Implementation of 2d navigation for a moving base using ROS (version 1.0)

Windel Bouwman

December 17, 2014

Abstract

This document describes the application of navigation software as available in ROS to the modified x80sv robot. Different ROS packages were evaluated on the robot. The move-base package was tuned for the x80sv and was tested on both real world and simulated environments.

Contents

List of symbols

Symbol	Unit	Description
l_{base}	$[m]$	Distance between the two wheels of the robot.
r_{wheel}	$[m]$	Radius of the wheel.
N_{enc}	$[-]$	Number of pulses per rotation of the encoders
v_{max}	$[m/s]$	Maximum forward velocity.
a_{max}	$[m/s^2]$	Maximum forward acceleration.
ω_{max}	$[rad/s]$	Rotational maximum velocity.
α_{max}	$[rad/s^2]$	Rotational maximum acceleration.
v	$[m/s]$	Forward velocity of the robot.
ω	$[m/s]$	Rotational velocity of the robot.
ω_l	$[rad/s]$	Rotational velocity of the left wheel.
ω_r	$[rad/s]$	Rotational velocity of the right wheel.
θ	$[rad]$	Rotation of the robot with respect to the world.

References

[1] ROS gmapping `http://wiki.ros.org/gmapping`

[2] x80sv software `https://github.com/SaxionLED/x80sv`

[3] Ompl plugin `https://github.com/windelbouwman/move-base-ompl`

[4] Ompl library `http://ompl.kavrakilab.org/`

[5] `http://wiki.ros.org/amcl`

[6] `http://wiki.ros.org/navigation/Tutorials/Navigation%20Tuning%20Guide`

1 Introduction

This document describes how the problem of 2d-navigation was solved on a real robot. The goal of navigation is to safely move a robot from a to b.

The task of navigation can be split up into the following tasks (see figure 1):

- Robot control. This task is concerned with controlling the seperate wheels of the robot. Typically this is done using PID control or something alike. On the x80sv this is done two times, for the left and right wheel.
- Kinematics takes care of odometry and translation from robot velocity to wheel velocity. By keeping track of wheel motions, the robot position can be calculated. This method is subjective to drift.
- Position and mapping, also known as SLAM, is the task of determining location of the robot in the world, and at the same time reconstructing the world.
- Global planning is the task of determining a path through a known map. This requires a search like A* or something like that.
- Local planning takes as input the global path and generated the appropriate motion commands for the robot control. It is some sort of setpoint generator. This layer is also responsible for obstacles. When an obstacle is observed, the path may be adjusted.
- Semantic interpretation is the process of converting a task into a sequence of locations to be reached. For example the conversion of the sentence "fetch beer" into a path from the current location to nearest fridge.

The rest of this document describes all the tasks listed above as applied to the x80sv using the robot operating system (ROS).

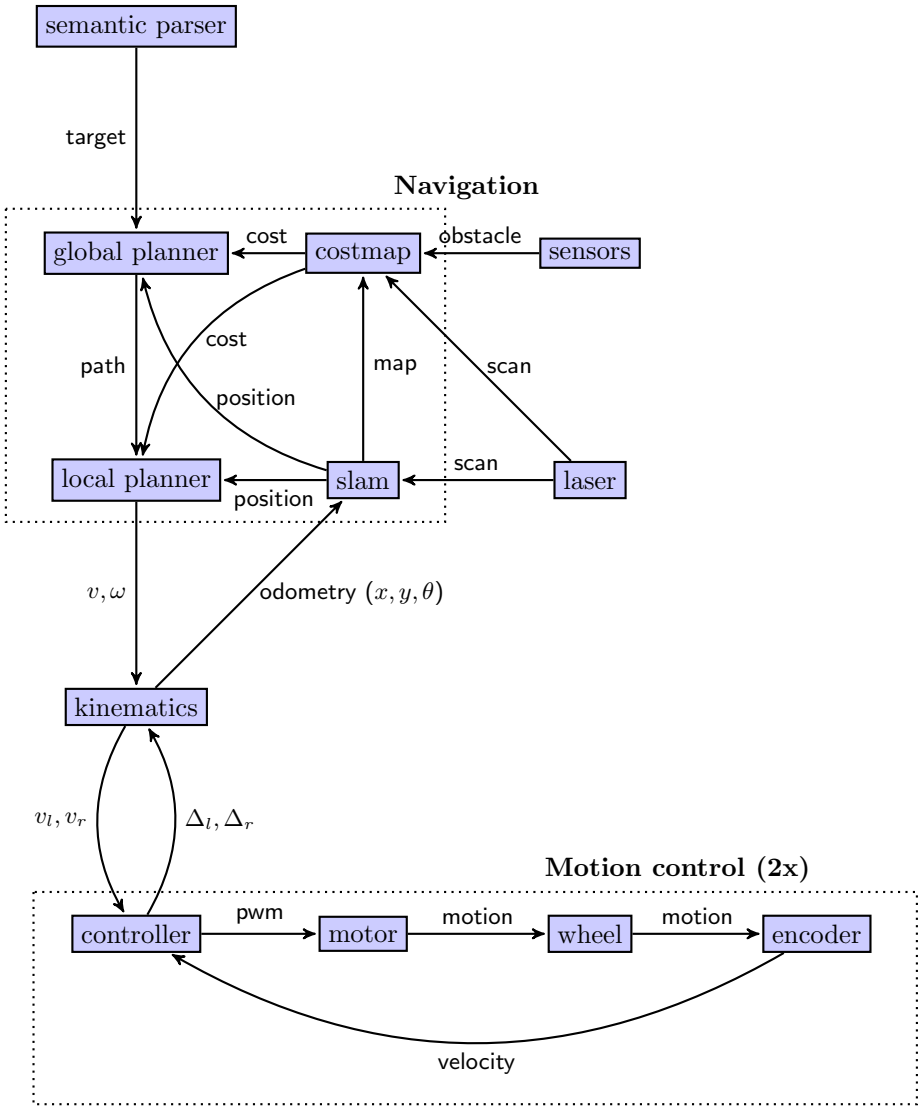


Figure 1: The building blocks involved in the task of navigation. Inside the rectangle is the core functionality.

