Universidade do Minho
Escola de Engenharia

Nino Sancho Sampaio Martins Pereira

Golf ball picker robot: path generation in unstructured environments towards multiple targets

Agosto 2014

Universidade do Minho
Escola de Engenharia

Nino Sancho Sampaio Martins Pereira

Golf ball picker robot: path generation in unstructured environments towards multiple targets

Tese de Doutoramento
Engineering Design and Advanced Manufacturing
Leaders for Technical Industries

Trabalho efectuado sob a orientação do
Professor Doutor Fernando Ribeiro

Agosto de 2014

To my beloved wife Teresa and my brave son Tomás, who are the ultimate reasons for everything I do and care about, and

to my parents who gave me all the support they could give, on top of the extraordinary model of noble human values.

ABSTRACT

The new TWIN-RRT* algorithm solves a motion planning problem in which an agent has multiple possible targets where none of them is compulsory, and retrieves feasible, “low cost”, asymptotically optimal and probabilistically complete paths. The TWIN-RRT* algorithm solves path planning problems for both holonomic and non-holonomic robots with or without kinodynamic constraints in a 2D environment, but it was designed to work as well with higher DOF agents and different applications. The new algorithm provides a practical implementation of feasible and fast planning especially where a closed loop is required. Initial and final configurations are allowed to be exactly the same. The TWIN-RRT* algorithm computes an efficient path for one sole agent towards multiple targets where none of them is mandatory. It inherits the low computational cost, probabilistic completeness and asymptotical optimality from RRT*. It uses efficiency as cost function, which can be adapted depending on the application. The TWIN-RRT* complies both with kinodynamic constraints and different cost functions. It was developed to solve a real problem where a robot has to collect golf balls in a driving range, where thousands of balls accumulate every day. This thesis is part of a bigger project, Golfminho, to develop an autonomous robot capable of efficiently collecting balls in a golf practice field.

RESUMO

O novo algoritmo TWIN-RRT* resolve problemas de planeamento de trajetórias em que um agente tem múltiplos alvos, onde nenhum deles é obrigatório, e produz um plano exequível, de "baixo custo" computacional, assintoticamente ótimo e probabilisticamente completo. O TWIN-RRT* resolve problemas de planeamento de trajetórias tanto para robôs holonómicos como não holonómicos com ou sem restrições cinemáticas e/ou dinâmicas num ambiente 2D, mas foi projetado para funcionar também com agentes com maiores graus de liberdade e em diferentes aplicações. O novo algoritmo fornece uma implementação prática de um planeamento viável e rápido, especialmente quando é necessário produzir uma trajetória fechada. As configurações iniciais e finais podem ser exatamente iguais. O algoritmo TWIN-RRT* calcula um caminho eficiente para um agente único em direção a múltiplos alvos, onde nenhum deles é obrigatório. Herda o baixo custo computacional, integralidade probabilística e otimização assintótica do RRT*. Usa a eficiência como função de custo, que pode ser adaptada em função das diferentes aplicações. Para além de diferentes funções de custo, o TWIN-RRT* também mostra conformidade com restrições cinemáticas. Foi desenvolvido para resolver um problema real em que um robô tem que recolher bolas de golfe num *Driving Range*, onde se acumulam milhares de bolas de golfe por dia. Esta tese é parte integrante do projeto Golfminho, para o desenvolvimento de um robô autónomo capaz de recolher eficientemente bolas num campo de práticas de golfe.

DECLARAÇÃO DE INTEGRIDADE

Declaro ter atuado com integridade na elaboração da presente tese. Confirmo que em todo o trabalho conducente à sua elaboração não recorri à prática de plágio ou a qualquer forma de falsificação de resultados.

Mais declaro que tomei conhecimento integral do Código de Conduta Ética da Universidade do Minho.

Universidade do Minho, 8 de Agosto de 2014

Nome completo: NINO SANCHO SAMPAIO MARTINS PEREIRA

Assinatura: ____

STATEMENT OF INTEGRITY

I hereby declare having conducted my thesis with integrity. I confirm that I have not used plagiarism or any form of falsification of results in the process of the thesis elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

University of Minho, The 8th August 2014

Full name: NINO SANCHO SAMPAIO MARTINS PEREIRA

Signature: ____

ACKNOWLEDGEMENTS

I would like to thank my main supervisor, Professor Fernando Ribeiro *(UMinho, Portugal),* who has been my main reference, encouraging me to prosecute my path, helping me to overcome the obstacles and steering my research towards the final goal.

I would like to acknowledge the important role of Professor António Cunha *(UMinho, Portugal)* who coached me on the first year and contributed on the design, building and growing of the MIT-Portugal program.

I thank my supervisor Professor Jorge Lino *(FEUP, Portugal)* who has been very supportive since the beginning and gave me important feedback in order to achieve the major milestones of the project.

I also thank my supervisor Professor Daniel Whitney *(MIT, USA)* who guided me on the important stages of the project and with his wisdom and experience contributed for the quality of my research.

To my colleges, Miguel Gomes, Ivo Moutinho, Pedro Silva, José Cruz, José Fernandes and João Costa, I would like to leave a word of gratitude and admiration. I cannot express how grateful I am for having them as co-workers and friends.

I would like to express my great appreciation to Professor Gil Lopes *(UMinho, Portugal)* for all his support and valuable contributions.

I would like to offer my special thanks to my colleges João Ferreira *(UMinho, Portugal)* and Alexandre Silva *(Univ. British Columbia, Canada)* who helped me implementing the algorithm and with whom I had the opportunity to discuss important details and ways to improve it.

Advice given by the *Federação Portuguesa de Golf* has been a great help to define the objectives, the constraints and the overall strategy.

Alejandro Perez *(MIT, USA)* provided me with very valuable information regarding sampling planning based algorithms and ways of quickly implementing them.

I am also particularly grateful to Professor Tijn van der Zant *(University of Groningen, The Netherlands)* for providing me all the technical and scientific resources during my stay in Groningen.

I would like to thank the following companies for their participation in the *Golfminho* project:

- *SAR – Soluções de Automação e Robótica, Lda;*
- *Mecanarte – Metalúrgica da Lagoa, Lda;*
- *Partis Consulting;*
- *Fibrauto, Fabrico de objectos em poliéster, Lda.*

My special thanks are extended to all the staff of *Tecminho - Associação Universidade-Empresa para o Desenvolvimento*, who played an important role regarding the project intellectual property and the technology transfer.

I am grateful to the *SPR – Sociedade Portuguesa de Robótica* and its affiliates for the remarkable contribution to the research in robotics field. Their example was a great source of inspiration and motivation.

I also thank all the teachers, supportive staff and affiliated companies of the MIT-Portugal Program, which, I think, notoriously contributed to increase the quality of teaching and research in Portugal.

I am grateful to the *University of Minho* and in particular to the *Dep. Electrónica Industrial da Escola de Engenharia* for providing me all the necessary conditions to do my research. I also recognise the helpful support given by all staff of *SAS – Serviço de Acção Social da Universidade do Minho*, namely regarding the use of the golf facilities for testing the *Golfminho* robot.

I would like to thank the *Portuguese Science Foundation - FCT "Fundação para a Ciência e Tecnologia"* for the PhD Grant No. SFRH / BD / 43008 / 2008.

