ENGINE: Mobile Robotics

Project: Explorer

Contents

1 Introduction

This is the manual for the maze exploration exercise of the course ”ENGINE: Mobile Robotics”. During this project you will learn how to work with:

- Derived C++ classes
- Virtual functions
- LaserScan Messages
- ROS Actions

In this project you should design and implement an high-level exploration algorithm that can explore as much area as possible in a given time. For this you can use any approach that you see fit for this task, e.g.:

- Rapidly Random Search Tree Exploration
- Wall-Follow Exploration
- Frontier Exploration

⋮

This project should provide you a basic idea of how exploration tasks are implemented in a mobile robotics system. Therefore you are going to use the Robot Operating System (ROS), the GAZEBO physics engine where a world and a mobile robot are simulated. All of this will be run inside a Docker container which will be built in the next section[1].

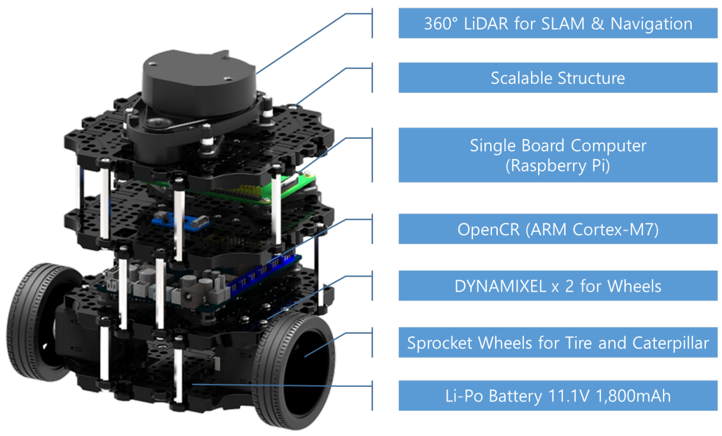
1.1 Simulation Overview

The remaining part of this chapter introduces the simulation environment GAZEBO as well as the simulated mobile robot Turtlebot3 Burger.

[1] If you have any questions feel free to email to engine@technikum-wien.at

1.1.1 Mobile Robot Turtlebot3 Burger

The TurtleBot3 Burger is a cost-effective, personalized robot kit driven using open source software developed by Robotis Bioloid[2]. It is equipped with necessary sensors to build a map of the environment and operate in it. Figure 1 visualizes the components of the Turtlebot3 Burger.



360° LiDAR for SLAM & Navigation
Scalable Structure
Single Board Computer (Raspberry Pi)
OpenCR (ARM Cortex-M7)
DYNAMIXEL x 2 for Wheels
Sprocket Wheels for Tire and Caterpillar
Li-Po Battery 11.1V 1,800mAh

Figure 1: System overview Turtlebot3 Burger modified taken from: https://www.slideshare.net/RobotisJapan/170520-10ros-tb3

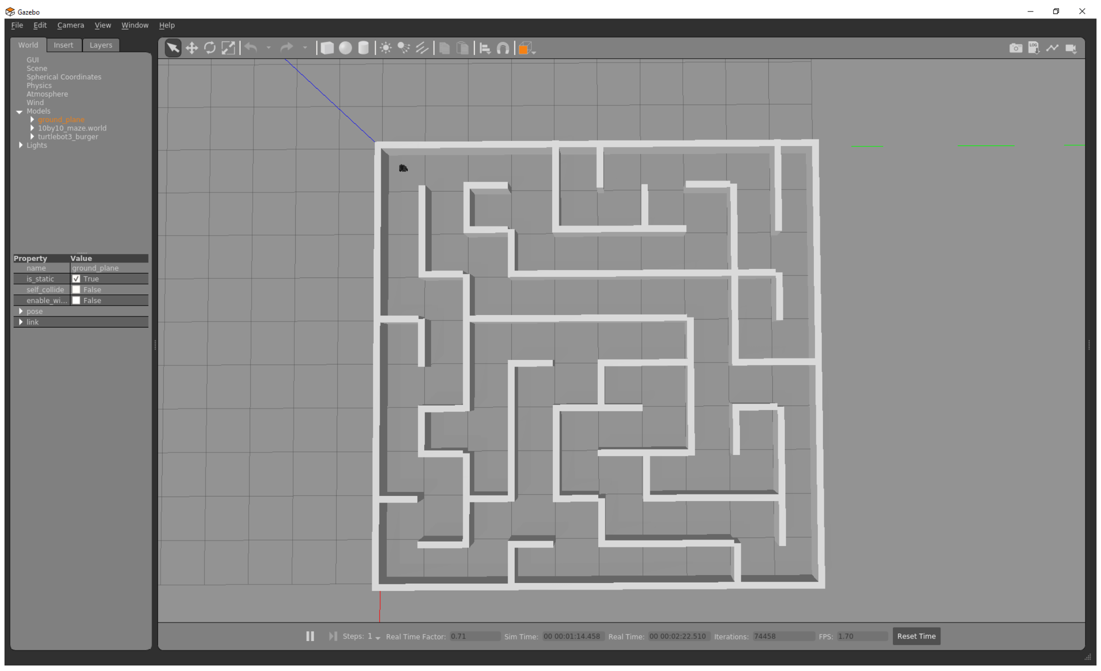
The 360°LIDAR is utilized during this exercise to build a map of the environment using gmapping[3] as well as to localise itself at the same time using AMCL[4]. This is known as the Simultaneous Localisation and Mapping[5] (SLAM) problem.

