TÉCNICO LISBOA

Benchmarking Mobile Robot Local Motion Planning

Maria Manuel Salgueiro Amaral Salgueiro

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisors: Prof. Pedro Manuel Urbano de Almeida Lima
Prof. Enrico Piazza

Examination Committee

Chairperson: Prof. João Fernando Cardoso Silva Sequeira
Supervisor: Prof. Pedro Manuel Urbano de Almeida Lima
Member of the Committee: Prof. Matteo Matteucci

December 2021

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Preface

The work presented in this thesis was performed at the Institute for Systems and Robotics of Instituto Superior Técnico (Lisbon, Portugal), during the period March-October 2021, under the supervision of Prof. Pedro Lima (IST) and PhD Candidate Enrico Piazza (Politecnico di Milano).

Acknowledgments

This thesis marks the end of an incredible 5 years journey in IST. During these years I lived some of the best moments of my life, there were some moments of frustration but most importantly I learned so many things that will forever help me and for this I'm very grateful and honored in being a graduate of this school.

First, I want to thank my supervisors: Prof. Pedro U. Lima for all the wise advises and valuable inputs trough this thesis and to Enrico Piazza for his everlasting patience and all the help. Without them, this thesis would never be possible.

During these years I made friendships that will last forever, I could not finish this journey without mentioning Margarida, Catarina, Filipa, Carolina, Luísa and of course the best flatmates I could ask for Teresa and Maria. To each and one of you: I don't know how I would have done it without you, you made me feel at home even when I was very far from it.

Speaking of home, a very special thanks to Euchi for being my best buddy and a huge support specially in the last moments of this journey.

Last but not least, to my family for all the support I could ever ask for and all the values that taught me that brought me here. A particular mention to my mom for being my biggest supporter and for having always the right words. This is for you.

Abstract

Benchmarking has been widely used in multiple fields and it has been playing a key role in pushing progress by quantifying improvements.

However, the situation is rather different in Robotics. This can be explained by the complexity of the systems and the consequently difficulty in the evaluation process. Other obstacle to the benchmarking process in robotics is the uniqueness of the hardware and software used in experiments, which makes it harder to replicate and compare the obtained results.

Despite all this, benchmarking in robotics could be beneficial and a great way to push progress and innovation even further. In this work, we are interested in benchmarking a specific functionality within the robot system, the local motion planning.

To achieve this, we tested two different local planners, Dynamic Window Approach (DWA) and Pure Pursuit, in simulated rooms with a simulated robot. The rooms used have different characteristics to provide a fair evaluation of the planners.

With the collected data from each simulation we were able to evaluate the performance of the planners by implementing different metrics. By comparing the results, we can understand how the planners perform in different situations and in which situations one performs better than the other.

More than a absolute evaluation of the planners, the goal is to understand how the performance of each planner is affected by different aspects and help researchers in their decision while choosing a local planner.

Keywords

Benchmarking; Local Motion Planners; Pure Pursuit; Dynamic Window Approach; Performance

Resumo

Benchmarking tem sido largamente utilizada em várias áreas, pelo facto de permitir uma quantificação de melhoramentos tem sido um factor importante em termos de inovação e progresso.

No entanto, a situação no campo da robótica é bastante diferente. Isto pode ser explicado pela complexidade dos sistemas e consequente dificuldade no processo de avaliação. Outro factor é a especificidade do hardware e software utilizado o que torna complicado a possibilidade de replicar e comparar resultados.

Apesar de tudo it, benchmarking em robótica poderia ser extremamente benéfica e uma ótima maneira de acelerar progresso e inovação. Com este trabalho, estamos interessados no processo de benchmarking de uma função específica, planeamento local.

Para atingir este objetivo, testámos dois algoritmos de planeamento local diferentes, Dynamic Window Approach (DWA) e Pure Pursuit, em salas simuladas com um robot simulado. As salas apresentam diferentes características o que torna a avaliação mais justa.

Com os dados adquiridos em cada simulação é possível avaliar a performance dos algoritmos de planeamento local atravé de diferentes métricas. Comparando os resultados, podemos perceber como é que cada algoritmo se comporta em diferentes situações e em que situações um dos algoritmos apresenta melhores resultados quando comparado com o outro.

Mais do que um método de avaliação absoluta dos algoritmos, o objetivo deste trabalho é perceber a performance de cada algoritmo e como é que esta é afetada por diferentes fatores.

Palavras Chave

Benchmarking;Algoritmos de Planeamento Local;Pure Pursuit;Dynamic Window Approach;Performance

Contents

1 Introduction **1**

1.1 Problem and Motivation . . . 3

1.2 Proposed Method . . . 4

1.3 Summary of Contributions . . . 5

1.4 Thesis Organization . . . 5

2 Theoretical Background **7**

2.1 Local Planners . . . 9

2.1.1 Dynamic Window Approach . . . 9

2.1.2 Pure Pursuit . . . 12

2.2 Benchmarking . . . 15

3 State of the Art **17**

4 Proposed Method **23**

4.1 Theoretical Framework . . . 25

4.1.1 Local Planners . . . 25

4.1.2 Parameters . . . 26

4.2 Metrics . . . 27

5 Experimental Implementation **29**

5.1 ROS Components . . . 31

5.2 Coordinate Frames . . . 33

5.3 Benchmarking Framework . . . 34

5.4 Robot Model . . . 35

5.5 Simulation . . . 36

5.6 Scenarios . . . 37

6 Experimental Results **41**

6.1 DWB . . . 43

6.2 Regulated Pure Pursuit . . . 44

6.3 Comparison Between Planners 45

7 Conclusions **51**

7.1 Conclusions 53

7.2 Future Work 53

Bibliography **55**

List of Figures

2.1 Target heading 11
2.2 Illustration of the Pure Pursuit algorithm 13
2.3 Pure Pursuit algorithm frame 14
2.4 Pure Pursuit algorithm frame 14

5.1 Navigation architecture diagram 32
5.2 Robot Operating System (ROS) graph with the nodes used in our implementation 32
5.3 Example of the ROS Transform Library (tf) tree implemented 34
5.4 Specifications of the Turtlebot3 Waffle model 36
5.5 Simulator view with a simulated scenario and the simulated robot model 37
5.6 Left: Voronoi diagram - the set of line segments separating the regions corresponding to different colors. Right: Voronoi diagram in the context of path planning 37
5.7 Set of scenarios used in the experiments pt.I 38
5.8 Set of scenarios used in the experiments pt.II 39
5.9 Set of scenarios used in the experiments pt.III 39

6.1 Total average execution time in function of total average path length for both planners . . 47
6.2 Average execution time in function of average path length for both planners, when the target pose is reached or not 47
6.3 Relation between average execution time and the different scenarios for DWB 48
6.4 Relation between average path length and the different scenarios for DWB 49

List of Tables

4.1 Common parameters between the local planners used 26

6.1 Evaluation of DWB by percentage of times that the target pose is reached 43
6.2 Evaluation of DWB by percentage of times that the target pose is reached in different scenarios 43
6.3 Evaluation of DWB by average execution time and standard deviation 43
6.4 Evaluation of DWB by average path length and standard deviation 44
6.5 Evaluation of Regulated Pure Pursuit (RPP) by percentage of times that the target pose is reached 44
6.6 Evaluation of RPP by percentage of times that the target pose is reached in different scenarios 44
6.7 Evaluation of RPP by average execution time and standard deviation 45
6.8 Evaluation of RPP by average path length and standard deviation 45
6.9 Evaluation of DWB and RPP by percentage of times that the target pose is reached in different scenarios 46
6.10 Evaluation of DWB and RPP by average execution time and standard deviation 46
6.11 Evaluation of DWB and RPP by average path length and standard deviation 46

Acronyms

AI	Artifical Intelligence
BT	Behavior Tree
DWA	Dynamic Window Approach
Nav2	Navigation 2
RoCKIn	Robot Competitions Kick Innovation in Cognitive Systems and Robotics
ROS	Robot Operating System
RPP	Regulated Pure Pursuit
tf	ROS Transform Library

1

Introduction

Contents

1.1 Problem and Motivation 3
1.2 Proposed Method 4
1.3 Summary of Contributions 5
1.4 Thesis Organization 5

We start by motivating the problem addressed in the thesis followed by a description of the method proposed. Then we summarize our contributions. At last, we outline the organization of the thesis.

1.1 Problem and Motivation

In the last few years there have been significant progress in multiple Artifical Intelligence (AI) fields, namely in machine learning, computer vision and speech processing. Multiple reasons can explain this progress, but there is one that tends to be overlooked: the development of a conducive research ecosystem. [1] The existence of multiple platforms like PyTorch and TensorFlow reduces the difficulty of starting research in those areas. These common platforms and datasets have led to standardized evaluations and benchmarks which translate into a better progress quantification in those areas. [1]

When it comes to robotics the progress and exciting innovations are also inarguable. However, when compared to other research areas it is harder for a new researcher to get started and contribute to progress in the field due to the lack of standard development tools.

As described in [2] the validation of robotic research was relatively straightforwared until a few years ago. If a researcher had published some of his work in a journal, mentioning the equations and solver details of the software used for example, it was possible for the readers to re-run numerical simulations in a short period of time. People in the field would have the chance to test, validate and possibly generalize the algorithm. A lot has changed in the robotics field since then. Machines are much more complex and the robots no longer live confined to specific environments, but are present in our daily life, from our houses to airports and even Mars.

The progress in the field of robotics have to, and will continue and it is our duty to understand how we can change the method of research in a way that leads to more and faster innovation. Quoting Steven Johnson [3]: “If you look at history, innovation does not come just from giving people incentives; it comes from creating environments where their ideas can connect”. The research community needs to exchange experiences, algorithms, pieces of code to replicate the experiments, find bugs or theoretical flaws and lastly to *benchmark*. This latter is the specific theme we address in this thesis.

Benchmarks were defined as “process of identifying the highest standards of excellence for products, services and processes and then making the improvements necessary to reach those standards” in [4], while the definition presented in [5] adds numerical evaluation of results (performance metrics) as a key element of benchmarking.

Benchmarks are important instruments to compare and evaluate the results of experiments. They can play an important role in various aspects as several areas of research require benchmarking to assess the value of new results against a reference performance. Benchmarking can also be of great help to scientists and technical developers, establishing clear technical targets to demonstrate a certain

readiness level of the technology. [6]

In this thesis the focus will be on the benchmarking of a specific functionality: robot local motion planning.

The path-planning problem is usually defined as follows: “Given a robot and a description of an environment, plan a path between two specific locations. The path must be collision-free (feasible) and satisfy certain optimization criteria”. In other words, path planning translates into generating a collision-free path in an environment with obstacles and optimizing it with respect to defined criterion. [7]

The path planning component is divided in two sections: (1) global and (2) local path planning. Global planning requires a map of the environment to calculate the best route. Depending on the analysis of the map, some methods solve the problem by assigning a value to each region of the roadmap in order to find the path with minimum cost. Another approach is by dividing the map into small regions (cells) called cell decomposition. After the success of finding the global path from the start to the goal, all the selected nodes are translated into positions in the reference axes.

