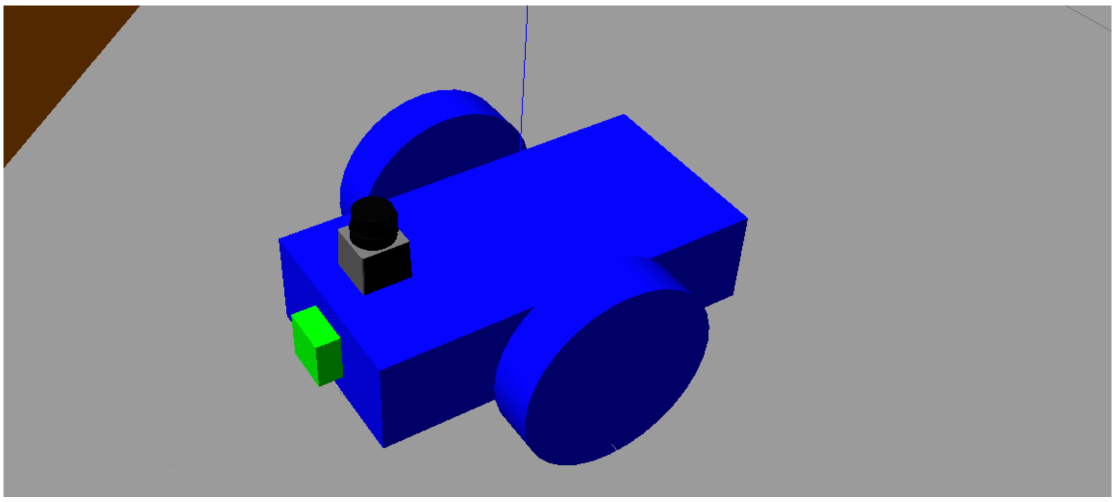


Fig. 1 Robot model

Basically, the robot model is a combination of several “links” connect by joints as described in the URDF file. In order to launch the robot model, a launch file was created and called by the udacity_world.launch.

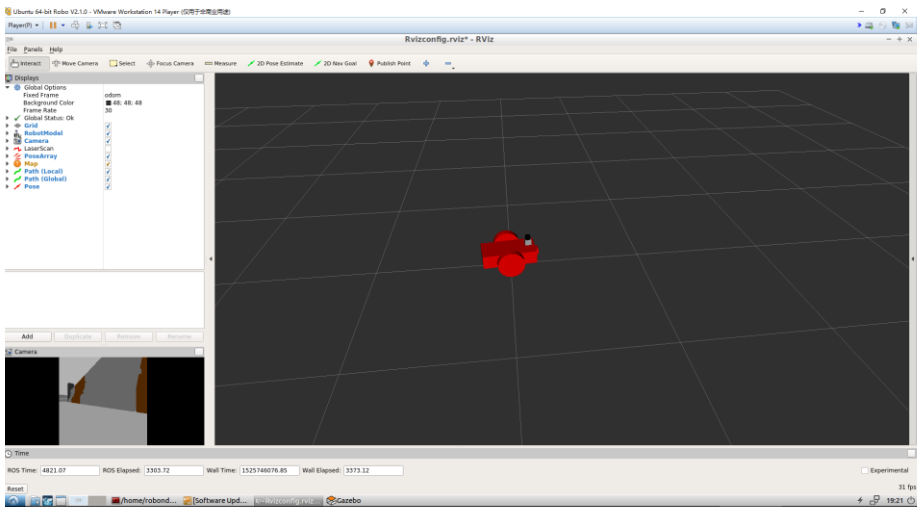
There are two sensors on the robot, a Hokuyo Lidar Rangefinder and a camera. The camera is simplified as a green box fixed to the front of the robot body, while the shape of the lidar is described by a mesh file (hokuyo.dae) which can be downloaded.

In order to collect sensor data such as images and laser-measured ranges, some plugins in Gazebo in employed. The plugin that drives the robot is called Differential Driver Controller, which is a controller that drives differential mobile base. It’s can be regarded as nothing but a ROS Node as well. The plug-in that support the camera and the lidar is pre-defined in Gazebo. The image taken by the camera is published to the image_raw topic and the laser range data is published to the udacity_bot/laser/scan topic. After all the setups and configurations, the simulation world and the robot model can be load by "*roslaunch udacity_bot udacity_world.launch*".