1.1.2 Simulation Environment GAZEBO

Robot simulation is an indispensable tool in the toolbox of every robotic engineer. A well-designed simulator allows you to quickly test algorithms, design robots, perform regression tests and train AI systems using realistic scenarios. Gazebo provides the ability to precisely and efficiently simulate populations of robots in complex indoor and outdoor environments. A robust physics engine, high-quality graphics and comfortable programmatic and graphical interfaces are available.
This exercises utilizes GAZEBO as a physics simulation engine to simulate the above mentioned actuators and sensors of the mobile robot as well as a automatically generated maze.

[2] http://emanual.robotis.com/docs/en/platform/turtlebot3/overview/
[3] http://wiki.ros.org/gmapping
[4] http://wiki.ros.org/amcl
[5] https://en.wikipedia.org/wiki/Simultaneous_localization_and_mapping

Figure 2[6] visualizes a running simulation of said maze and mobile robot.



Property	Value
name	ground_plane
is_static	✓ True
self_collide	False
enable_wi...	False

Steps: 1 Real Time Factor: 0.71 Sim Time: 00 00:01:14.458 Real Time: 00 00:02:22.510 Iterations: 74458 FPS: 1.70

Figure 2: Simulation overview

1.2 Build the docker container

The remaining part of this chapter walks you through the building procedure of the docker container.

1.2.1 Prerequisites

1. Install Docker (see: click me)
2. Install VcXsrv as X11-Server (only on Windows) (see: click me)

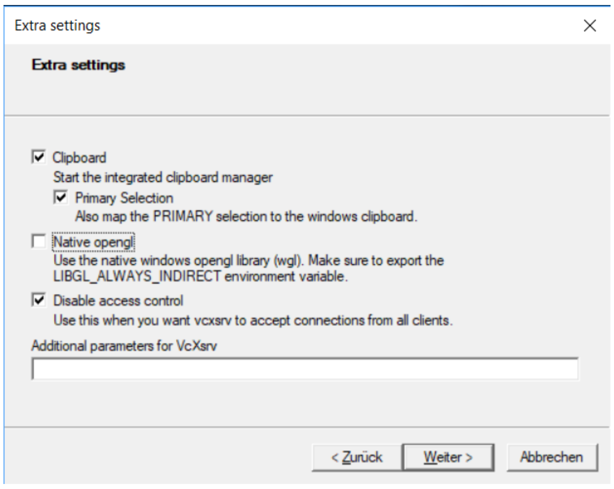
1.2.2 Build Container

1. Download all necessary files from **[here](link_zu_datein)**
2. Open downloaded folder in explorer
3. Double click on "build_docker_container.bat"

[6]Note that when you start the explorer node as described below (see 3) this Graphical User Interfaced will not be opened due to performance optimisation

1.2.3 Run Container

- Start VcXsrv (XLaunch) with following configuration[7]:



Extra settings

Extra settings

[✓] Clipboard
Start the integrated clipboard manager
[✓] Primary Selection
Also map the PRIMARY selection to the windows clipboard.
[ ] Native opengl
Use the native windows opengl library (wgl). Make sure to export the LIBGL_ALWAYS_INDIRECT environment variable.
[✓] Disable access control
Use this when you want vcxsrv to accept connections from all clients.
Additional parameters for VcXsrv

Figure 3: Configuration for the VcXsrv X-Server

- Double click on "run_docker.bat"
- Start programming!