In order to transform the global path into suitable waypoints, the local planner creates new waypoints taking the dynamic obstacles and the vehicle constraints into consideration. The map is reduced to the surroundings of the vehicle and is updated as the vehicle is moving around so that a new path can be calculated at a specific rate. The sensors are unable to update the map in all regions and therefore it is impossible to use the whole map. Besides this, a large number of cells would raise the computational cost. Local path planning algorithms can be implemented in a way of seeking the goal and avoiding nearby obstacles. Another approach is to follow a path planned before by the global path planning. With the local map and the global waypoints, the local planner is able to generate strategies to avoid dynamic obstacles and use the waypoints as guidance for the trajectory.

Local path planning means that the path planning is done while the robot is moving, which means that the algorithm is capable of producing a new path dynamically in response to changes in the environment. If no obstacles are on the way, the shortest path between the start point and the end point is a straight line. Therefore the robot will proceed along this path until an obstacle is detected. At this point the algorithm responsible for the path planning must be able to find a feasible path around the obstacle. After avoiding the obstacle, the robot continues to navigate toward the end-point along a straight line until another obstacle is detected or the end position is reached. [8]

1.2 Proposed Method

In order to tackle the problem presented in 1.1 we will use a benchmark framework that evaluates different local planners in a multitude of scenarios against different metrics. This variety enables a better understanding of where to use a certain local planner in detriment of other, which can be crucial in

situations where robots play a key role.

Our benchmark evaluates the performance of different local planners in seven scenarios with various characteristics. In order to obtain the evaluation, a robot is designed and simulated in various scenarios using a simulator tool to represent the reality as close as possible. Our goal is to use the data provided by these simulations to make an evaluation in terms of multiple metrics, specifically average execution time and average path length.

1.3 Summary of Contributions

Making use of the proposed methods explained above, we were able to:

- Test two local planners in various simulated scenarios and using a simulated robot;
- Implement different metrics: average execution time, average path length and number of times that the robot reaches the goal;
- Presentation and discussion of the experimental results.

1.4 Thesis Organization

This thesis is organized as follows. Chapter 2 contains a review of local motion planning algorithms, While chapter 3 presents the state of the art in benchmarking mobile robot local planning. In Chapter 4 the benchmarking methodology is described and chapter 5 details the experimental implementation. In Chapter 6 we present the achieved results and chapter 7 finishes off with concluding this thesis by summarizing the work and pointing out directions for future research.

2

Theoretical Background

Contents

2.1 Local Planners 9
2.2 Benchmarking 15

This chapter starts with presenting a review on local motion planning algorithms, that are required for the objectives of this thesis: Dynamic Window Approach (DWA) and Pure Pursuit algorithm. Lastly, an overview of benchmarking is presented.

2.1 Local Planners

2.1.1 Dynamic Window Approach

The DWA is a local motion planner for mobile robots proposed in [9]. Unlike other methods of this type, DWA is directly derived from the dynamic features of the robots and is designed with the goal of dealing with constraints imposed by limits on velocity and acceleration of the robot. The search for commands controlling the robot is carried out directly in the space of velocities. [9]

Through mathematical manipulation, the trajectory of the robot can be approximated by a sequence of circular arcs, that are commonly referred as curvatures. Each of these curvatures is uniquely determined by a velocity vector, (v_i, w_i), where v_i is the linear and w_i the angular velocity. From this point on, we will refer to this vector as velocity. To generate a trajectory to a given goal, velocities have to be determined for each of the next n time intervals. This would result in an exponential search space for these vectors in terms of the considered intervals.

To avoid complexity, the DWA considers a short time interval and assumes that the velocities will stay constant in the remaining time intervals, which equals to assuming zero accelerations in that time interval. This approximation is done based on three observations: (1) the reduced search space is two dimensional, which easily allows to reach a solution, (2) the search is repeated after each time interval and (3) velocities will stay constant unless new commands are given. [9]

Restrictions on the velocity are imposed when there is an obstacle in the closer environment of the robot. For example, when specific pairs of velocities could lead to a collision if followed by the robot. The space set of possible velocities (both linear and angular), V_s, can be divided into the pairs of velocities that lead to feasible trajectories and into the ones that lead to non-feasible trajectories. The part of the set that contains the velocities that lead to feasible trajectories is called the set of admissible velocities. A velocity is considered admissible if the robot is able to stop before it reaches the obstacle. Having this said, the set of admissible velocities V_a, is defined as

$$V_a = \{(v, w) \mid v \leq \sqrt{2\, dist(v, w)\, \dot{v_b}} \ \wedge \ w \leq \sqrt{2\, dist(v, w)\, \dot{w_b}}\} \tag{2.1}$$

where $dist(v, w)$ is the distance to the closest obstacle on the corresponding curvature defined by (v, w), and the pair $(\dot{v_b}, \dot{w_b})$ represents the accelerations of braking. The distance is obtained by the product of

the angle γ between the robot and obstacle with the radius r of the curvature:

$$dist(v, w) = \gamma \cdot r \tag{2.2}$$

The search space is then reduced to the dynamic window. This set only contains the velocities that can be reached within the next time interval. Considering t the time interval where the accelerations ($\dot{v}$ and $\dot{w}$) will be applied and (v_a, w_a) the actual velocity, the dynamic window can be defined as:

$$V_d = \{(v, w) \mid v \in [v_a - \dot{v}t,\ v_a + \dot{v}t] \ \wedge \ w \in [w_a - \dot{w}t,\ w_a + \dot{w}t]\} \tag{2.3}$$

This dynamic window is set around the center of the actual velocity and the extension of the set depends on the accelerations that are feasible. The curvatures outside the dynamic window can not be reached by the robot "immediately" and therefore are not considered.

The search space used in DWA after the restrictions imposed above can be obtained by

$$V_r = V_s \ \cap V_a \ \cap V_d \tag{2.4}$$

After having determined the resulting search space from Equation (2.4), an optimal velocity is selected. In order to achieve this, the objective function is defined as

$$G(v, w) = \sigma\ (\alpha \cdot heading(v, w) + \beta \cdot clearance(v, w) + \gamma \cdot velocity(v, w)) \tag{2.5}$$

and the pair (v, w), that achieves the maximum value of $G(v, w)$ is chosen as the optimal. The objective function takes three main criteria into account: the heading, clearance and velocity. We now take a deeper look into these criteria.

Target Heading

This criterion is a measure of progress towards the goal location (maximal if the robot moves directly towards the target). The target $heading(v, w)$ measures the alignment of the robot with the target direction. It is obtained by the angle of the target point relative to the robot's heading direction. Due to the change of this direction with different velocities, the target heading θ is computed for a predicted position as shown below:

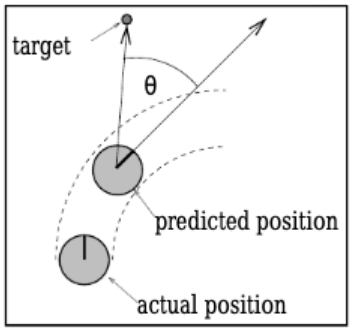

Figure 2.1: Target heading [9]

The predicted position is determined by assuming that the robot moves with the selected velocity in the next time interval and considering the kinematic dynamics, the target heading is computed at the position that is reachable by the robot when a maximal deceleration is performed after the next interval. [9] This results in a smooth turning to the target when the robot avoids an obstacle.

Clearance

$Clearance(v, w)$ is the distance to the closest obstacle on the trajectory, or in other words, the closest obstacle that intersects with the curvature. The value of clearance is set to a large constant if no obstacle is present.

Velocity

The $velocity(v, w)$ function has the role of evaluating the progress of the robot in the trajectory. It can be described as the forward velocity of the robot and is simply a projection on the translational velocity v.

The values of all three criteria are then normalized to $[0, 1]$ and the objective function is computed. Depending on the values of α, β and γ of 2.5 these criteria can have different influences on the objective, but all of them are equally crucial. If only the clearance and the velocity were maximized, the robot would travel into free space but not towards a goal. On the other hand, if heading was the only criteria taken into account, the robot would get stopped by the first encountered obstacle and would not be able to move away from it. In the last years, some improvements to this local planner have been suggested. More recently, and worth mentioning, in [10] some problems that arise from the use of DWA are mentioned. The main issues are the ones that come when dense objects appear on the robot's global planning path. When this is the case, the robot is easily trapped. The robot is incapable to judge the density of the environment and therefore needs more turning, acceleration and deceleration operations to get out of trouble. This lack of perception in the density of objects will stop the robot from surrounding the dense area and instead the robot will enter it. The native DWA explained above, only considers

factors such as heading angle, target distance and speed. It does not consider the density of objects in the environment, which will inevitably lead to the robot being easily trapped and therefore, very time-consuming. Considering this, the papers suggests an improvement by adding a density evaluation factor to the evaluation function. With this feature it is possible to evaluate if a certain obstacle is in the dense area and solve the problem that the local trajectory planning algorithm DWA can not avoid dense objects in advance. The density factor is responsible to predict the dense area based on the update of the information retrieved by the sensor, pre-perceive the distribution of dense objects and reduce the number of times the robot falls into dense areas. According to the results shown in [10], this improvement results in a more efficient DWA and can be widely used in robots that perform their tasks in dense environments.

2.1.2 Pure Pursuit

Pure Pursuit is a well-known planner that follows a given path, thus it is usually denominated as a geometric path tracking method. The Pure Pursuit algorithm calculates the angular velocity to reach a target point using the current position, linear velocity of a robot and the curvature passing the target point. [11] One of the first detailed descriptions of the implementation of the Pure Pursuit algorithm was described in [12].

Geometry Deviation

The Pure Pursuit approach enables the determination of the curvature that will drive the robot to a goal point.

After retrieving the initial position, a search for the target waypoint or goal point as previously mentioned, is performed. The target waypoint is a point that lies on the path and on a given search circle. The radius of this circle is set as a parameter, the lookahead distance. After this, an arc that joins the current point and the goal point is constructed and the chord length of this arc is the lookahead distance. This acts as constraint so that it is possible to determine a unique arc that joins the two points.

In the Figure below we can see the point (x, y), which is one lookahead distance, l from the origin. The point (x, y) is constrained to be on the path. The objective is to calculate the curvature of the arc that joins the origin to (x, y) and whose chord length is l.

