Titile: Algorithms for local planning of robot motion and obstacles detection in the ROS environment

Abstract in English language

The main goal of this work is to compare several local planning algorithms (planners).

The assumption is to compare, two algorithms which are already implemented in ROS environment and two selected motion planning algorithms.

Based on the performed research of the available motion planning approaches, two algorithms have been selected, Potential field based algorithm and BUG0 algorithm (Chapters 2-3).

In order to achieve the main goal of this master thesis, the whole test environment based on ROS has been created. The Gazebo2 simulator and the Pioneer 3-DX robot model have been used in that order. The Gazebo2 simulator and the robot model have been configured with the ROS environment compatibility (Chapter 4).

Selected algorithms have been implemented in Python 2.7 programming language. Implemented algorithms and ROS algorithms have been configured with previously created test environment (Chapters 5-6).

The robot working area became the rectangular building wit dimensions, 100x30[m]. About 40 obstacles, with different size, have been created in the building (Chapter 7.1).

Next, the tests have been performed, in the prepared working area, in order to obtain the optimal parameters sets for each algorithm.

The algorithms, with their optimal parameters configuration, have been compared in the speed test. The fastest algorithm was the Potential Field based algorithm (Chapter 7.3).

Abstract in Polish language

Celem niniejszej pracy magisterskiej jest porównanie algorytmów lokalnego planowania ruchu robotów mobilnych.

Założeniem jest porówananie dwóch już zaimplementowanych algorytmów w środowisku ROS (Ros Base local planner i TEB local planner) oraz dwóch wybranych podejść lokalnego planowania ruchu.

Na podstawie przeprowadzonego przeglądu dostępnych podejść lokalnego plannowania ruchu, wybrano dwa algorytmy: algorytm wykorzystujący metodę pól potencjałów oraz algorytm BUG0 (Rozdziały 2-3).

Aby zrealizować cel niniejszej pracy magisterskiej stworzone musiało zostać graficzne środowisko testowe zgodne ze środowiskiem ROS. W tym celu użyto symulatora Gazebo2 oraz modelu robota mobilnego Pioneer 3-DX. Symulator Gazebo oraz model robota zostały skonfigurowane ze środowiskiem testowym (Rozdział 4).

Wybrane uprzednio algorytmy zostały zaimplementowane w języku Python 2.7. Zaimplementowane algorytmy oraz algorytmy ROS zostały skonfigurowane ze środowiskiem testowym (Rozdział 5-6).

Obszarem roboczym robota został prostokątny budynek o wymiarach, 100mx30m. W budynku utworzono około 40 przeszkód różnej wielkości (Rozdział 7.1).

Następnie przeprowadzono testy mające na celu dobranie optymalnych parametrów dla każdego z algorytmów w wybranym obszarze roboczym (Rozdział 7.2).

Algorytmy z optymalnymi ustawieniami zostały porównane w teście szybkościowym. Najszybszym algorytmem okazał się algorytm bazujący na metodzie potencjałów (Rozdział 7.3).

Contents

Chapter 1

Introduction

Path planning algorithms can be applied in many fields of the industry, automotive-autonomous cars, military-autonomous drones or in agriculture-autonomous harvesters. At this moment the advanced path planning systems are becoming more and more popular.

Driver assistance systems have become a field of research for the biggest companies in the world such as Tesla, Uber, Toyota, Volvo or Google. There are some rumours that Apple is also working on its own autonomous car project. Tesla company is after the testing stage and car models such as 'S' and 'X' have implemented fully autonomous autopilot and are already in production. Google engineers are testing their own fully autonomous car system, Google autonomous cars have driven more than 1 mln kilometres so far, and there is no alarming information about any issues. Google, Uber, Tesla and other companies are working on fully autonomous car projects, but there are a lot of companies that implement not so advanced autonomous systems, such as collision avoidance modules, parking assistance, autonomous braking system etc. One of such companies is Volvo. We can conclude that the direction of increasing the autonomy of vehicles and reducing human intervention is the direction in which the car market follows. According to the forecast prepared by Fortune Magazine [1], the autonomous car market will hit 21 Million cars by 2035.

Another important area of planning algorithms are military applications. Computer computation power constantly increases so as possibilities to store more data. The US army has created drones which are fully automated. Based on the information from GPS, gyroscopes and sonars such drones are able to move from the start point to the predefined goal point without collision with static (trees, walls) or dynamic obstacles (birds, other drones etc.). There are also concepts to create drones, which will be able to define their position based on the stored maps of local environment and data from sonars about its nearest area.

Planning algorithms are also a field of research for agriculture. Several scientific works and articles have been presented in that field of research, such as [2], in which main problem is to ensure coverage for big agriculture areas, or [3], where path planning is presented as a solution for Japanese problem in the agriculture industry. Based on the article prepared by Avantech Magazine [4], there are already autonomous harvesting systems available on the market. According to that information and information from Popular Science Magazine [5] in the nearest future the fully autonomous farms are going to become a reality.

The number of path planning applications increases every year. Planning can be performed offline (when we have full knowledge about the global environment) or online

(when only information about the local environment obtained from sonar or laser scanner is available). The area of interest of this master thesis are local planning algorithms.

The main goal of this work is to compare several local planning algorithms (planners): two algorithms which are already implemented in ROS as ROS packages (*Ros Base local planner* and *TEB local planner*) and two externally implemented approaches/algorithms. It is a requirement that algorithms should be implemented in compliance with ROS environment. It is also demanded that a robot implementing such an algorithm must be able to define and execute the non-collision path between the start point and the goal point in defined working area (rectangular building with dimension 100x30 [m] with medium obstacle density).

In order to achieve these objectives, the whole test environment based on ROS should be created and configured. The test environment should contain the graphical simulator, which should perform visualisation of robot model and robot working area. The robot model should satisfy the non-holonomic mobile robot assumptions. External algorithms should be chosen based on the research of known motion planning algorithms. ROS local planners (*Ros Base local planner* and *TEB local planner*) and two selected planning algorithms should be implemented and configured with created test environment. Afterwards, the comparison tests should be performed in order to determine the best local planning algorithm.

Chapter 2

Motion planning

Steven M. LaValle said that the one of the most important needs in robotics is to have algorithms which convert the high-level task description from humans into low-level description of how to move. Such algorithms can be called *path planning algorithms*. LaValle introduced also the terms *motion planning* and *trajectory planning* which are often used in robotics for naming the process of non-collision movement from the predefined start position to the desired goal position with physical robot constraints satisfaction [6].

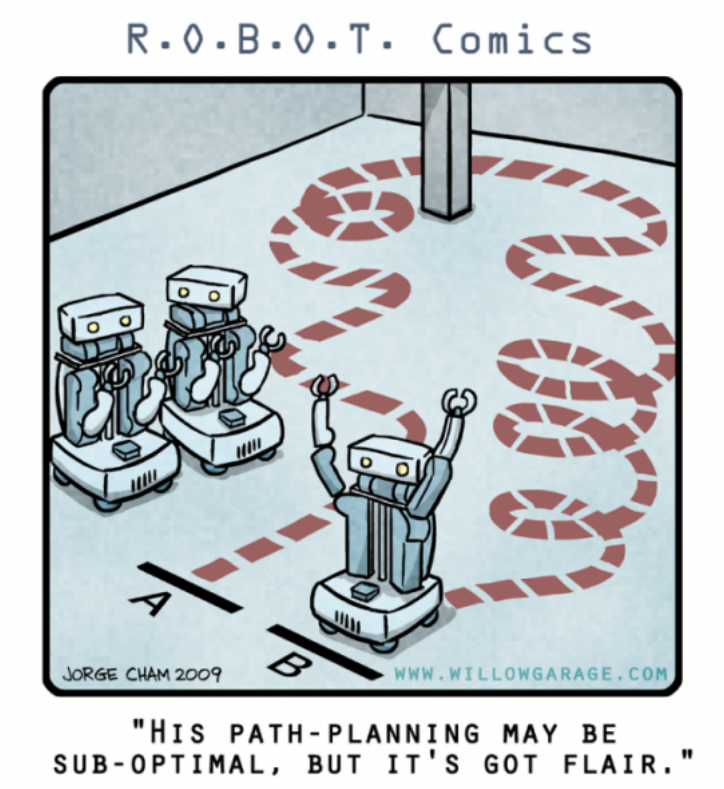

Figure 2.1: Path planning [10].

A classical version of the motion planning problem is usually presented as *Piano Mover's Problem*. In [8] J.T. Schwartz and M. Sharirr introduced the Piano's Mover's Problem as follows: "given a body(piano) and a region bounded by a collection of walls (room), either find a continuous motion connecting two given positions and orientations of robot during which robot avoids collisions with the walls, or else establish that no such

motion exists."

In [6] the input for the algorithm was a precisely designed model of the house and piano. The algorithm was responsible for determining how to move the piano between rooms without any collision. Robot motion planning usually focuses only on the translations and rotations required to move the piano. Trajectory planning term refers to the problem of taking the solution from the robot motion planning algorithm as input and determining how to move along the solution in a way that respects the robot mechanical limitations.

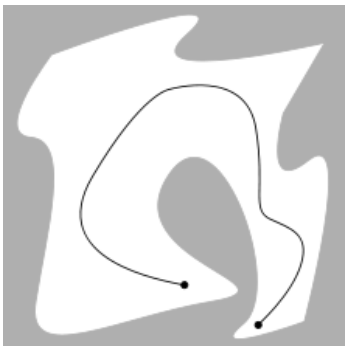

Figure 2.2: Example of a valid path.

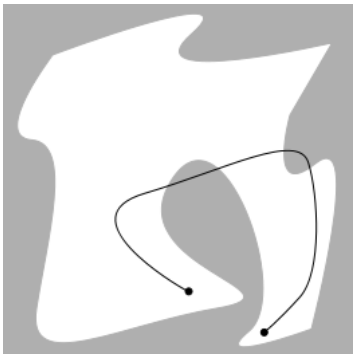

Figure 2.3: Example of an invalid path.

2.1 Robot workspace representation

In the previous chapter the basic motion planning problem was introduced. The robot works in a precisely designed working space (see Figure 2.4). The solution of the motion planning problem (the path) is presented in the robot *configuration space* (see Figure 2.5).

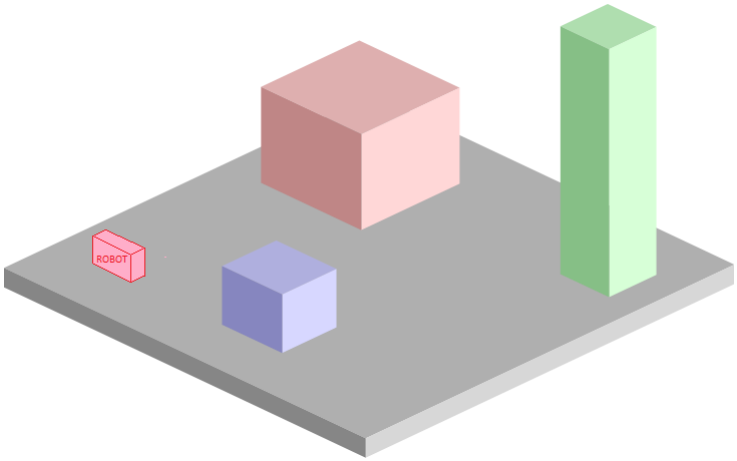

Figure 2.4: Example of robot workspace.