2 ROS concepts

2.1 Basics

ROS is a robotic operation system. Is provides infrastructure to build robot applications. A ROS system consists of nodes, these are running programs performing some task. Nodes can interchange data via topics via a publish subscribe mechanism. Other concepts are services and parameters.

2.2 Rviz

Rviz is an indispensible tool when debugging a robotic system. With rviz one can visualize a robot and its environment. The tool consists of a main window and a pane to the left where various data visualizers can be added.

2.3 Rqt

Another helpful tool when debugging a ros system is rqt. This tool is a plugin container where plugins can be selected. Plugins exists for tf tree visualization, node interconnect, topic inspection, logging viewer, diagnostics viewer and more.

2.4 Tf

The tf (`http://wiki.ros.org/tf`) system of ROS is used to describe the various parts of a robot in space. The TF-tree of a robot is the relative position of all bodies of a robot with respect to eachother.

2.5 Gazebo

Gazebo is a physics simulator, which can be in combination with ROS. In gazebo entire worlds can be modelled. In this world, all kind of objects can be inserted. For example a motor can be inserted, which is coupled with a ROS topic. Gazebo is a physics simulator, so every friction and contact force can be modelled.

2.6 Urdf

To describe robotic systems in ROS the urdf file format is used. An urdf file is a xml file with tags describing links and joints of a robot. The file format can be used by both gazebo and rviz. It also contains information about how a robot must be visualized by textures, shapes and colors. The physical parameters are also described with this file.

To simplify the writing of urdf, the macro language xacro can be used. Xacro files can be automatically expanded into urdf files via the xacro script. Usually this is done when the urdf is loaded.

3 Robot setup

The robot used in this setup is the x80sv of drrobot. This robot has three wheels, of which two are controlled. Other sensors are range sensors, infrared and ultrasonic. The robot is extended with a laser range sensor (LRS) and a

laptop with ROS installed. The LRS and the controllerboard of the x80sv are connected via usb-serial cables. The controllerboard of the x80sv is provided with the robot.

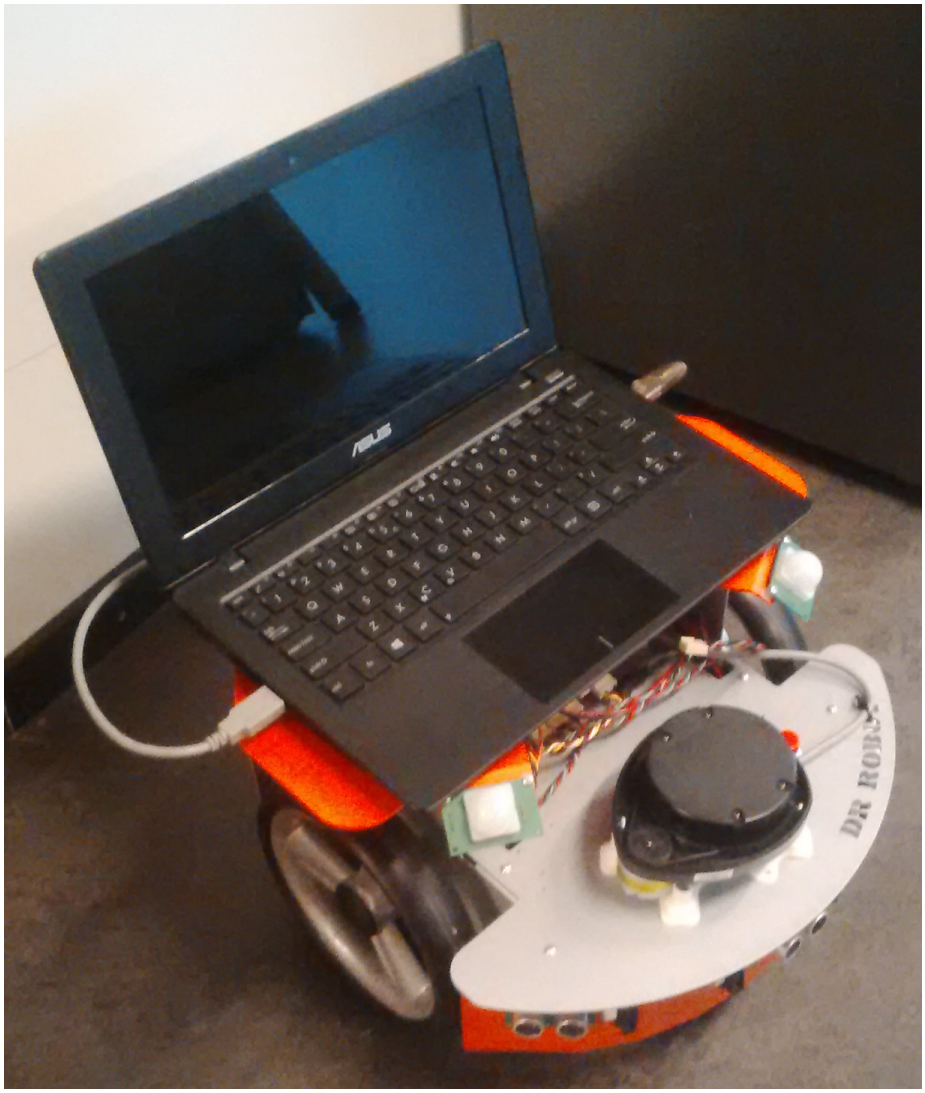


Figure 2: Photo of the x80sv equiped with laptop and LRS

Parameter	Value	Description
N_{enc}	756 [-]	Encoder ticks per wheel revolution
r_{wheel}	0.085 [m]	Wheel radius
l_{base}	0.283 [m]	Distance between left and right wheel

3.1 Installation

To use the robot, install ROS indigo on ubuntu 14.04 on the robot laptop. Download the x80sv software from the github repository [2]. Follow instruction

in the readme.