My research is part of the project GOLFmINHO n°1583 which was co-financed by European community fund *FEDER (Fundo Europeu de Desenvolvimento Regional),* approved by *QREN (Quadro de Referência Estratégico Nacional, Portugal 2007.2013)* and supported by *ON.2 (o Novo Norte – Programa Operacional da Região Norte), CCDRn (Comissão de Coordenação e Desenvolvimento Regional do Norte)* and *ADI (Agência de Inovação).*

"Do not go where the path may lead, go instead where there is no path and leave a trail."

Ralph Waldo Emerson

PREFACE

"The machine does not isolate man from the great problems of nature but plunges him more deeply into them."

Antoine de Saint-Exupery

I long for a world in which Man can engage freely in noble activities as philosophical thought, the arts, or the search for understanding of our own existence.

Humans have always been struggling for a better life. It is craved in our DNA this quest for higher quality of life. Our most powerful tool to achieve it is science and technology. We managed to live longer and healthier by the use of drugs and medicines. We managed to set us free from dangerous, tedious or painful tasks by the use of automata.

It is essential to preserve and promote the scientific and technological research because the gains that we take from it enhance the elevation of the human species. It is also important to promote the use of the knowledge to do the good, to improve our lives and to harmonize our relationship with nature.

I believe that robotics has an important role to play in this century, regarding those premises.

It is up to us to make the most out of it and to do it for good purposes.

I hope this thesis to be a small contribution in that direction…

CONTENTS

ACRONYMS AND ABBREVIATIONS

ACO - Ant Colony Optimization

AI – Artificial Intelligence

CPU - Central Processing Unit

DC - Direct current

DOF – Degrees Of Freedom

EST - Expansive-Spaces Tree planner

EXPTIME - exponential amount of time

GA - Genetic Algorithms

GPS - Global Positioning System

ICC - Instantaneous Centre of Curvature

KD - Tree - k-dimensional tree data structure for organizing points in a k-dimensional space.

LK - Lin Kernigham algorithm

OPP - Opportunistic Path Planner

PRM – Probabilistic Roadmap planner

PSPACE Polynomial amount of space

RPP - Randomized Path Planner

RRT - Rapidly-exploring Random Tree planner

RRT* - Asymptotically Optimal Rapidly-exploring Random Tree planner

RRT-CONNECT - Rapidly-exploring Random Tree CONNECT planner

SRT - Sampling-Based Roadmap of Trees planner

TPP - Traveling Purchaser Problem

TSP - Travelling Salesman Problem

TWIN-RRT* - TWIN Asymptotically Optimal Rapidly-exploring Random Tree planner

UAV - Unmanned Aerial Vehicle

SYMBOLS

2D - Two dimension

3D - Three dimensions

A - Amperes

Ah - Amperes hour

C_{free} - free configuration space

Km/h - Kilometres per hour

L - length of the robot (from rear axis to front turning wheel)

m - meters

mm - millimetres

NP - Non-polynomial amount of time

NP-hard - Non polynomial time hard

O - number of operations

P - Polynomial amount of time

q - configuration

Q - configuration space or set of all possible configurations

rpm - rotations per minute

Ta - Forward Tree

Tb - Backwards Tree

V - Volts

V_l - Left wheel speed direction

V_r - Right wheel speed direction

W - workspace

x - agent configuration

X - agent configuration space

X_{free} - obstacle free region

X_{goal} - goal region

x_{init} - initial configuration

X_{obs} - obstacle region

δt - elapsed time between two poses of the robot

θ – direction of the robot movement

ρ - curvature radius

ρ_{min} - minimum curvature radius

σ - path

ϕ - yaw turning angle

ϕ_{max} - maximum yaw turning angle

ω - angular velocity

Keywords: Field Robots, AI, Path planning, Motion planning, RRT, RRT*, TWIN-RRT*, asymptotically optimal, efficient planning, multiple targets, closed loop planning.

LIST OF FIGURES

LIST OF TABLES

LIST OF EQUATIONS

CHAPTER I

INTRODUCTION

This document presents a work in the field of Robotics and specifically related to path planning algorithms. Robotics is a broad field which incorporates principles of electronic engineering, mechanical engineering, computer science, amongst others (Fig. 1). Within computer science, the artificial intelligence (AI) branch is the most relevant for robotics.

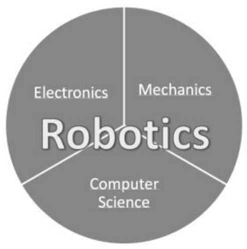


Fig. 1 – Robotics integrates electronics, mechanics and computer science

"Artificial intelligence is that activity devoted to making machines intelligent, and intelligence is that quality that enables an entity to function appropriately and with foresight in its environment." (Nils J. Nilson in Quest for Artificial Intelligence)

As Nils J. Nilson pointed out [1], artificial intelligence deals with the art of making machines not only able to act in accordance with the environment, but also able to perceive it beforehand.

Humans have the capability of gathering information about the environment and use that perception to plan and act over it. Human actions are in essence as simple as that, but humans also verify, correct and adjust in real-time, reacting to unpredictable situations, adversities or unexpected external factors. These are the challenges of artificial intelligence.

The work presented in this document relates to the planning phase, when the agent, having relevant information about the environment, is capable of planning a feasible and appropriate action in the environment.

Path planning is especially important for mobile robots, not only for wheeled robots but also for manipulators. This document focuses on wheeled non-holonomic robots and path planning on a 2D environment (or workspace).

Some authors prefer to use the expression “motion planning” when a mobile robot is the agent, but in artificial intelligence literature the expression “path planning” is more common. Both terms will be used indistinguishable throughout this document. The same reasoning applies to the words “robot” and “agent” which have the same meaning for the current purpose.

This thesis is structured in chapters covering different aspects of the research. Chapter I presents a brief explanation of the new TWIN-RRT* algorithm and its main features. It introduces the initial technical problem to be solved and presents a case-study defining the framework in which the project took place.

Chapter II gives an overview on how related problems, namely target localisation and kinematic constraints were dealt with. These subjects do not constitute the core of the research but significant developments were accomplished while solving the case-study problem. This chapter also presents the metric used to evaluate the algorithm performance and how efficiency is defined.

Chapter III gives a comprehensive overview of path planning concepts and existing methods. The path planning problem is further detailed. It also provides important background regarding the type of planner and related methods suitable for solving the technical problem.

Chapter IV presents the TWIN-RRT* in detail as well as the discretisation process used for the case-study. It shows how the TWIN-RRT* derives from RRT* and inherits its most valuable features. Also, a new variation of KD-Trees is presented which enables using direction representation together with 2D Euclidean space representation.

Chapter V comprises the complete set of experiments carried out to better understand and characterise the behaviour of the TWIN-RRT* algorithm. It proves the new algorithm suitability and efficiency in solving the initial technical problem.

Chapter VI summarises the main contributions of this thesis, presents a reflection about the strategy followed and gives some indications for further work.

1.1. The new TWIN-RRT* algorithm

A strategy was developed to generate feasible efficient paths for one agent towards multiple targets. A new random sampling algorithm based on RRT* was developed. It grows two RRT* trees from the same starting configuration (see Fig. 2). For that reason it was named TWIN-RRT*.