4.2 Config the launch file for RViz

In the *robot_description.launch* file, two nodes are added. The first one is the *joint_state_publisher* that publishes joint state messages for the robot. The second one is the *robot_state_publisher*, which handles the task of publishing the location and orientations of all the links in the robot to the transform tree. The transform tree is just another ROS node that keeps track of multiple coordinate frames over time and maintains the relationship between coordinate frames in a tree structure buffered in time [3], and lets the user transform points, vectors, etc between any two coordinate frames at any desired point in time.

To start RViz together with Gazebo, "*<node name="rviz" pkg="rviz" respawn="false"/>*" is added to the *udacity_world.launch*. The robot and sensor data in Rviz are shown in the figure below.



Global Options	
Fixed Frame	odom
Background Color	■ 48; 48; 48
Frame Rate	30
Global Status: Ok	
Grid	✓
RobotModel	✓
Camera	✓
LaserScan	
PoseArray	✓
Map	✓
Path (Local)	✓
Path (Global)	✓
Pose	✓

ROS Time: 4821.07 ROS Elapsed: 3303.72 Wall Time: 1525746076.85 Wall Elapsed: 3373.12

Fig. 2 Robot in RViz

4.3 Load the map

The map in the simulation environment is pre-constructed by Clearpath Robotics. This map is loaded by the *udacity_world.launch* file.

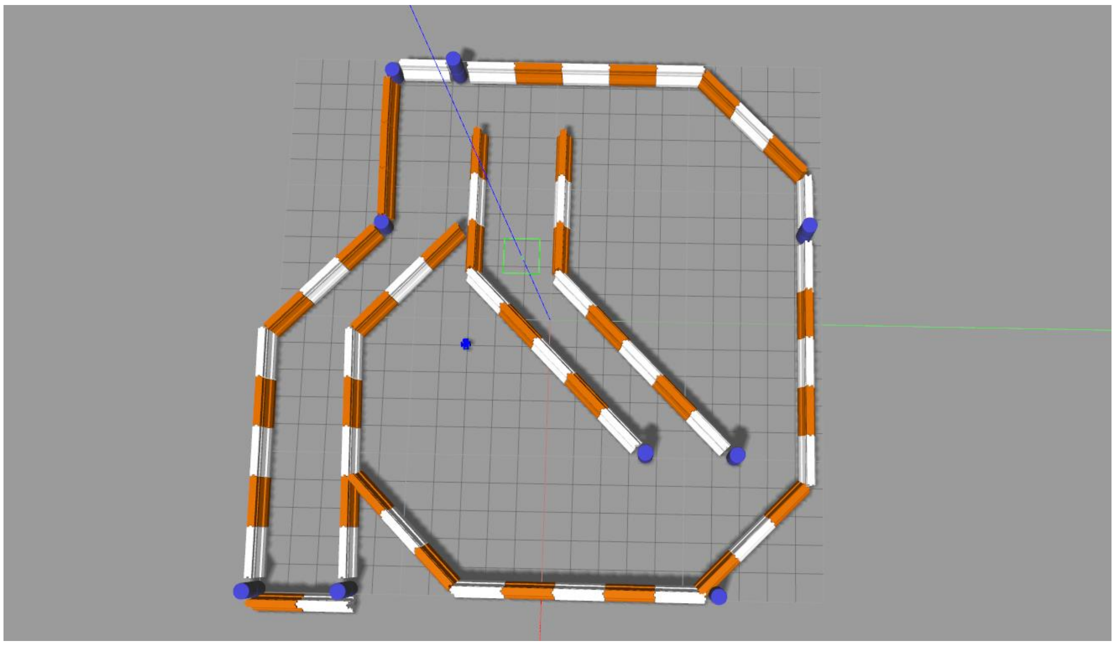


Fig.3 Map in the simulation environment

4.4 AMCL package

AMCL is a probabilistic localization system for a robot moving in 2D. It implements the adaptive (or KLD-sampling) Monte Carlo localization approach (as described by Dieter Fox), which uses a particle filter to track the pose of a robot against a known map [4].

The AMCL package is added to the project by create a new launch file named as *amcl.launch*. There are three nodes employed in the file.

The first node is the 'map_server' which links the 'map' and 'odom' frames.