3.2 Robot description

To use the robot with the ROS system, an urdf model must be created. The model of the x80sv is located in the folder `x80sv_description`. The xacro macro system is used to simplify the writing of the urdf file. The urdf description contains a description of what links and joints the robot consists of. The model is used by rviz for visualization and by gazebo to construct the physical model of the robot to simulate it. The weights, shapes and materials of the links are also specified in this file.

3.3 Simulation

Instead of trying to run everything on the real robot, a simulated version of the x80sv was created.

With this simulation it is possible to run the exact same navigation software with the real robot as well as with the simulated variant.

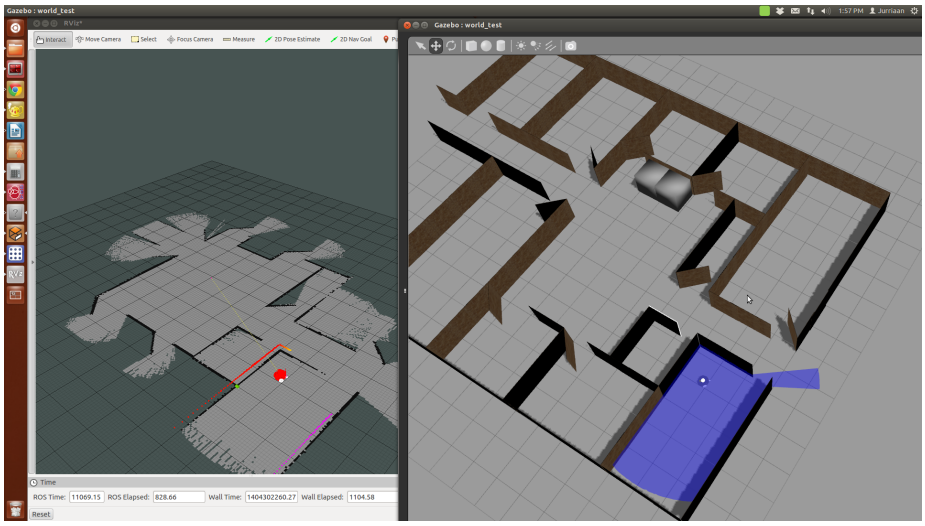


Figure 3: Gazebo (right) and rviz (left) running a simulated robot

4 Navigation on the x80sv

There is an existing framework for 2d mobile base navigation `http://wiki.ros.org/move_base`. Another framework is the skynav framework developed at saxion `https://github.com/SaxionLED/skynav` Those two should be interchangable. They both publish the ***cmd_vel*** topic, subscribe to sensors such as laser, infrared and ultrasonic, and they receive navigation goals from a higher level module. For this project the move-base navigation stack was selected.

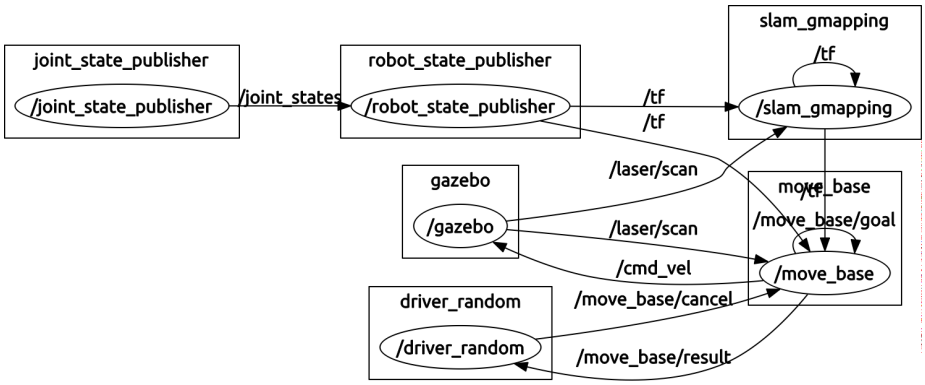


Figure 4: The nodes involved in the simulation

4.1 Kinematics

Two-wheeled vehicles have the following kinematic equations of motion. Given Δ_{left} and Δ_{right} as the traveled distance for the left and right wheels, we can calculate the traveled distance forward and rotation.

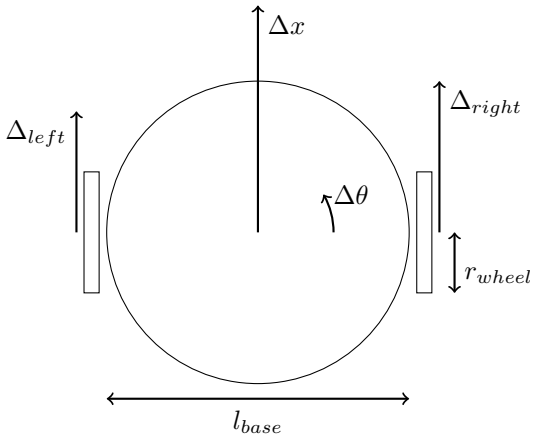


Figure 5: Robot kinematics

$$\Delta x = \frac{\Delta_{left} + \Delta_{right}}{2} \tag{1}$$

$$\Delta\theta = tan^{-1}(\frac{\Delta_{right} - \Delta_{left}}{l_{base}}) \tag{2}$$

Given these traversed deltas we can now compute the new position and orientation of the robot in world coordinates.

$$x_{x+1} = x_n + \cos(\theta_n)\Delta x \tag{3}$$

$$y_{x+1} = y_n + \sin(\theta_n)\Delta x \tag{4}$$

$$\theta_{x+1} = \theta_n + \Delta\theta \tag{5}$$

4.1.1 Control

The velocity control is done by the controllerboard of the robot. The commands to the controllerboard are wheel velocities in encoder ticks. A conversion from linear and angular velocities of the robot (v and ω) to wheel velocities (ω_{left} and ω_{right}) is required.

$$\omega_{left} = \frac{v - \frac{1}{2} * \omega * l_{base}}{r_{wheel}} \tag{6}$$

$$\omega_{right} = \frac{\frac{1}{2} * \omega * l_{base} + v}{r_{wheel}} \tag{7}$$

In the next step, the wheel velocities are translated into encoder velocities.

$$leftWheelCmd = \frac{-\omega_{left} * N_{enc}}{2\pi} \tag{8}$$

$$rightWheelCmd = \frac{\omega_{right} * N_{enc}}{2\pi} \tag{9}$$

These equations are implemented in the real robot drivers (`https://github.com/SaxionLED/x80sv/tree/master/x80sv_driver`). In case of simulation, these equations are done by gazebo.