The RRT* is, in turn, an asymptotically optimal planning algorithm [2-5] derived from RRT (rapidly-exploring Random Tree planner). The original RRT retrieves fast solutions even for many degrees of freedom and different constraints [6, 7].

In TWIN-RRT* the initial and final configurations are allowed to be exactly the same. The algorithm retrieves a closed loop path starting at the initial configuration and ending at the final configuration. The resulting path maximises efficiency by maximising the ratio (*targets visited)/ (distance travelled)* while avoiding obstacles. It also satisfies the restrictions and conditions imposed, like for example, the minimum number of targets to visit, the minimum distance to travel, the maximum distance to travel, the maximum time to spend in path calculation and minimum efficiency gain among others. A detailed explanation of the algorithm is described in section 4.2.

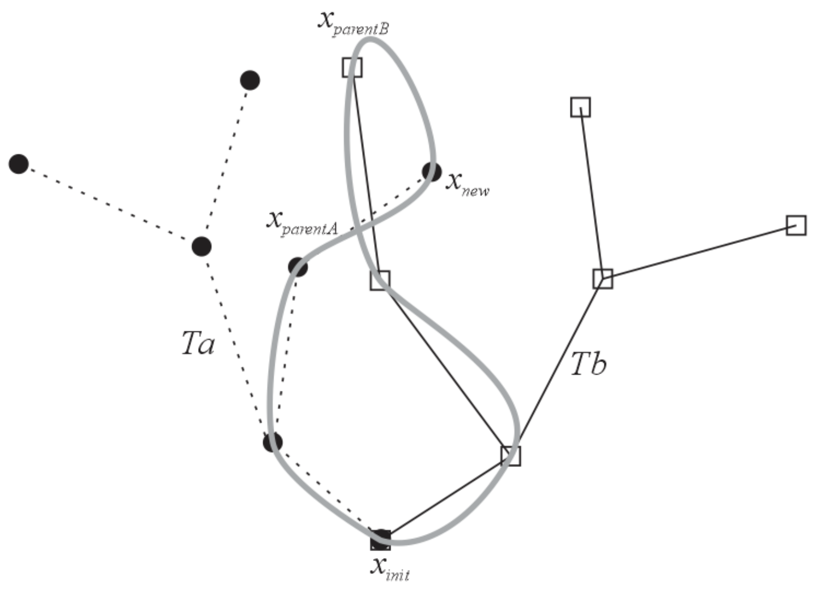


Fig. 2 – TWIN-RRT* representation having two trees growing from the same root

1.2. Case-Study

A case-study is included to illustrate the current research. This will hopefully enable a better understanding of the type of problem being solved and their subtleties.

The algorithm solution presented later on this document is extensive to other type of problems and situations. In order to maintain a common reference, the case-study will serve as guide along the entire document.

Case-study:

In golf practice fields, usually called *Driving Ranges*, several thousands of balls are played every day. Typically a medium size (300m x 100m) field accumulates over 10,000 balls per day. Since golf balls are not disposable or biodegradable, they must be collected, washed and returned to the dispensing machine where players get again the balls to play (Fig. 3). The golf ball picking task is usually performed by a worker operating a buggy with a collecting device on a trolley. This trolley comprises a set of discs arranged in parallel so that when rolling over the terrain the balls are arrested between two discs and are pulled into the basket by the rotating movement of the discs.

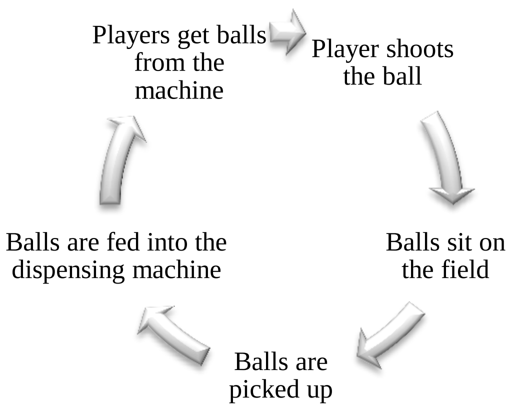

Players get balls from the machine
Player shoots the ball
Balls sit on the field
Balls are picked up
Balls are fed into the dispensing machine

Fig. 3 - Golf ball cycle on a golf practice field

Three or four times a day (sometimes more) the operator drives the buggy throughout the field and recovers many of the balls lying on the field, most of the times while players are still shooting balls. A golf ball weighs 45g and may reach about 250km/h and for that reason, the operator has to wear special protections. Usually the buggy has a special protective unbreakable glass (Fig. 4).

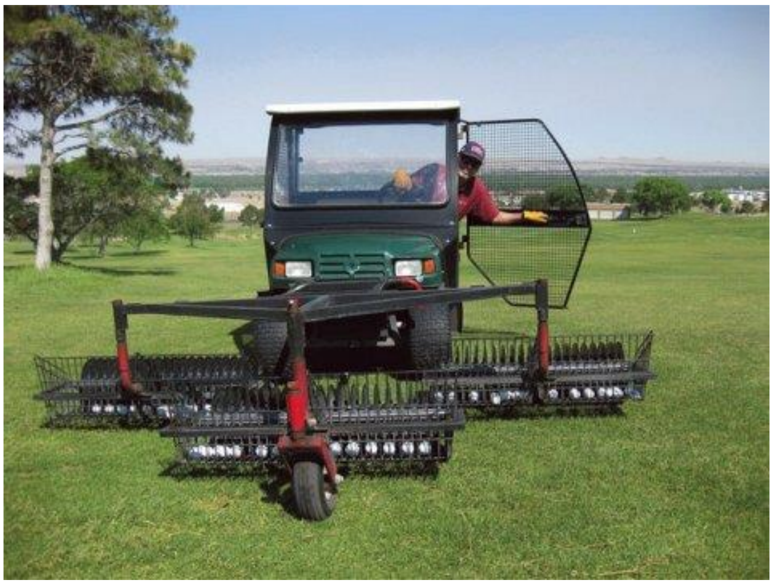


Fig. 4 – Golf ball picking task performed by buggy

Buggies generally operate at 5km/h, they are capable of collecting 4,000 balls/hour and are equipped with high capacity baskets (1,800 balls).

In some countries, where human labour is cheap, the picking task is still performed without the use of machines. Workers carry special protections and they manually collect thousands of golf balls per day (Fig. 5). This is considered to be a tedious and dangerous task [8].

New approaches have recently emerged and attempt to perform this task using autonomous or remote controlled devices. Their primary goal is to reduce human risk exposure and simultaneously reduce operational and maintenance costs [8-10]. Some of them use vision systems to find and pursuit the golf balls [8, 9]. Others use gesture based commands to drive robots on the field [10]. As pointed out before, the need for dedicated and specialized vehicles to pick up golf balls is becoming imperative to reduce overall operational costs and to decrease the risk of health problems for humans.