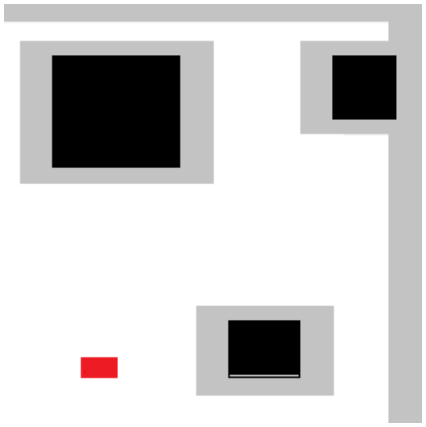

Figure 2.5: Example of a configuration space. The configuration space for a rectangular translating robot (pictured red). White area is a collision free area (*Cfree*), gray area is the area where exists collision possibility (*Cobs*), dark-gray area is the obstacle area (*Cobs*).

It is important to define the robot state space. The robot configuration space for motion planning is a set of available transformations (positions, orientations) that may be applied to the robot. The dimension of the conguration space is defined by the number of robot degrees of freedom . The representation of motion planning configuration space is very important [7].

In the next chapter several known motion planning algorithms and configuration space representations will be introduced.

Chapter 3

Path planning algorithms

Chapter 2 has introduced the term *motion planning*. A motion planning algorithm takes the precisely designed model of workspace and returns the path. Such an algorithm can be called "planner". In the next chapters names such as "path planning algorithm, "motion planning algorithm" and "planner" will be used interchangeably.

Steven M. LaValle has proposed a planner general definition: "A planner simply constructs a plan and may be a machine or a human. If the planner is a machine, it will generally be considered as a planning algorithm. In many circumstances it is an algorithm in the strict Turing sense, however, this is not necessary. In some cases, humans become planners by developing a plan that works in all situations" [23].

As it was mentioned previously, path planning algorithms can be divided into two groups:

- *Local(Online) path planning algorithms*: Robot configuration space is defined based on the data from sonars or laser scanners.
- *Global(Offline) path planning algorithms*: Robot configuration space is defined based on the global map of the workspace.

The subject of this work are local motion planning algorithms. In the next sections, several known algorithms and motion planning approaches will be described. Algorithms/approaches will be compared based on two kinds of criteria [11]:

- *Feasibility*: Given plan is not necessarily efficient. However, it gives assurance for arrival to the goal state.
- *Optimality*: Given path is optimal. The optimality criteria is defined by user.

3.1 BUG Algorithm

The BUG algorithm assumes only local knowledge about the robot working space and the predefined global goal point. The advantage of using this algorithm as the planner is that it can be modified, and already exist a lot "versions" of this algorithm like BUG0, BUG1, BUG2, I-BUG etc. The configuration space for BUG algorithm is simple, a robot can take any non-collision path in the state space [12].

In the next subsection several BUG algorithms will be introduced. The behaviour of every BUG algorithm version will be presented on the the same configuration space (see Figure 3.1). Presented configuration space has been designed only for the BUG0 algorithm description purposes.

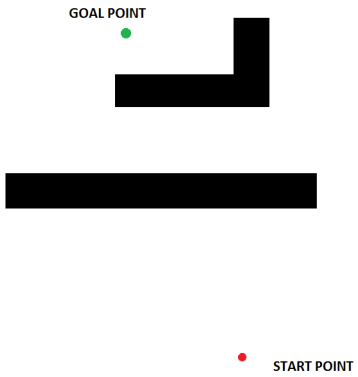

Figure 3.1: Robot configuration space.

3.1.1 BUG0 Algorithm

BUG0 algorithm is the simplest version of the BUG algorithm. At the beginning a robot moves toward the global goal point. When an obstacle is detected, the robot starts to follow the obstacle from the predefined side (assume it will be right side). The robot follows the obstacle until it can head toward the goal point again (see Figure 3.2, orange line is the robot path) [13].

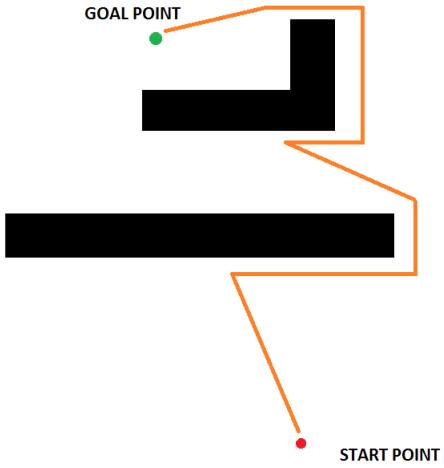

Figure 3.2: Robot configuration space with defined the path resulting from BUG0 algorithm.

BUG0 algorithm fulfils the feasibility criteria. However, it's not optimal. It will always follow the obstacle from the predefined side which is the biggest disadvantage of this algorithm.

3.1.2 BUG1 Algorithm

BUG1 Algorithm is similar to BUG0 but it has some modifications. The robot moves straight to the goal until it detects an obstacle. Robot saves its the position. The robot starts to follow the obstacle from the predefined side (assume right side). The robot

follows the obstacle till it gets to the saved point. Then the robot turns back and starts to follow the previously generated "avoidance" path, until robot can head toward the goal point again (see Figure 3.3) [14].

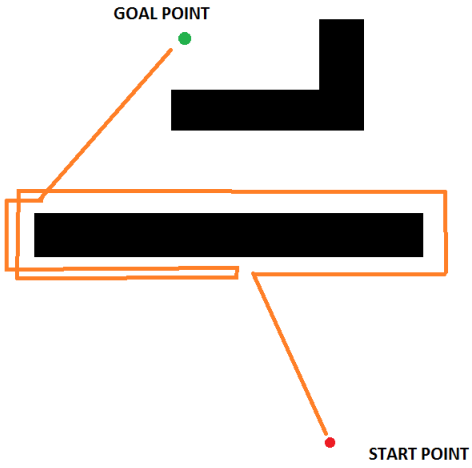

Figure 3.3: Robot configuration space with defined path using BUG1 algorithm.

Feasibility criteria has been fulfiled. The path defined by a BUG1 algorithm is much longer than the path obtained by BUG0 algorithm. This version of the algorithm is less efficient than version "0".

3.1.3 BUG2 Algorithm

In the first step, BUG2 algorithm defines the straight path between the start and goal point (see the grey line in Figure 3.4). Then robot follows the obstacle from the predefined side until it gets back to the global path which has been defined in the first step [16].

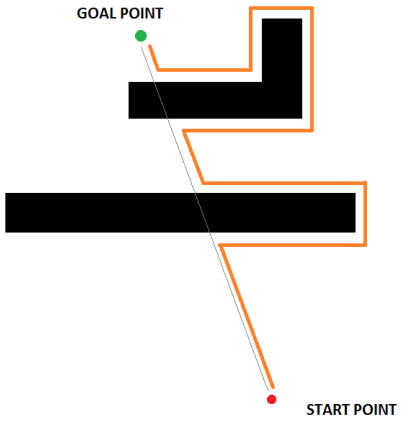

Figure 3.4: Robot configuration space with the path resulting from BUG2 algorithm.

This version algorithm also fulfils the feasibility criteria. Based to optimality criteria BUG2 algorithm is less optimal than BUG0 algorithm. It generates much longer path.

3.2 Potential Field Method

The main idea of a the potential field method is taken from nature, for example, a charged particle navigating a magnetic field or ball rolling on a hill. The main idea is based on the strength of the magnetic field, where the charged particle can reach the source of the magnet field. The same behaviour has its application in the motion planning. We can simulate the same effect by creating the configuration space as an artificial potential field that will attract the robot to the goal [15].

Jean-Claud Latombe has described Potential Field approach: "It treats the robot represented as a point in configuration space as a particle under the influence of an artificial potential field U, whose local variations are expected to reflect the "structure" of the free space" [19].

The potential configuration space is represented as a set of attractive potential points connected with the free space, which pulls the robot toward the goal configuration, and qcset of the repulsive potential points connected with the non-free space (obstacles), which push the robot away from the obstacles (see Figure 3.5).

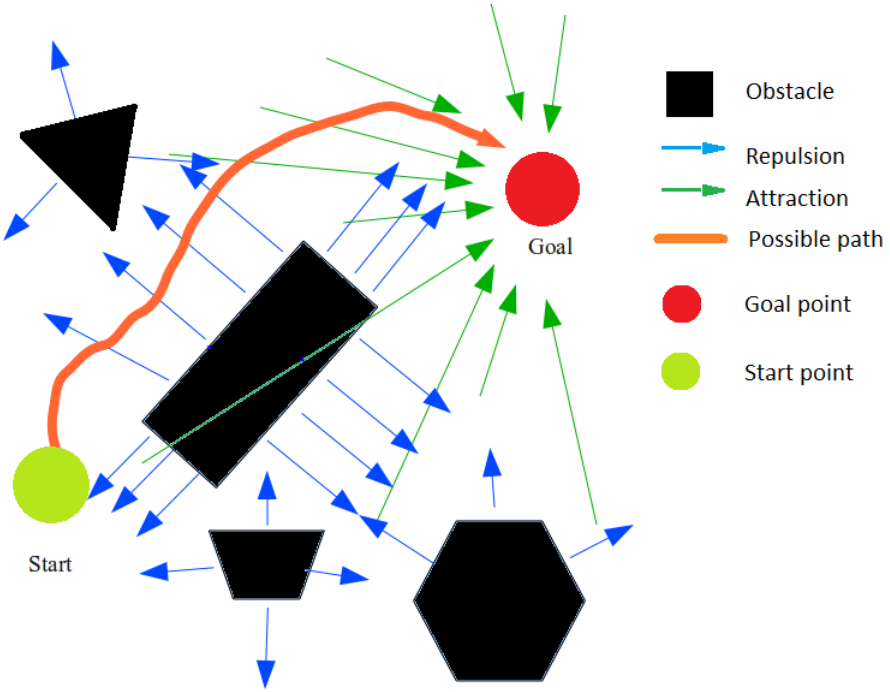

Figure 3.5: Potential field method.

The potential field based algorithm takes the configuration space represented as described above. A robot implementing such an approach is moving toward the goal through the most attractive points. The clue of the good algorithm quality is the correct representation of the configuration space. Unfortunately, such an algorithm gives no assurance that the robot always reaches the goal. The robot can reach only the local maximum and never reach the global goal. Because of that, the feasibility criteria is not fulfiled.

3.3 Best-First Algorithm

Best first algorithm is a search algorithm. Its configuration space is represented as the graph of nodes. The algorithm explores the graph expanding the most promising nodes according to the specified rule. Judea Pearl described the Best First algorithm as estimating the most promising node N by a heuristic evaluation function $f(n)$ which, in general, may depend on the description of n, the information of the goal, the information gathered by the search up to that point, and most importantly, on any extra knowledge about the problem domain [24].

The configuration space for Best First algorithm could be accomplished by using *Visibility Graph Method*. The principle of the Visibility Graph method was defined by Latombe in [22]. The path is defined as a polygonal line which connects the start configuration with the goal configuration through vertices's of non-free configuration space which are the graph nodes (see Figure 3.6). Such operation defines possible non-collision paths between the start and goal position.

