

Technical autonomous systems (TAS)

Introduction to the hard- and software for the course

Contents

1 Overview	2
1.1 Hardware	3
1.1.1 Motors and controllers	3
1.1.2 Arduino board	4
1.1.3 Linux-Board	4
1.1.4 Laserscanner	5
1.1.5 Inertial measurement unit (IMU)	6
1.2 Software	6
2 The TAS-package	8
2.1 Installing the TAS-package	9
2.2 Hardware drivers	10
2.2.1 wiimote	10
2.2.2 wii_control	11
2.2.3 rosserial_python	11
2.2.4 hokuyo_node	11
2.2.5 xsens_driver	11
2.3 TAS-Odometry	12
2.3.1 hector_mapping	12
2.3.2 tas_odom	13
2.4 Transformations	13
2.5 Navigation	14
2.5.1 map_server	15
2.5.2 amcl	15
2.5.3 move_base	16
2.5.4 tas_autonomous_control	16
3 Additional topics	17
3.1 Shell scripts	17
3.2 External RVIZ	17
3.3 VNC	17
3.4 Recording and playing data	17
3.5 Integrated Development Environments (IDE's)	17
3.6 Version control with git	17
3.7 Arduino code	17
3.8 Switching remote controller	17
Bibliography	18

1 Overview

The following chapter will give a quick overview of the hard- and software which is currently used on the remote-controlled car. The whole system can be seen in picture 1.1

