Implementation of 2d navigation for a moving base using ROS (version 0.1)

Windel Bouwman

December 4, 2014

Contents

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 robot used was moving base, the x80sv.

2 Navigation overview

The task of navigation can be split up into the following tasks (see also 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.
- Odometry calculation. 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).

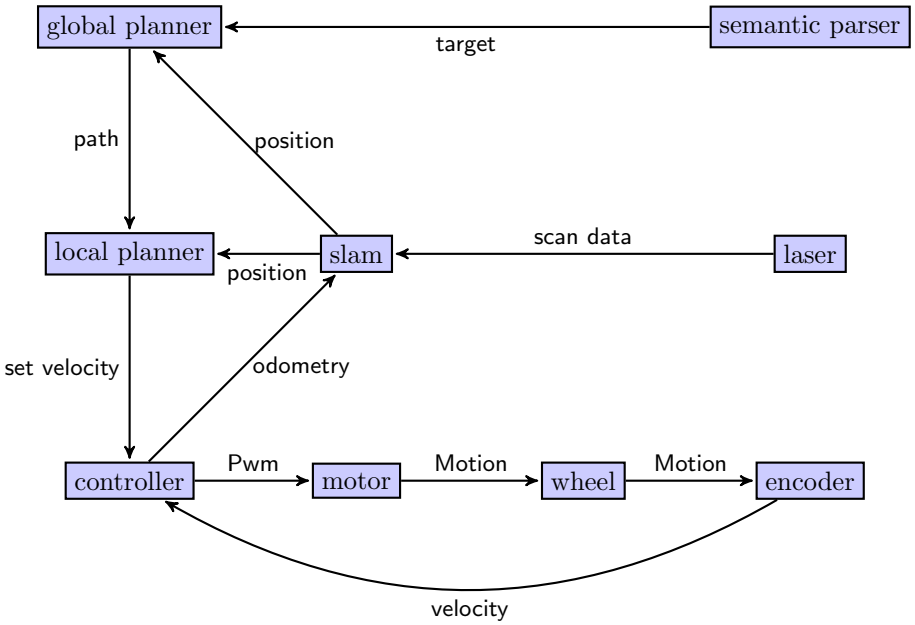


Figure 1: Subjects involved in the task of navigation

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.