The second node the ‘amcl’ package. As for the amcl package depend on the odometry data and the lidar data by the robot, the *scan* topic in the amcl package is remapped by the *udacity_bot/laser/scan* topic. Some of the parameters of amcl node are also set in the same file.

The third node is the *move_base* node. The goal of this node is to drive the robot to the goal position. The parameters of this node are mainly defined in the *yaml* files under */config*.

These three nodes can be launched by *roslaunch udacity_bot amcl.launch*. At this moment, the arrange of the ROS system is shown in the figure below.

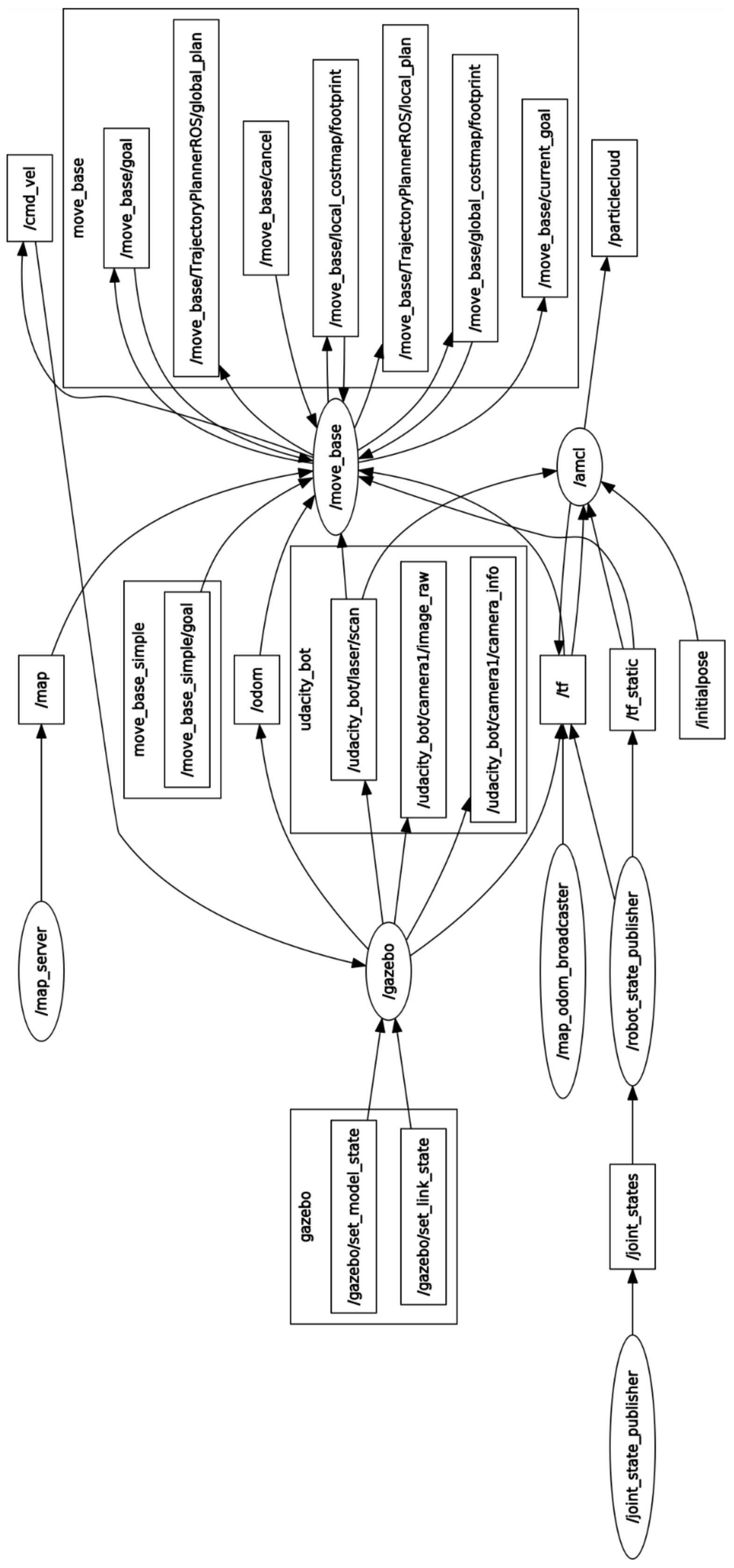


Fig. 4 Arrangement of the ROS system: node and topics.