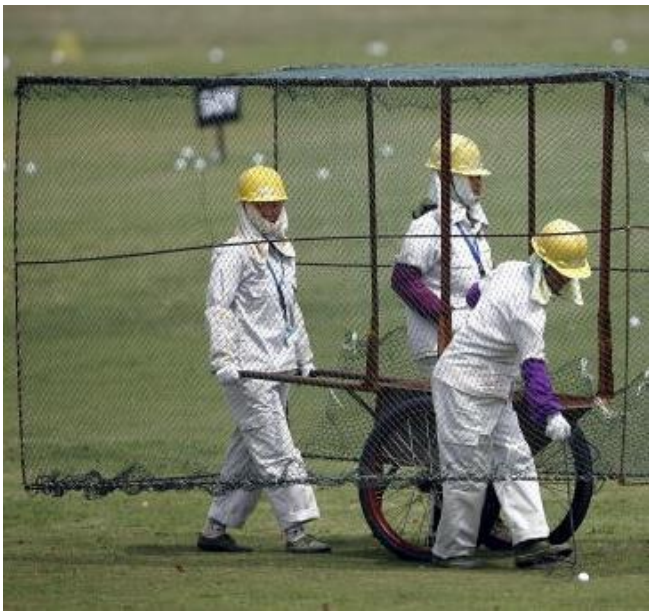


Fig. 5 – Workers collecting golf balls

There is an autonomous robot available in the market which aims to accomplish this task [9]. It collects golf balls while randomly moving in a wire limited perimeter. This machine has a 400 balls capacity container, weighs about 65 kg and operates at speeds up to 3.6 km/hour. Its simple approach relies on the fact that golf balls may be anywhere on the field and their distribution is random. The simplest version of this robot does not possess means to retrieve the exact golf ball locations and does not even determine its own location. It navigates around the limited perimeter and picks up all the golf balls in its way. This is still an effective machine but nevertheless significantly inefficient.

Researchers are seeking ways to endow machines the capability of recognizing and locating objects in a fast an accurate way for real time applications [10, 11].

Computer vision systems are amongst the most effective ways of doing so. Nevertheless, they often require delicate calibration processes [12-14] and are unsuitable for use in external environments where light variations, rain and dirt pose additional problems and severely affect the results accuracy [14].

The main problem which remains unsolved is how to autonomously perform the golf ball picking task in an efficient manner, by simultaneously eliminating the human exposure to risk, reducing costs by operating without human intervention and at the same time improving the efficiency when compared to random navigating systems.

1.3. The need

Opposite to what one might think, there is no need to collect all the golf balls on the field. The important goal is to prevent the golf ball dispensing machine from getting starved (refer to Fig. 3). The main requirement consists of always guaranteeing balls availability for players.

Consequently there is the need for some method and/or device to constantly replenish the machine by picking up the golf balls and putting them back into the dispensing machine.

With this idea in mind, an autonomous device was designed to accomplish the golf ball picking task. Apparently it seems not to be much different from existing solutions, but conceptually it is. This new device has some built in new technologies that enable two major important features: prediction of golf balls localisation and efficient path planning.

1.4. The technical problem

It is assumed that targets location is given a priori. The problem can then be defined as follows: Given a set of golf balls located at different positions on the field, the problem is how to collect at least a certain amount of balls while maximising the efficiency, where efficiency is defined as the ratio between the number of balls collected over the distance travelled (see equation 1).

$$Efficiency = \frac{(number\ of\ balls)}{(distance\ travelled)} \tag{1}$$

Note: a certain amount of balls only implies a number but not a sub-set of specific balls, i.e., the aim is to collect a number of balls, amongst any balls on the field, so that any ball can be collected and none of them is mandatory.

Restrictions may apply in the form of minimum number of targets, minimum distance travelled or maximum time spent. A necessary condition is that the initial and final positions are the same.

1.5. Domain of research

The main research is focused on efficient path planning towards multiple targets. The path planning or motion planning is a sub-domain inside artificial intelligence[1], which has captured the attention of many brilliant researchers for the past decades. The piano mover's problem [15] is a classic example of a path planning problem. Considering any three-dimensional object and a set of known obstacles, the task is to find a collision-free path from an initial configuration to a final configuration. In this case, the object is allowed to move freely in space, having all the possible degrees of freedom the space allows. In three dimensions, the object is allowed to translate in x, y and z, and rotate along x, y and z axis, thus having 6 degrees of freedom. For a 2 dimensional space a similar object will have 3 degrees of freedom: translation in the x and y axis and rotation along the z axis. As premises to address this kind of problems, the obstacles are completely known (position and dimensions) and are considered to be stationary. The planned path is assumed to be perfectly executed by the system, i.e. it does not account for uncertainty[2]. The algorithm that solves this kind of problems is called the planner. It may be classified according to its optimality, completeness and computational complexity [16].

The form of the planner strongly depends on the properties of the agent (robot) and the environment. An omnidirectional robot is able to move freely in any direction of its configuration space. A non-holonomic robot is subject to direction and velocity constraints (e.g. a car) and its kinematic and dynamic properties must be taken into account by the planner. Most of path planning algorithms focus on finding a path for one agent from an initial configuration to a final configuration. Recently, researchers have been paying more attention to planning for multiple agents towards multiple targets.

The work presented in this document focused on finding a path planning algorithm for one agent towards multiple targets, adding the important fact that none of the targets is really compulsory. This slight difference has a major impact on the design of the planning algorithm.

The chosen path should be the one that maximises the number of targets while minimising the distance. The goal is thus to maximise the efficiency (see equation 2).

[1] *"Artificial intelligence (AI) is that quality that enables an entity to function appropriately and with foresight in its environment." (Nilsson, 2010)*

[2] *"Uncertainty may be purely a consequence of a lack of knowledge of obtainable facts. That is, you may be uncertain about whether a new rocket design will work, but this uncertainty can be removed with further analysis and experimentation. At the subatomic level, however, uncertainty may be a fundamental and unavoidable property of the universe. In quantum mechanics, the Heisenberg Uncertainty Principle puts limits on how much an observer can ever know about the position and velocity of a particle. This may not just be ignorance of potentially obtainable facts but that there is no fact to be found. There is some controversy in physics as to whether such uncertainty is an irreducible property of nature or if there are "hidden variables" that would describe the state of a particle even more exactly than Heisenberg's uncertainty principle allows." (Wikipedia)*

1.6. Social impact

"Technology is teaching us to be human again." Simon Mainwaring

Scientific and technological contributions aim to provide man a better life. Machines and robots in particular have substantially decreased human exposure to risk and simultaneously improved production and quality of products.

The small contribution laid down on this document is aligned with the idea of having machines that can free humans from hard, dangerous and tedious labour.

Planning algorithms have been successfully applied in industry and academy. Besides robotics, planning algorithms have also been useful in manufacturing, drug design, and aerospace applications [17].

In particular, the planning algorithm that is presented in this document explicitly focuses on efficiency, which is a major concern nowadays, both for economic and environmental reasons. A sustainable environment depends on efficient tasks and that is the relevant feature of the new proposed algorithm. It aims not only to solve a problem but also to do it in an efficient manner.

It is expected to be relevant in the field robotics area, especially in particular applications such as agriculture, surveillance or land exploration.

1.7. Project Milestones

The project was subdivided in five major milestones. Within each milestone a set of objectives were previously established. Fig. 6 shows the main objectives corresponding to each phase. The first milestone is accomplished by completing phase I. All subsequent milestones are accomplished as soon as all the previous objectives are completed. Each phase depends on the previous one, but within one phase the tasks can be interchanged. The project was developed roughly according to this structure but it does not mean that each phase was terminated before starting another one. Often it was necessary to go back and adjust previous calculations or specifications detailed in a previous phase. This is how it generally goes in practice and it has to be seen as a good practice to adjust and correct whenever required, along the research project.