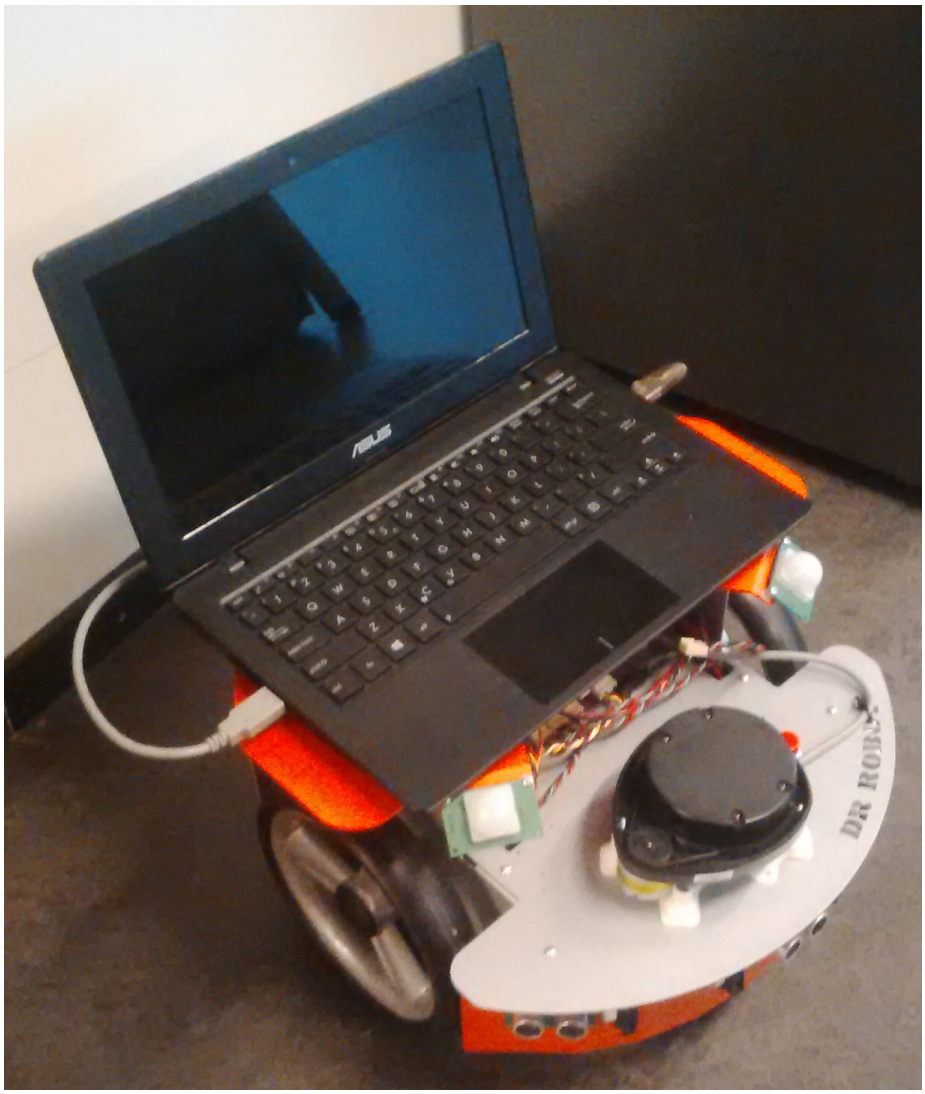


Figure 2: Photo of the x80sv equiped with laptop and LRS

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 (`https://github.com/saxionled/x80sv`). Follow instruction in the readme.

3.2 Kinematics

Two-wheeled vehicles have the following kinematic equations of motion:
TODO

3.3 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 to wheel velocities is required.

$$v_{left} = (2 * v_{linear} - v_{angular} * wheelDistance)/(2 * wheelRadius) \quad (1)$$

$$v_{right} = (v_{angular} * wheelDistance + 2 * v_{linear})/(2 * wheelRadius) \quad (2)$$

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

$$leftWheelCmd = -motorDir * v_{left} * encoderOneCircleCnt/(2\pi) \quad (3)$$

$$rightWheelCmd = motorDir * v_{right} * encoderOneCircleCnt/(2\pi) \quad (4)$$

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.

3.4 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:

$$d_{left} = calculateMovementDelta(mtr0) \quad (5)$$

$$d_{right} = calculateMovementDelta(mtr1) \quad (6)$$

$$averageDistance = (d_{left} + d_{right})/2 \quad (7)$$

$$\delta\theta = atan2((d_{right} - d_{left}), wheelDis_{)}; \quad (8)$$

$$\delta x = averageDistance * cos(\theta); \quad (9)$$

$$\delta y = averageDistance * sin(\theta); \quad (10)$$

$$\theta+ = \delta\theta \quad (11)$$

$$x+ = \delta x \quad (12)$$

$$y+ = \delta y \quad (13)$$

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.

3.5 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.

3.6 Robot limitations

In order to perform good navigation, certain robot parameters must be known. These include the robot maximum acceleration and maximum velocity for both 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 1. 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 3.

From figure 3 the following robot limitations are determined:

Parameter	Value
$vmax_{linear}$	0.4
$amax_{linear}$	1.0
$vmax_{rotational}$	2.0
$amax_{rotational}$	5.0

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 ROS concepts

4.1 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.

4.2 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.