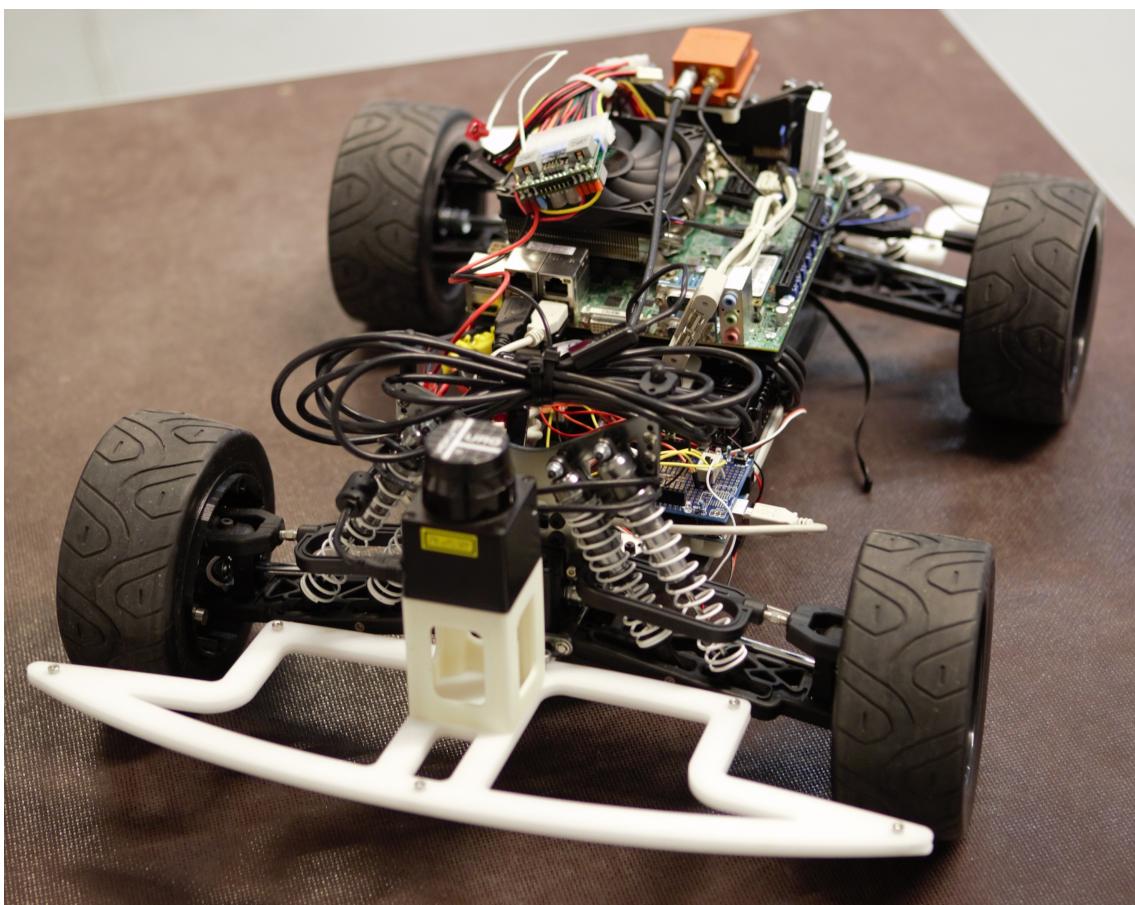


Abb. 1.1: Picture of the rc-car

1.1 Hardware

This section will give a short introduction to the hardware which can be found on the car.

1.1.1 Motors and controllers

On the car we can find two electrical motors, one for driving the car and one for the steering.

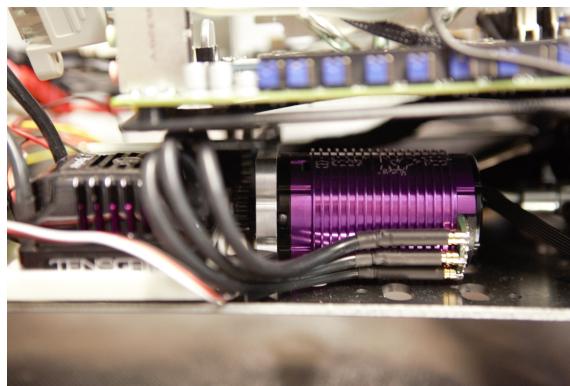


Abb. 1.2: HACKER Skalar 8 brushless motor

In picture 1.2 we can see the electrical brushless motor which is used to drive the wheels. The motor is connected to the wheels over a gear and two differentials. Table 1.1 gives some details of the used type. More information can be found here:

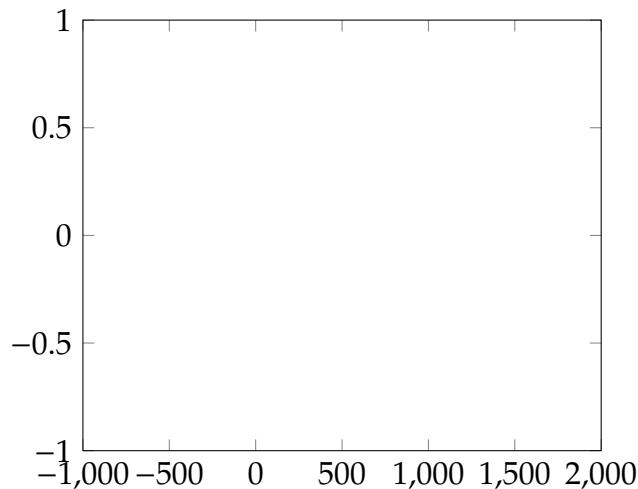
<http://www.hacker-carline.de/produkte/skalar-motoren/hacker-skalar-10/>

For steering, there is a second motor at the front of the car (see picture ??).

Both motors bring its own controllers which take a PWM-signal as input to set the desired revolution rate or angle. The signal is shown in picture ?. The input signals are provided by a little Arduino board (see next section 1.1.2).

Name:	HACKER Skalar 8 1750
max. Power (W):	1650
max. RPM (1/min):	25900
max. voltage (V):	14.8
max. current (A):	110

Table 1.1: Hacker Skalar 8 - Details



1.1.2 Arduino board

To provide the signals to the motor controllers there is a little Arduino board on the car which can be seen in picture ???. It receives the PWM-values from the mainboard of the car via USB and sets them to the output pins which are connected to the controllers.

1.1.3 Linux-Board

For computing there is a Supermicro X10SLV motherboard with a Intel Core i7 processor on it. To control the system we use Ubuntu 14.04 and a special software framework called ROS (see section 1.2). Table 1.2 gives a short overview of the computer system. More information can be found on:

<http://www.supermicro.com/products/motherboard/Core/H81/X10SLV.cfm>

Motherboard:	Supermicro X10SLV
Processor:	Intel Core i7
Memory:	4GB DDR-RAM
	200GB SSD Hard Disc

Table 1.2: Computer system - Details

TODO: Correct entries in table (fake values)!