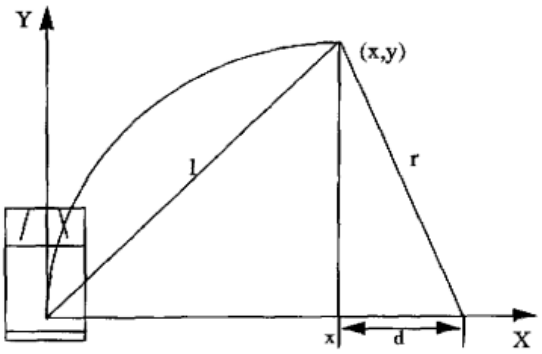

Figure 2.2: Illustration of the Pure Pursuit algorithm [12]

The next step is the calculation of the curvature of the circle passing the vehicle position and the target waypoint. Considering the Figure (2.2), the curvature k is obtained by

$$\begin{aligned} x + d &= r \\ x^2 + y^2 &= l^2 \end{aligned} \tag{2.6}$$

The first equation from the ones above describes the circle of radius l. This area includes the possible locations of goal points. The second one states that the radius of the arc and the x offset are independent and differ by d.

From above and with the goal of relating the curvature of the arc to the lookahead distance, the following equations hold

$$\begin{aligned} d &= r - x \\ y^2 + (r - x)^2 &= r^2 \\ x^2 + r^2 - 2rx + y^2 &= r^2 \\ l^2 &= 2rx \\ r &= \frac{l^2}{2x} \\ k &= \frac{2x}{l^2} \end{aligned} \tag{2.7}$$

Implementation

The pure pursuit algorithm is responsible to produce the steering angle required to bring the vehicle back to the reference path. The classic frame that describes this algorithm can be seen below:

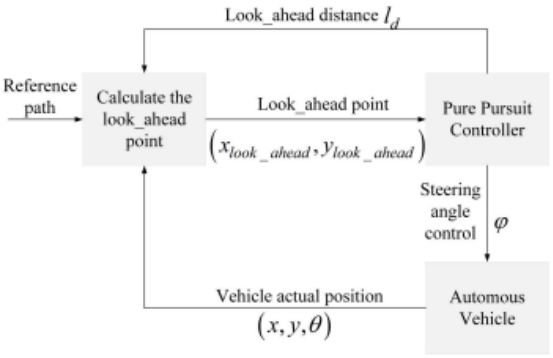

Figure 2.3: Pure Pursuit algorithm frame [13]

The first step of the algorithm is to determine the current location that consists of a two dimensional position and the orientation. After retrieving this, the next step is to find the path point closest to the vehicle. As mentioned before, the goal point should be within one lookahead distance of the vehicle. However it is possible that multiple points fall in this category. If this is the case the vehicle should steer towards the closest point one lookahead distance from its current location which commences up the path.

The next step is to find the goal point. For this, the planner starts by moving up the path and calculating the distance between that path point and the vehicle's current location. The moment this goal point has been found, it has to be transformed to the vehicle's local coordinates since path point locations are recorded in the global frame. By using the Equation 2.6, the curvature is calculated and it is transformed into steering wheel angle by the controller. At last, the position of the vehicle is updated.

To better understand how the steering angle and the rest of the variables are obtained Figure 2.4 shows a schematic representation of the algorithm:

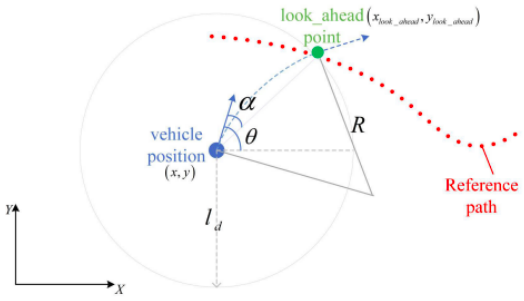

Figure 2.4: Pure Pursuit algorithm schematic [13]

First, the angle α is obtained:

$$\alpha = \left| \theta - \arctan\left(\frac{y_{lookahead} - y}{x_{lookahead} - x} \right) \right| \tag{2.8}$$

The angle α is the angle between the body of the robot and the line that connects the lookahead point and the rear wheel position of the robot. In the above expression, $x_{lookahead}$ and $y_{lookahead}$ describe the position of the lookahead point corresponding to the robot position, x and y describe the position of the robot and θ is the heading of the vehicle.

Another variable that can be retrieved with geometric manipulation is the radius of curvature R that the vehicle needs to follow:

$$R = \frac{l_d}{2\sin\alpha}, \tag{2.9}$$

where l_d is the lookahead distance.

By using an Ackermann geometric model for the robot, the steering angle can be obtained by:

$$\phi = \arctan\left(\frac{2L\sin\alpha}{l_d}\right), \tag{2.10}$$

where L is the wheelbase of the vehicle.

2.2 Benchmarking

By allowing repeatable measurements to quantify improvements, benchmarking has been playing a key role in providing improvements in multiple fields. A good example of the previous statement is the existence of benchmarks in the industry - vendors in those have the highest level of interest in developing credible benchmarks. [14] Without the evaluations tools, it would be impossible for the vendors to validate system comparisons during the development process of new products. It would also be impossible to gain recognition from the experts in the area or from the public for significant technology advances. The importance of benchmarking in industry is so high that there are even organizational structures that are responsible for creating and maintaining reliable tests.

Moving to scientific areas, benchmarking tools play a key role in multiple fields, for example in computer vision, as mentioned in [15].

The situation in the field of robotics is completely different from the ones presented above. Currently, comparative research in robotics has focused majorly on competitions between systems which have shortcomings in the ability to advance research. [16]

As robots become more complex and moving into more complex environments, the evaluation process becomes more difficult. Measuring progress can be hard in any discipline. However, in the field of robotics this evaluation seems to face particular difficulties.

Nowadays, robots are systems with a high degree of complexity. The interaction between components, both hardware and software, properties, environment and the whole system performance is non trivial and hard to isolate. The hardware and software used in research are, most of the times, unique,

which makes it even harder to replicate and compare the obtained results, or to identify which part of the system contributes to make progress in a certain task. [17]

In [18], benchmarks are described as “precisely defined, standardized tasks”. This definition includes three aspects of benchmarks: task - the robot has to perform a given mission; standard - the benchmark is accepted by a significant set of experts in the field; precise definition - the task is described exactly, especially the execution environment, the mission goal and limiting constraints. [14] It is also necessary to include one important feature: a numerical evaluation of the performance.

Benchmarking is also tightly connected to replicability, implying reproducibility and repeatability of experiments. [19] Reproducibility can be defined as the possibility to verify the results of a certain experiment in an independent way; repeatability underlines the fact that obtaining a single result is not enough to ensure the success of experiments and that multiple repetitions are needed for statistical significance. [19]

Another interesting point, from [19], is the split of definition between benchmarks, being defined for tasks and functionalities. The first one takes the context of an experiment and evaluates the performance of the overall execution of a task while the last evaluates the performance of a functionality as the best performance among the remaining options that provide the functionality.

In sum, benchmarking involves measuring and comparison as well as a quantification of results. Additional features can include a widespread acceptance with possible use by a relevant group, a relation to reality and applicability to problems of the real world. [14]

3

State of the Art

In this chapter, we present the state of the art in benchmarking mobile robot guidance.

Benchmarking local planners is non-trivial for several reasons. First, since the planners rely on sampling, the performance can not be evaluated from a single run. Instead, benchmarks need to be run multiple times to obtain a distribution of the chosen performance metric. Besides this, the optimization for different metrics depends on the application and there is no universal metric to assess the performance of local planner algorithms across all benchmarks. [20] Despite this, some general infrastructures for comparing different planning algorithms have been proposed in order to overcome the problem of scarce literature in this area [21]. Here we single out some that motivated the methods proposed in the thesis.

As one of the first references to benchmarking in robotics, [22] introduces a method for measuring odometry errors of mobile robots, focusing on a quantitative evaluation of systematic and non systematic errors. In [23] an early benchmark for robot navigation is presented. However, the way to evaluate the robot performance in a changing environment is not taken into account.

By considering a static environment, [24] presents an evaluation of two specific robot tasks, namely path planning and obstacle avoidance.

One of the most used approaches consists of the evaluation of performance taking the results within the same environment into consideration - namely in the same moment with the same constraints. [2] This is the case of robotic competitions. Competitions, such as the DARPA Grand Challenges and RoboCup can be considered as an option. Inside the RoboCup initiative, we find RoboCup@Home interesting to mention, this competition defines a live competition of service robots that need to fulfill a series of tests in a domestic environment. [25] Instead of defining a final challenge, this competition is defined as a benchmarking process that guides the system development from simple tasks towards successively more complex tasks in realistic environments. [25]

Competitions like the ones mentioned above provide an opportunity for researchers to benchmark different approaches and understand the relative advantages of each one, considering a defined measure of success. However, competitions lack the possibility to repeat an experiment with different methods. [26]

With the aim of developing competitions that come close to scientific experiments while providing an objective performance evaluation of robot systems, the Robot Competitions Kick Innovation in Cognitive Systems and Robotics (RoCKIn) project is born. [27] This new approach aims to tackle the issue that competitions provide benchmarking at the system-level based on a single high-level measure by moving to more sophisticated benchmarking activities, while retaining some of the traditional values of competitions as producing a rank among alternative solutions at competition time, assigning prizes and awards to the best teams and push for progress. By using the results of the first RoCKIn competition, the authors were able to show how this project can provide a set of tools to enable the replicability of experiments involving autonomous robots.

In the last few years, with the awareness of how important benchmarking is to objectively compare different solutions, other approaches have been proposed. One example can be the one proposed in [28], it addresses the problem of evaluating automated guided vehicles with different degrees of autonomy by defining benchmark tools to grade the performance of each approach.

The benchmark methodology described in [28] includes the definition of different types of scenarios, including wide and narrow corridors, small and large isles, zig-zag maneuvering and scenarios with moving objects (other vehicles or people). For each scenario described above, a set of pairs with start or goal positions are also defined. Another relevant defined set is the one containing the maximum speeds, that the vehicle can move when the driver fully accelerates. At last, random tests are generated with combinations of start/goal positions and maximum speeds. The number of tests for each algorithm is given by:

$$N = \frac{n!\, m}{2!(n-2)!} \tag{3.1}$$

where n is the dimension of the set of pairs start/goal positions and m is the dimension of the set of maximum speeds. [28]

Crucial aspects like robustness, efficiency, safety and comfortability are also evaluated. Robustness is measured by the number of experiments in which the vehicle reaches the goal without any collision, efficiency as a measure of mean time, velocity or mean path length. Safety is measured in terms of average minimum distance to obstacles and risk of collisions, comfortability in terms of the type of the described trajectory and variations of acceleration (jerk).

More recently, in [29] a complete and principled evaluation framework for comparing the performance of local planners is established. To achieve this, multiple simulation scenarios were developed and different types of evaluation metrics were proposed.

Indoor scenarios were used to make an overall evaluation of the local planners. This category includes large-scale-office- and small-scale-family-house-like scenarios. Narrow space scenarios (like U-shaped or Z-shaped corridors) were designed to challenge the flexibility and smoothness of the local planners. In the previous categories the complete prior map is given as the input to local planners. To test the adaptability, the authors blurred the maps individually and input the incomplete map to the local planners.