1.3 Available Components

The files in ”catkin_ws/fhtw_maze_explorer” provide you with a skeleton ROS package which you will use throughout this project. This folder is included inside of the docker environment (/root/catkin_ws/src/fhtw_maze_explorer). It contains documentation, source-code as well as a GAZEBO world for simulation.

If you are not using the provided docker container but your own ubuntu computer make sure to install (either via binary or source) the following ROS packages:

- move-base*
- turtlebot3-*
- dwa-local-planner
- global-planner
- gmapping
- rosdoc-lite (only if you want to build the documentation)

[7]Make sure that VcXsrv is enabled for both public networks and private networks in the firewall.

- gdbgui (only if you want to debug your ROS node)

The files in ”fhtw_maze_explorer” are described in the following paragraph:

1. *Readme.md*: Contains a brief overview of the implemented functions in the fhtw_explorer base class. A more human read-able version of this is attached to this document appendix A. To view in depth documentation you can look at the documentation inside the source-code or use the doc folder containing the documentation in html-style. To view it just open the ”annotated.html” file in doc/html with a browser (Only work required if you want the documentation)

2. *CMakeLists.txt*: This file contains all necessary information regarding the compiler and linker[8]. ROS uses this file to create executables and libraries.(No extra work required)

3. *package.xml*: This file defines properties about the package such as the package name, version numbers, authors, maintainers, and dependencies on other catkin packages. (No extra work required)

4. *src/* : This folder contains the ready to use source codes
   - *main.cpp* : Defines the ROS node and initializes the explorer class. Here you can also set the Logging-Level (Info, Debug, Warn,...). Here you need to include your derived class. See ”*inlcude/fhtw_explorer/example.hpp*”
   - *fhtw_explorer.cpp* : The declaration of the fhtw_explorer base class. (No extra work required, but you can adapt it if you want.)

5. *inlcude/fhtw_explorer/* :
   - *fhtw_explorer.hpp* : The corresponding headerfiles for the fhtw_explorer base class. (No extra work required, but you can adapt it if you want.)
   - *example.hpp* : Is an example file that shows how to override the base class functions. Here you will have to override the base class sample and compare_nodes function to implement your explorer algorithm.

6. *launch/* :
   - *maze_explorer.launch* : This is the launchfile for the explorer package. It spawns the GAZEBO world and the turtlebot, starts the tfs for the turtlebot, the SLAM node (gmapping) as well as the move_base stack. Further it starts rviz for visualization and finally it starts the explorer node.
     If you set the `debugging` parameter to true, your browser will open and debugging session of gdbgui[9] will launch.

7. *maps/* : Contains the GAZEBO world.

[8]http://wiki.ros.org/catkin/CMakeLists.txt
[9]For a basic tutorial see: https://www.gdbgui.com/gettingstarted/

8. *config/* : Contains the configuration files for the explorer, which are loaded via the launch-file to the ROS parameter server, and the configuration for the rviz visualization.
9. *cfg/* : Defines the parameter that can be dynamically changed via rqt_reconfiugre.
10. *scripts/* : Contains all scripts that handle the maze generation as well as the bringing up the simulation environment.

1.4 Implemented Software Overview

The following section provides a brief overview of how the ROS components in this project work together using the following figure 4.