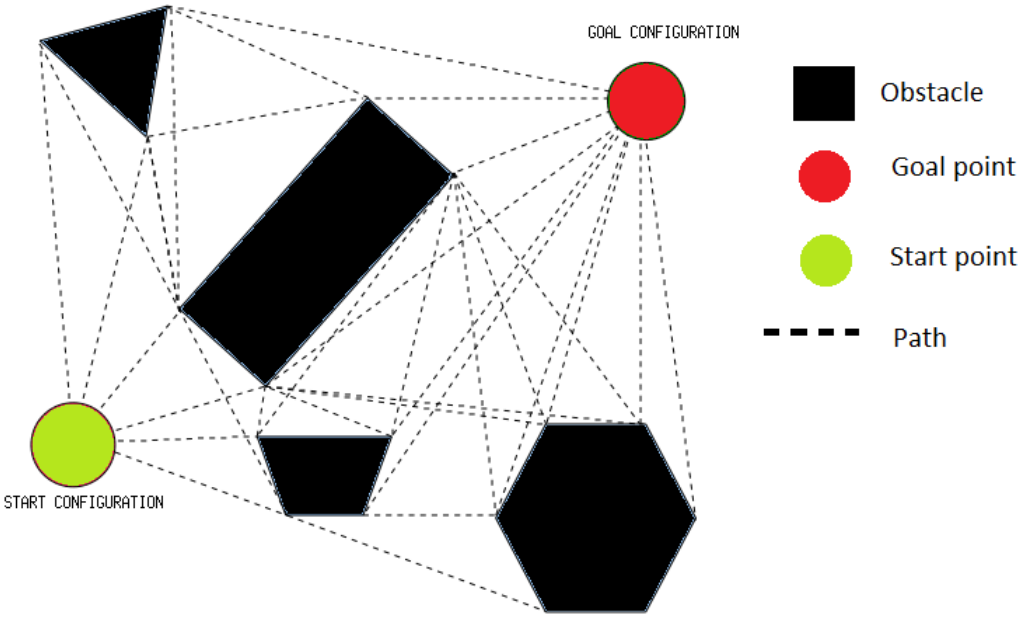

Figure 3.6: Configuration space obtained by Visibility Graph method.

The quality of such an algorithm increases proportionally according to the information about the robot workspace. In other words, the algorithm is better when the robot has global the knowledge about the environment.

3.4 Reward-Based Method/Markov Decision Process

Reward-Based Algorithm assumes that in each position of the configuration space, the robot can choose between different actions. However, the result of each action is not the same. The robot gets the reward if it reaches the goal and gets nothing (or a negative reward) if it gets into collision with an obstacle. The Markov Decision Process (MDPs) is popular mathematical framework, which is often used really in Reward-Based Algorithm [17].

A Markov Decision Process model contains:

- A set of possible world states (robot configuration space): **S**
- A set of possible actions (move toward, turn left, turn right): **A**
- A real valued reward function: **R(S,A)**
- A description of each action's effects in each state: **T**

In every state S, a robot can choose between the defined set of actions A. A robot implementing MDP will choose this action, which gives the biggest possibility of positive reward [18].

The canonical example for the MDP is a grid world. Such representation of the configuration space can be used by Grid-Based Method. The configuration space is divided into grid cells, each grid cell dimension is the same as the dimension of the robot. We can obtain the map with free grid cells, points contained in Cfree (white), and non-free cells (black), contained in Cobs set (see Figure 3.7). The robot is allowed to move between grid points as long as the line between them is completely contained in Cfree (see Figure 3.7).

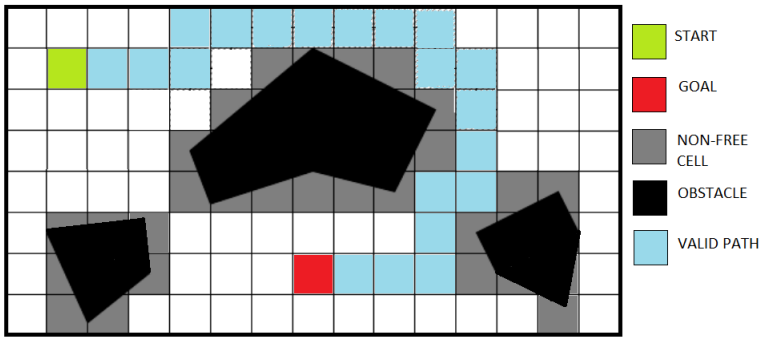

Figure 3.7: Representation of the configuration space obtained by Grid-Based method.

3.5 Summary

The presented algorithms fulfil the assumed criteria in different extent. However, only two of them have been selected for further consideration.

The *Potential field method* has been selected because of its popularity, fulfilment of the optimality criteria and a wide range of applications. *BUG0 Algorithm* has been selected because of its efficiency in the goal point achievement (feasibility criteria) and the possibility of its adjustment to a variety of environments.

Chapter 4

Test environment

A precisely designed test environment was necessary for performing the comparison tests of implemented motion planning algorithms (planners).

One of the most important tools in the roboticist's toolbox is a robot graphical simulator. A graphical simulator compatible with the ROS framework was needed for creating such an environment. *Gazebo* simulator has been chosen based on its popularity and full compatibility with ROS framework. A robot working space has been created by using gazebo simulator.

Another important part of the test environment is a robot. As assumed earlier a nonholonomic mobile robot has been considered. The mobile robot *Pioneer 3-DX* has been selected because of its popularity and compatibility in ROS. An additional advantage is that real pioneer robots have ROS framework on-board.

The robot eye on the world was a laser scanner. *Hokuyo LMS100* laser scanner has been chosen based on its compatibility with ROS framework and the simulator. Also, this type of a laser is easily configurable in the ROS environment. The laser has been mounted on the top of the robot model.

4.1 Robot Operating System (ROS)

Robot Operating System is known as a flexible framework for writing robot software. It is a collection of tools and libraries. The main goal of the ROS framework is to simplify the task of creating complex and robust robot systems across a wide variety of robotic platforms. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more. ROS is licensed under an open source, BSD license [25].

4.1.1 ROS distribution

An ROS distribution is a versioned set of ROS packages. There are several ROS distributions at the time of writing this master thesis:

- ROS Kinetic
- ROS Jade
- ROS Indigo
- ROS Hydro

- ROS Groovy

The main goal of each distribution kind is to give the developers possibility to work with a stable framework [26].

ROS Indigo distribution has been used in the test environment due to its compatibility with the Gazebo2 graphical simulator, EOL (End-Of-Life) date (May 2019) and the stability recommendations.

4.1.2 ROS packages

Software in ROS is organized in packages. A package may contain ROS nodes, an external library (ROS-independent), datasets, configuration files, a third-party piece of software, or anything else that can logically construct a useful ROS framework module. The main goal of ROS framework packages is to create whole functionality in an easy-to-consume manner so that software can be easily reused. ROS framework has implemented more than 2000 packages which can be used for variety purposes [27].

4.1.3 ROS nodes

A node is a program or a process, which performs the computation. Nodes are connected together into graphs and communicate with each other using streaming topics, RPC services, and the Parameter Server. A robot control system will usually contain many nodes. For example, the first node is responsible for control of a laser range-finder, the second node controls the robot's wheel motors, the third performs localisation, the fourth node performs path planning, the fifth provides a graphical view of the system, and so on [28].

4.2 Gazebo Simulator

A well-designed simulator gives the possibility to test planning algorithms rapidly on properly designed robots and environment. This reduces the adaptation time of the planner final version to the real robot in real workspace (robot environment).

Gazebo2 Simulator is one of the most popular simulators, which give such opportunity. It offers precisely designed robot models. Another big advantage of Gazebo simulator is that this simulator is free and has a big vibrant community [29].

Gazebo simulator provides useful features:

- *Dynamics Simulation*: Simulation is provided by using multiple high-performance physics engines including ODE, Bullet, Simbody, and DART.
- *Advanced 3D Graphics*: Gazebo provides the realistic rendering of environments including textures, lighting and shadows.
- *Sensors and Noise*: Gazebo simulator is able to generate sensor data with noise from variety equipment such laser range finders, 2D/3D cameras, Kinect-style sensors, contact sensors, force-torque, and more.
- *Plugins*: User is able to develop custom plugins for the robot, sensor, and environmental control.

- *Robot Models*: Many robot models are available in the gazebo simulator including PR2, Pioneer2 DX, iRobot Create, and TurtleBot. The user is also able to build own robot model.
- *TCP/IP Transport*: Gazebo provides the possibility for running simulation on remote servers, and interface to Gazebo through socket-based message passing using Google Protobufs.
- *Cloud Simulation*: User is able to use CloudSim tool to run Gazebo on Amazon, Softlayer, or your own OpenStack instance.
- *Command Line Tools*: Command line tools simplify simulation introspection and control.

At the time of writing this master thesis, there are several releases of the Gazebo simulator. The Gazebo2 release has been chosen due to its compatibility with ROS Indigo distribution.

4.3 Pioneer 3-DX description

The Pioneer robots are one of the most popular the world's research mobile robots. The Pioneer's robot versatility, reliability and durability have made it the reference platform for robotics research and application involving:

- mapping
- teleoperation
- localization
- monitoring
- reconnaissance
- vision
- manipulation
- autonomous navigation
- multi-robot cooperation and other behaviors
- general robotics

Pioneer 3-DX is a small, lightweight two-wheeled mobile robot (see Figure 4.1). P3-DX has two differential motors with encoders, one battery and HOKUYO laser mounted on the front-top.

Figure 4.1: Pioneer 3-DX [31].

In Figure 4.2 specific dimensions of Pioneer 3-DX mobile robot have been presented.

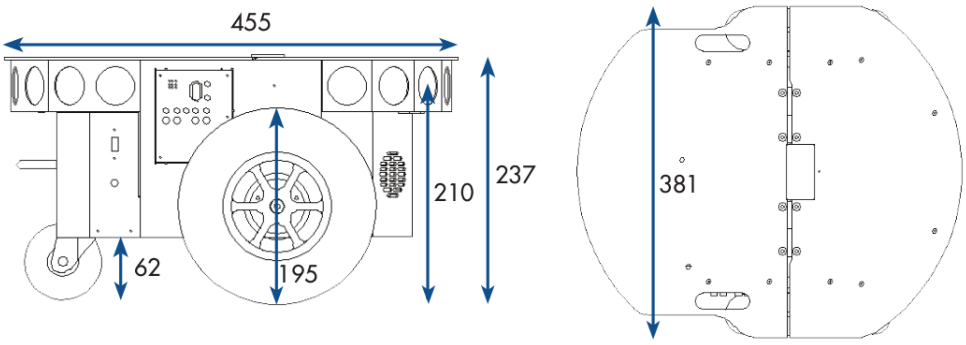

Figure 4.2: Pioneer 3-DX dimensions [30].

4.4 Hokuuyo LMS100 laser

The model of Hokuyo LMS100 laser was used for workspace monitoring and obstacle detection purposes. The laser has been configured for simulation purposes, and the configuration parameters are as follows:

- Angle range of the laser visibility :=π[rad]
- Maximal distance of the laser visibility:=5[m]
- The number of samples (N) which is taken in each measurement:=500 (the angle range is divided by the number of samples)

The laser scanner publishes data about the local environment as matrix of sample (each sample(i) get a value (data[i]) according to the distance from obstacle):

$$laser\_data(i) = data[i] \qquad i = 0...N-1 \tag{4.1}$$

N: the number of samples, *data[i]*: Sample values, every sample can take value in the range (0-5)[m]

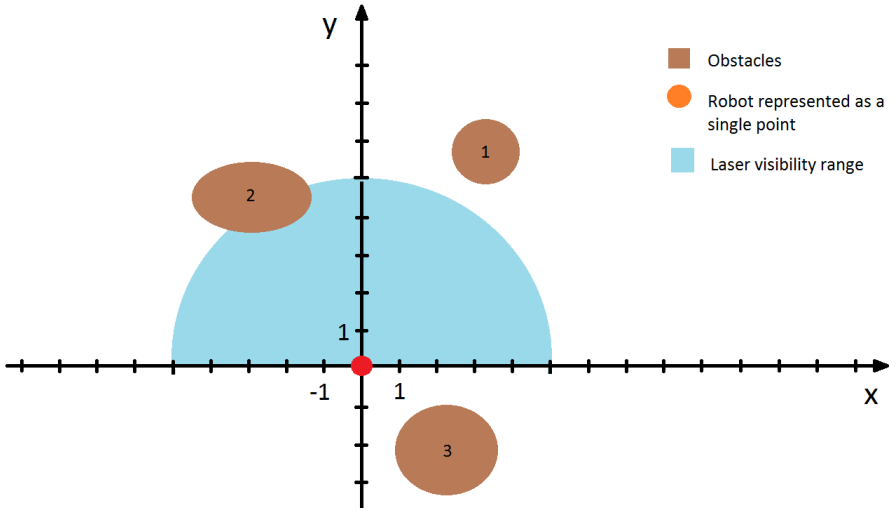

Figure 4.3: Robot representation (red point) in the configuration space with laser visibility range marked.