At last, the robustness of local planners is challenged in dynamic scenarios that include simulations where only two people are walking, but also a more complex one where six people are walking around in an open space environment. In this last one, the robot needs to pass through the crowd to reach the goal at the other, reproducing a crowded scene in the real world. This type of scenarios challenges the safety, flexibility and real-time performance of local planners.

The evaluation of the performance of the local planner is done from different aspects by using different metrics.

Safety metrics are used to evaluate the security performance of the local planner. These metrics can be evaluated in terms of minimum distance to the closest obstacle and percentage of time spent by the robot in the surrounding area of the obstacles.

In [29], d_o is defined as the minimum distance to the closest obstacle and p_o as the percentage of time spent by the robot in the area around obstacles, and the safety metric is expressed as:

$$d_o = min\{d_i\}, 1 \leq i \leq n \tag{3.2}$$

$$p_o = \frac{\sum(t_b - t_a)}{t_n - t_1} \times 100\% \tag{3.3}$$

where a and b represent the indices of the timestamps that satisfy $d_k \leq d_{safe}, a \leq k \leq b$, where d_k is the distance to the closest obstacle for $i = k$ and d_{safe} is the preset safe distance to obstacles.

Another type of metrics is presented: efficiency metrics. These are used to evaluate motion efficiency, described by how quickly the local planner guides the robot to the local, and computational efficiency, which evaluates the real-time performance of local planners.

The motion efficiency uses the total travel time, T:

$$T = t_n - t_1 \tag{3.4}$$

And the computational efficiency is given by the average time consumption of a single local planner iteration:

$$C = \frac{1}{n}\sum_{i=1}^{n} c_i \tag{3.5}$$

Lastly, [29] also refers to smoothness metrics that are used to assess the quality of motion commands given by the local planners. The overall smoothness performance is evaluated by the path and velocity smoothness. The path smoothness is defined as:

$$fps = \sum_{i=2}^{N-1} ||\Delta x_{i+1} - \Delta x_i||^2 \tag{3.6}$$

where $\Delta x_i = x_i - x_{i-1}$ represents the displacement vector at $x_i = (x_i, y_i)^T$. The velocity smoothness is measured by computing the average of acceleration:

$$fvs = \frac{1}{n-1}\sum_{i=1}^{n-1} \left| \frac{v_{i+1} - v_i}{t_{i+1} - t_i} \right| \tag{3.7}$$

4

Proposed Method

Contents

4.1 Theoretical Framework 25
4.2 Metrics 27

This chapter presents the method used in this thesis. First, the theoretical framework is explained. Then we describe the parameters that were modified to achieve our goals. At last, we explain the metrics used to evaluate the performance.

4.1 Theoretical Framework

As previously mentioned, our goal with this thesis is to establish a principled evaluation framework that allows an objective comparison between the performance of local planners, particularly those mentioned in Chapter 2.

The application of the benchmark is done in open source algorithms of the previously mentioned planners. We start by presenting a description of those algorithms.

4.1.1 Local Planners

DWB

In Robot Operating System (ROS) 1, the navigation repository provides an implementation of the DWA algorithm, explained in 2.1.1. The ROS 2 implementation rewrites and extends the functionality of the previous one. In order to differentiate both, the ROS 2 version of this algorithm is called DWB.

In a simplified way, the goal of a local planner is to provide the command of velocities that will make the robot move towards the goal, by taking the global planner and local costmap into account. DWA achieves this via sampling, which means that it generates feasible velocity commands and evaluates them according to various metrics and selects the one with the best score, until the robot reaches the goal.

The changes from the previous version to the current one are based on the form of the evaluation. Assuming we are evaluating a given command to see if it collides with any obstacles in the costmap, the key question, according to the author of the package, is where the robot will drive using the command. In order to achieve this, the position, velocity and the kinematics of the robot need to be known. The evaluation is then done by considering the trajectory which contains not only the velocity, but an array of multiple sample poses that are more likely to be reached by the robot, and not only the isolated velocities. [30]

Regulated Pure Pursuit

This package implements a variant on the Pure Pursuit algorithm, mentioned in 2.1.2, to track a path. It is called Regulated Pure Pursuit (RPP) due to its additional regulation terms on collision and linear speed. [31]

The RPP implements a variation on the original Pure Pursuit that makes it possible to regulate linear velocities in high curvature turns and help reduce overshoot at high speeds. It implements heuristics so the robot can slow in proximity to other obstacles and nearby potential collisions. This algorithm also implements an adaptive lookahead point feature that allow velocities to be scaled in order to enable more stable behavior in a larger range of translational speeds. [31]

4.1.2 Parameters

The configurations for the planners have a multitude of parameters, from dynamical constraints of the robot to plugins used to check if the goal was achieved or not.

When it comes to the parameters to use as variables for the benchmarking data analysis a few considerations are worth having. The values used were the default ones that can be consulted in [32] with the exception of the ones that are common to both planners. Those were set to equal values to obtain fairest comparisons. Those parameters can be seen in the following table:

Table 4.1: Common parameters between the local planners used

Parameter	f	a_{max}	a_{min}	v_{max}	v_{min}	θ_{max}	θ_{min}
Value	20	2.5	-2.5	0.26	0	1.0	0
Unit	Hz	m/s^2	m/s^2	m/s	m/s	rad/s	rad/s

Wherein a_{max} and a_{min} represent, respectively, the maximum linear acceleration and maximum linear deceleration; v_{max} and v_{min} represent, respectively, the maximum and minimum linear velocity; θ_{max} and θ_{min} represent, respectively, the maximum and minimum angular velocity and f is the frequency chosen to run the planner.

In the process of the robot navigation, from the start to the goal pose, the local planner is called n times. In order to be able to evaluate the performance of the local planners later on, we need to save data every time the local planner is called.

The local planners are then tested by choosing different target positions in different scenarios. These scenarios are simulated environments with different characteristics, some are simulated office-like rooms and others are simulated rooms with a large area of free space. This variety is also key for the evaluation of the local planner performance since robots can be used in a variety of rooms and it is crucial to choose the planner accordingly to this.

When the robot completes the navigation task and using the scripts explained in Chapter 5, the following intermediate data of the whole navigation process is saved: timestamp of the moment where the target pose is accepted, t_0, timestamp of the moment where the target pose is reached, t_n and the robot position (x_i, y_i), for $1 \leq i \leq n$. This data is then used to compute metrics that will evaluate the performance of the local planners, as we describe in the following section.

4.2 Metrics

Metrics serve as the evaluation needed to assess the performance of our benchmarking objects. In our case, we use metrics as tools to evaluate the performance of the local planners. In order to do this, different metrics were chosen. The first metric used to assess the performance was the number of times that the robot reaches the target pose and in which scenarios this happens, more or less frequently. This metric is an important tool to evaluate local planners, the results presented in this thesis can help other users to choose the local planner accordingly to the type of scenario in which they are going to use their robot.

The execution time was used as a metric and can be described as the time that goes from the moment when the target pose is accepted, to the time where is actually reached or not, like shown below:

$$T = t_n - t_0, \tag{4.1}$$

where t_n represents the timestamp of the call in which the robot reaches the target pose (or not) and t_0 represents the timestamp of the call in which the target pose is accepted.

This metric was used in different situations, the first one being the total execution time independently of the fact that the target pose is reached or not and then only for the situations where the target pose is reached. Lastly, only for the situations where the target pose is not reached.

This can be useful to users so that they can choose the local planner considering their goals. If the main goal is to reach a certain target pose within a certain time limit, this evaluation will help them in making the choice of the local planner.

Besides the execution time, another metric that was used to evaluate the performance was the path length. This is done by splitting the path into small segments and considering the general length of one segment. Since dx and dy are infinitesimally small, the length dL can be approximated using the Euclidean Distance. By summing all these consecutive segments of the curve, we get the integral:

$$L = \int_{t_0}^{t_n} \sqrt{\left(\frac{dx}{dt}\right)^2 + \left(\frac{dy}{dt}\right)^2}\, dt \tag{4.2}$$

Like mentioned before for the execution time, this metric was also assessed in total and in two different cases, one when the target pose is reached and one when it is not.

5

Experimental Implementation

Contents

5.1 ROS Components 31
5.2 Coordinate Frames 33
5.3 Benchmarking Framework 34
5.4 Robot Model 35
5.5 Simulation 36
5.6 Scenarios 37

This chapter describes our approach. First, we present the packages used for the work developed. Lastly, the robot models and scenarios used in the simulations are presented.

5.1 ROS Components

The ROS is a set of open source software libraries and tools for building robot applications. Since the start of ROS in 2007, there have been lots of changes in robotics and in the ROS community. With this in mind and with the goal of adapting to those changes, ROS 2 project was launched, trying to leverage what is great about ROS 1 and improving what is not. [33] The work developed during this thesis was done using ROS 2 and, in specific, the Foxy Fitzroy distribution. [34]

Currently, the most used navigation package on ROS 2 framework is Navigation 2 (Nav2). [32] Nav2 uses behavior trees to call modular servers to complete an action. Examples of actions are the computation of a path, control effort, recovery or any other navigation related action. Each of these is a separate node that communicate with the Behavior Tree (BT) over a ROS action server. The expected inputs to Nav2 are a map source, BT XML file and any relevant sensor data sources. The package will then provide valid velocity commands for the robot to follow.

The Nav2 package includes several others by default such as:

- map_server: responsible to supply the map data to the navigation stack while being able to make changes to the current map;
- global_planner: defines a path planner from the location of the robot to the desired goal. It takes the current occupancy grid map and costmaps to avoid static obstacles into account;
- local_planner: creates a trajectory from the plan generated from the global planner that allows the robot to safely navigate without hitting obstacles towards the goal;
- costmap_2d: handles all the costmaps. The costmaps are occupancy grids produced from sensor data and from the map. It also generates the inflated cost area based on the parameters set in the configuration file.