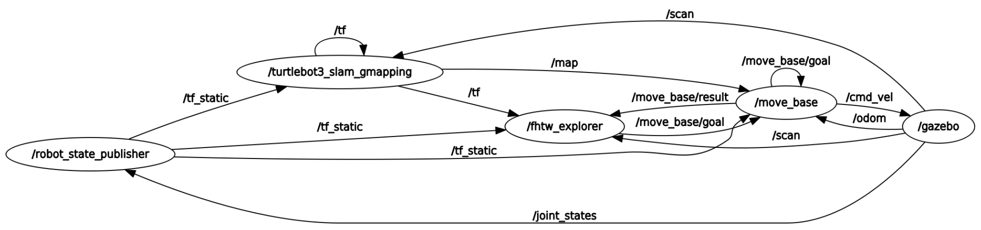


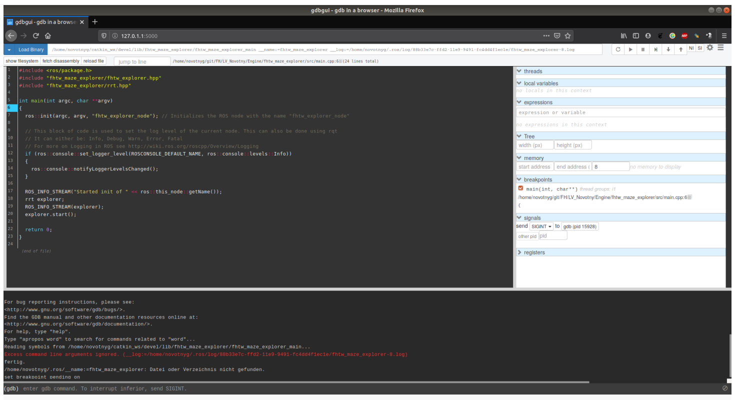
Figure 4: Rosgraph

- `robot_state_publisher`: This node loads a URDF Turtlebot model, where all tfs of the turtlebot are described. It therefore provides static tf broadcasters for the Turtlebot's coordinate transformations.
- `turtlebot3_slam_gmapping`: Is the SLAM (gmapping) node used during this project. It subscribes to the LIDAR messages (/scan) and publishes the current pose of the robot to the fhtw_explorer node.
- `fhtw_explorer`: This is the exploration node. It publishes a goal via a MoveBase ActionServer (/move_base/goal) and subscribes to the MoveBase Feedback (move_base/result).
- `move_base`: Handles the path planning and goal execution. Publishes velocity commands (/cmd_vel) to gazebo to control the robot and subscribes to the odometry (/odom) published by the simulated robot.

1.5 Debugging with gdbgui

gdbgui is a browser-based frontend to gdb, the gnu debugger. You can add breakpoints, view stack traces, list variable values of current stack and more in C, C++, Go, and Rust!

Since it is a web-based frontend you have to utilize a browser of your choice and simply paste *<docker container ip>:5000*[10] in the url bar of your browser. An example is listed in the following figure.



```
#include <ros/package.h>
#include "fhtw_maze_explorer/fhtw_explorer.hpp"
#include "fhtw_maze_explorer/rrt.hpp"

int main(int argc, char **argv)
{
  ros::init(argc, argv, "fhtw_explorer_node"); // Initializes the ROS node with the name "fhtw_explorer_node"

  // This block of code is used to set the log level of the current node. This can also be done using rqt
  // It can either be: Info, Debug, Warn, Error, Fatal
  // For more on Logging in ROS see http://wiki.ros.org/roscpp/Overview/Logging
  if (ros::console::set_logger_level(ROSCONSOLE_DEFAULT_NAME, ros::console::levels::Info))
  {
    ros::console::notifyLoggerLevelsChanged();
  }

  ROS_INFO_STREAM("Started init of " << ros::this_node::getName());
  rrt explorer;
  ROS_INFO_STREAM(explorer);
  explorer.start();

  return 0;
}
```

```
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from /home/novotnyg/catkin_ws/devel/lib/fhtw_maze_explorer/fhtw_maze_explorer_main...
Excess command line arguments ignored. (__log:=/home/novotnyg/.ros/log/88b33e7c-ffd2-11e9-9491-fc4dd4f1ec1e/fhtw_maze_explorer-8.log)
fertig.
/home/novotnyg/.ros/__name:=fhtw_maze_explorer: Datei oder Verzeichnis nicht gefunden.
set breakpoint pending on
(gdb) enter gdb command. To interrupt inferior, send SIGINT.
```