In Figure 4.3 the laser visibility range has been marked based on the set of configuration parameters. From the figure we can conclude that robot is able to detect obstacles win the distance up to 5m.

4.5 Pioneer 3-DX model configuration

Robot model for odometry purposes is represented as a single point (the center of the robot rigid body) in the configuration space (see Figure 4.4). The position of this point is defined with respect to the fixed center of the map (point x=0,y=0). The robot position is computed by *Odometry* ros node, this process uses data from the motion sensors (encoders) to estimate the robot position change over time. *Odometry* is a ROS mechanism (one of the already implemented ROS packages). Therefore we can assume that the robot position is always defined perfectly.

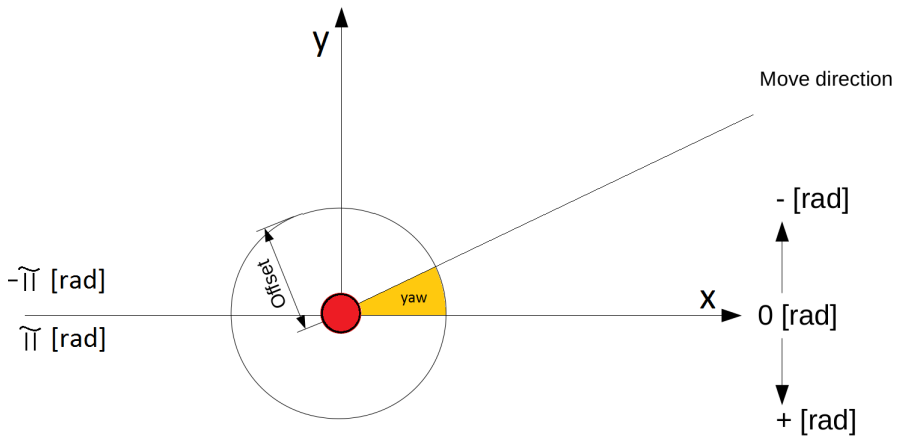

Figure 4.4: Robot model representation (red point) in the configuration space with respect to its global x-y frame.

The robot performs a uniformly accelerated curvilinear motion. The minimal and maximal values for velocities and accelerations have are as follows:

- *maximum linear velocity*: 0.5 [m/s]
- *minimum linear velocity*: 0.0 [m/s]
- *maximum rotational velocity*: 0.5 [rad/s]
- *minimum rotational velocity*: -0.5 [rad/s]
- *acceleration linear*: +/- 0.5 $[m/s^2]$
- *acceleration rotational*: +/- 0.5 $[m/s^2]$

The robot is able to move forward, turn right and turn left while moving backward is not supported. Figure 4.5 presents the robot visual kinematic chart for rotational and linear movement.

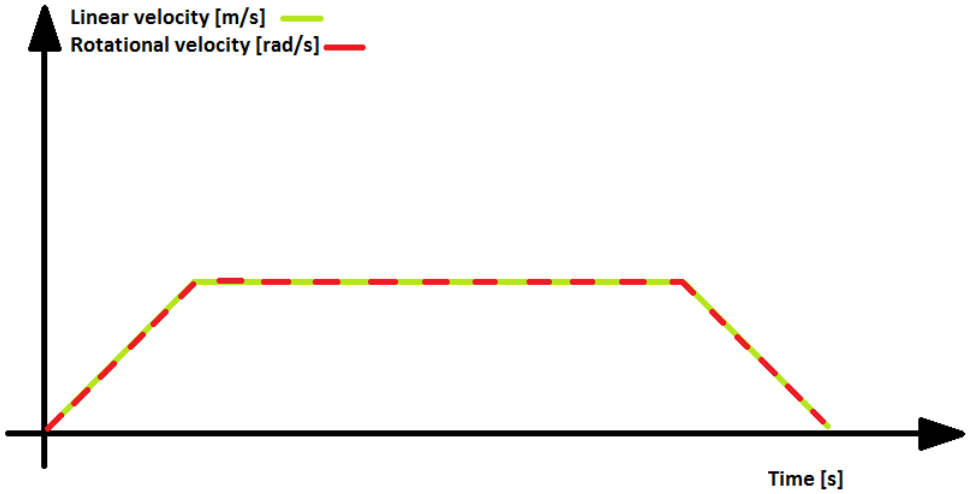

Figure 4.5: Robot model kinematic chart.

4.6 Simulation

In the figure 4.6, ROS computation graph for the whole test environment simulation has been introduced. Gazebo Simulator (*gazebo*), Pioneer 3-DX model (*pioneer1*) and the implemented planner are represented as ROS nodes. Gazebo simulator node publishes the laser scanner and odometry data messages to the listener topics from the robot model node:(*pioneer1/scan/*) and (*pioneer1/odom/*). The implemented planner, according to the information about the desired goal pose and data from the laser scanner and odometry topics computes the proper velocities and publishes it to the *pioneer1/cmd_vel* topic. Then the computed velocities are published from the robot model node to the gazebo simulation node. Next, the velocities are set in the robot graphical model in Gazebo simulator, forcing it to move so that we are able to see if the planner computes the proper velocities.

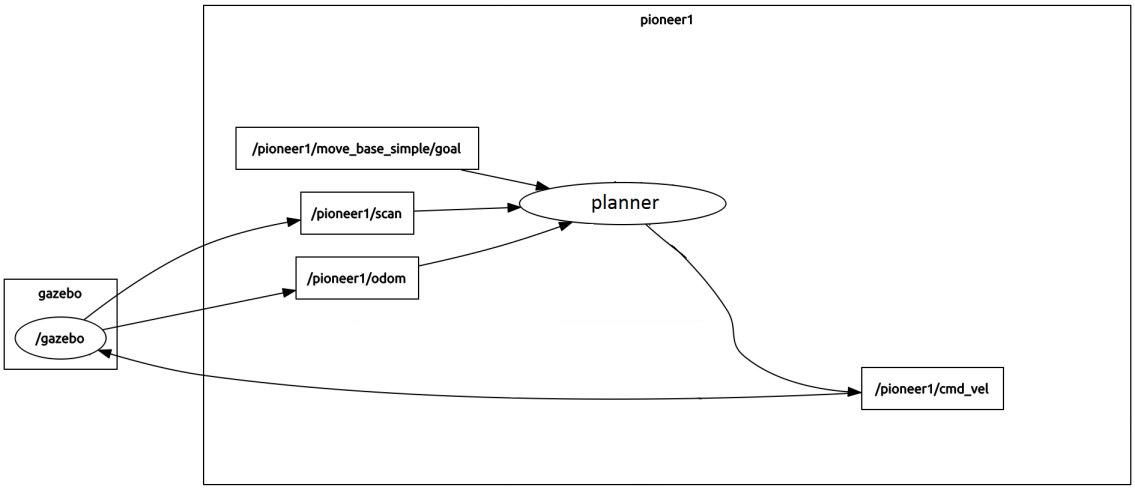

Figure 4.6: ROS computation graph for working test environment simulation.

Chapter 5

ROS local planners

There are several local planners already implemented as ROS packages, including *BASE local planner* or *TEB local planner*. In this chapter, the algorithms which are used by the mentioned planners will be described.

5.1 Base local planner

The configuration space representation for Ros Base local planner is obtained by using the Sampling-Based approach. The robot working area is prtitionedby number N of samples. Only these samples which are connected to free space are saved. These samples will be used by the planner as milestones. Next, the path can be defined as the connection of milestones (see Figure 5.1) [20].

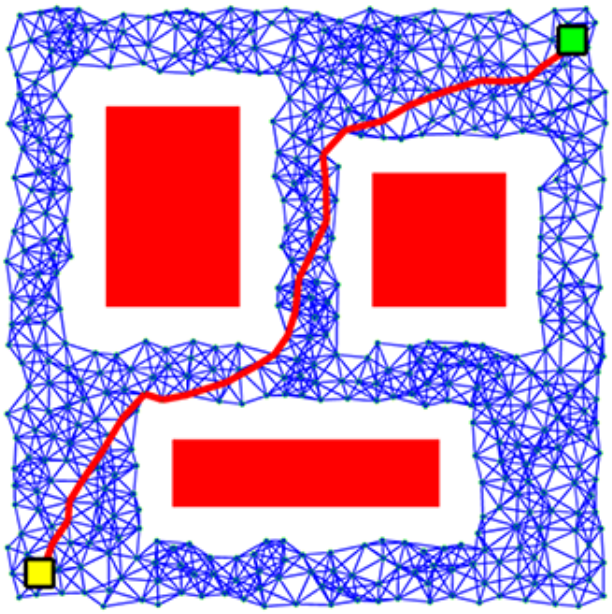

Figure 5.1: Application of Sampling-Based approach [21].

5.1.1 Algorithm overview

The base local planner implements the Trajectory Rollout algorithm. The algorithm defines the path through the milestones from start to goal [32].

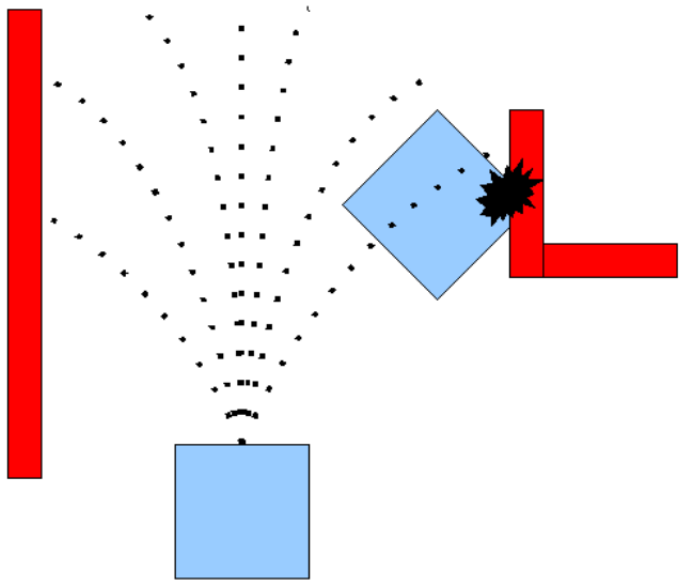

Figure 5.2: Graphical illustration of how Trajectory Rollout algorithm works. The black dot lines are the possible paths obtained from the set of velocities [32].

The basic idea of the Trajectory Rollout algorithm is as follows [32]:

1. Discretely sample the robot model configuration space (dx,dy,dtheta).
2. For every set of sampled velocities, perform a simulation. This means that for every obtained set of velocities (*dx,dy,dtheta*), the simulation from start to goal should be performed to predict what will happen if such set of velocities will work for very short period of time.
3. Evaluate each obtained path using to the distance from obstacles, distance to the goal, speed, etc criteria. Remove collision trajectories from the set.
4. Choose the best trajectory and configure the proper set velocity in the robot model.
5. Execute the selected trajectory and repeat.

5.1.2 Implementation

The implementation is practically the same for all already implemented local planners in ROS.

Figure 5.3, presents the ROS computation graph with the working base local planner. Planner(*pioneer1/movebase*) gets the information from laser(*pioneer1/scan*) and robot odometry(*pioneer1/odom*). Based on the obtained data, the planner computes the set of proper velocities(*pioneer1/cmd_vel*).