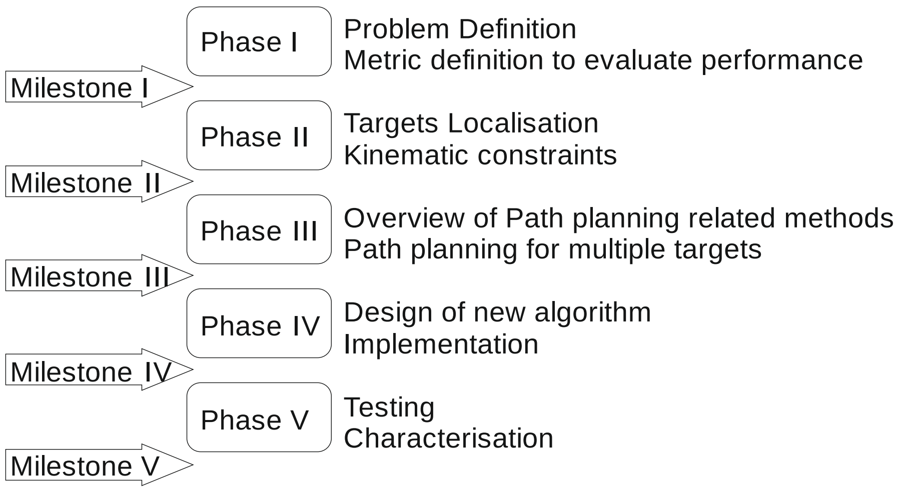


Fig. 6 – Milestones of the project

The problem definition and metric to evaluate performance were briefly presented on section 1.4 but a more detailed explanation is presented on section 4.1.

To construct a path, both the properties of the agent and the properties of the algorithm have to be priory established. Chapter II focuses on the agent properties that influence the planner performance, including targets localisation and kinematic constraints. Chapter III describes the properties of the algorithm that best suits the problem needs and includes an overview of path planning related methods. Chapter IV covers the Phase IV and presents the new algorithm and Chapter V covers Phase V, providing an extensive set of experiments carried out to test and characterise it.

CHAPTER II

TARGET LOCALISATION AND ROBOT KINEMATICS

Regarding the agent, it is necessary to know its degrees of freedom, its kinematics and its dynamics. There are planners which perform very well for few degrees of freedom but become unfeasible for higher number of degrees of freedom [18, 19]. The agent kinematics is important for steering and obstacle avoidance. Non-holonomic robots have to deal with an extra effort to avoid obstacles while conforming to their kinematic limitations [20-22].

This chapter presents information regarding the physical platform (the robot) to be used with the algorithm. The algorithm is designed to be independent of the kinematic restrictions, nevertheless, considering the real problem posed in the case-study, it is pertinent to present here the relevant aspects related to robot constraints.

Also, the problem of dealing with the target localisation is presented in this chapter. A new approach is described - the Hybrid Localisation System - that aims to deal with the lack of precise information about the targets localisation.

2.1. Non mandatory targets

It is not mandatory to reach a specific target or group of targets. Actually none of the possible targets is mandatory. This is apparently a simple nuance, but it changes the paradigm of path planning. Categorising this situation as a path planning problem is possibly arguable and some may consider it as an optimization problem instead.

It is not the main focus to reach a certain number of targets, but a minimum value can be imposed in the form of a restriction.

In this case, as in any other path planning problem the targets locations must be determined before establishing a plan.

2.2. Targets and robot localisation

For the planning phase it is imperative to know the exact location of the targets, and for the real implementation it is also imperative that the agent knows its localisation.

The unstructured environment of the presented case-study does not provide any information about the targets localisation. An unstructured environment is a dynamic environment where changes in obstacle location or target location may occur [23]. Additional interference of other external factors, such as slopes, holes, slippery terrain and many others should also be considered. These environments pose a significant challenge due to their complexity and inherent uncertainty. In unstructured environments, a robot cannot rely on a detailed and accurate *a priori* model of the environment [24].

It is not the focus of this document to address all these questions. But since the current case-study poses an interesting and challenging problem related to the golf balls localisation, it is worth to mention a new strategy developed to infer the targets approximate location based on prediction and statistical records. This method (see section 2.4) was developed to overcome the lack of precise information about targets location. It was designed to provide a feasible way to do it in a reliable manner.

Not knowing the exact targets location is at least as critical as not knowing the obstacles location, for path planning purposes. Although the exact target location is not known, it is believed that a statistically based prediction provides a reasonable approach to solve the case-study problem. Section 2.4 presents an explanation on how target location is determined. About the robot localisation it suffices to say that it uses a standard GPS based system which was carefully chosen to suit the application needs.

2.3. Robot design

To better understand the next section, it is important to make a reference to the platform running the algorithm. The robot structure helps to understand the design choices detailed in this chapter.

The robotic platform, named Golfminho (see Fig. 7), was designed for autonomous operation. The mechanical chassis was designed to hold two DC motors directly coupled to the two rear wheels. The robot movement is controlled by electric differential. This type of coupling reduces the mechanical losses. The free wheel at the front of the robot prevents it from getting stuck when facing higher slopes.

The robot structure was made compatible with commercially available gangs (set of disks that pick up the golf balls). A set of 34 polyethylene disks with a diameter of 29.4 cm each, are mounted on a steel axle. The gang is attached to the front structure and has three degrees of freedom (40° Roll, 10° Pitch and 20° Yaw amplitudes). The Yaw amplitude was limited to 20° in order to prevent damage to the gang disks. The balls are recovered by the gang disks when they roll over the balls. Since the disks rotate as the robot moves, the balls get stuck in between the disks and travel up as the disks rotate. When the balls reach the top of the basket a wedge forces them to detach from the disks and they drop into the basket. The maximum width is 1.65 m. The whole structure (mechanical chassis, motors, batteries, wheels and gang) weighs about 120 kg and is prepared to pull up to 200 kg. Its capacity is about 1200 balls which means 54 kg extra.