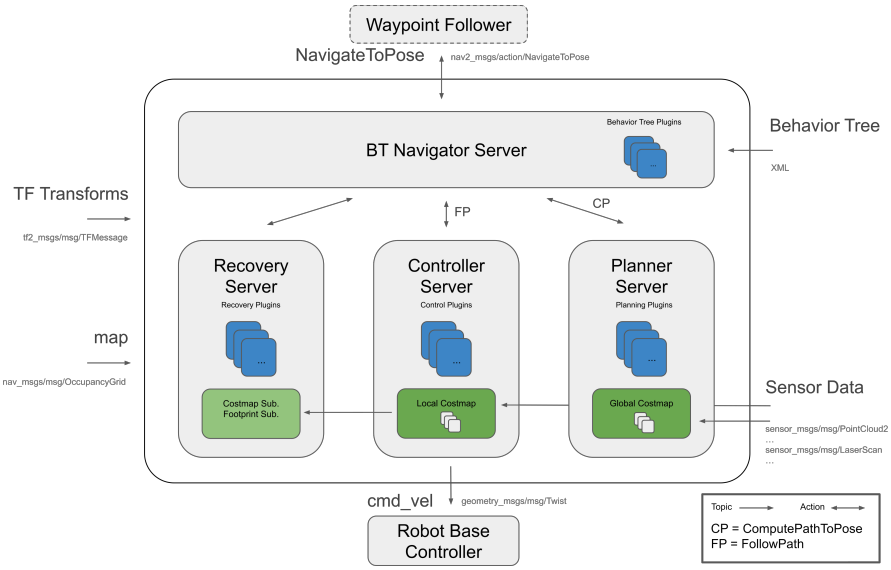

Figure 5.1: Navigation architecture diagram [32]

Another ROS component worth mentioning is the graph, which is a network of elements processing data together at one time. This graph includes nodes and each of it is responsible for a single, module purpose. Each node can send and receive data to other nodes via topics, services, actions or parameters. [35] The graph of our network is easily accessible. However the high number of nodes that exist would make the comprehension of it highly difficult. The following diagram shows a simplified version of our graph:

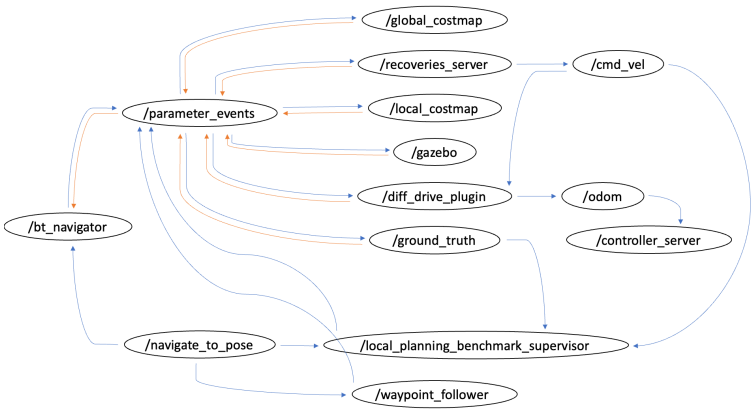

Figure 5.2: ROS graph with the nodes used in our implementation

The BT navigator receives a goal pose and navigates the robot to the specified destination. To do so, the module reads an XML description of the BT from a file as specified by a node parameter. In our case this description is navigate to pose. It navigates from a starting point to a single point goal. The waypoint follower module implements a waypoint following method using the navigate to pose action server.

Each node will provide a topic on which parameter events will be published. This topic monitors parameters for changes.

The costmap is responsible for building a 2D costmap of the environment. This costmap consists of several ”layers” of data about the environment and those are updated by the observations from the sensors. A plugin can be added to allow the layers to be inflated via an inflation radius based on the robot footprint.

The recoveries are used to get the robot out of certain situations, like being too close to a wall.

In Nav2, local planners are denominated as controllers (in previous versions of ROS they were denominated simply as local planners) and are the way to follow the computed global path or complete a local task. [32] The controller will attempt to compute feasible control efforts based on the access it has to the local environment representation through the sensor.

The ground truth node is responsible to retrieve all the poses of the robot during the whole navigation process. The local planning benchmark supervisor node is the one that calls the functions described in 5.3. The connection between these nodes is important since we will use the data from the ground truth node in our benchmark framework to calculate the chosen metrics.

5.2 Coordinate Frames

There are many components in a robotic system. Those components move in relation to each other and it is crucial to know the position of each component relative to the others. This is important since there are lots of measurements that are taken in various frames and need to be transformed to a common frame to combine those measurements. ROS contains a library to tackle this problem, the ROS Transform Library (tf).

The main purpose of this library is to connect every frame of the system in a tree graph structure where each frame is a node and the transformation between two frames is represented as edges of the graph.

The base tf tree used in our implementation can be seen below:

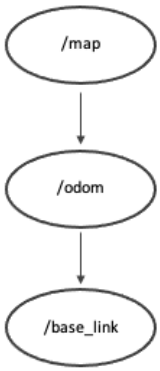

Figure 5.3: Base tf tree implemented

The robot position is measured relative to a global reference frame, the map frame. The odom frame originates where the robot starts at. The base link frame is defined as the center of the robot's rear axle and moves with the robot. The transform between frames is crucial, for example if an obstacle is detected in the base link, it is important to know where it is in the map.

5.3 Benchmarking Framework

The benchmarking framework used in this thesis relies on multiple scripts responsible for a certain number of tasks, from selecting a configuration to shutdown the process when the benchmarking is concluded. The scripts used in this thesis are a modified version from the ones available at [36]. In this case the scripts were used to benchmark localization. However, in our case our goal is to benchmark local planners and for this reason some modifications were required. In this section, we will give an overview of the used scripts and explain why certain changes were done in order to achieve our goals.

The main script, labeled as supervisor, is responsible to assure the consecutive steps of the benchmarking. In a simplified way, we can explain it as the script responsible for starting the navigation stack, which will create a path between two poses. If the path is feasible, the supervisor will then finish and save the relevant data in files that will then be used to analyze the performance of the local planners.

The other script, the execute script, contains all the specifications used in the benchmarking. With this, it is possible to choose which scenario and local planner will be used in a certain simulation among others. At last, the script denominated as run, is responsible to create the files that are going to be used by the navigation stack and then for the ones related to ROS, according to the specifications present in the execute script. When the benchmarking is concluded, it shuts down.

The main modifications performed in the benchmark framework that served as model for the one used in our work are made at the supervisor script. In the initial script, a traversing path was created taking the cost of each node into account. In our case, we simply use a list of all the vertices and we

randomly choose one as our target pose. Another important modification is the change of behavior from follow path to navigate to pose, with follow path the entire path was taken and followed by the planner. In the current implementation we used navigate to pose that navigates from a starting point to a single goal point.

5.4 Robot Model

Many mobile robots use a drive mechanism known as differential drive. This mechanism consists of two drive wheels mounted on a common axis, and each wheel can be independently driven either forwards or backwards. Although the velocity of each wheel can vary, for the robot to perform rolling motion it must rotate around a point that lies along their common left and right wheel axis. This point is known as instantaneous center of curvature.

Considering w as the rate of rotation, that is the same around that point for both wheels; R the distance from the instantaneous center of curvature to the middle point between wheels; l the distance between the centers of the two wheels and V_r, V_l, respectively, the right and left wheel velocities along the ground, the following can be stated for any time t:

$$\begin{aligned} R &= \frac{l}{2}\frac{V_l + V_r}{V_r - V_l} \\ w &= \frac{V_r - V_l}{l} \end{aligned} \tag{5.1}$$

It is interesting to note that the differential drive robot can not move in the direction along the axis, with this being called a singularity. Slight changes in the velocity of each wheels can have an enormous effect on the motion of differential drive vehicles. Small errors in the relative velocities between the wheels can also effect the robot trajectory. [37]

The model used in our experiments, that is an example of a differential drive, was the Turtlebot3 Waffle and the description of the robot that was used was the one available in [32]. The TurtleBot3 Waffle is a small and low cost mobile robot, some of the specifications can be seen in the following figure.

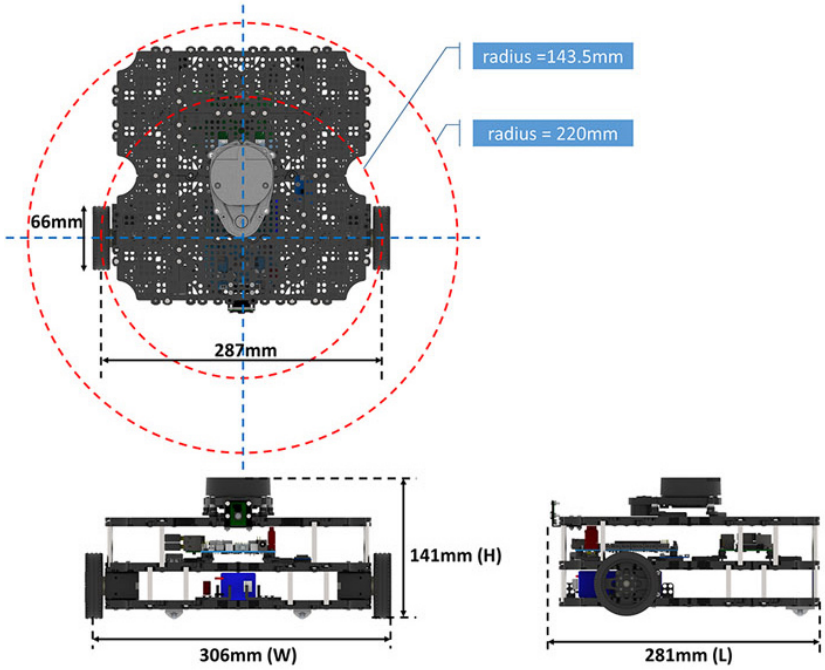

Figure 5.4: Specifications of the Turtlebot3 Waffle model [38]

5.5 Simulation

As mentioned before, ROS is used as the interface for the robot. This interface combined with a simulator results in a very useful robot simulator. The simulator used in this work was Gazebo, due to its popularity among the ROS community. Another interesting note is the fact that Gazebo can use high-performance physics engines and provide realistic renderings of environments including high-quality lighting, shadows and textures. [39]

The implementation of the simulated robot is done by implementing sensors and physical parts, for example the wheels, connected by links and joints. The robot is designed through the XML macros language Xacro. The footprint of the robot is set to a circle with a radius of 0.22m. On this basis, a lidar sensor is mounted on the robot. The sensor has a maximum and minimum range of $3.5m$ and $0.12m$, respectively, and an update rate of $5Hz$.

It is important to note that this design is taking the real dimensions of the robot into account so that the simulated robot is as similar as the real one which is crucial in the case of collecting performance data from a real robot instead of a simulated one. In addition, the robot pose is obtained from the ground truth provided by Gazebo to avoid the influence of localization errors.

Besides this, different simulated scenarios are used as well. These scenarios represent different types of environments where a robot might be used, from open spaces to office-like rooms.

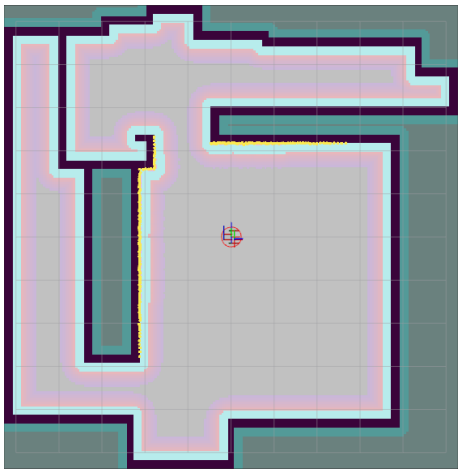

Figure 5.5: Simulator view with a simulated scenario and the simulated robot model

5.6 Scenarios

The regions in which the robot moves contain different types of obstacles, for example walls and corridors. Each of these can be represented as polygons. In order to find the Voronoi diagram for this set of polygons, it is possible to use an approximation and compute the diagram for a set of discrete points. A Voronoi diagram can be defined as the set of line segments separating different regions, where line segments are defined by the points that are equidistant from two data points and the vertices are points equidistant from more than two data points, as seen in the left figure in 5.6. In the same image on the right side we can see the Voronoi diagram in the context of path planning: find the point q^*_{start} of the Voronoi diagram closest to q_{start}, then find the point q^*_{goal} of the Voronoi diagram closest to q_{goal}, finally compute the shortest path from q^*_{start} to q^*_{goal} on the Voronoi diagram.