4.1.2 Odometry

The task of the odometry system is to keep track of the robot position using wheel encoder data. To implement this for a two wheeled robot the following formulas are used:

$$d_{left} = calculateMovementDelta(mtr0) \tag{10}$$

$$d_{right} = calculateMovementDelta(mtr1) \tag{11}$$

$$averageDistance = (d_{left} + d_{right})/2 \tag{12}$$

$$\delta\theta = atan2((d_{right} - d_{left}), wheelDis_{)}; \tag{13}$$

$$\delta x = averageDistance * cos(\theta); \tag{14}$$

$$\delta y = averageDistance * sin(\theta); \tag{15}$$

$$\theta + = \delta\theta \tag{16}$$

$$x + = \delta x \tag{17}$$

$$y + = \delta y \tag{18}$$

These equations are implemented in the real robot drivers (`https://github.com/SaxionLED/x80sv/tree/master/x80sv_driver`). In case of simulation, these equations are done by gazebo.

4.2 Gmapping

Simultaneous localization and mapping (SLAM), is a required component for navigation. The task of the SLAM module in a robotic system is determining the robots position in space, and while doing that, determine a map of the environment. There exist several packages to do this.

For the x80sv the package ***ros-indigo-gmapping*** [1] was used. This node takes as input the odometry data from the control layer and the laser scan data. It then is capable of generating a map and determining the drift that occurred since odometry start.

The tf tree is visualized in figure 6. The map frame is the world frame. The gmapping node determines the transform between map and odom frame. The odom frame is the frame in which the robot started to drive. The transform betweet odom and base footprint is the transform as recorded by the odometry system. In the case of the simulation, gazebo takes care of this transform. When using the real robot, the x80 driver records the transformation. From the base footprint the rest of the mobile base consists of links which have static transforms.

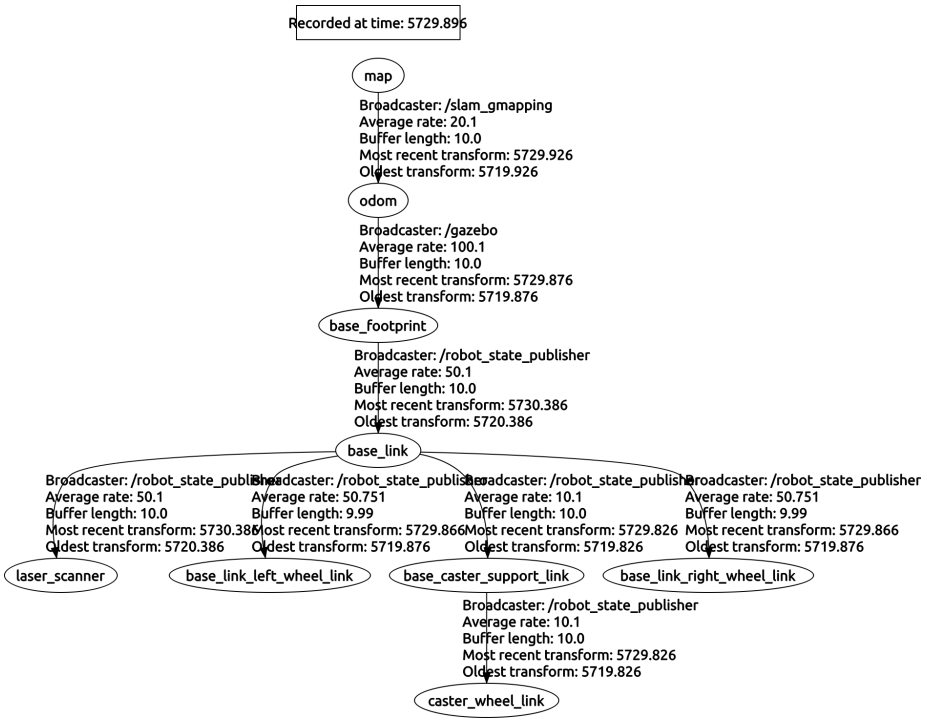


Figure 6: tf tree visualized with rqt tf tree plugin

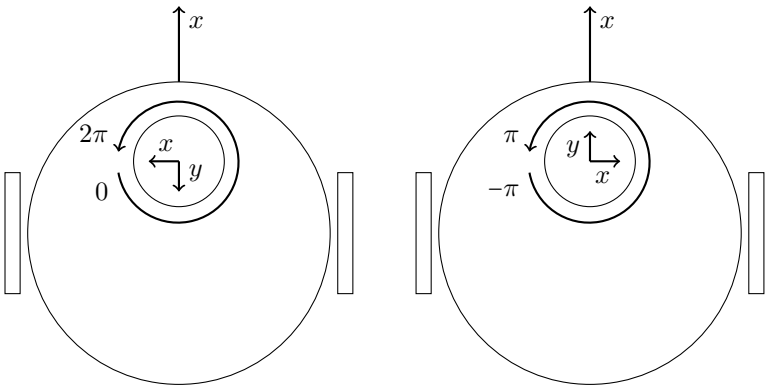
4.2.1 Laser orientation

The orientation of the laser is important for the gmapping node. The gmapping node assumes that the scan data is symmetric around zero angle.