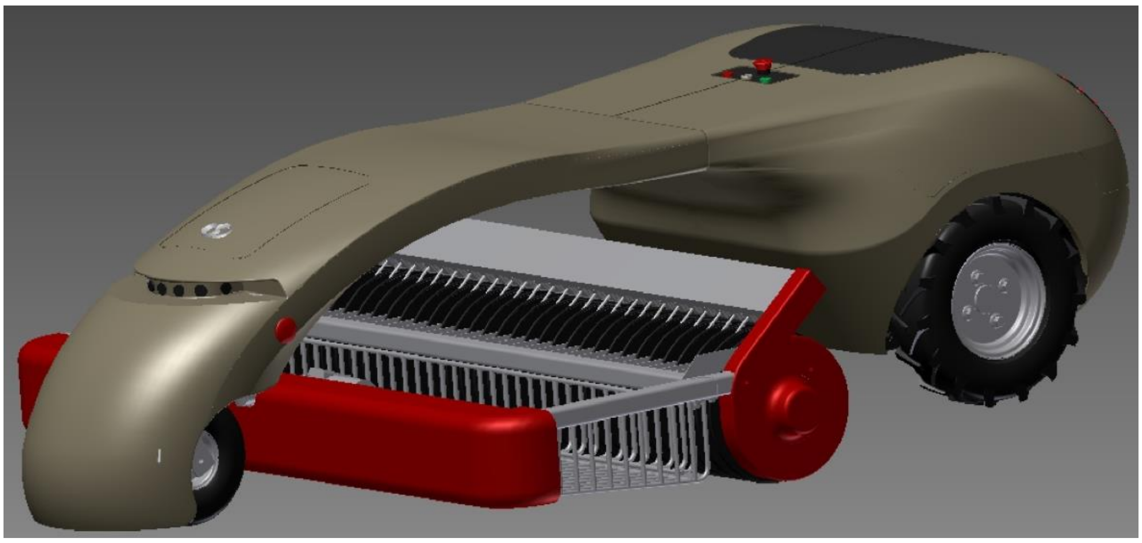


Fig. 7 – Golfminho robot

A remote control enables manual control of the robot, should be required. The manual operation gives the user full control of the robot movement. Besides that, the remote control has bidirectional communication using the Zigbee protocol. This means that the robot may send alerts to this device, notifying the operator when necessary.

The robot is equipped with sensors to provide the necessary information for moving the robot safely and efficiently. These include a gyroscope, temperature and humidity sensors, encoders coupled to the motors, sonars, GPS, compass, touch sensors, voltage and current sensors.

A ball counter device using micro-switches provides information about the number of balls the robot collects. Sonars are used to detect obstacles so they can be avoided autonomously. The differential GPS system informs the robot about its location on the field.

Two 24V DC motors, each one attached to one wheel, are used to control the robot movement. An electrical differential is used to control robot heading direction. These DC motors have permanent magnets with electric brake and 1:30 gearbox incorporated. Each motor consumes 800 W and has a gearbox output of 100 rpm. Using 400mm diameter wheels, the maximum linear speed is 7.5 km/h.

Eight cells of 3.6 V, 160 Ah lithium ion batteries are used to power the robot. With these specifications, the robot operates for about 8 hours, after which it needs a 3 hours recharge. Nominal current is 80 A, and maximum current is 480 A.

The Golfminho robot does not possess any means to accurately detect all targets location. Even if it did, that information would quickly become obsolete as the players throw more balls at the field, during practice time. The hybrid localisation system described next provides an answer for this problem.

2.4. Hybrid Localisation System

The notion that statistical reasoning is relevant to robot motion planning has for long been acknowledged by researchers [25].

A statistically based predictive system was developed to estimate the most probable target location at a given moment – the Hybrid Localisation System. This system, which is explained next, works together with the main robot system and uses relevant and updated information about the number and the location of previously reached targets. It uses both real and virtually generated data, hence named “hybrid”, to predict the actual balls location on the field.

The hybrid localisation system provides a virtual map containing previewed golf balls locations, which is continuously being updated, based on real information gathered by the robot. The system logs relevant data, such as date, time and GPS data for each collected ball. The data is then processed to retrieve statistically relevant information, such as the distribution probability, the frequency at which balls are sent to the field and respective variations along the day, week, month and year. A virtual map is generated and continuously updated.

Fig. 8 shows a diagram of the operating system. It includes a Real Item Database, a Trigger

Unit, a Virtual Item Generator, a Virtual Item Positioner a Virtual Map and an Area Clearance module. The Trigger Unit is a software module that generates a signal to start an event. The signal may have a fixed or variable frequency. This frequency is continuously adjusted based on the Real Item Database information. The Virtual Item Generator is another software module which generates a virtual item (golf ball) whenever it receives the trigger signal from the Trigger Unit. The virtual item generated has the same characteristics (which can be detected or input to the system) of a real one (e.g. dimensions and weight).

The Virtual Item Positioner is responsible for allocating the item generated by the Virtual Item Generator in the Virtual Map. It assigns a position to the virtual item on the Virtual Map.

The Virtual Map retains an updated and statistically significant prediction of the golf balls on the field. It constitutes the most important output of the system.

The Area Clearance is a module that acts on the Virtual Map removing items from a specific or entire area of the field. While navigating on the field, the robot sweeps areas that will be cleared from the Virtual Map, i.e. whether or not it picks balls in reality, virtually it collects all the balls in that same area. This last effect could be interpreted as an “augmented virtuality”, in the same way as one conceives an augmented reality.

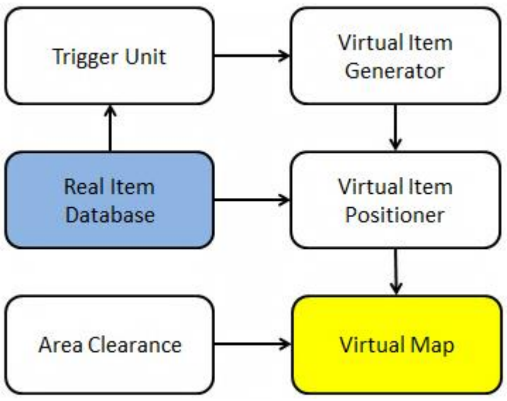


Fig. 8 - Hybrid Location System Diagram

The process steps of the hybrid location system are depicted in Fig. 9. There are two processes, A and B running in parallel. In process A, the Trigger Unit updates the rate at which signals are generated based on the information fetched from the Real Item Database. The Virtual Item Generator receives the signal and creates a virtual item. The Virtual Item Positioner then attributes a position to that item based on the information of the Real Item Database. The item is finally represented on the Virtual Map.

The process B runs in parallel with the process A and updates the *Virtual Map* based on the motion information. First, the current position is acquired and then the covered area is determined by using the previous position and the width of the robot. Finally the correspondent area on the virtual map is cleared, i.e. all the virtual items located in that area are removed from the *Virtual Map*.

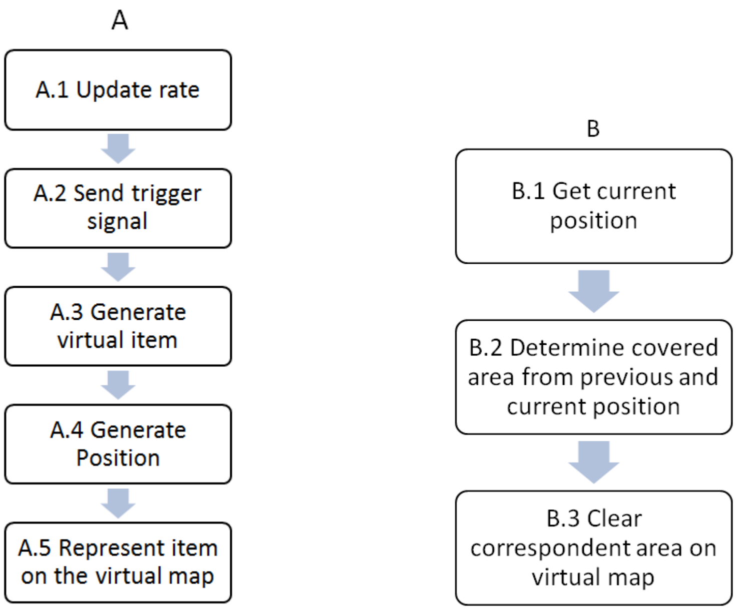

A
A.1 Update rate
A.2 Send trigger signal
A.3 Generate virtual item
A.4 Generate Position
A.5 Represent item on the virtual map
B
B.1 Get current position
B.2 Determine covered area from previous and current position
B.3 Clear correspondent area on virtual map

Fig. 9 – Process flowchart

It is important to remember that the *Real Item Database* is being updated as the real robot moves and determines the exact location of a target (e.g. by collecting it and registering the current location, date and time).