1.1.4 Laserscanner



Abb. 1.3: Hokuyo URG-04LX-UG01

At the front of the car we can find a Hokuyo URG-04LX-UG01 laserscanner (see picture 1.3). The scanner is used for location in a given 2-D map or to create maps while the car is moving through the environment. For detailed information about the scanner refer to:

https://www.hokuyo-aut.jp/02sensor/07scanner/urg_04lx_ug01.html

Name:	Hokuyo URG-04LX-UG01
Measuring area:	20 to 5600mm(white paper with 70mm×70mm), 240°
Accuracy:	60 to 1,000mm : ±30mm, 1,000 to 4,095mm : ±3 percent of measurement
Scanning time	100ms/scan

Table 1.3: Laserscanner - Details

1.1.5 Inertial measurement unit (IMU)

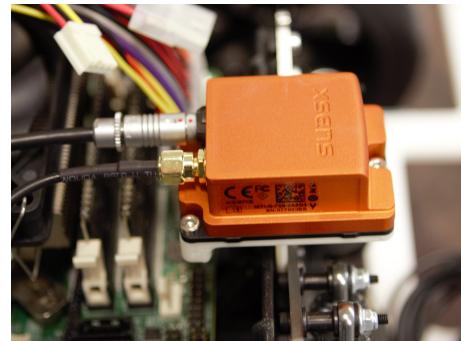


Abb. 1.4: Xsens MTI-G-700

At the back of the car there is an X-Sens MTI-G-700 Inertial Measurement Unit (IMU). It contains sensors for accelerations in every spatial direction, for angle velocities and orientation about each axis and a GPS-module. Table 1.4 gives some details of the module. For more information see:

<http://www.xsens.com/products/mti-g-700/>

Name:	Xsens MTI-G-700
Output frequency:	Up to 2000Hz
Standard full range gyro:	450 °/s
Standard full range acc	50 m/s ²
In-run bias stability gyro	10 °/h

Table 1.4: IMU - Details

1.2 Software

The operating system currently running on the linux board of the car is Ubuntu 14.04. To work with the hardware there is installed a software framework called Robot Operating System (ROS) which provides a big number of libraries and tools to work with robots. On the car we use the currently newest version: ROS Indigo¹.

At the beginning of the course you should get familiar with the functionality of ROS. A good start are the official tutorials on:

<http://wiki.ros.org/>

After working with the tutorials you should understand the concept of nodes and topics and be able to create and build your own package. To work with the car

¹Since ROS Fuerte there are two different build systems. We use the catkin build system

there is a package on GitHub you can download and install on your system. See the next chapter for more information.

2 The TAS-package

This chapter will show you how to install the TAS-package on the car and give you some details about the different components. The package contains several self-written nodes and launchfiles to run the hardware (Laserscanner, IMU, Arduino) and the so called Navigation Stack properly on the car. The Navigation Stack provides tools for high level tasks like path planning and localization. The TAS-package also brings some shell scripts to make the work a little more comfortable. The folder structure can be seen in picture 2.1.