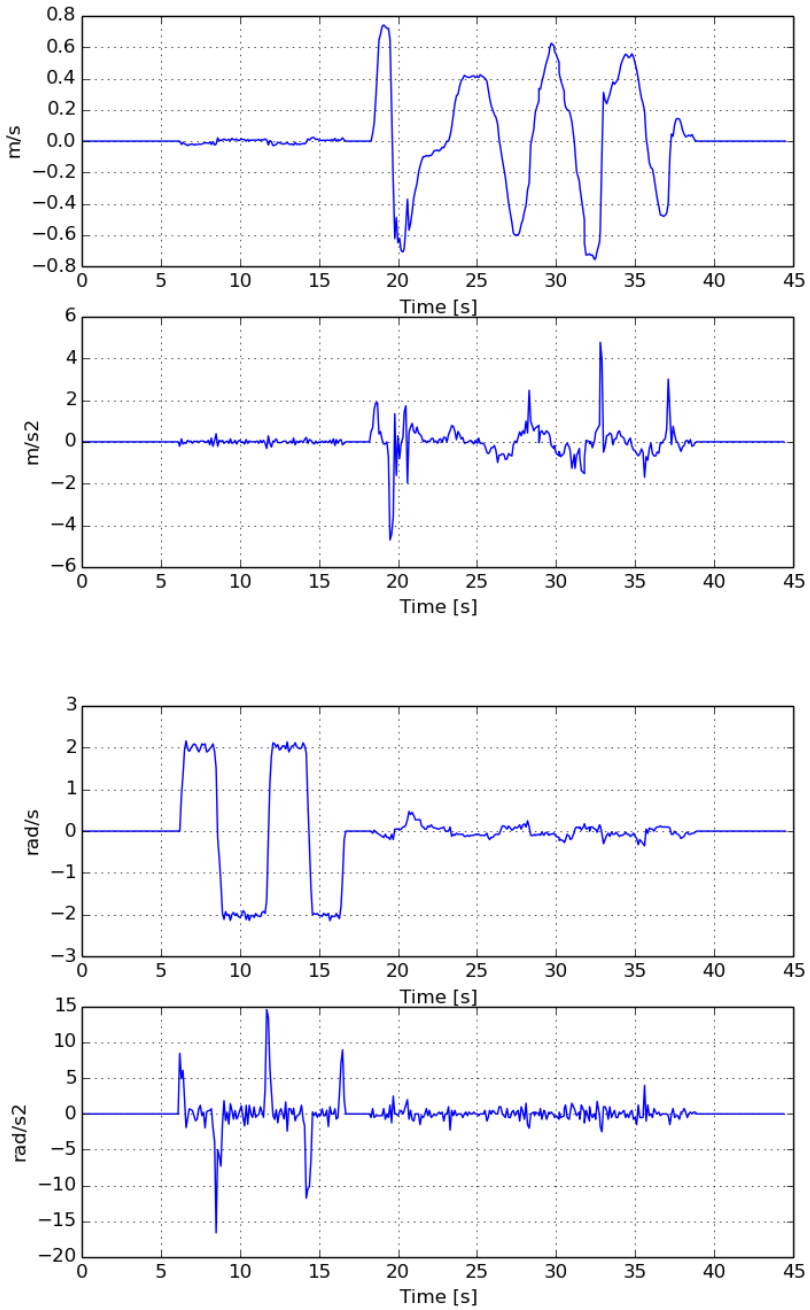


Figure 3: Measured velocities

4.3 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.

5 Slam

Simultaneous localization and mapping (SLAM), is a required component for navigation. The gmapping node was used in the case of the x80sv.

5.1 gmapping

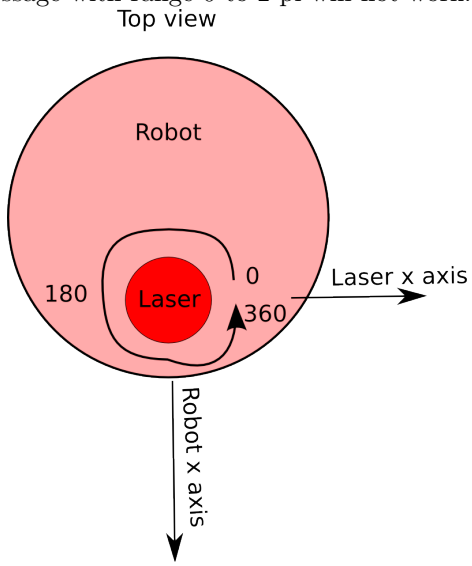
For slam the standard package ***ros-indigo-gmapping*** [1] can be used.

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

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.

5.2 Laser orientation

The orientation of the laser is important for the gmapping node. The node assumes that scan is symmetric around zero angle. This means that a laserscan message with range 0 to 2 pi will not work. A range from -pi to pi will work!



6 Simulation

Instead of trying to run everything on the real robot, a simulated version of the x80sv was created. This was done using the gazebo simulator. Gazebo is a physics simulator, which can be used with ROS.