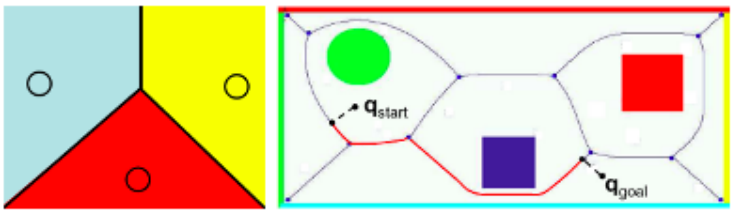

Figure 5.6: Left: Voronoi diagram - the set of line segments separating the regions corresponding to different colors. Right: Voronoi diagram in the context of path planning [40]

The boundaries of the obstacles can be approximated by a number of points that result from subdividing each side of the original polygon into small segments and subsequently compute the Voronoi diagram for this collection of approximated points. In the end, we are left with edges that form a good approximation of the generalized Voronoi diagram. In this diagram the start and target pose of the robot

are defined and the planner comes to action and defines the path. This method generates a route that for the most part remains equidistant between the obstacles closest to the robot and gives the robot a relatively safe path to travel along. [41] It is important to note that the start pose remains the same for all the simulations and the target pose is chosen randomly to avoid bias towards more easily accessible positions that could effect the evaluation of the local planner performance.

In order to achieve a complete and reliable benchmark test for local planners, it is necessary to use different scenarios with different characteristics to clearly evaluate the performance. Also, considering the application in the real world, it can be very interesting to know which local planner performs better in a certain scenario. In this thesis, some of the different scenarios used are the ones available in [36].

Airlab is a small scenario composed of small corridors on two sides but with an open area in the middle section. This environment can be used to understand the performance of the planner in a small environment.

7A-2 is an open-space area with wide hallways but with a few small rooms that can be accessed by small doors which can effect the performance of the planner.

Office_b is a typical office scenario made by many small rooms and small hallways. There are also small doors connecting the rooms.

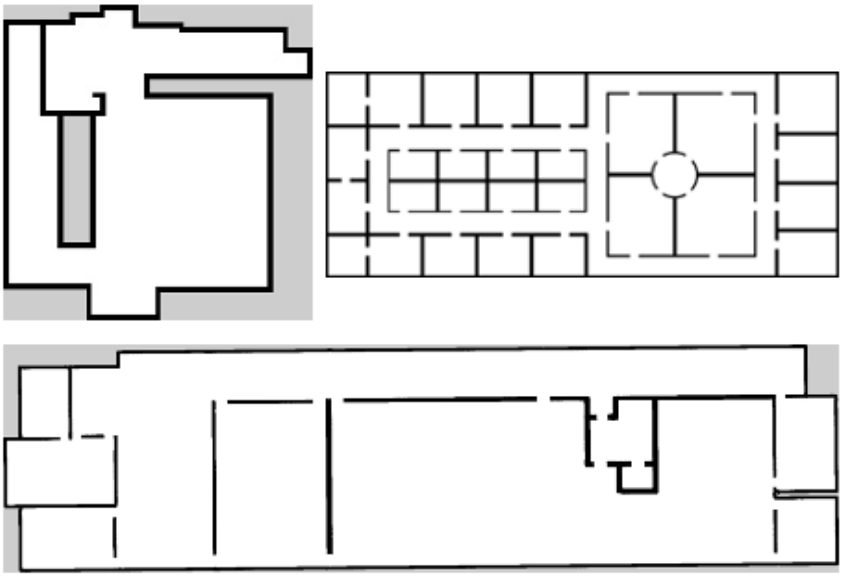

Figure 5.7: Scenarios used in the experiments. Top left: **Airlab** (size $11.7 \times 11.4m$); Top right: **Office_b** (size $60 \times 24.5m$); Bottom: **7A-2** (size $26.4 \times 90.4m$). Figures not to scale.

Intel has different types of rooms and hallways, there are also some obstacles along the way. Despite having a large area, the existence of obstacles can effect the overall performance.

Fr079 has one big hallway connecting multiple smaller rooms, some of those rooms are connected

either by small or unfeasible passages (due to the presence of obstacles between the rooms).

Mexico is a wide-open scenario with a large area of free space which facilitates the movement of the robot, despite this it also has rooms that contain obstacles.

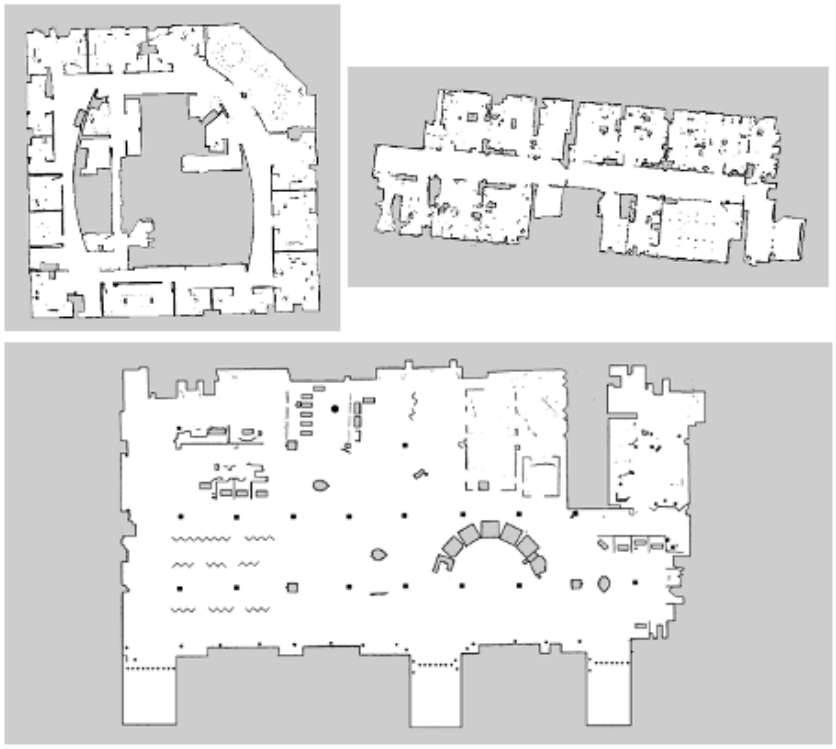

Figure 5.8: Scenarios used in the experiments. Top left: **Intel**; Top right: **Fr079**; Bottom: **Mexico**. Figures not to scale.

Turtlebot_world is a small and simple scenario in a hexagonal shape composed with small cylinder obstacles. This scenario, available in [32], was also used in the experiments:

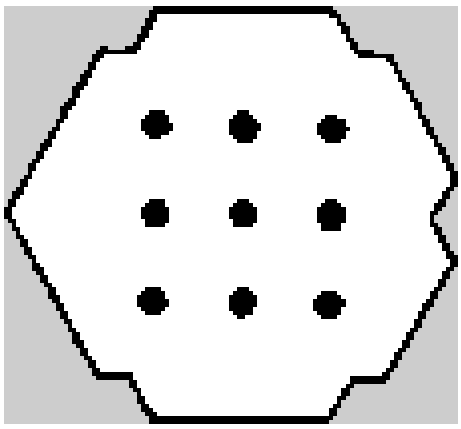

Figure 5.9: Scenario used in the experiments: **Turtlebot_world**. Figure not to scale.

6

Experimental Results

Contents

6.1 DWB 43

6.2 Regulated Pure Pursuit 44

6.3 Comparison Between Planners 45

In this chapter the experiments results obtained during this thesis are presented and discussed.

As mentioned in Chapter 5, for this benchmark we used different local planners, different scenarios and metrics were chosen to evaluate the performance. The main goal of this thesis is to benchmark local planners and having this in mind two different local planners were chosen. The use of different metrics and scenarios is key to assure a fair evaluation.

We now present the results obtained by using DWB and RPP. To obtain these results, 1000 runs were done using the benchmark framework mentioned in Chapter 5. [1]

6.1 DWB

In this section the results regarding DWB are presented. First, we start with the results concerning the percentage of times that the target pose was reached:

Table 6.1: Evaluation of DWB by percentage of times that the target pose is reached

Target Pose Reached	Percentage of Times
Yes	91.83
No	8.17

Considering the same metrics but taking the scenarios used in the runs into consideration, the following results are obtained:

Table 6.2: Evaluation of DWB by percentage of times that the target pose is reached in different scenarios

Scenario	Percentage of Times Target Pose is reached
7A-2	95.83
Turtlebot_world	100
Airlab	100
Intel	94.20
Office_b	100
Mexico	55.07
Fr079	95.89

In the next table the planner is evaluated in terms of average execution time and respectively standard deviation:

Table 6.3: Evaluation of DWB by average execution time and standard deviation

	Average Execution Time	Standard Deviation
Total	61.545	55.374
If Target Pose is Reached	63.206	47.403
If Target Pose is Not Reached	42.86	109.074

[1] some of the simulations can be seen here: https://youtube.com/playlist?list=PL0ljxiq3uuzVjuzg8eZeF0Usc_ZY_Rr3o

Now the planner is evaluated in terms of average path length and respectively standard deviation. The results are the following:

Table 6.4: Evaluation of DWB by average path length and standard deviation

	Average Path Length	Standard Deviation
Total	14.19	12.648
If Target Pose is Reached	14.845	12.016
If Target Pose is Not Reached	6.827	16.612

Having these results, we can conclude that the planner was successful in the vast majority of times with exceptions in bigger environments. However, it is also interesting to note that when the target pose is not reached, the average execution time and the average path length is considerably smaller which is a positive takeout since it means that when the path is not feasible the amount of wasted time is small.

6.2 Regulated Pure Pursuit

In this section the results regarding RPP are presented. Firstly, the results concerning the percentage of times that the target pose was reached are presented:

Table 6.5: Evaluation of RPP by percentage of times that the target pose is reached

Target Pose Reached	Percentage of Times
Yes	90.51
No	9.49

Considering the same metrics but taking into consideration the scenarios used in the runs the following results are obtained:

Table 6.6: Evaluation of RPP by percentage of times that the target pose is reached in different scenarios

Scenario	Percentage of Times Target Pose is reached
7A-2	96
Turtlebot_world	100
Airlab	100
Intel	93.06
Office_b	95.83
Mexico	58.33
Fr079	91.04

In the next table the planner is evaluated in terms of average execution time and standard deviation respectively:

Table 6.7: Evaluation of RPP by average execution time and standard deviation

	Average Execution Time	Standard Deviation
Total	36.967	63.117
If Target Pose is Reached	35.69	25.434
If Target Pose is Not Reached	49.135	188.751

Now, the planner is evaluated in terms of average path length and standard deviation respectively. The results are the following:

Table 6.8: Evaluation of RPP by average path length and standard deviation

	Average Path Length	Standard Deviation
Total	15.658	20.242
If Target Pose is Reached	15.897	12.657
If Target Pose is Not Reached	13.372	57.748

With these results, we can conclude that the planner was overall successful in reaching the target pose despite exceptions in the bigger environments with obstacles. However, in this case the average execution time when the target pose is not reached is larger than when it is reached which can be a disadvantage and can be seen as a waste of time. Regarding the average path length, the results for the total average path length and when the target pose is reached are similar to the ones obtained with DWB. However, when the target pose is not reached the average path length when RPP is used is considerably larger than when using DWB.