The virtual map is expected to have approximately the same number of items present on the real field. The error will be smaller for larger number of items. The distribution of the predicted items will be closer to the real distribution as the number of items increases. The discrepancy between the virtual and real positions will decrease as the real database increases in size.

It is expected that the robot performance will continuously be improved as it operates repeatedly.

Guessing the golf balls location is only part of the solution. Another, and perhaps even more challenging problem is to determine the most efficient pathway to collect the maximum amount of balls over a determined distance travelled or time. Typically a path planning problem deals with a starting position and one target position. In this case there is also a starting and ending position with many possible targets, but none of them is compulsory (Fig. 10). This problem shares many similarities with the TPP (Traveling Purchaser Problem). It is a NP-hard problem studied in theoretical computer science. TPP was first described by Ramesh [26] in 1981. He also proposed an exact algorithm which solves the problem optimally. However, TPP solutions have always been computationally very expensive [27]. Since then some attempts have been made to deal with that complexity [28-31].

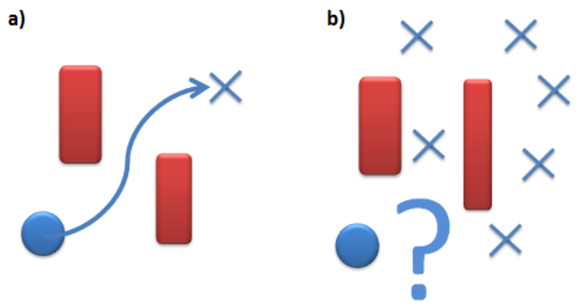


Fig. 10 – Planning towards: a) one target and b) multiple targets

The robot of the case-study uses a very compact CPU and a small processing unit that cannot run expensive time consuming algorithms. Moreover, the planning algorithm must also take into account the kinematic constraints of the robot. A new fast and efficient motion planning algorithm which provides a solution for a multi-target problem was developed. The foundations of the new algorithm are presented in the next chapter.

Next follows a brief presentation about the particular kinematic constraints of the robot used in the present case-study. It is important to mention that, although the analysis concerns a specific construction of a robot, the approach is common to other non-holonomic robots.

2.5. Kinematic Analysis

Instead of having a steer axle, the Golfminho robot has a free wheel at the front. It moves by electric differential applied to the rear wheels (Fig. 11). When turning right the rear left wheel moves at higher speed than the rear right wheel. The direction of the movement (Ø) is obtained taking into account the linear speed of both left and right wheels, V_l and V_r.

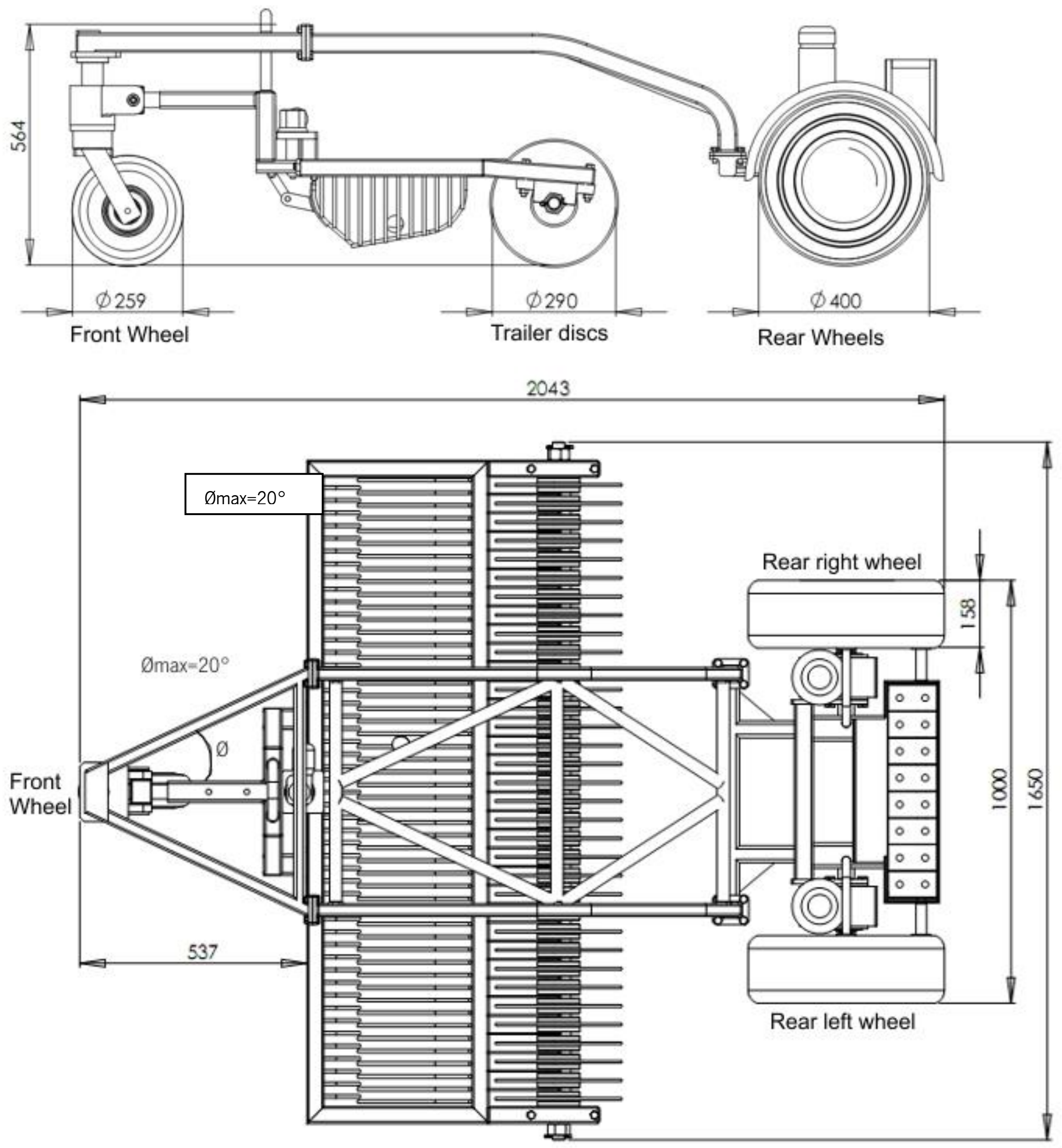


Fig. 11 – Golfminho chassis technical drawing

Many mobile robots use a drive mechanism known as differential drive. It consists of 2 drive wheels mounted on a common axis, and each wheel can independently be driven either forward or backward. [32]

The Golfminho Robot drive mechanism results from a combination of simple differential drive and an Ackerman car model [32]. It differs from the tricycle mechanism because the traction wheels are on the rear. Both wheels are mounted on a common axis and each one may be driven either forward or backward at different speeds. The front wheel does not limit the movement but the gang does.