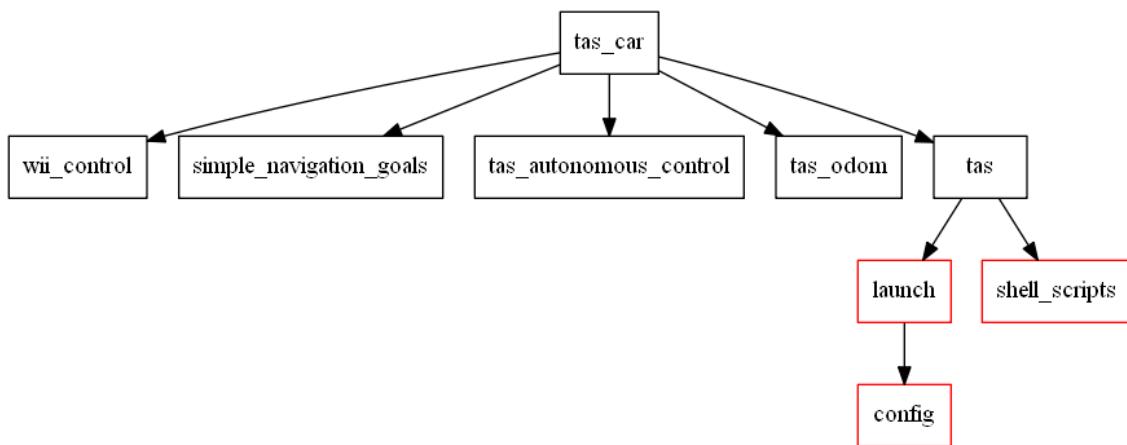


Abb. 2.1: Folder structure of the TAS-package

2.1 Installing the TAS-package

As it was mentioned in the previous chapter the TAS-package can be found on GitHub. To install it open a terminal and switch to your catkin workspace directory. In this example we assume your working directory in `~/catkin_ws`. To get and build the package on your car type in:

```
# cd ~/catkin_ws/src
# git clone https://github.com/LSR-TAS/tas_car
# cd ..
# catkin_make
```

If there occurred no errors, the package has been built and the car is now ready to run. Before running anything you should also do the following preparations to make your work in the terminal a little more comfortable:
Open a terminal and type in:

```
# sudo gedit ~/.bashrc
```

The `.bashrc` file is called automatically everytime you open a terminal in Ubuntu. Add the following lines to the end of the textfile¹:

```
PATH=~/catkin_ws/src/tas_car/tas/shell_scripts:"$PATH"
source ros_setup
```

TODO: Add `ros/opt..../setup.bash` to `ros_setup` file, and change file extension!

The first command will make it possible to run scripts from the specified folder without giving the full path. The second command sources the `ros_setup` file from the `shell_scripts` directory (see picture 2.1 for the folder structure). This script mainly sets the correct access rights to read and write from the USB ports (see ?? for more detailed information).

¹Eventually you have to change the path of your catkin workspace directory



2.2 Hardware drivers

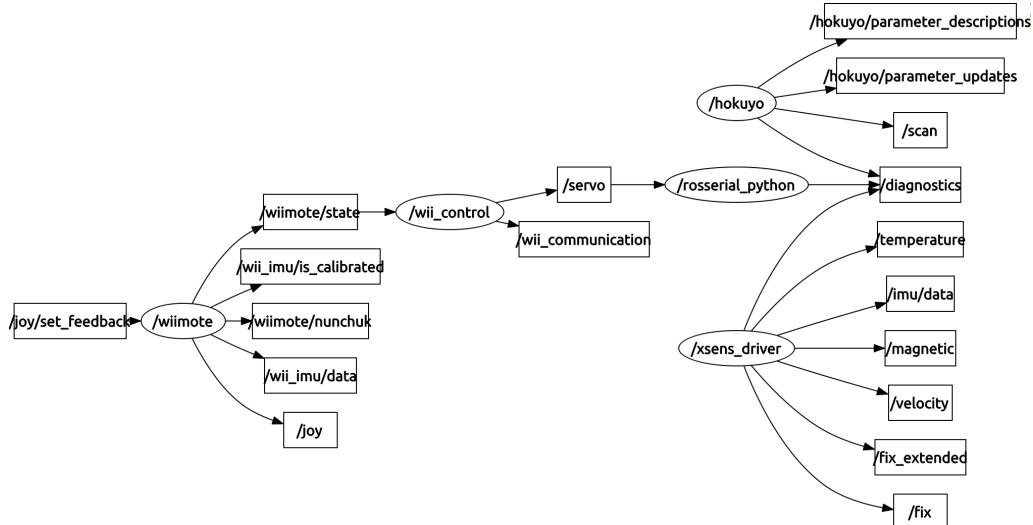


Abb. 2.2: Overview of the hardware drivers

To run the hardware drivers open a terminal and type in:

```
# rosrun tas/launch  
# roslaunch hardware.launch
```

This will run several nodes to communicate with the hardware which will be explained in detail in the following sections. To establish the connection to the wii-controller press the two buttons at the bottom of the controller (see picture ??) for about 5 seconds until you see a notification in the terminal window. The car now can be driven around manually.

In picture 2.2 you can see a graphical overview of the running nodes and topics. If you open a terminal and type in:

```
# rosrun rqt graph rqt graph
```

you should get the same picture.



2.2.1 wiimote

The `wimote` node reads various data from the wii-controller via bluetooth and publishes it to the appropriate topic. We use these as control inputs for the car.