gdbgui - gdb in a browser - Mozilla Firefox
127.0.1.1:5000
Load Binary
/home/novotnyg/catkin_ws/devel/lib/fhtw_maze_explorer/fhtw_maze_explorer_main __name:=fhtw_maze_explorer __log:=/home/novotnyg/.ros/log/88b33e7c-ffd2-11e9-9491-fc4dd4f1ec1e/fhtw_maze_explorer-8.log
show filesystem
fetch disassembly
reload file
jump to line
/home/novotnyg/git/FH/LV_Novotny/Engine/fhtw_maze_explorer/src/main.cpp:6 (24 lines total)
threads
local variables
expressions
Tree
memory
breakpoints
main(int, char**)
/home/novotnyg/git/FH/LV_Novotny/Engine/fhtw_maze_explorer/src/main.cpp:6
signals
send SIGINT to gdb (pid 15928)
other pid
registers

Figure 5: gdbgui frontend with url set to: ”*127.0.0.1:5000*”

To find out more about debugging with gdbgui have a look at the following links:

- https://www.gdbgui.com/
- http://www.yolinux.com/TUTORIALS/GDB-Commands.html

2 Algorithms

The following chapter provides you with pseudo-code for each of the initially mentioned exploration algorithms as well as additional food for thought, but keep in mind that these algorithms are only guidelines. If you want to implement a different algorithm or adapt an algorithm feel free to do so.
Which ever algorithm you chose, keep in mind that the LIDAR simulated on the Turtlebot only has a range of 3.5m.

2.1 Rapidly Random Search Tree Exploration [easy]

Rapidly-exploring random tree (rrt) is a search algorithm that randomly searches high-dimensional search spaces for possible paths. In robotics, the algorithm and its variations are often used for motion planning or exploration tasks.
The pseudo-code 1 gives an overview of the steps needed to implement an rrt for exploration.

[10]You can get the ip of your running container by running ”*hostname -I*” inside the container

Algorithm 1 RRT Exploration

```
1: procedure RRT
2:     for $< itr \leq MAXitr >$ do
3:         $\vec{n} = \langle \vec{n}, get\_position() \rangle$                ▷ $\vec{n} \ldots visited\ nodes$
4:         $i \sim \mathcal{U}(0, n)$                ▷ Take random sample from LIDAR msg $\mathcal{M}$
5:         $p' = gen\_pose(\mathcal{M}[i])$                ▷ Generate a pose using the sample and $\mathcal{M}$
6:         $p = transform\_pose(p')$                ▷ Transform p' to map coordinate frame
7:         if $< compare\_nodes(p, \vec{n}) >$ then
8:             $sendGoal(p)$
9:             while $< curr\_time - start\_time <= timeout >$ do
10:                $wait\_for\_feedback()$
11:            end while
12:        end if
13:    end for
14: end procedure
```

2.2 Wall Follow Exploration [normal]

Exploration strategies differ in the extent to which the Development map is used to control the explorative movements. Wall trackers solely rely only on the measurements of distance sensors, which measure the distances to the walls to explore an environment.
The following pseudo-code 2 gives an overview of the steps needed to impement an wall-follower for exploration.

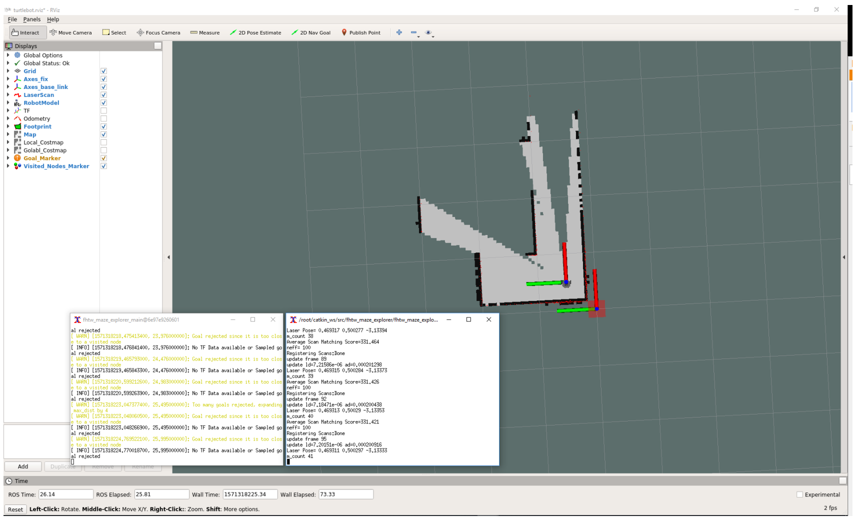
Algorithm 2 Wall Follow Exploration

```
1: procedure FOLLOW_WALL
2:     $find\_nearest\_wall(\mathcal{M})$                ▷ Find the nearest wall using the LIDAR msg $\mathcal{M}$
3:     $drive\_to\_nearest\_wall(\mathcal{M})$    ▷ Geometrically calculate the desired distance to the wall
   and publish it.
4:     while $< itr \leq MAXitr >$ do
5:         $calc\_next\_pose()$                ▷ Here is the actual wall following executed
6:         $itr + +$
7:     end while
8: end procedure
```