6.3 Comparison Between Planners

When comparing local planners, execution time and path length can be highly important. In a real life situation robots can be required to perform tasks where they need to be fast and also reach the target pose by following a short path. Having this in mind, we compared the behavior of both local planners in terms of average execution time and average path length in a general way, then considering only when the target pose is reached and lastly, when the target pose is not reached.

To better understand the difference in the performance of the local planners, the results presented in 6.1 and 6.2 are combined:

Table 6.9: Evaluation of DWB and RPP by percentage of times that the target pose is reached in different scenarios

Scenario	Percentage of Times Target Pose is reached	
	DWB	RPP
7A-2	95.83	96
Turtlebot_world	100	100
Airlab	100	100
Intel	94.20	93.06
Office_b	100	95.83
Mexico	55.07	58.33
Fr079	95.89	91.04

Table 6.10: Evaluation of DWB and RPP by average execution time and standard deviation

	Average Execution Time		Standard Deviation	
	DWB	RPP	DWB	RPP
Total	61.545	36.967	55.374	63.117
If Target Pose is Reached	63.206	35.69	47.403	25.434
If Target Pose is Not Reached	42.86	49.135	109.074	188.751

Table 6.11: Evaluation of DWB and RPP by average path length and standard deviation

	Average Path Length		Standard Deviation	
	DWB	RPP	DWB	RPP
Total	14.19	15.658	12.648	20.242
If Target Pose is Reached	14.845	15.897	12.016	12.657
If Target Pose is Not Reached	6.827	13.372	16.612	57.748

With these results, we can see that the percentage of times that the target pose is reached is the same for both planners in small environments such as Airlab and Turtlebot_world. The results obtained for the other scenarios are also similar, it is worth mentioning the difference for the Office_b - an office-like room, where DWB reaches the target pose every time and RPP only reaches it in 95.83% of the times. Regarding average execution time, RPP, in average, needs a smaller amount of time to reach the target pose which can be seen as an advantage. However, the execution time when the target pose is not reached is bigger than when it is reached. This can be problematic since the user will need more time until figuring out that the path is not feasible. Regarding average path length, the results are very similar for the measurements when the target pose is reached and in total. However and once again, the value is rather different when the target pose is not reached.

Besides the tables above we decided to present the results in the following way:

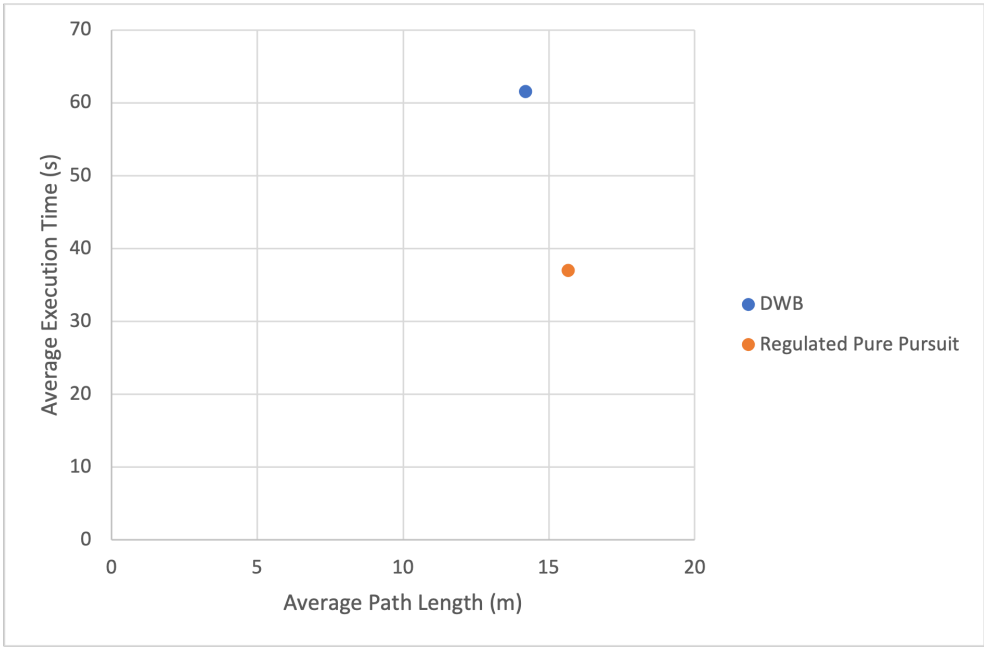

Figure 6.1: Total average execution time in function of total average path length for both planners

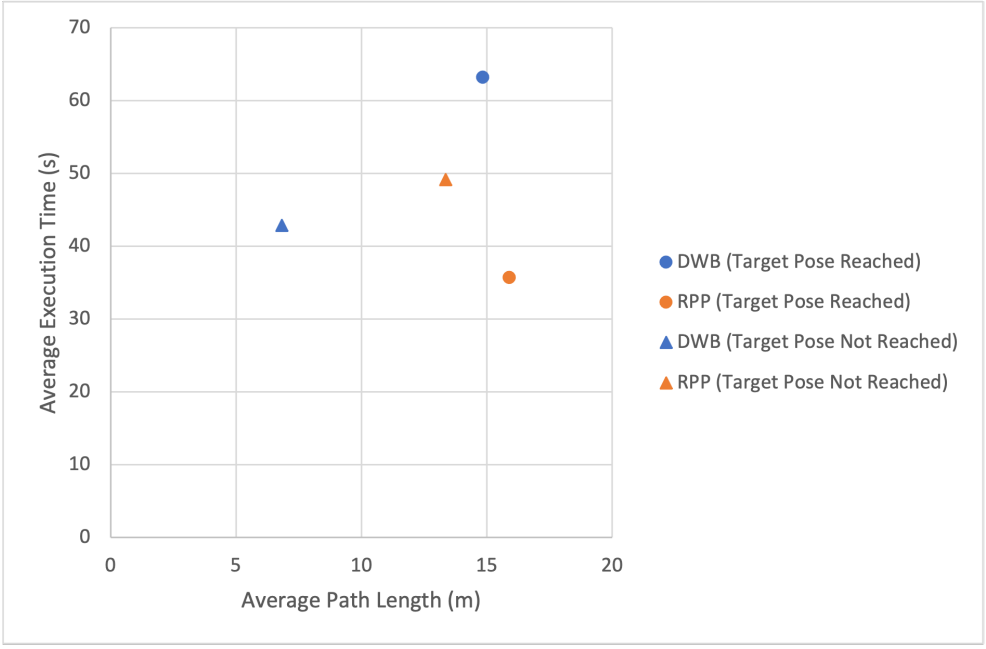

Figure 6.2: Average execution time in function of average path length for both planners, when the target pose is reached or not

Analyzing the results above, we can conclude that overall the average path length is slightly bigger when using RPP. The difference in average execution time between the planners is larger. In this case DWB has a much higher average execution time than RPP.

In the case where the target pose is reached, the results are the same as the above. The situation is rather different in the case where the target pose is not reached since RPP has a higher average execution time and the difference between average path lengths also increases with RPP having the largest one, once again.

After this analysis, we considered that it was important to know the relation between the average execution time and the different scenarios for both planners. The following results were obtained:

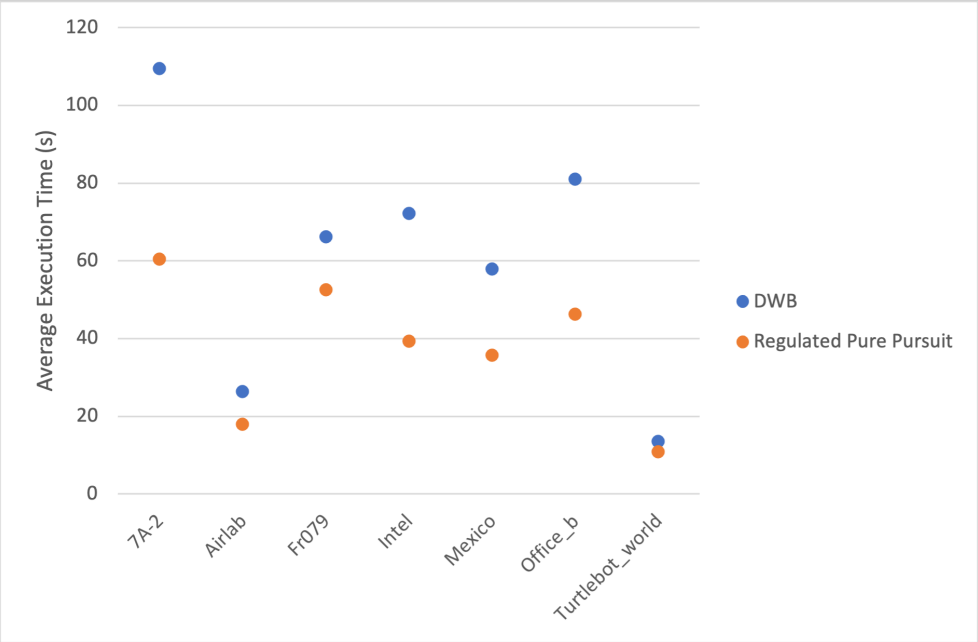

Figure 6.3: Relation between average execution time and the different scenarios for DWB

Considering the above, the average execution time is bigger for DWB overall. However, in small and relatively simple scenarios (like Airlab and Turtlebot_world) the results are very similar unlike the difference in more complex scenarios that is quite remarkable.