This was discovered during experiments with the gmapping node. The minimum and maximum range $0..2\pi$ did not work. A range from $-\pi..\pi$ did work!

The scan data contains the same bytes, but the minimum and maximum angle as defined in the scan message was adjusted. Also the reference frame of the laser scanner had to be adjusted. All these changes could be done without the laser scanner being physically remounted!

These changes also resulted in the simulated and the real robot behave the same.



(a) Laser orientation and scan angles that did not work

(b) Laser orientation and scan angles that did work

Figure 7: Laser reference frame and minimum and maximum angles appeared to be important for the gmapping node

4.3 Move base

The ***ros-indigo-move-base*** package provides a sort of infrastructure for 2d mobile base navigation. It provides a structure into which plugins can be inserted. Among plugins are costmap function, recovery behaviors, local planner and global planner.

4.4 Costmap2d

The ***ros-indigo-costmap-2d*** is a ros package that can create a costmap of the environment of the robot. The costmap is then used by both global and local planning to determine the path. The costmap consists of several input layers. The static layer takes the map as determined by gmapping and inflates it somehwat. The obstacle layer can incorporate different sensors. Rviz can be used to visualize these costmaps.

4.5 Local planning

To implement local planning on the x80sv, the navigation tuning guide was followed closely [6].

4.5.1 Robot physical limits

In order to perform good navigation, certain robot parameters must be known. These include the robot maximum acceleration and maximum velocity for both

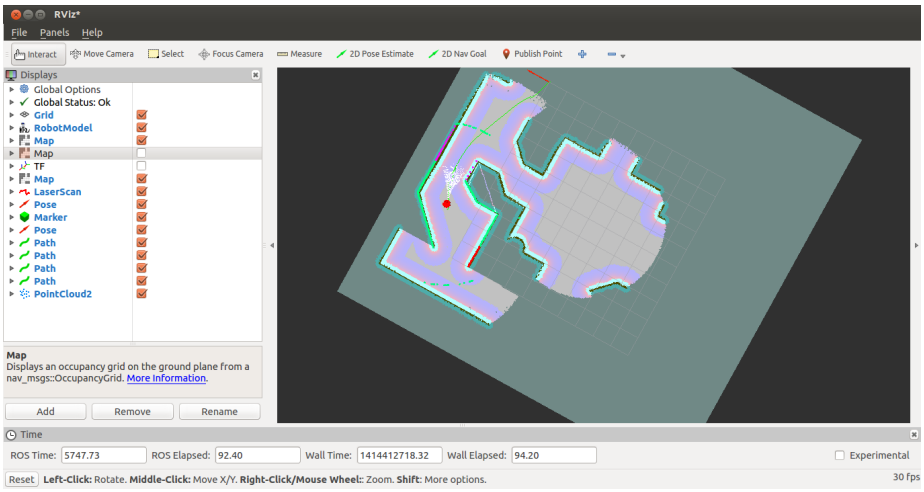


Figure 8: Navigation visualized in rviz

linear and rotational motion. To Determine this, the robot was driven at its maximum speed, and the velocities as measured by the encoders was logged. This was done using the ROS x80 driver, see listing 4.8.2. At line 1, the real robot driver is launched. The robot can now be controlled via rqt. Line 2 starts a recording of the odom and cmd_vel topics into a bag file. Now, the robot was moved using the robot steering plugin of rqt. At line 3 the rosbag file is converted to a csv file. Line 4 activates a script that plots the csv file into figure 9.

From figure 9 the following robot limitations are determined:

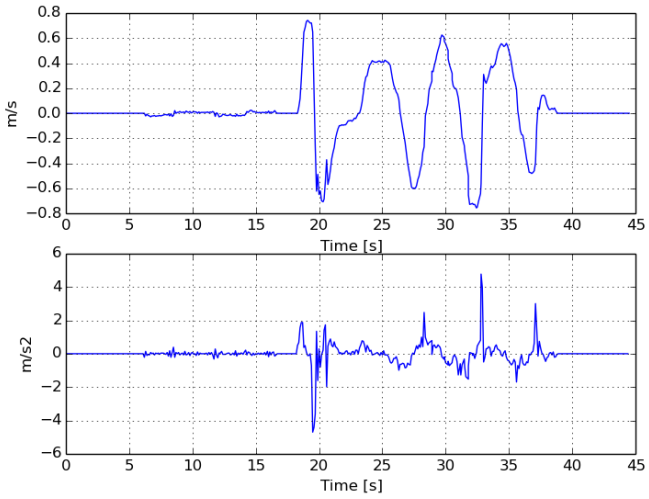
Parameter	Value
v_{max}	$0.4[m/s]$
a_{max}	$1.0[m/s^2]$
ω_{vmax}	$2.0[rad/s]$
α_{max}	$5.0[rad/s^2]$

Listing 1: commands to determine robot limitations

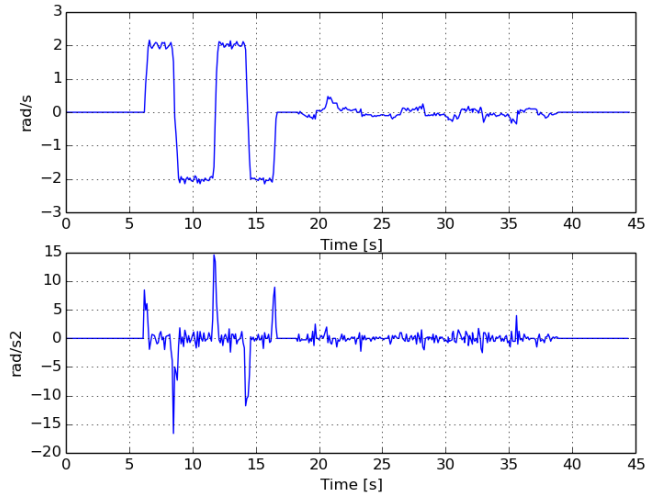
```
1 launch x80sv_bringup real_robot_driver.launch
2 rosbag record odom cmd_vel
3 rostopic echo -b 2014-12-01-14-44-27.bag -p
    /odom/twist > measurement.txt
4 python3 plot_acceleration.py
```

4.5.2 base local planner

The default planner of the ***ros-indigo-move-base*** package for ROS is the base local planner. This planner uses dynamic window approach (DWA) to plan a path. This means that from the current location several path options are simulated in advance and the one with the least cost is selected. This planner has several parameters that must be tuned to a specific robot.