3 Exercise Walk-through

The following chapter walks you through the mandatory steps of creating and testing your implemented algorithms.

1. After you have started the container (see 1.2.3) you can validate if everything is setup correctly by executing the command ”`start-maze.sh`”. After you pressed "enter" on the "Finished maze generation, Press enter to start gazebo" message, a number of windows should appear which are depicted in figure 6.



```
fhtw_maze_explorer_main@6e97e9260601
al rejected
[ WARN] [1571318218.475413400, 23.976000000]: Goal rejected since it is too clos
e to a visited node
[ INFO] [1571318218.476841400, 23.976000000]: No TF Data available or Sampled go
al rejected
[ WARN] [1571318219.465793000, 24.476000000]: Goal rejected since it is too clos
e to a visited node
[ INFO] [1571318219.465843300, 24.476000000]: No TF Data available or Sampled go
al rejected
[ WARN] [1571318220.599212600, 24.983000000]: Goal rejected since it is too clos
e to a visited node
[ INFO] [1571318220.599263900, 24.983000000]: No TF Data available or Sampled go
al rejected
[ WARN] [1571318223.047377400, 25.495000000]: Too many goals rejected, expanding
 max_dist by 4
[ WARN] [1571318223.048060500, 25.495000000]: Goal rejected since it is too clos
e to a visited node
[ INFO] [1571318223.048266900, 25.495000000]: No TF Data available or Sampled go
al rejected
[ WARN] [1571318224.769522100, 25.995000000]: Goal rejected since it is too clos
e to a visited node
[ INFO] [1571318224.770018700, 25.995000000]: No TF Data available or Sampled go
al rejected
```

```
/root/catkin_ws/src/fhtw_maze_explorer/fhtw_maze_explo...
Laser Pose= 0.469317 0.500277 -3.13394
m_count 38
Average Scan Matching Score=331.464
neff= 100
Registering Scans:Done
update frame 89
update ld=7.21586e-06 ad=0.000201298
Laser Pose= 0.469315 0.500284 -3.13373
m_count 39
Average Scan Matching Score=331.426
neff= 100
Registering Scans:Done
update frame 92
update ld=7.18471e-06 ad=0.000200438
Laser Pose= 0.469313 0.50029 -3.13353
m_count 40
Average Scan Matching Score=331.421
neff= 100
Registering Scans:Done
update frame 95
update ld=7.20151e-06 ad=0.000200916
Laser Pose= 0.469311 0.500297 -3.13333
m_count 41
```

ROS Time: 26.14 ROS Elapsed: 25.81 Wall Time: 1571318225.34 Wall Elapsed: 73.33

Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click:: Zoom. Shift: More options.

Figure 6: On an successful build the following windows are going to start

   As time goes on the mobile robot explores the map using an rrt algorithm.
   The sample code only works until the workspace is built for the first time, because then the binary files are overwritten.

2. Implement your explorer algorithm! For this use the `example.hpp` (see 5)

3. To test your explorer code execute the following steps:

   a) cd /root/catkin_ws #change directory to your catkin workspace

   b) catkin_make #build the workspace including the fhtw_maze_explorer package

   c) start-maze.sh [-Options] #run the maze-generator and launch GAZEBO, RVIZ and your ROS node

      - Options:

        [-s generate a maze of size 10 x 10]

        [-m generate a maze of size 25 x 25]

        [-l generate a maze of size 50 x 50]

        [-d start debugging for maze_explorer]

4 Lessons Learned

After you have completed the project you should be capable of answering the following questions:

- What is the difference between actions and services in ROS?
- How are LaserScan messages structured?
- How can you debug a ROS node?
  - How can `rosnode info` help with that?