After this, we decided to do the same as previously presented but this time considering average path length. The results are the following:

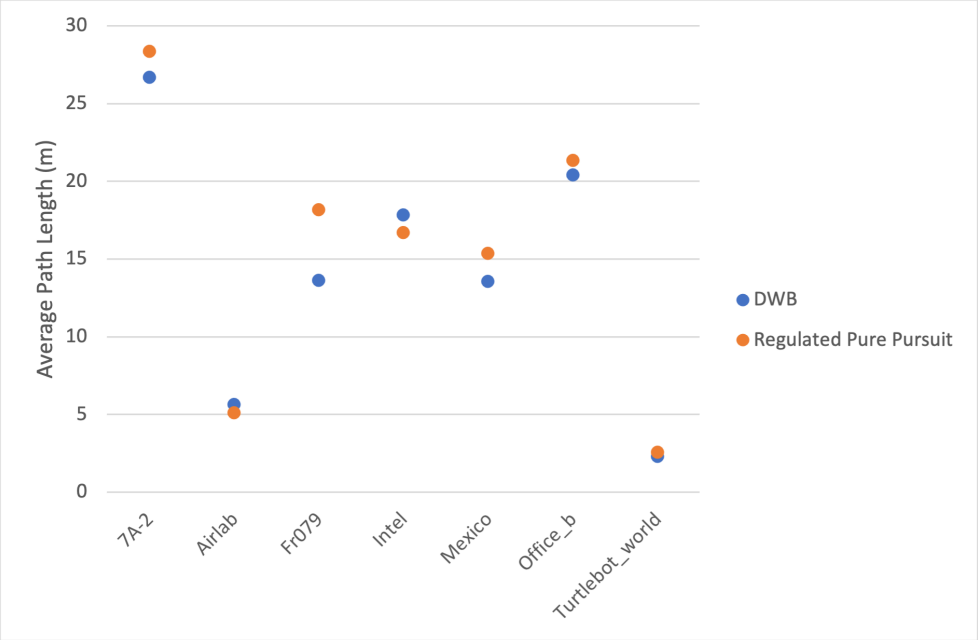

Figure 6.4: Relation between average path length and the different scenarios for DWB

The results were very similar in this case with the exception of the use in the larger scenarios (Mexico and Fr079) where the average path length obtained by using RPP was higher than the one achieved by using DWB.

7

Conclusions

Contents

7.1 Conclusions 53
7.2 Future Work 53

This chapter summarizes our work and suggests possible extensions.

7.1 Conclusions

Our goal with this work was - more than creating a method for absolute evaluation of local planners - to help researchers in the robotics field to choose their local planner accordingly to the situation where it is going to be used. We tested two local planners suitable for differential drive robots, one of the most commons drive mechanisms, and made simulations in different scenarios with different characteristics, from open-space to office-like rooms.

The comparison of the local planners was not possible if tools for evaluation were not used. For this reason, we implemented different metrics: average execution time, average path length and number of times that the pose is reached to be able to help the decision of when to use a certain local planner.

7.2 Future Work

As mentioned before, metrics are a crucial part in benchmarking so one simple task that deserves attention in the future is the implementation of more metrics, namely metrics that will also help in making an evaluation of the scenario used, for example minimum and mean passage width and the number of obstacles.

Regarding obstacles, one of the next steps could consist in setting up moving obstacles which would reflect another type of common scenarios where a robot could be used in a room with people moving.

Despite our effort in creating a simulated robot as close as possible to a real model, the use of the last would be also beneficial. Another suggestion for future work is the use of a different drive model, for example a car drive mechanism due to its high level of utilization.

Lastly, the number of times the simulation is run and the number of scenarios could be increased which will lead to a higher reliability in the results.

Bibliography

[1] Adithyavairavan Murali, Tao Chen, Kalyan Vasudev Alwala, Dhiraj Gandhi, Lerrel Pinto, Saurabh Gupta, Abhinav Gupta, "PyRobot: An Open-source Robotics Framework for Research and Benchmarking," *Carnegie Mellon University*, 2019.

[2] G. Antonelli, "Robotic research: are we applying the scientific method?" *Frontiers in Robotics and AI, Volume 2*, 2015.

[3] S. Johnson, *Where Good Ideas Come from: The Natural History of Innovation*. Penguin Books, 2010.

[4] Martin Hägele, Kai Pfeiffer, "RoSta Deliverable D 4.1: Report on State of the Art on Benchmarks for Mobile Manipulation and Service Robots," *Fraunhofer IPA*, 2007.

[5] R. Dillmann, "Benchmarks for Robotics Research," *European Robotics Network*, 2004.

[6] Diego Torricelli, Jose L. Pons, "Benchmarking: The Keystone of Robotics Innovation," 2019.

[7] K. Sugihara, J. Smith, "Genetic Algorithms for Adaptive Motion Planning of an Autonomous Mobile Robot," *Proceedings of the IEEE International Symposium on Computational Intelligence in Robotics and Automation*, 1997.

[8] Kamran H. Sedighi, Kaveh Ashenayi, Theodore W. Manikas, Roger L. Wainwright, Heng-Ming Tai, "Autonomous Local Path Planning for a Mobile Robot Using a Genetic Algorithm," *Proceedings of the 2004 Congress on Evolutionary Computation*, 2004.

[9] Dieter Fox, Wolfram Burgard, Sebastian Thrun, "The Dynamic Window Approach to Collision Avoidance," *IEEE Robotics and Automation Magazine*, 1997.

[10] Xiquan Mai et al, "An improved dynamic window approach for local trajectory planning in the environment with dense objects," *Journal of Physics: Conference Series 1884 012003*, 2021.

[11] Hiroki Ohta, Naoki Akai, Eijiro Takeuchi, Shinpei Kato and Masato Edahiro, "Pure Pursuit Revisited: Field Testing of Autonomous Vehicles in Urban Areas," *IEEE 4th International Conference on Cyber-Physical Systems, Networks, and Applications*, 2016.

[12] R. Craig Conlter, “Implementation of the Pure Pursuit Path Tracking Algorithm,” *The Robotics Institute of Carnegie Mellon University*, 1992.

[13] Rui Wang, Ying Li, Jiahao Fan, Tan Wang and Xueta Chen, “A Novel Pure Pursuit Algorithm for Autonomous Vehicles Based on Salp Swarm Algorithm and Velocity Controller,” *IEEE Access, Volume 8*, 2020.

[14] Rüdiger Dillmann, “KA 1.10 Benchmarks for Robotics Research,” *European Robotics Network*, 2010.

[15] Danica Kragic, Ville Kyrki, Patric Jensfelt and Frank Lingelbach, “Drawing a parallel: Benchmarking in Computer Vision and Robotics,” *European Robotics Network - EURON*, 2006.

[16] Jonathan Weisz, Yipeng Huang, Florian Lier, Simha Sethumadhavan and Peter Allen, “RoboBench: Towards Sustainable Robotics System Benchmarking,” *IEEE International Conference on Robotics and Automation*, 2016.

[17] Vincent C. Müller, “Measuring progress in robotics: Benchmarking and the ‘measure-target confusion’,” *University of Leeds*, 2018.

[18] S. Hanks, M. E. Pollack and P. R. Cohen, “Benchmarks, Test Beds, Controlled Experimentation, and the Design of Agent Architectures,” *AI Magazine Volume 14, Issue 4*, 1993.

[19] Pedro U. Lima, “A Probabilistic Approach to Benchmarking and Performance Evaluation of Robot Systems,” *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[20] Mark Moll, Ioan A. Sucan, Lydia E. Kavraki, “Benchmarking Motion Planning Algorithms,” *IEEE Robotics and Automation Magazine*, 2015.

[21] Christoph Sprunk, Jörg Röwekämper, Gershon Parent, Luciano Spinello, Gian Diego Tipaldi, Wolfram Burgard and Mihai Jalobeanu, “An Experimental Protocol for Benchmarking Robotic Indoor Navigation,” *Department of Computer Science, University of Freiburg, Germany; Microsoft Robotics, Microsoft Corporation, USA*, 2014.

[22] J. Borenstein and L. Feng, “Umbmark: A benchmark test for measuring odometry errors in mobile robots,” *Proc. SPIE, 2591:113–124*, 1995.

[23] R. Knotts, I. Nourbakhsh and R. Morris, “Navigates: A benchmark for indoor navigation,” *Int. Conf. and Exp. on Robotics for Challenging Environments*, 1998.

[24] W. Nowak, A. Zakharov, S. Blumenthal, and E. Prassler., “Benchmarks for mobile manipulation and robust obstacle avoidance and navigation,” *BRICS Deliverable D3.1*, 2010.

[25] Sven Wachsmuth, Dirk Holz, Maja Rudinac, Javier Ruiz-del-Solar, “RoboCup@Home – Benchmarking Domestic Service Robots,” *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[26] Zhi Yan, Luc Fabresse, Jannik Laval, Noury Bouraqadi, “Building a ROS-Based Testbed for Realistic Multi-Robot Simulation: Taking the Exploration as an Example,” 2017.

[27] Francesco Amigoni, Emanuele Bastianelli, Jakob Berghofer, Andrea Bonarini, Giulio Fontana, Nico Hochgeschwender, Luca Iocchi, Gerhard K. Kraetzschmar, Pedro Lima, Matteo Matteucci, Pedro Miraldo, Daniele Nardi and Viola Schiaffonati, “Competitions for Benchmarking Task and Functionality Scoring Complete Performance Assessment,” *IEEE Robotics and Automation Magazine*, 2015.

[28] Hector Yuste, Leopoldo Armesto and Josep Tornero, “Benchmark Tools for Evaluating AGVs at Industrial Environments,” *TIEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010.

[29] Jian Wen, Xuebo Zhang, Qingchen Bi, Zhangchao Pan, Yanghe Feng,Jing Yuan, Yongchun Fang, “MRPB 1.0: A Unified Benchmark for the Evaluation of Mobile Robot Local Planning Approaches,” 2020.

[30] D. Lu. (2019) (accessed: 30.10.2021). [Online]. Available: https://github.com/locusrobotics/robot_navigation/tree/master/dwb_local_planner

[31] S. Macenski. (2021) Regulated pure pursuit. (accessed: 30.10.2021). [Online]. Available: https://github.com/ros-planning/navigation2/tree/main/nav2_regulated_pure_pursuit_controller

[32] (accessed: 30.10.2021). [Online]. Available: https://navigation.ros.org/

[33] (accessed: 30.10.2021). [Online]. Available: http://docs.ros.org/en/rolling/

[34] (accessed: 30.10.2021). [Online]. Available: https://docs.ros.org/en/foxy/Releases/Release-Foxy-Fitzroy.html

[35] (accessed: 30.10.2021). [Online]. Available: http://docs.ros.org.ros.informatik.uni-freiburg.de/en/crystal/Tutorials/Understanding-ROS2-Nodes.html

[36] (accessed: 30.10.2021). [Online]. Available: https://github.com/AIRLab-POLIMI/performance_modelling

[37] M. J. Gregory Dudek, *Computational Principles of Mobile Robotics*. Cambridge University Press, 2010.

[38] (accessed: 30.10.2021). [Online]. Available: https://www.roscomponents.com/en/mobile-robots/215-turtlebot-3-waffle.html

[39] E. Ackerman, *Latest Version of Gazebo Simulator Makes It Easier Than Ever to Not Build a Robot*. IEEE Spectrum. IEEE, 2016.

[40] (accessed: 30.10.2021). [Online]. Available: http://www.cs.cmu.edu/afs/andrew/course/15/381-f08/www/lectures/motionplanning.pdf

[41] (accessed: 30.10.2021). [Online]. Available: https://www.cs.columbia.edu/~pblaer/projects/path_planner/