(a) Measured linear velocity and calculated acceleration



(b) Measured rotational velocity and calculated acceleration

Figure 9: Measured velocities

4.5.3 DWA local planner

The dwa local planner is also a standard ROS planner. It is contained in the package ***ros-indigo-dwa-local-planner***. It is simpler than the base local planner, that is why this package is chosen as currently the best solution.

4.5.4 Smooth planner

The smooth planner is own work, and is located at github (`https://github.com/SaxionLED/x80sv/tree/master/smooth_local_planner`). This planner follow the global path and when confronted with an obstacle simply reports that the plan has failed and waits for new commands from the global planner. This planner can be extended to include smooth motion profiles to the wheel base.

4.6 Global planning

4.6.1 Global planner

The ***ros-indigo-global-planner*** package contains a global planner that uses simple search for an optimal path given a costmap. It is a plugin for use with move-base. This was the global navigation plugin that was used for the x80sv.

4.6.2 Navfn planner

The ***ros-indigo-navfn*** package provides another plugin for move-base for global navigation. This is an existing ROS module. It was tried on the x80sv, and is an alternative to ***ros-indigo-global-planner***.

4.6.3 Move-base-ompl

Ompl is a motion planning library using random trees. Random trees follow the idea of randomly exploding a tree of all possible state of a robot given its current position and vehicle dynamics. During this project a wrapper was made around the ompl library, so that it is usable as a global planner. The node works as a plugin, but the results are poor. This must be investigated further. [3] [4]

4.7 Recovery behavior

Sometimes the robot may come in a situation where the navigation fails. In this case, the ***ros-indigo-move-base*** package has something that is called recovery behaviors. When both local and global navigation fail in steering the robot, several recovery behaviors can be configured.

One of those is the rotate recovery behavior. This behavior rotates the robot about its axis and records the map using its lases scanner. Obstacles that were closeby, may be removed by this action, so after a rotation normal navigation can be retried.

4.8 Commanding the planner

Once the global and local planners are in place, a navigation goal must be given. The move-base node can be commanded via the actionlib package. An action cannot be easily given via command line, but is possible using rviz. In rviz select "2d nav goal" and click a point on the map where the robot should go. Other nodes that publish navigation actions are the random driver and the frontier exploration.

4.8.1 Random driver

During this project, a python script was developed to steer the robot to predefined positions. This was done by using a simple action client object and sending these objects goals. By using simple action client, the goal could be sent and the result checked.

4.8.2 frontier exploration

The ***ros-indigo-frontier-exploration*** package contains a node that checks the map and commands the move-base system to various goals in order to explore unknown space. To use this package, install it, and launch it with:

```
1 roslaunch frontier_exploration global_map.launch
```

Then in rviz, click publish point and click four points in the map. This will create a polygon in which the frontier exploration will explore. To view the polygon, add a polygon marker in rviz for the exploration_polygon_marker topic. Finally publish a point inside the polygon. The move base system will now receive goal positions.

5 Results

The odometry on the real robot was checked by driving the robot around and visualizing the laser in rviz with a decay time of about 30 seconds. All scans were registered over each other.

The move-base package was tried both in a simulated and a real world environment. See figure 11.

A sequence of images was recorded using the hector_geotiff library.

Listing 2: Recording the map as generated by gmapping

```
1 rosrun hector_trajectory_server
     hector_trajectory_server
2 rosrun hector_geotiff geotiff_node map:=dynamic_map
3 rostopic pub -r 1 /syscommand std_msgs/String "data:
     'savegeotiff'"
```