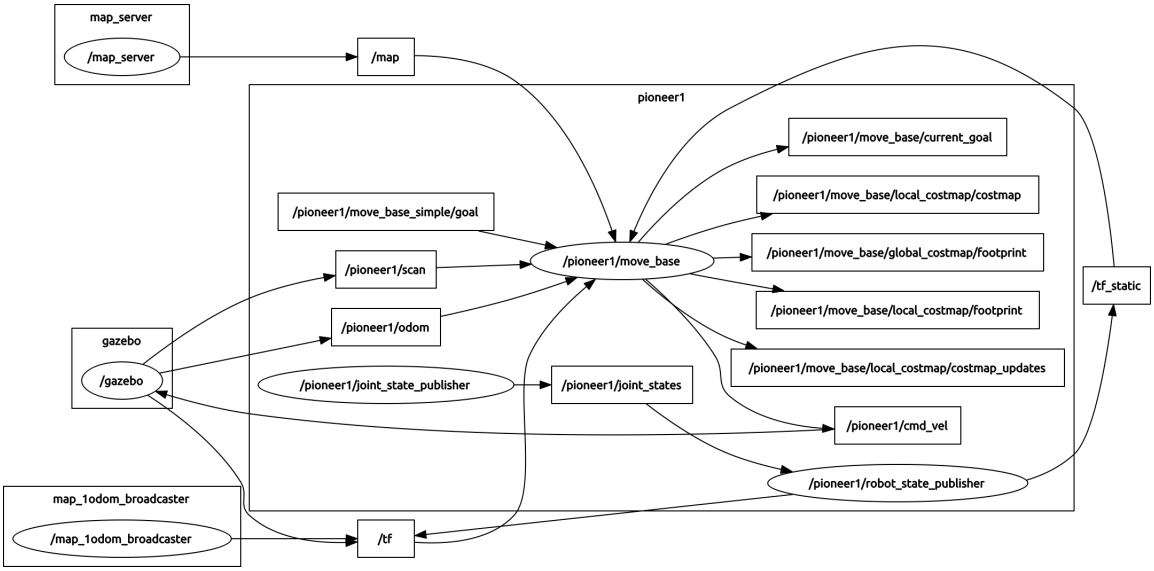

Figure 5.3: Base Local Planner ROS computation graph.

5.1.3 Parameters

There exists a big number of customization parameters for base local planner. BLP customization parameters are divided into several groups such as:

- robot configuration
- goal tolerance
- forward simulation
- trajectory scoring
- oscillation prevention
- global plan

The most important customization parameters that is, the parameters that have the biggest influence on the planner behaviour, should be chosen. These are the number of samples used for velocities computations:

- *vx_samples*: The number of samples used in x velocity space exploration
- *vtheta_samples*: The number of samples used in theta velocity space exploration

5.2 TEB local planner

5.2.1 Algorithm overview

Configuration space representation for TEB local planner is obtained by the same approach as for Base local planner (*Sampling-Based approach*). The planner uses Time-Elastic-Band algorithm for obstacle avoidance purposes.

The algorithm has been described in [33]. Consider the global and local reference frames of the mobile vehicle that are presented in Figure 5.4. The motion planning problem is to move the robot from point A(x_0, y_0) to point B(x_N, y_N) without any collision with the obstacles.

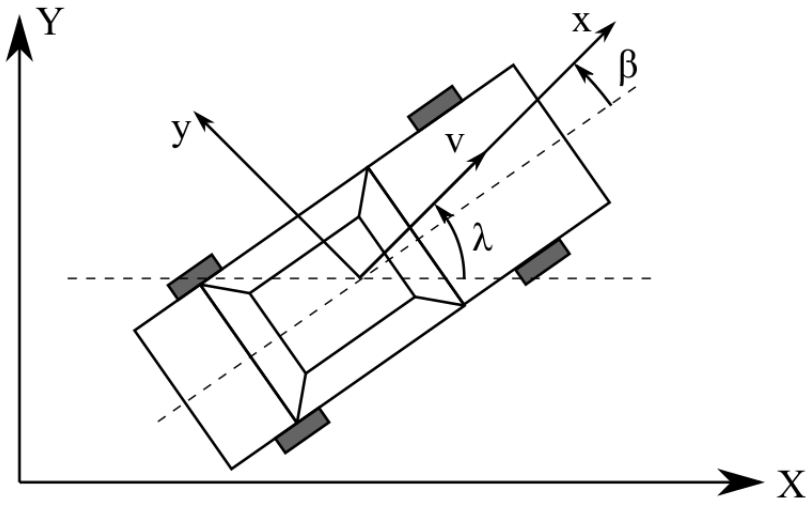

Figure 5.4: Global and local mobile vehicle frames [33].

TEB is created by fixed number N of way-points ($x_0 ... x_N$,$y_0 ... y_N$).
Path from A to B can be represented as a set of vehicle poses (way-points):

$$Q = \begin{pmatrix} x_i \\ y_i \end{pmatrix} \qquad i = 0...N \tag{5.1}$$

Two way-points are apart by a specified time interval t. The time interval set T can be presented as:

$$T = \Delta t_i \qquad i = 0...N-1 \tag{5.2}$$

Time Elastic Band consists of two sets (way-poitns and time intervals):

$$B := (Q, T) \tag{5.3}$$

The band is computed by minimizing the function:

$$f(B) := \sum_k \gamma_k \tau_k(B) \tag{5.4}$$

which is a sum of multiple objectives γ_k and cost function τ_k(previously defined penalties for constraint violations). According to that, the optimal path is given by:

$$B* = minf(B) \tag{5.5}$$

The vehicle velocities and accelerations are computed from differences between consecutive points. For example the equation for velocities computation is given by:

$$Vi = \sqrt{\frac{(x_{i+1} - x_i) + (y_{i+1} - y_i)}{\Delta t_i}} \tag{5.6}$$

5.2.2 Implementation

It was mentioned previously that ROS local planners implementations are practically the same. Time-Elastic-Band algorithm obtains optimal set of velocities based on the received data from robot odometry and laser scanner topics.

5.2.3 Parameters

As BASE local planner, TEB planner also has configuration parameters, which allow the user to customize the planner to it's own needs. The parameters are divided into several groups [33]:

- robot configuration
- goal tolerance
- trajectory configuration
- obstacles
- optimization
- planning in distinctive topologies
- miscellaneous parameters

The customization parameters have been selected based on their influence on TEB planner behaviour. They are listed below:

- *min_ samples*: minimum number of samples used for defining the trajectory for avoiding the obstacle
- *min_ obstacle_ dist*: the minimal distance between the optimak path and the obstacle

Chapter 6

Implemented local planners

6.1 BUG0 local planner

BUG0 algorithm has been described in the chapter 3.1.1. BUG0 planner uses this algorithm for obstacle avoidance purposes. Only one change has been introduced in the algorithm. At the beginning, the planner defines the optimal side from which the robot will follow the obstacle (the side of the obstacle whose vertex is closer to the global path).

6.1.1 Implementation

BUG0 local planner has been implemented in Python 2.7 language.

The planner generates a set of velocities based on the received data from the odometry and laser scanner topics (see Figure 6.1).

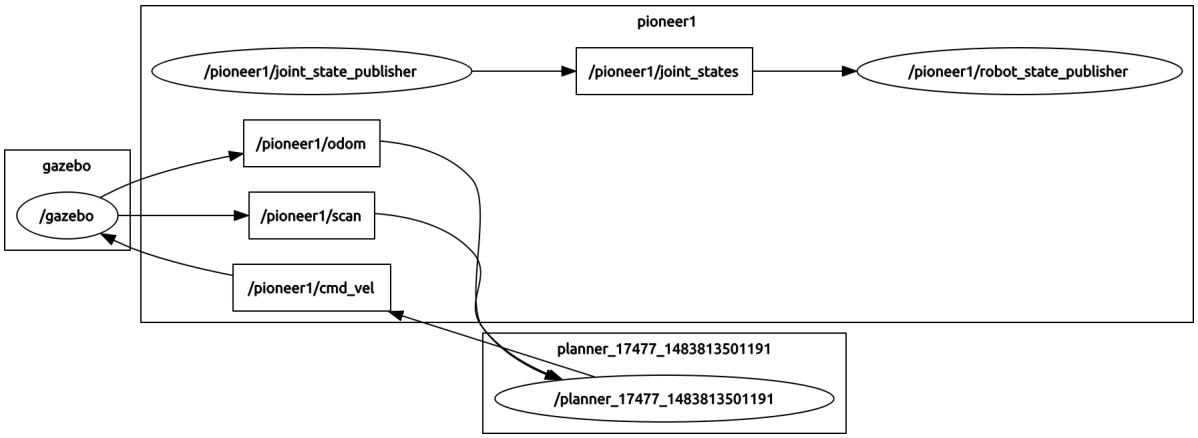

Figure 6.1: ROS computation graph for BUG0 Local Planner.

In Figure 6.2, the flow diagram for BUG0 local planner has been presented.

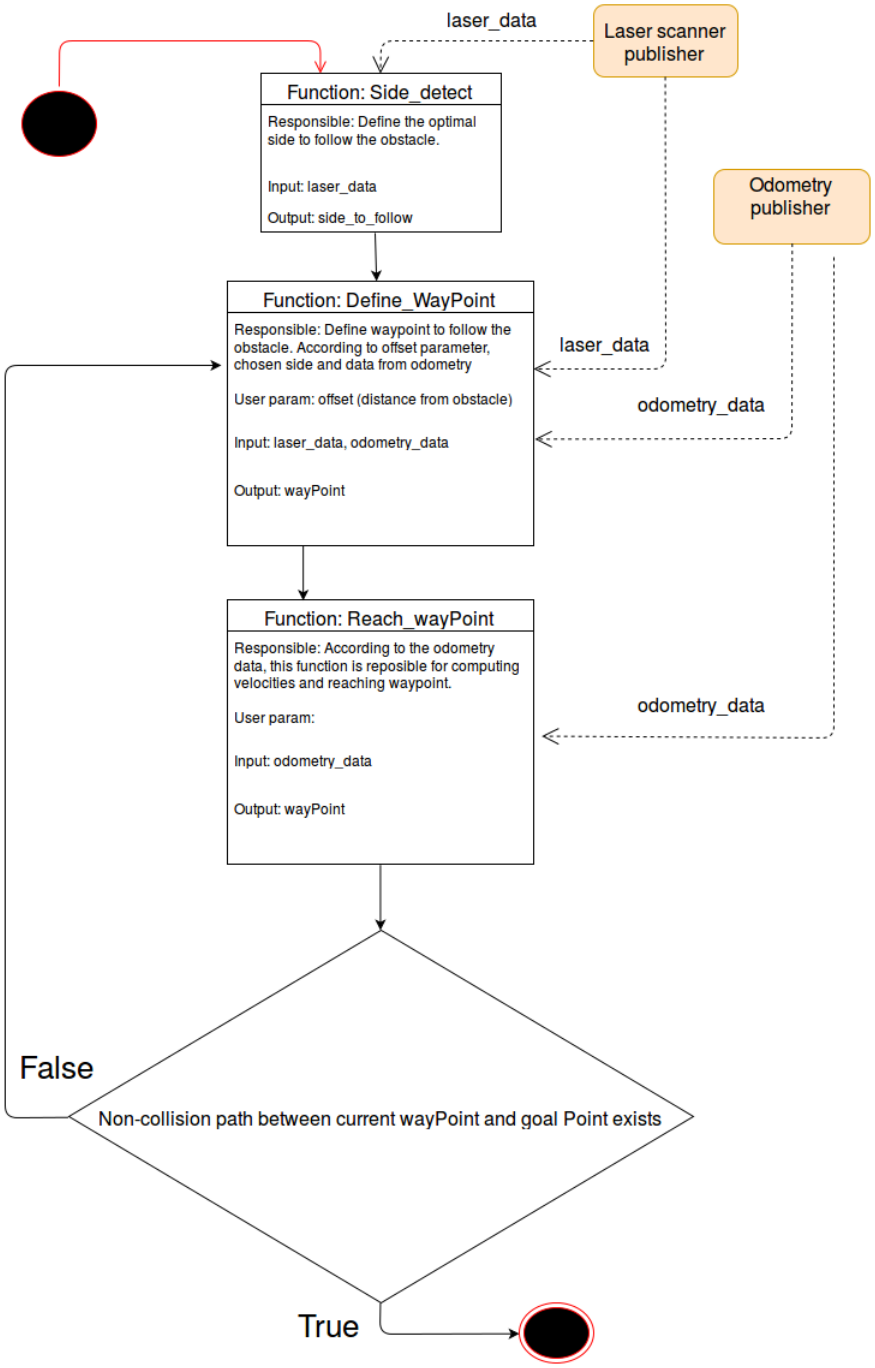

Figure 6.2: Flow chart for BUG0 local planner.