<http://wiki.ros.org/wiimote?distro=indigo>

2.2.2 wii_control

The `wii_control` node takes the inputs from the controller which are published by the `wiimote` node. From these inputs it calculates the PWM values for the motor controllers and publishes them to the `/servo` topic.

2.2.3 rosserial_python

The `rosserial_python` node takes the PWM values from the `/servo` topic and sends them to the arduino. It mainly implements the communication channel between nodes and topics over a (virtual) serial port. For correct communication there has to be installed the so called `rosserial_client` on the arduino. For more information refer to:

<http://wiki.ros.org/rosserial>

2.2.4 hokuyo_node

The `hokuyo_node` is the driver for the laserscanner. It reads the measured data from the scanner via USB and publishes it to the `/scan` topic.

http://wiki.ros.org/hokuyo_node

2.2.5 xsens_driver

The package provides a driver for several different IMUs from the XSens company. It reads data from the sensor via USB and publishes it to the `/imu/data` topic just like the `hokuyo_node` does with the laser data.

http://wiki.ros.org/xsens_driver

2.3 TAS-Odometry

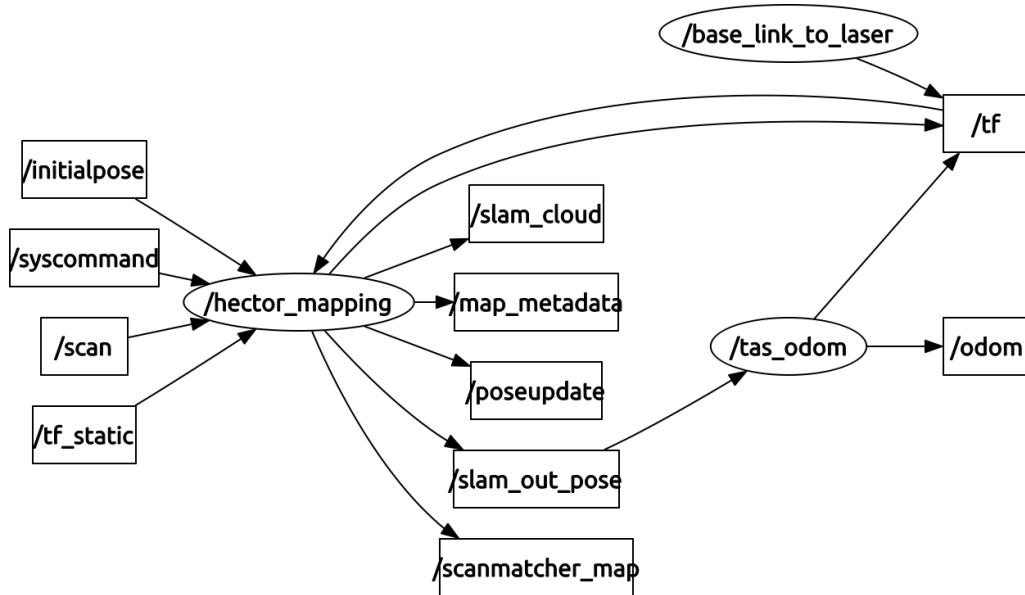


Abb. 2.3: Overview of the TAS-Odometry

Currently there is no real odometry on the car available. For the navigation packages though odometry data is required (see picture 2.4). This was solved by implementation of a "fake-odometry". It uses position and orientation estimated from the laserscanner data, calculates the velocities and provides it as odometry data.

To launch the fake odometry open a terminal and run the `odom.launch` file:

```
# rosrun tas/launch
# roslaunch odom.launch
```

Picture 2.3 shows the running nodes and topics for the fake odometry.

2.3.1 hector_mapping

The `hector_mapping` node implements a SLAM-algorithm² to build a map and localize the car in it simultaneously at runtime. Unlike other solutions for localization (see the `amcl` package in section 2.5.2) it only needs data from the laserscanner and does not require any odometry inputs.

The estimated position and orientation is published to the `/poseupdate` and the `/slam_out_pose` topics. The messages on the first topic additionally contain the covariances.

²Simultaneous Localization and Mapping

For correct calculation the node needs information about the relation between coordinate frame of the laserscanner and the coordinate frame of the car (refer to ?? for an introduction to transformations and frames).