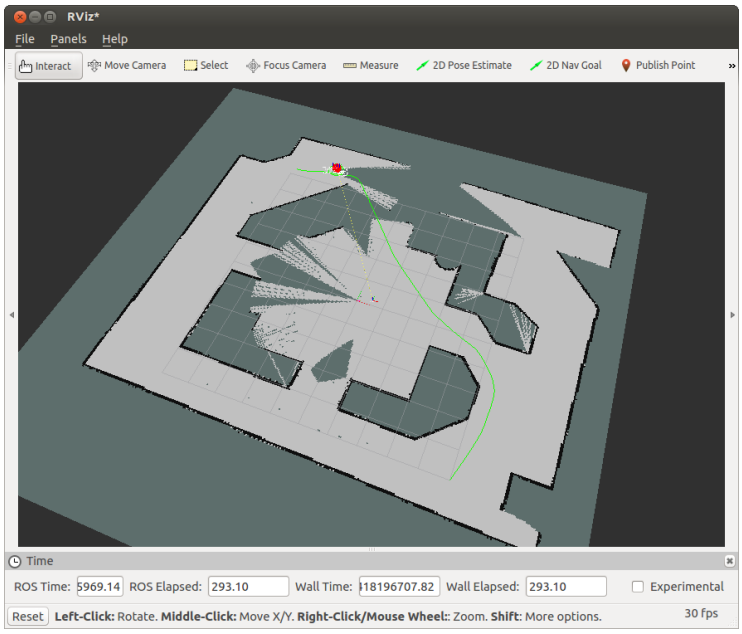
5.1 Issues encountered

5.1.1 Extrapolation error

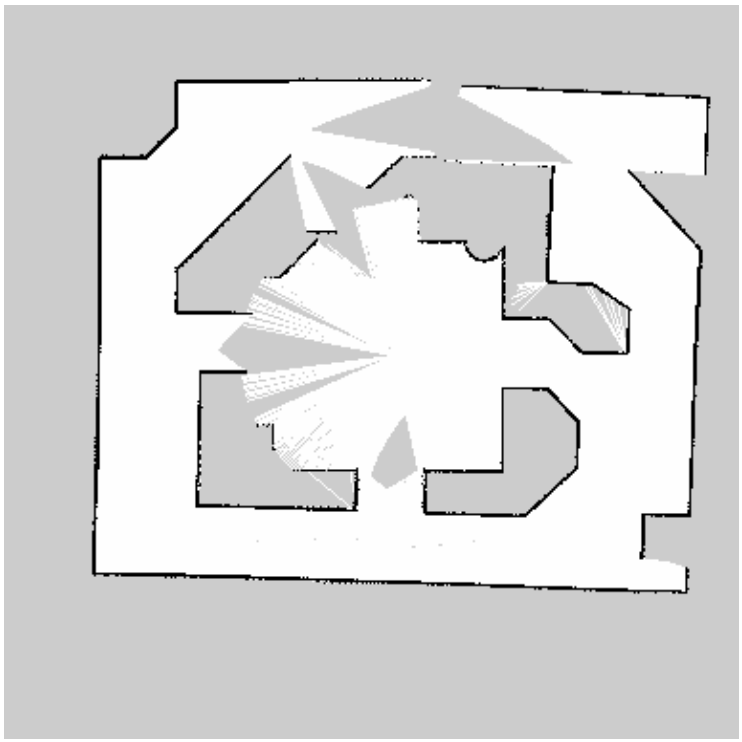
An issue encountered was that when running the navigation software on the robot, the move-base node reports an error:

Listing 3: Error message when running on different pc

```
1 [ INFO] [1418207278.346072074]: Got new plan
2 [ERROR] [1418207278.346767687]: Extrapolation Error:
     Lookup would require extrapolation into the
     future.  Requested time 1418207278.408950434 but
     the latest data is at time 1418207278.374471708,
     when looking up transform from frame [odom] to
     frame [map]
```