Based on the information included in Figure 6.2, the most important steps can be distinguished as follows:

1. Get data from laser scanner.
2. Define side to follow the obstacle based on the information from laser scanner.
3. Start following the obstacle from the chosen side, in every iteration(every way-point on the obstacle avoidance path), check if there exists a non-collision local path between current point and goal point. Repeat this step until such path no more exists.

6.1.2 Parameters

The biggest influence on the BUG0 planner behaviour have parameters included below:

- *yaw_ tolerance*: The tolerance in radians between current yaw and desired yaw angles.
- *min_ obstacle_ dist or offset*: The minimal distance from the obstacle to be followed.

6.2 Potential Field Local Planner

In chapter 3.2 the Potential Field approach in motion planning has been described. Obstacles push robot away and free points on the non-collision path attract the robot. The same behaviour has been used in the implementation of the Potential Field local planner.

6.2.1 Implementation

As BUG0 local planner, also this planner has been implemented in Python 2.7 language.

The planner gets data from laser scanner and odometry publishers (see Figure 6.1) and based on the obtained information, it calculates the optimal path for the robot.

As it was mentioned, the planner uses the information from the laser scanner (equation 4.1) and the odometry process (*x,y,yaw* in global frame). The final planner optimisation function is assumed to be a multiple of cost function(*cost_ func(i)*) and laser data function(*laser_ data(i)*) from equation 4.1:

$$PotencialFunc(i) := cost\_func(i) * laser\_data(i) \qquad i = 0...N-1 \tag{6.1}$$

The cost function is defined with respect to the information from odometry and goal point position. The way-points, which are closer to the desired path between the current position and goal point, are more attracted than the points which are not. The cost function is the only customization parameter for that planner.

The main goal is to maximise the final function:

$$*PotencialFunc := max(PotencialFunc(i)) \qquad i = 0...N-1 \tag{6.2}$$

Using equations (6.1 and 6.2) the planner calculates the most attractive way-point to move. Based on the odometry, the planner computes the velocities necessary to reach previously obtained way-point (see Figure 6.3).

Figure 6.3 presents the flow diagram for the Potential Field local planner.

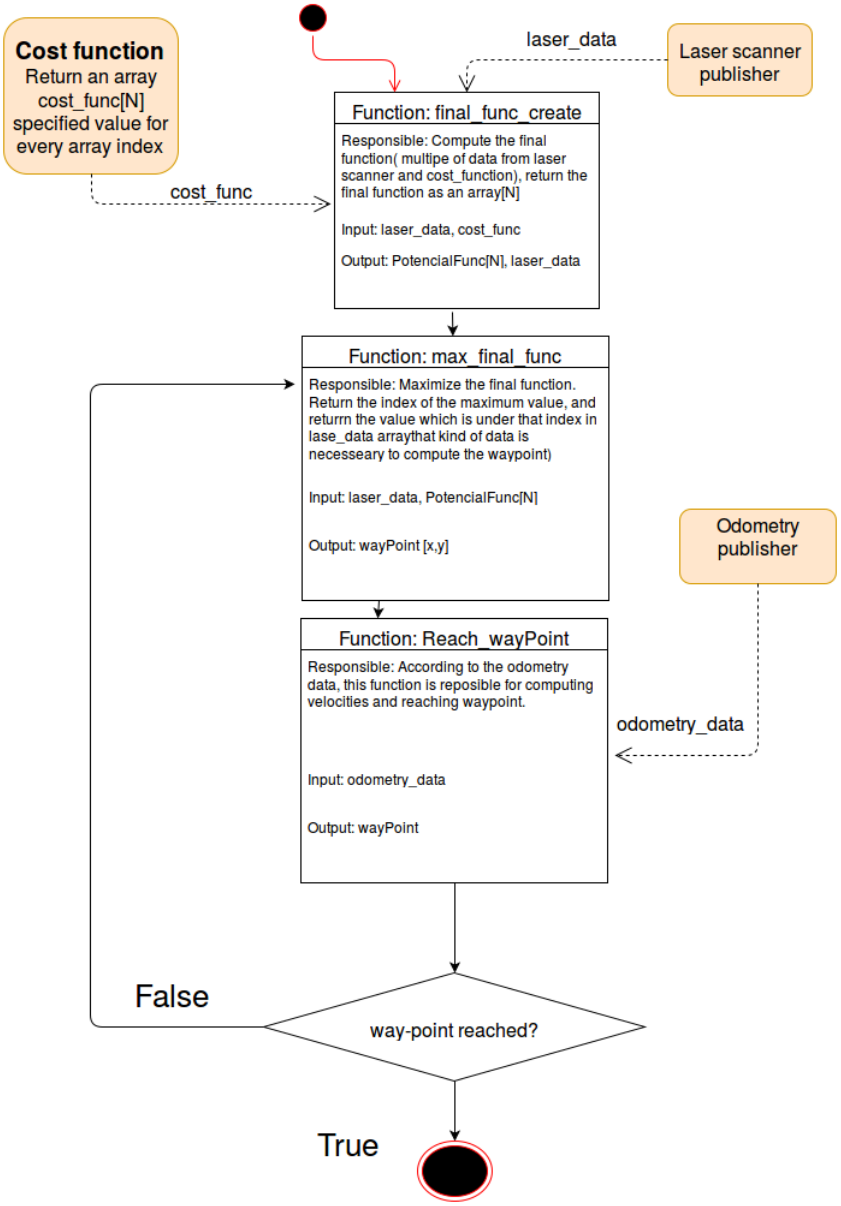

Figure 6.3: Flow chart for Potential Field local planner.

6.2.2 Parameters

In the potential field based algorithm, the cost function has the biggest affect on the algorithm behaviour:

- *cost_function*: the cost function

Chapter 7

Tests and evaluation

Both planners, the global and the local have been implemented in the robot. According to them, the robot can be in the following six states:

- start state: the map of the environment has not been sent to robot yet
- end state: robot reached the goal point
- global path planning state: robot is stopped and global planner is working
- local path planning state: robot is stopped and local planner is working
- global planner movement state: the robot is moving from current point to the goal point
- avoiding obstacle state: the robot moves from current point to the point defined by local planner

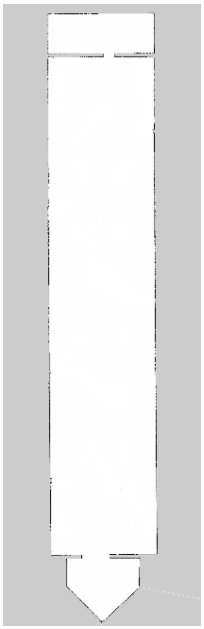

Figure 7.1: Robot test area without any obstacles.

At the beginning, the robot gets the map of the working area without any obstacles (see Figure 7.1). Based on the map, the global planner computes a global path from the start point to the goal point. In this case, the global path is just a straight line between these points. The local planner turns on when the robot detects the obstacle in its local area (see Chapter 4.3). In such a situation, the global planner turns off and the local planner is responsible for defining the obstacle avoidance path. In Figure 7.2, the reachability graph for the robot model is introduced. As it has been mentioned, the robot can be in several states:

- *START*-robot did not receive the map of the environment.
- *END*-robot reached the goal.
- *GP*-global planner is working.
- *LP*- local planner is working.
- *GPM*-robot is moving from point to point.
- *AO*-avoiding obstacle state.

Switching between states is allowed only if a particular event occurs (e1,e2,e3,e4,e5,e6);

- e1: Map has been sent to the global planner.
- e2: The path from start to goal has been defined by the global planner.
- e3: Robot has reached the goal point.
- e4: Robot has detected the obstacle on his way.
- e5: The obstacle avoidance path has been defined by the local planner.
- e6: Obstacle has been avoided, there exists a non-collision path in the robot local area from current point to global point.

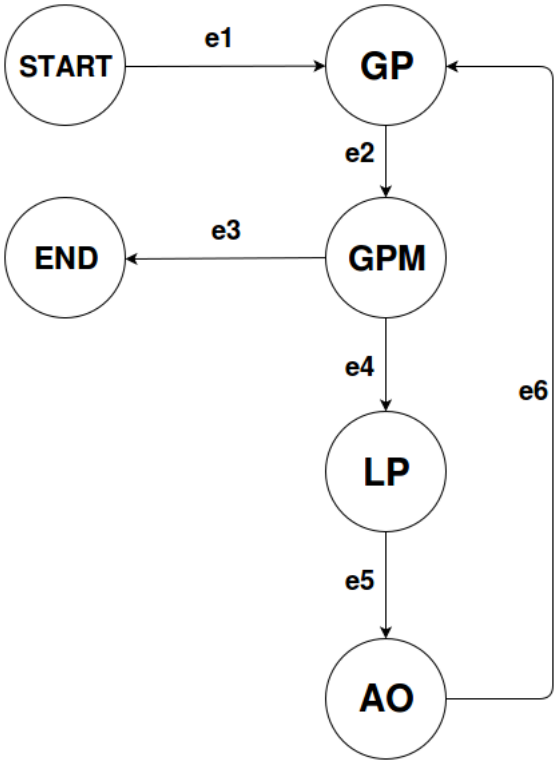

Figure 7.2: The robot reachability state graph.

7.1 Path planning area

The implemented planners have been compared with respect to their behaviour in the created test environment (see Chapter 4).

Figure 7.1 presents a bird's-eye view of the robot working area without any obstacle. The test area is a longitudinal building with dimensions 100x30[m], including several columns and obstacles (see Figure 7.3). There are three kinds of obstacles in the robot working area:

- jersey barrier (white or orange), dimensions 1,5x0,5[m]
- blue cylinder with diameter 0,5[m]
- column with dimensions 0,5x0,5[m]

As the configuration space for mobile robots is two dimensional, the height of the obstacles has been omitted.

In Figure 7.3, one can also notice the robot model (bottom right corner) with blue blow of laser visibility range.