A Implemented Functions- Readme.md

A.1 Bringup Simulation

To bringup the simulation with different parameters see: 3

A.2 FHTW Explorer Node

This readme file gives a short overview about the functionality of the fhtw_explorer ROS package. For an in depth documentation see annotated.html (open this with a browser) or see the documentation directly in the code (Not recommended)

A.2.1 1. fhtw_explorer Class

Defined in fhtw_explorer.hpp and implemented in fhtw_explorer.cpp.
This is the parent class from which students should derive their children's classes. It is not recommended to edit the class members, but feel free to do.
The following functions are provided by this class:

- fhtw_explorer: Constructor of Class
- callback_reconfigure: Dynamically updating the configuration variables defined in config.yaml respectively explorer.cfg
- callback_goal_done: Recieve the state of a finnished goal of the move_base Action Server
- callback_goal_feedback: Recieve feedback of the current goal
- callback_goal_active: Callback once on sending goal
- pub_marker_goal: Publish a visualisation marker in rviz of the current goal
- pub_marker_nodes: Publish a visualisation marker of the already visited nodes
- callback_lidar: Store the current LIDAR message
- plan: Executes all planning functions

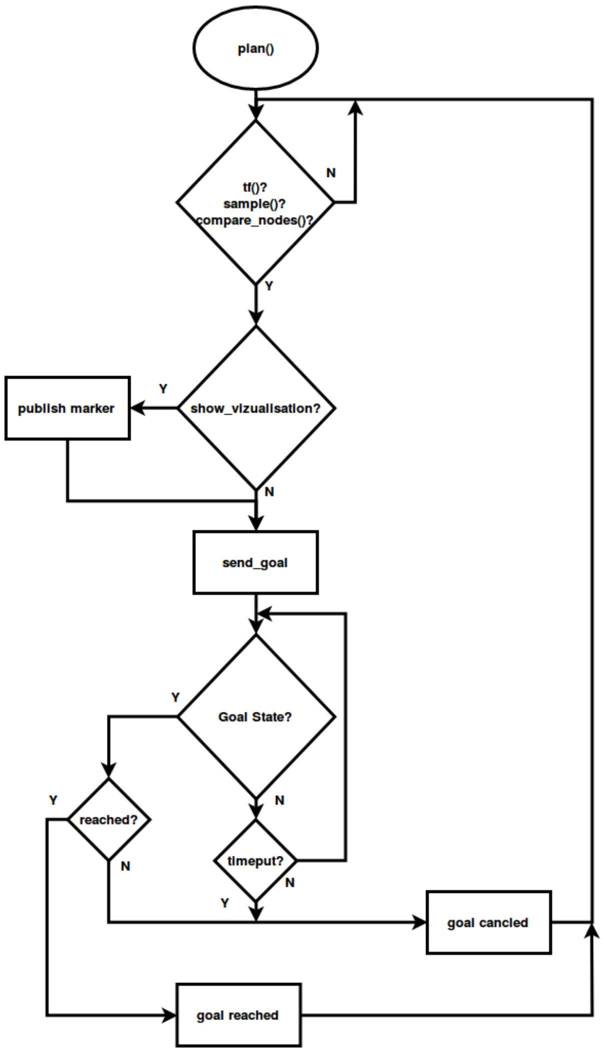


Figure 7: Overview of planning functions

- get_tf: Get the transform between the map_frame and robot_frame
- append_nodes: Appends the current robot pose (in relation to map_frame) to visited_nodes

The following functions **must** be overwritten by the students child class as they are (pure) virtual:

- ~fhtw_explorer: The Destructor of the class. If you allocate (new) any memory you must free (del) it here
- compare_nodes: This is a function that compares the current sampled goal `temp_goal` with the already visited nodes `visited_nodes`. Returns `true` if the current goal is accepted, else `false`
- sample: This function is used to generate the next goal (`temp_goal`) for the move_base Action Server. Return `true` if you want to continue else return `false`