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

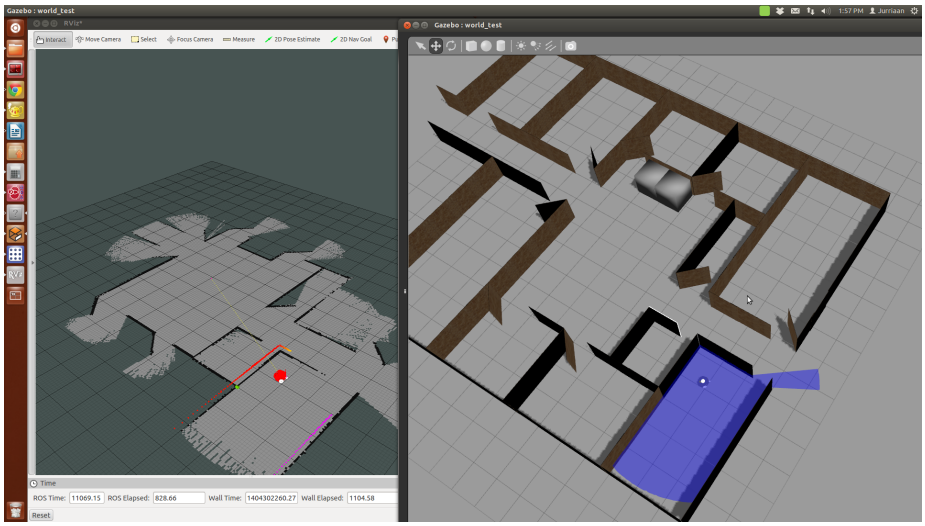


Figure 4: Gazebo (right) and rviz (left) running a simulated robot

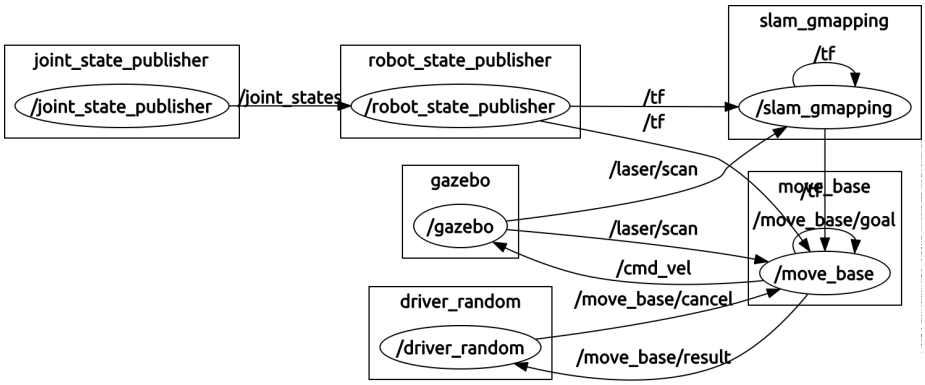


Figure 5: The nodes involved in the simulation

7 Navigation

7.1 Framework

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`

7.2 skynav

TODO

7.3 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.

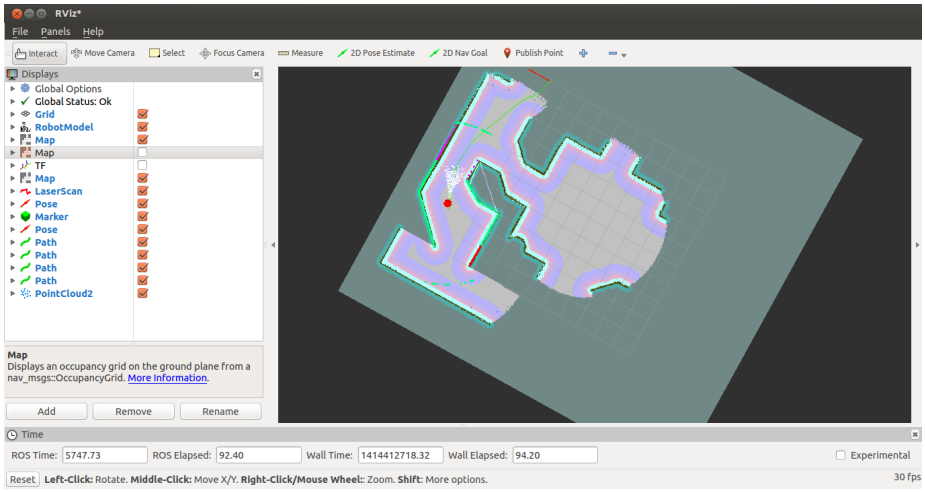


Figure 6: Navigation visualized in rviz

7.4 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.

8 Global planning

8.1 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.

8.2 navfn planner

The ***ros-indigo-navfn*** package provides another plugin for move-base for global navigation.

8.3 move-base-ompl

This package contains a plugin wrapper for 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.

`https://github.com/windelbouwman/move-base-ompl`
`http://ompl.kavrakilab.org/`

9 Local planning

9.1 base local planner

The default planner of the 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.

9.2 DWA local planner

The dwa local planner is also a standard ROS planner. It is contained in the package ***ros-indigo-dwa-local-planner***.

9.3 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.

10 Future work

- The skynav navigation stack could be transformed to work via the plugin system of move base ros package.