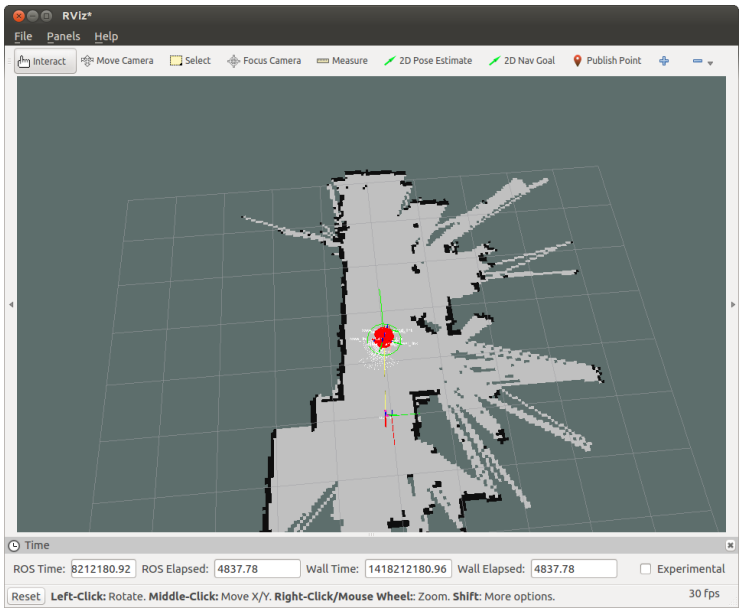


(a) rviz view

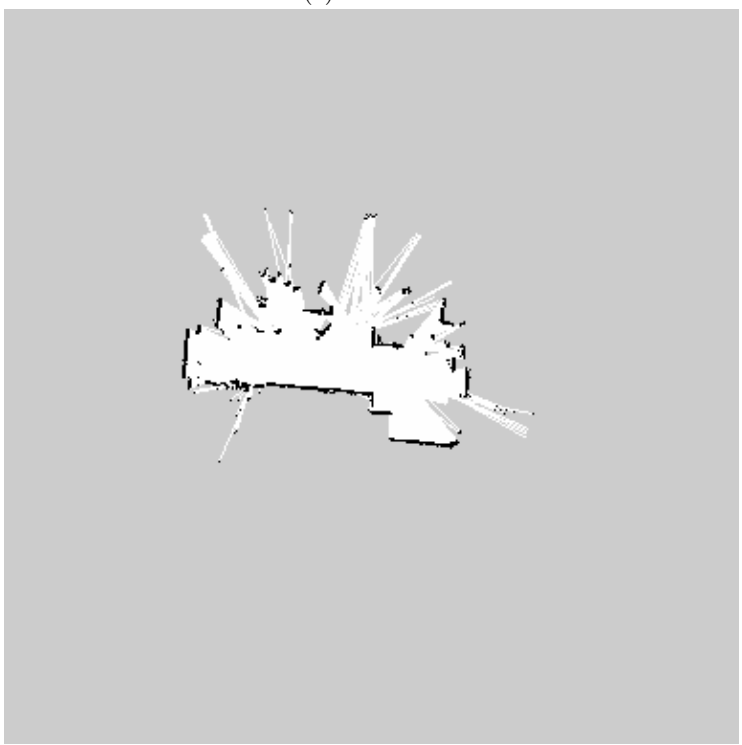


(b) resulting map

Figure 10: Experiments in the simulator



(a) rviz view



(b) resulting map

Figure 11: Experiments in the real world

3

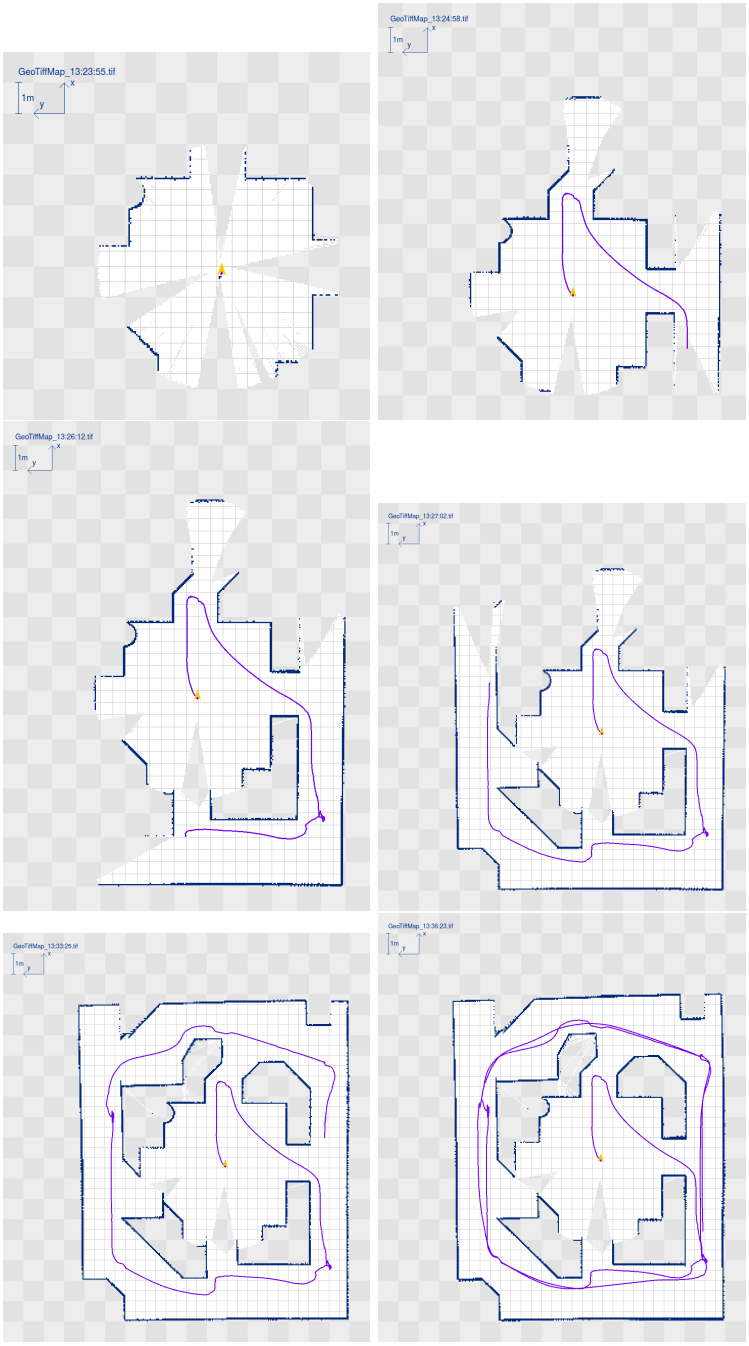


Figure 12: Simulated robot exploration and planning

```
4 [ERROR] [1418207278.346911345]: Global Frame: odom
     Plan Frame size 31: map
5
```

```
6 [ WARN] [1418207278.347037572]: Could not transform
     the global plan to the frame of the controller
7 [ERROR] [1418207278.347137469]: Could not get local
     plan
8 Registering Scans:Done
9 [ INFO] [1418207278.596014219]: Got new plan
```

The source of this error is that the time synchronization between the laptop and the pc is accurate enough. One solution was to run all navigation software on the laptop on the robot. Another solution is to always use the map frame. Usually the odom frame is used for local navigation, but we can use the map frame as well.

6 Conclusions

The odometry and the gmapping localization were realized on the x80sv platform. The ***ros-indigo-gmapping*** package yields good results for mapping and positioning. The move-base package was used to navigate the robot. Several alternative planners were tried. The ***ros-indigo-global-planner*** package was the best global planner. The ***ros-indigo-dwa-local-planner*** was the best evaluated local planner.

7 Future work

- The skynav navigation stack could be transformed to work via the plugin system of move base ros package. The advantage is that this navigation stack is better suitable with move-base. The disadvantage is that the limitations of the move-base package will be encountered.
- The black box controller PCB could be replaced by a new design, which is simpler and contains a known controller for the wheels. This has the advantage that the controller is known and can be modified.
- The laptop on the robot should be replaced by an embedded solution, capable of running the navigation stack.
- The smooth-planner and ompl planner could be further fine-tuned and extended. This would result in a wider range of available planners, and thus more flexibility.
- As an alternative for gmapping, the hector-slam module and amcl [5] could be tried as an alternative. It is now unknown what the capabilities of these modules are.
- The semantic processor should be implemented.
- The initial map and robot position problems should be implemented.

A Demo manual

Install ros indigo the software from github [2] onto the robot laptop and the host pc.

Make sure the networking settings are setup correct. This means the the file `/etc/hosts` must contain something like this:

```
1 127.0.0.1   localhost
2 192.168.10.102 x80sv
```

Test this by doing a ping command from the host pc to the x80sv laptop and vice versa:

```
1 user@hostpc$ ping x80sv
```

To run the demo, the roscore program must be started on the robot:

```
1 user@x80sv$ roscore
```

On the host pc make sure that the `ROS_MASTER_URI` environment variable is set.

```
1 user@hostpc$ export ROS_MASTER_URI=http://x80sv:11311/
```

To demo the robot with the navigation software, follow the following steps:
Launch the robot driver in a new terminal window on the robot:

```
1 user@x80sv$ roslaunch x80sv_bringup
    real_robot_driver.launch
```

Now open rqt and use the robot steering plugin to verify that the robot moves.

If the robot moves correctly, launch the navigation stack (either on the robot itself or another laptop or PC in the case of ros multimaster mode):

```
1 user@x80sv$ roslaunch x80sv_navigation
    x80sv_navigation.launch
```

Now open rviz and give a navigation goal by using the 2d nav goal tool in rviz. To visualize different topics rviz, in the view panel click add, and add the robot model, the laserscan message and the map.

```
1 user@hostpc$ rviz
```