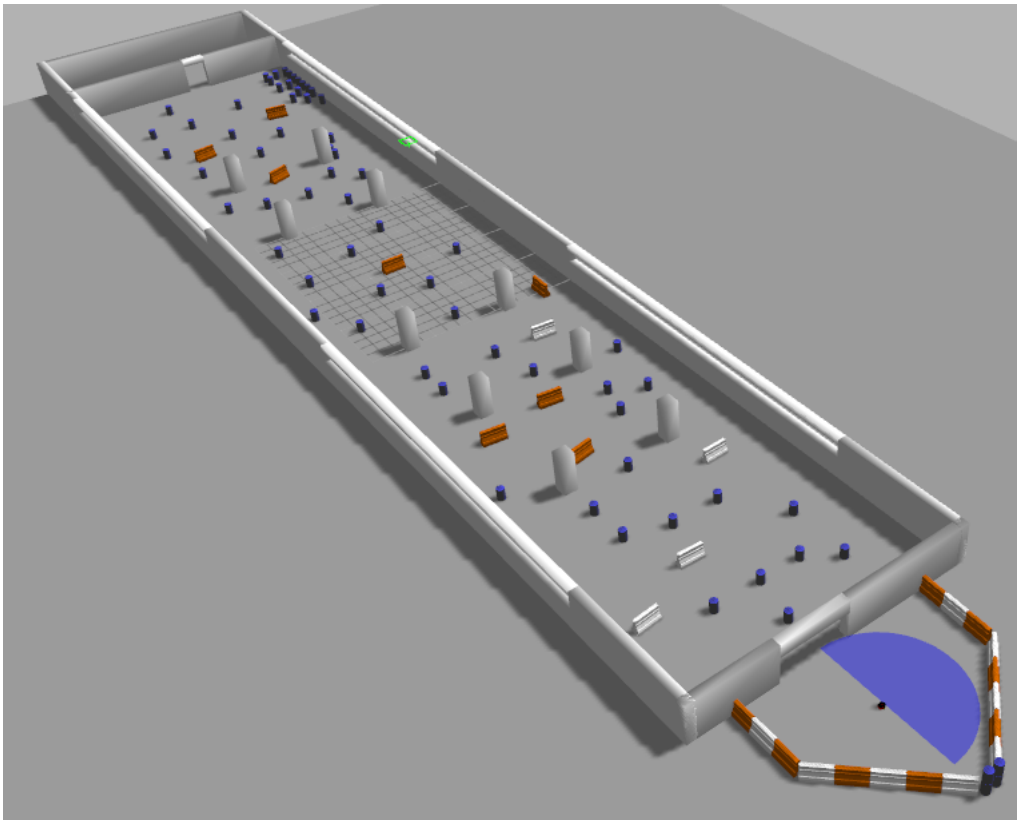

Figure 7.3: Robot test environment.

7.2 Tests and comparison

Each implemented planner was tested with respect to the change of its unique configuration parameters. The maximum and minimum variation of the translational and rotational velocities were fixed for all the planners with the values given below (see Chapter 4.3):

- max rotational velocity: 0.5 [rad/s]
- min rotational velocity: -0.5 [rad/s]
- max translational velocity: 0.5 [m/s]
- min translational velocity: 0.0 [m/s]

The main goal of the performed tests was to obtain the optimal set of parameters for each planner in the created test environment.

7.2.1 Base local planner

Below once again the list of chosen configuration parameters for the base local planner is presented:

- *vx_samples*: The number of samples used in x velocity space exploration
- *vtheta_samples*: The number of samples used in theta velocity space exploration

It is not hard to notice that the chosen parameters are the number of samples used for computation purposes in the exploration of the transitional and rotational velocities. Too few samples may cause problems with defining the proper set of velocities. The start values for the test are as below:

- *vx_samples*: 1
- *vtheta_samples*: 1

With that set of parameters, the robot was unable to obtain proper velocities. The first move which robot should take is a turn in proper goal direction. Based to that information the value of *vtheta_samples* has increased:

- *vx_samples*: 1
- *vtheta_samples*: 5

The robot implementing the trajectory rollout algorithm customised with the presented set of parameters was able to rotate but the number of samples for the both rotational and transitional velocities exploration was still to small. In the next iteration, the values of both parameters (*vtheta_samples* and *vx_samples*)have been increased:

- *vx_samples*: 2
- *vtheta_samples*: 10

In this configuration, the robot was able to turn around but still the problem with obtaining a proper yaw and moving in the goal direction occurred. In the next iteration, the values of both parameters got increased once again:

- *vx_samples*: 3
- *vtheta_samples*: 15

Finally, the robot was able to obtain a proper direction and start to move to the goal. The main problem of this planner customization were oscillations. The possibility of the obstacle avoidance was practically excluded. The oscillations occurred because the number of *vtheta_samples* was not sufficient. Based on this conclusions, *vtheta_samples* was increased in the next test iteration:

- *vx_samples*: 3
- *vtheta_samples*: 20

The robot behaviour was much better. Only small oscillations have occurred. The robot was able to avoid single obstacles without problems. However, the robot had problems when obstacle density in its local area was higher (2-4 obstacles). In such a situation, the robot started turning in place. At the end, the obstacles were avoided but this operation took too much time. According to this observation, both parameters were increased to almost double times.

- *vx_samples*: 5
- *vtheta_samples*: 35

The robot worked correctly. Obstacles were avoided even when their density in the robot local area was bigger. The set of customization parameters presented above is optimal for the Base local planner. The planner customised by this set of parameters will be used in the final comparison of the planners.

7.2.2 Time-Elastic-Band Local Planner

Time-Elastic-Band local planner customisation parameters have been introduced in chapter 5.2.3:

- *min_samples*: minimal number of samples used in defining the trajectory for avoiding obstacles
- *min_obstacle_dist*: minimal predefined distance between the optimal path and the obstacle

At the beginning the smallest possible values were set to the chosen parameters:

- *min_samples*: 1
- *min_obstacle_dist*: 0.05[m]

The robot was able to move but the collision avoidance was not working. The trajectory was beeing changed even when no obstacle was detected. One sample was not enough for defining a proper trajectory by using the Time-Elastic-Band approach. In the next iteration, the number of the samples was increased:

- *min_samples*: 2
- *min_obstacle_dist*: 0.05[m]

The robot was able to avoid obstacles but its movement was interrupted by the calculation of the new trajectories every 1-2 sec. The number of samples had to be increased once again:

- *min_samples*: 5
- *min_obstacle_dist*: 0.05[m]

Now, the robot had a stable movement. Obstacles were avoided properly. The main problem with this set of customisation parameters was that the robot was avoided the obstacles too close. This sometimes caused collisions. In the next iteration, the minimal distance from the obstacle was increased:

- *min_samples*: 5
- *min_obstacle_dist*: 0.2[m]

The robot movement was stable, without oscillations. The defined avoidance trajectory was no more changed without a purposes (e.g. a new obstacle on the path). However, the same behaviour occurred as in the case of the Base local planner. The robot had big problems with defining a proper trajectory when obstacle density in its local area increased (more obstacles than 2). According to this observation, the number of samples used for defining the proper trajectory had to be increased:

- *min_samples*: 10
- *min_obstacle_dist*: 0.2[m]

Increasing the number of trajectory samples was the solution for the problem. The robot was able to reach the goal without any problems. This set of parameters will be used in the final comparison of the planners.

7.2.3 BUG Local Planner

BUG algorithm and planner implementations have been introduced and described in chapters 3.1.1 and 6.1.1. BUG planner customization parameters have been described in chapter 6.1.2 and selected according to the same rules as for the configuration parameters of other planners. Selected parameters are listed below:

- *yaw_tolerance*: tolerance in radians between current yaw and desired yaw
- *min_obstacle_dist or offset*: minimal distance from the obstacle to be followed

The test started with the lowest possible set of the parameter values:

- *yaw_tolerance*: 0.05 rad[rad]
- *min_obstacle_dist or offset*: 0.1[m]

The robot was not able to obtain a proper yaw angle accuracy. Consequently, the robot was unable to move and was just rotating in place. The yaw angle tolerance had to be increased:

- *yaw_tolerance*: 0.1 rad[rad]
- *min_obstacle_dist or offset*: 0.1[m]

With that kind of yaw angle accuracy, the robot was able to move from the start position. However, its movement was interrupted for the yaw angle correction purposes. Due to this, the robot was unable to define a proper non-collision trajectory. The yaw angle tolerance was increased once again:

- *yaw_tolerance*: 0.2 rad[rad]
- *min_obstacle_dist or offset*: 0.1[m]

The robot was able to move without any interruptions, the obstacle was avoided properly but the distance from the obstacle was too small. A small distance from obstacles together wit a big yaw angle error tolerance caused collisions with obstacles. The minimal distance from obstacle was increased:

- *yaw_tolerance*: 0.2 rad[rad]
- *min_obstacle_dist or offset*: 0.2[m]

The robot movement was clear and caused no collisions. The robot was able to reach the goal every time, but this set of parameters was not optimal for the predefined robot working area (see chapter 7.1). Getting back to the movement took too long after the robot had stopped (too small yaw angle tolerance). Also, the distance from an obstacle was too small for that kind of area. The robot wasted a lot of time for calculating the closest path. According to these conclusions, both parameters were increased:

- *yaw_tolerance*: 0.3 rad[rad]
- *min_obstacle_dist or offset*: 0.5[m]

The robot was able to avoid obstacles. Its path was optimal for that kind of a working area. This set of configuration parameters will be used in the final comparison of the planners.

7.2.4 Potential Field local planner

The Potential Field local planner has been described in Chapter 6.2. The cost function is the customization 'parameter' for this planner. It is assumed that the cost function can only be a rectangular function. An example of a rectangular function is presented below:

$$cost\_func(i) = \begin{cases} value\_1, for : i \in (0, range\_1 > \\ value\_2, for : i = 0 \\ value\_3, for : i \in (range\_1, range\_2 > \end{cases} \tag{7.1}$$

Thus, customisation for the Potential Field local planner narrows to designing the cost function (see equation 7.1).

The customisation process for this planner was started in a different way than those for the other planners. The start value and the ranges for the *cost_func(i)* had the highest possible values. The first iteration of the customisation *cost_func(i)* had the following form (according to the description of Chapter 4.3.1: $i \in < 0, 499 >$) :

$$cost\_func(i) = \begin{cases} 1, for : i \in < 0, 199) \\ 2, for : i \in < 199, 299 > \\ 1, for : i \in (299, 499 > \end{cases} \tag{7.2}$$

The cost function was most restrictive as for maintaining the trajectory defined by the global planner. The most attractive points were the closest to the global trajectory. This kind of cost function was quite good. However, bigger obstacles caused some problems. In order to make cost function less restrictive, the differences between the values of the ranges were decreased:

$$cost\_func(i) = \begin{cases} 1, for : i \in < 0, 199) \\ 1.2, for : i \in < 199, 299 > \\ 1, for : i \in (299, 499 > \end{cases} \tag{7.3}$$

Less restrictive cost function was the solution for avoiding bigger obstacles. However, such a cost function was not optimal. Once again a problem with higher obstacle density in robot local area occurred. The number of ranges had to be increased:

$$cost\_func(i) = \begin{cases} 1, for : i \in < 0, 99) \\ 1.1, for : i \in < 99, 199) \\ 1.2, for : i \in < 199, 299 > \\ 1.1, for : i \in < 299, 399) \\ 1, for : i \in (399, 499 > \end{cases} \tag{7.4}$$

The robot was able to reach the goal. The obstacles were avoided even when their density was higher. The main issue in the cost function customised like that was that sometimes the robot would not choose the optimal trajectory. The robot tended to defined the trajectories which were too close to the small obstacles or too far from the bigger obstacles. Based on the above information, the decision about increasing the number of

the ranges was taken.

$$cost\_func(i) = \begin{cases} 1, for : i \in < 0, 99) \\ 1.1, for : i \in < 99, 149) \\ 1.15, for : i \in < 199, 229) \\ 1.2, for : i \in < 229, 269 > \\ 1.15, for : i \in < 269, 299) \\ 1.1, for : i \in < 349, 399) \\ 1, for : i \in (399, 499 > \end{cases} \tag{7.5}$$

The cost function was customised properly. The robot was able to avoid every obstacle on its path. The Potential Field local planner with so customised cost function will be used in the final comparison.

7.3 Final comparison - speed test

In this section, all the previously presented planners with their optimal customisation parameters sets will be compared. Which is the best way for comparing planner algorithms with fixed maximum rotational and transitional velocities? Of course a race! In this chapter a speed test will be presented. Each planner will be tested in the same working area (see Chapter 7.1), which will be implemented in the same robot model (see Chapter 4.3) with the same start point (red dot in Figure 7.4) and goal point (green dot in Figure 7.4).