For more details about the parameters and settings of the package see:

http://wiki.ros.org/hector_mapping

2.3.2 tas_odom

The tas_odom node subscribes to the `/slam_out_pose` topic (see previous chapter 2.3.1) and calculates the velocity from the estimated position by determining the difference quotient. Position, orientation and velocity is then published to the `/odom` topic. See the following page for information about the content of odometry messages:

http://docs.ros.org/indigo/api/nav_msgs/html/msg/Odometry.html

Additionally it publishes the transformation from the odometry coordinate frame to the frame of the car. See the next section for more details about transformations.

TODO: Velocity has to be in base_link frame/tas_odom publishes twist with velocity in odom frame!

2.4 Transformations

2.5 Navigation

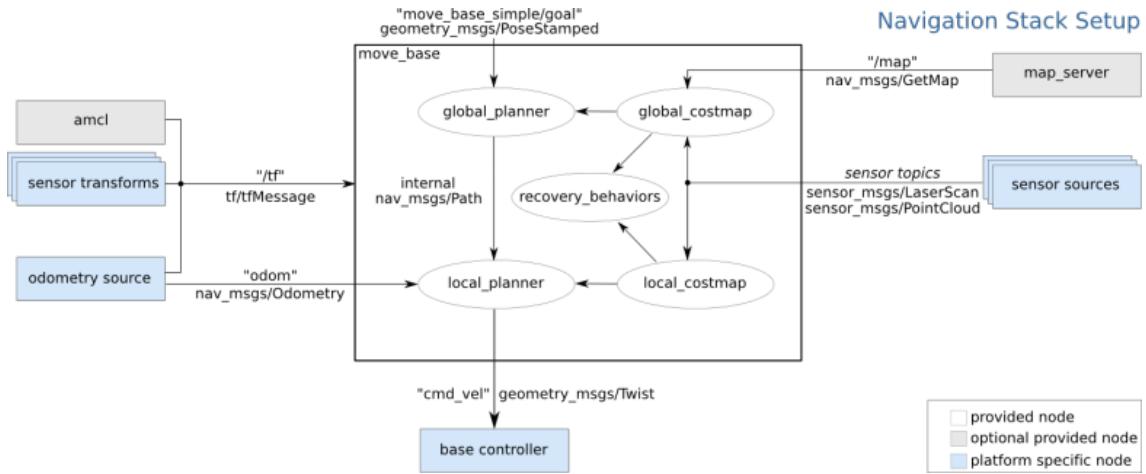


Abb. 2.4: Basic Navigation Stack

In ROS in general the Navigation Stack takes sensor inputs from odometry, laser and other sources and outputs velocity commands to reach a given goal. Picture 2.4 shows the parts of the basic Navigation Stack which is installed on the car.

To run the Navigation Stack open a terminal and type in:

```
# rosrun tas/launch
# roslaunch move_base.launch
```

The launch file will run all necessary parts of the Stack. Keep in mind that the hardware drivers and the TAS-Odometry package have to be launched before (see ?? for hardware drivers and ?? for odometry).

To specify a goal for the car you can use the visualization tool RVIZ. Open a terminal and type in:

```
# rosrun tas/launch
# roslaunch rviz rviz -d config/rviz/tas_rviz.rviz
```

This will run rviz with the configuration file in the config directory. At the beginning you have to give an initial estimation of the position to the `amcl` node. After that you can give goals to the `move_base` node which the car will try to reach (see the marked buttons in picture 2.5). To actually drive the car you have to push the C-button on the wii-nunchuk.

Alternately you can launch hardware drivers, TAS-Odometry, the Navigation Stack and the RVIZ tool at once by launching the `run_rviz.launch` file.