4.5 Parameter Tuning

Most of the parameters of the system are saved in the *yaml* files under *config,* while some other parameters are hold in the file of *amcl.launch*. The configuration of the parameters are shown in the table below.

Table 1 Configurate of Parameters

AMCL	
min_particles	20
max_particles	500
initial_pose_x	0
initial_pose_y	0
laser_min_range	0.5
laser_max_range	10
laser_z_max	0.05
laser_model_type	likelihood_field_prob
laser_likelihood_max_dist	2
transform_tolerance	0.2
move_base	
yaw_goal_tolerance	0.05
xy_goal_tolerance	0.05
acc_lim_x	2.5
acc_lim_y	2.5
acc_lim_theta	3.2
update_frequency	15
publish_frequency	15
sim_time	1
meter_scoring	FALSE
pdist_scale	0.6

- *min_particles* and *max_particles* define the range of number of particles for the particle filter. Although a higher number of particles may improve the performance of the system, it might be too computational heavy. The config of these two parameters are shown in Table 1.

- transform_tolerance helps determine the longevity of the transforms being published for localization purposes.
- *initial_pose* is the initial location of the robot.
- sim_time is the amount of time to forward-simulate trajectories in seconds
- pdist_scale is the measure that how much the controller should attempt to reach its local goal.

5. Results

The results of the project are shown in the figures below.