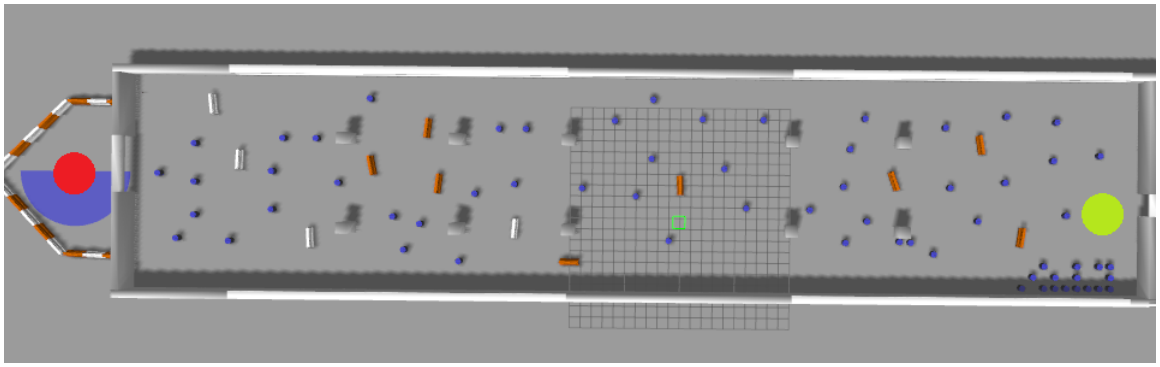

Figure 7.4: Robot Speed test working area with star t(red) and goal (green) points.

There is almost 90 meters in a straight line between the start and the goal point, also the robot has more than 30 obstacles on his way to avoid.

The result of the performed speed test are included below:

1. Potential Field Local Planner: 8min 11s
2. Base Local Planner: 8min 37s
3. Time-Elastic-Band Local Planner: 9min 26s
4. BUG Local Planner: 10min 13s

The difference between the speed test results obtained by the Potential Field local planner and the Base local planner is less than 4%, so we can simply conclude that the quality of these planners in employed area is similar.

The TEB local planner has problems with avoiding more than one obstacle in its local area. Such an operation took a few seconds more than the same operation for faster planners. It caused a delay that affected for the result of the TEB Local Planner. A robot implementing the TEB planner was almost 15% slower than the fastest planner.

The slowest planner was the BUG0 local planner. Such a rank, in view of the behaviour of the algorithm, is not surprising. For the robot implementing BUG0 planner the whole route from the start to the goal position took almost 2 minutes more than for the Potential Field local planner.

Chapter 8

Summary

8.1 Test environment

The main goal of this master thesis was to compare several local planners. For this purpose, a whole test environment had to be designed and created. The Gazebo graphical simulator with the Pioneer 3-DX robot model was configured and connected with the Robot Operating System (ROS) framework.

The Pioneer 3-DX robot model was configured and designed with real robot compatibility, which made the implemented planner portable between the simulation and real environments.

A robot working area has been designed and created in Gazebo Simulator. The final version of the robot working area was a rectangular building, 100 meters long and 30 meters wide. In the building, there were more than 40 obstacles of different sizes. The robot working area was created for the simulation purposes. The main assumption in its design was to enable testing of the planner behaviour in various situations, for example, characterized by:

- Obstacle high/low density.
- Long working time (big working area).
- Versatility; since different planners were to be tested, the working area should be equally difficult for different planner types.
- Flexibility; the working area can be changed even in the simulation time, an obstacle can be added or removed as needed.

Next, the whole graphically designed test environment was connected with ROS. The Gazebo simulator and the Pioneer 3-DX graphical model became represented as nodes in the ROS framework.

The so designed test environment can be used for the purposes of Local or Global planner prototyping, with one or multiple robots in the working area.

The created test environment is flexible. This means that the user is able to change and design the robot working area for its own purposes. The robot model can also be customised and configured as needed. The Pioneer 3-DX model can be substituted by another robot model available in the Gazebo libraries, or the users can create and design their own robot model.

8.1.1 Problems

I faced several problems associated with the test environment designing part.

First of all, there are a lot of previously created Pioneer 3-DX models compatible with Gazebo and ROS. The main problem was that none of them had the Hokuyo Laser configured in the proper place, on the front top of the robot. Consequently, I had to edit the robot model script according to my own needs. There are no tutorials or manuals devoted to the set up and customization of robot models in Gazebo.

Gazebo and ROS compatibility depends strictly on which ROS distribution and Gazebo version are in use. For example, Gazebo 7 is not fully compatible with ROS Indigo. This caused a lot of problems as in the beginning. I tried to use them in such configuration. After a few failed attempts, only the ROS Indigo and Gazebo 2 combination turned out to be sufficiently stable.

An initial assumption for the ROS framework was to implemented it in real robots. This caused a lot of problems with the ROS local planner configuration in the Gazebo simulator as well as with the Pioneer 3-DX robot model connection. The impact of this assumption was visible at every step of working with the ROS framework.

The Gazebo simulator requires a lot of computing power for working properly. Therefore all tests in Gazebo must be executed on sufficiently powerful computers.

The Gazebo simulation time is not configurable. The user is not allowed to decrease or increase the simulation time according to his or her purposes.

8.1.2 Conclusions

The Gazebo simulator is an advanced tool for rapid prototyping with very high compliance to the real environment and real robots. In this master thesis, such real-world compliance quality wasn't necessary. The main goal of the following master thesis was to compare several planners. For such purposes a graphical simulator using grid maps and robots represented as single points would have been sufficient.

The graphical simulator was too much advanced, which caused many problems with the configuration and connection with ROS, and with the planners implementation.

8.2 Local Path Planning Algorithms

A specific goal of this master thesis was to compare two ROS algorithms and two self-implemented algorithms. This goal has been reached. The compared algorithms are listed below:

- Base Local Planner
- Time-Elastic-Band Local Planner
- BUG0 Local Planner
- Potential Field Local Planner

To achieve this goal, the algorithms were implemented and configured according to the comparison needs. The developed planners were tested and compared in the previously created test environment.

8.2.1 Conclusions

The Potential Field algorithm proved to be the fastest in the created working area. This kind of planning approach proved to be most suitable for the tested type of working area (a big area with medium obstacle density).

The Base local planner produced a really close result to that of the Potential Field planner. The Trajectory Rollout algorithm is also considered as an algorithm for big area environments, so the good result could have been predictive.

The biggest surprise was the result obtained for the Time-Elastic-Band. This planner is considered as more advanced and more effective than the Sample Base planner. However, the problems with finding a proper path in the higher obstacle density area have caused such low rank of this planner on the comparison list. Interestingly such problems occurred after a long time of the planner operation, and therefore obstacles were not detected in the initial tests.

Fortunately , after some more research and additional tests, the cause of the problem was found. Gazebo node after some time of working has stopped publishing laser scanner data for almost 1 second. Due to that, the planner did not have sufficient information for obstacle avoidance. Such a problem occurred only for the TEB local planner, which showed that its configuration was not optimal.

The rank of BUG0 planner was easy to predict. This planner ensures reaching the goal, but in the optimal way. Also, the working area was not totally appropriate for this planner. The BUG0 algorithm would better show its advantages in a small area with high obstacle density. In contrast to the above, the robot implementing the Potential Field approach had big problems with reaching the goal, which was caused by the local extremes occurrence.

Bibliography

[1] IHS Markit, *Autonomous vehicle sales forecast to reach 21 mil. globally in 2035, according to IHS Automotive*, www.ihs.com

[2] Sanjiban Choudhury , Sanjiv Singh , *A Coverage Planning Algorithm for Agricultural Robots*, August 2009

[3] Tsutomu MAKINO, Hiroshi YOKOI, Yukinori KAKAZU, *Development of a Motion Planning System for an Agricultural Mobile Robot*, HOKKAIDO University, N-13, W-8,kita-ku, Sapporo, Japan

[4] Advantech, *Autonomous Harvesting Technology Realizes Intelligent Agriculture*, www.advantech.com/logistics

[5] Sarah Fecht, *The World's First Fully Robotic Farm Opens In 2017*, Popular Science Magazine, www.popsci.com

[6] Steven M. LaValle,*Planning Algorithms* 2006, Cambridge University Press, pages: 19-20.

[7] Steven M. LaValle,*Planning Algorithms* 2006, Cambridge University Press, pages: 97,144.

[8] J.T. Schwartz and M. Sharir, *On the "Piano-Movers" Problem: I. The case of a two-dimensional rigid polygonal body moving amidst polygonal barriers.Communications on Pure and Applied Mathematics*, pages: 345–398.

[9] Buniyamin N., Wan Ngah W.A.J., Sariff N., Mohamad Z., *A Simple Local Path Planning Algorithm for Autonomous Mobile Robots*, INTERNATIONAL JOURNAL OF SYSTEMS APPLICATIONS, ENGINEERING & DEVELOPMENT

[10] Jorge Cham, *R.O.B.O.T. Comics: Path Planning*, Willow Garage 2009

[11] Steven M. LaValle,*Planning Algorithms* 2006, Cambridge University Press, pages: 34.

[12] Howie Choset, *Robotic Motion Planning: Bug Algorithms*, Robotics Institute 16-735 , slides: 1-5

[13] Howie Choset, *Robotic Motion Planning: Bug Algorithms*, Robotics Institute 16-735 , slides: 6-0

[14] Howie Choset, *Robotic Motion Planning: Bug Algorithms*, Robotics Institute 16-735 , slides: 11-17

[15] Hani Safadi, *Local Path Planning Using Virtual Potential Field*, McGill University School of Computer Science 2007

[16] Howie Choset, *Robotic Motion Planning: Bug Algorithms*, Robotics Institute 16-735 , slides: 18-24

[17] Wolfram Burgard, Cyrill Stachniss, Maren Bennewitz, Kai Arras *Robot Motion Planning*, Freiburg University 2011

[18] Bob Givan, Ron Parr *An Introduction to Markov Decision Processes*, Purdue Universitu, Duke University

[19] Jean-Claude Latombe, *Robot Motion Planning* , 1991, Kluwer Academic Publishers, pages: 295-297.

[20] Steven M. LaValle,*Planning Algorithms* 2006, Cambridge University Press, pages: 185-195.

[21] Weria Khaksar, Khairul Salleh Mohamed Sahari and Tang Sai Hong,*Application of Sampling-Based Motion Planning Algorithms in Autonomous Vehicle Navigation*, INTECH

[22] Jean-Claude Latombe, *Robot Motion Planning* , 1991, Kluwer Academic Publishers, pages: 153-156.

[23] Steven M. LaValle,*Planning Algorithms* 2006, Cambridge University Press, pages: 20-22.

[24] Pearl J. *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Addison-Wesley, 1984. p. 48.

[25] ROS.org, *About ROS*, www.ros.org

[26] ROS.org, *ROS distributions*, www.ros.org

[27] http://wiki.ros.org/Packages, *ROS Wiki*, www.ros.org

[28] http://wiki.ros.org/Nodes, *ROS Wiki*, www.ros.org

[29] gazebosim.org *Why Gazebo?*,www.gazebosim.org

[30] MobileRobots, *Datasheet: Pioneer 3DX-P3DX-RevA 09366-P3DX Rev. A*, www.mobilerobots.com

[31] MobileRobots, *Pioneer P3-DX*, www.mobilerobots.com

[32] ROS.org, *base_local_planner*, wiki.ros.org

[33] Martin Keller, Frank Hoffmann, Torsten Bertram, Carsten Hass, Alois Seewald, *Planning of Optimal Collision Avoidance Trajectories with Timed Elastic Bands* 19th World Congress The International Federation of Automatic Control Cape Town, South Africa. August 24-29, 2014