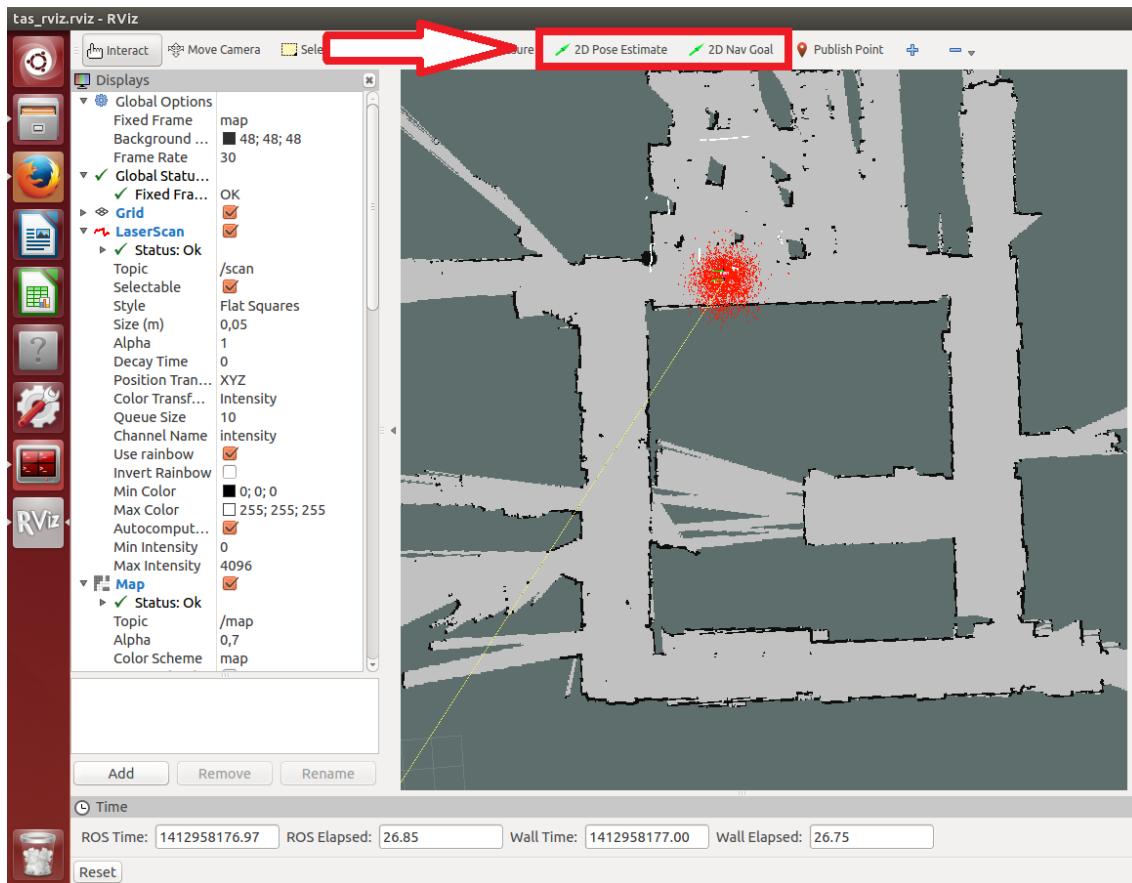


Abb. 2.5: RVIZ visualization tool

2.5.1 map_server

The `map_server` node takes a 2-D map of the environment and provides it as a ROS-service to other nodes. For every map there has to be a configuration file. The map file and the configuration file are located in the `config` directory of the TAS-package (see picture ?? for the folder structure). The path of these files can be changed in the launchfile. For more information about the parameters of the `map_server` package refer to:

http://wiki.ros.org/map_server

2.5.2 amcl

The `amcl` node implements different Monte Carlo localization algorithms to estimate the position of the robot. It requires a laserscanner, odometry and a map of the environment. The position and orientation with its covariances is published to the `/amcl_pose` topic.

The node takes odometry data in form of the transformation between the odom-



etry frame and the coordinate frame of the car. See section ?? for information about the `tf` package. Refer also to the package documentation on:

<http://wiki.ros.org/amcl>

2.5.3 move_base

2.5.4 tas_autonomous_control

3 Additional topics

3.1 Shell scripts

3.2 External RVIZ

3.3 VNC

3.4 Recording and playing data

3.5 Integrated Development Environments (IDE's)

3.6 Version control with git

3.7 Arduino code

TODO: Put arduino code in repository!

3.8 Switching remote controller

Bibliography

Zu erledigen:

Correct entries in table (fake values)	4
Add ros/opt..../setup.bash to ros_setup file, and change file extension	9
Velocity has to be in base_link frame/tas_odom publishes twist with velocity in odom frame	13
Put arduino code in repository	17