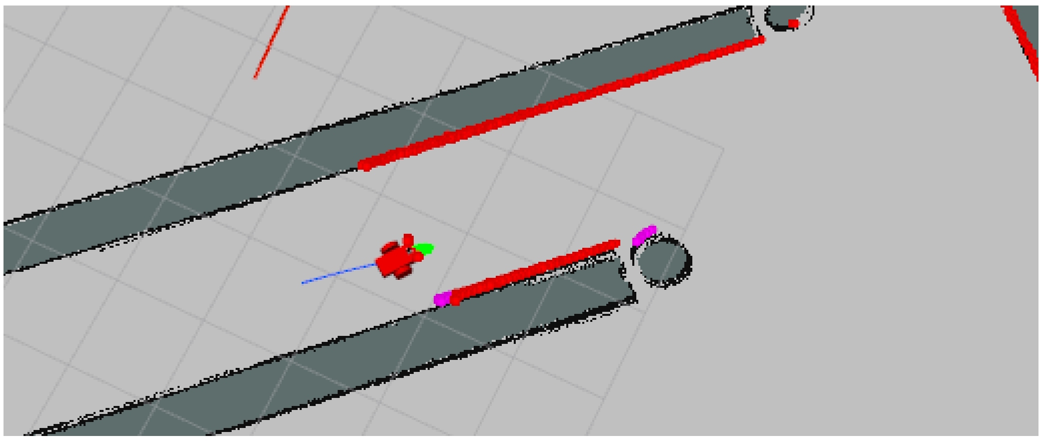


Fig.5 Robot heading to the goal

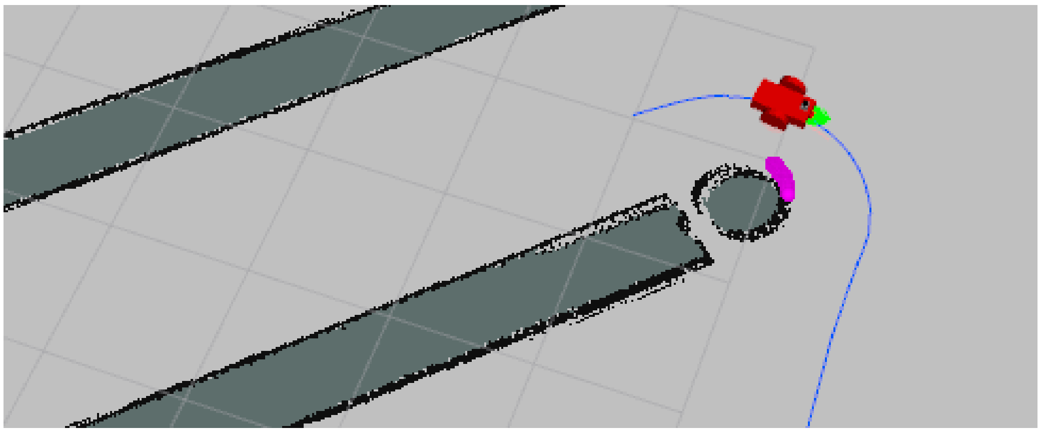


Fig.6 Robot heading to the goal

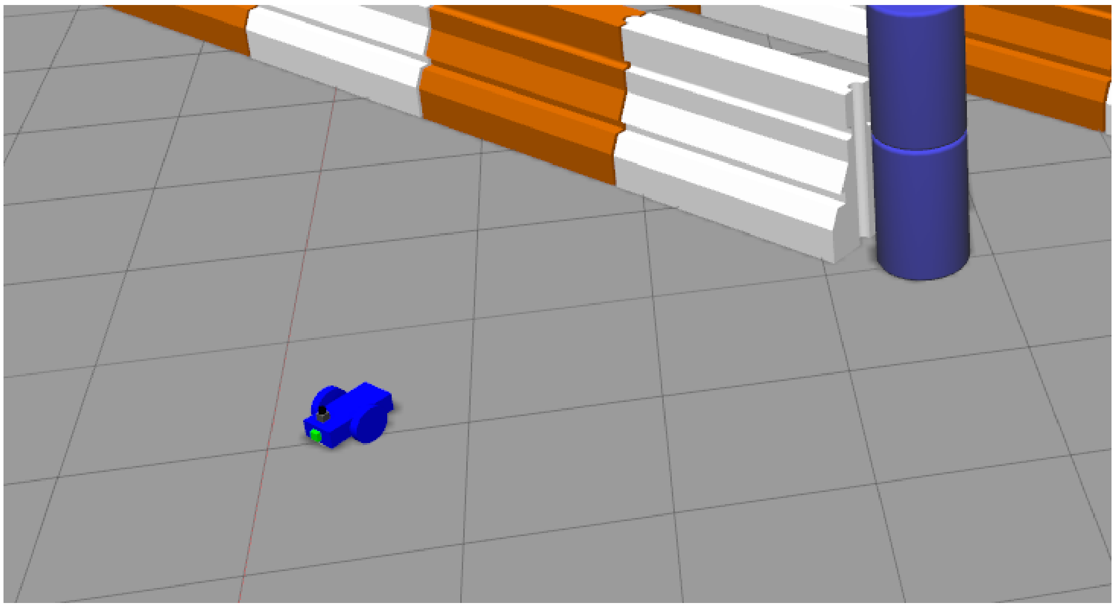
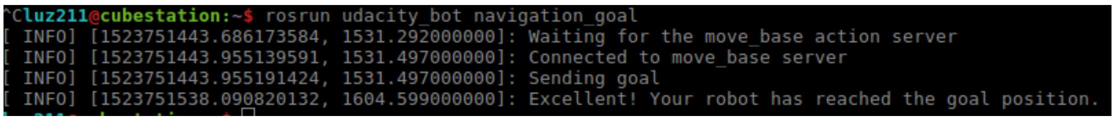


Fig.7 Robot heading to the goal in Gazebo



```
^Cluz211@cubestation:~$ rosrun udacity_bot navigation_goal
[ INFO] [1523751443.686173584, 1531.292000000]: Waiting for the move_base action server
[ INFO] [1523751443.955139591, 1531.497000000]: Connected to move_base server
[ INFO] [1523751443.955191424, 1531.497000000]: Sending goal
[ INFO] [1523751538.090820132, 1604.599000000]: Excellent! Your robot has reached the goal position.
```

Fig.8 Results of the navigation goal

6. Discussion

The results show that the robot reached the goal in the simulation environment. The AMCL node works well in this project. However, for a kidnapped robot problem the AMCL is not the best choice of location estimators, because it may bring some local errors. Furthermore, the locally similarity of the environment should be noticed for AMCL.

AMCL can be used for AGU robots that deliver parts in an indoor factory environment.

7. Future Work

Additionally, the influence of parameters on the accuracy of AMCL need to be tested. Furthermore, I am building a small cart based on Rapsberry Pi with the same ROS setup and planning to test it in the real world.

Reference:

[1] http://wiki.ros.org/urdf

[2] http://wiki.ros.org/diff_drive_controller

[3] http://wiki.ros.org/tf

[4] http://wiki.ros.org/amcl