Varying the velocity of each wheel, the robot may turn right or left, rotating about a pivot point known as ICC - Instantaneous Centre of Curvature, lying along their common left and right wheel axis (see Fig. 12). Note that in Fig. 12 the trailer is not represented, but the limitation it imposes is ($\emptyset_{max}$=20°) because it influences the vehicle kinematics.

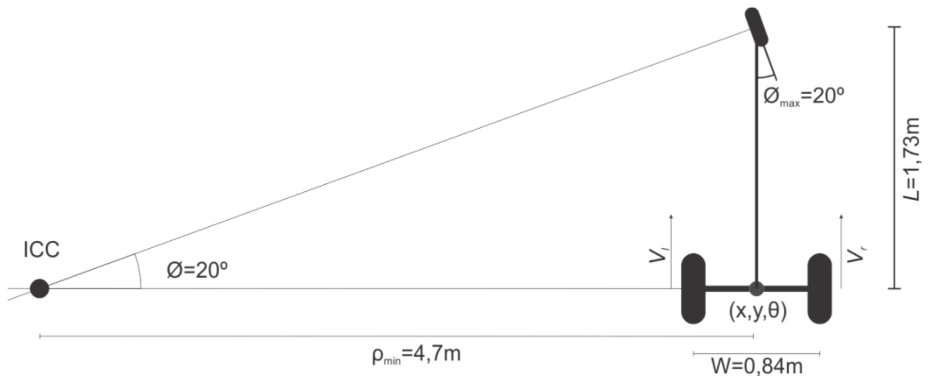


Fig. 12 - Left turn differential drive representation

The maximum Yaw angle is limited to about 20°. This value is used to calculate for the minimum curvature radius (ρ_{min}, see equation 4).

$$\rho_{min} = \frac{L}{tan(\emptyset)} = \frac{1{,}73m}{tan(20°)} = 4{,}7m \tag{2}$$

Note: Ø = Gang Maximum Yaw Angle= 20°

The ICC point is determined by the intersection point of the line along the axis of the front wheel and the line along the axis of the rear wheels (**Error! Reference source not found.**).

The angular velocity, ω, and the curvature radius, ρ, may be calculated solving the pair of equations, being W the width of the rear axis:

$$left\ turn: \begin{cases} \omega\left(\rho + \frac{W}{2}\right) = V_r \\ \omega\left(\rho - \frac{W}{2}\right) = V_l \end{cases} \quad or\ \ right\ turn: \begin{cases} \omega\left(\rho - \frac{W}{2}\right) = V_r \\ \omega\left(\rho + \frac{W}{2}\right) = V_l \end{cases} \tag{3}$$

Solving for ω and ρ, the following expressions are obtained:

$$\rho = \frac{W}{2} \times \frac{V_r + V_l}{V_r - V_l} \qquad \omega = \frac{V_r - V_l}{W} \tag{4}$$

Also the relation between V_r and V_l for any case may be calculated. For the case of Golfminho Robot, this means that when turning left at maximum angle (Ø=20°), $\frac{V_l}{V_r} = 83{,}5\%$.

It is possible to determine the robot next pose (*x',y',θ'*), knowing the velocity of each wheel (*V_l* and *V_r*), the current pose (*x,y,θ*) and the elapsed time *δt*.

The following general equations apply for this type of differential robot [25, 32, 33]:

Next pose for a left turn:

$$ICC = [x - \rho\, sin(\theta),\ y + \rho\, cos(\theta)] \tag{5}$$

At t + δt:

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} \cos(\omega\delta t) & -\sin(\omega\delta t) & 0 \\ \sin(\omega\delta t) & cos(\omega\delta t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x - ICCx \\ y - ICCy \\ \theta \end{bmatrix} + \begin{bmatrix} ICCx \\ ICCy \\ \omega\delta t \end{bmatrix} \tag{6}$$

Next pose for a right turn:

$$ICC = [x + \rho\, sin(\theta), \qquad y - \rho\, cos(\theta)] \tag{7}$$

At t + δt:

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} \cos(\omega\delta t) & \sin(\omega\delta t) & 0 \\ -\sin(\omega\delta t) & cos(\omega\delta t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x - ICCx \\ y - ICCy \\ \theta \end{bmatrix} + \begin{bmatrix} ICCx \\ ICCy \\ -\omega\delta t \end{bmatrix} \tag{8}$$

Three cases are possible for these drives:

1. if $V_l = V_r$, then there is a forward linear motion in a straight line. ρ becomes infinite, and there is effectively no rotation - ω is zero;
2. if $V_l = -V_r$, then ρ = 0, and there is a rotation about the midpoint of the wheel axis - rotate in place;
3. if $V_l = 0$, then there is a rotation about the left wheel. In this case $\rho = \frac{l}{2}$. Same is true if $V_r = 0$.

The fact that the non-holonomic robot is not able to move in the direction of its main axis (e.g. the Golfminho rear axis containing the traction wheels) is referred to as a singularity [16].

A differential drive robot is also difficult to control because a small change in the velocity of one of the wheels causes a significant change in the direction. This is the source of many problems commonly labelled as uncertainty in motion execution. This is a subject that falls out of the scope of this document, but should be taken into consideration at the implementation phase.

2.6. Dubins Car

Non-holonomic robots have therefore an important limitation regarding the maximum turning angle. This kinematic limitation poses an additional problem to the planning algorithm which is of extreme importance, because a given path is only feasible if it complies with the agent physical restrictions. Aware of this problem, researches usually implement simplifications on the robot kinematics in order to achieve higher performance of the planner while maintaining the planning compliant with those kinematic constraints. The Dubins curves are such an example [34].

The basic idea of Dubins curves is to consider that the agent possesses only three possible movements, called primitives, namely move forward, turn left and turn right. When turning left or right the agent can only turn at the minimum curvature radius. Each movement can have a fixed or variable length, depending on the implementation. Fig. 13 shows a reachability map for two steps with a fixed length.

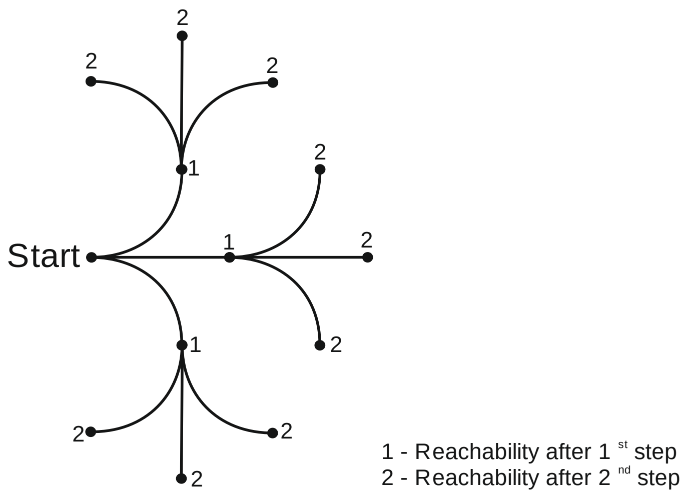

Start
1
2
1 - Reachability after 1 st step
2 - Reachability after 2 nd step

Fig. 13 – Reachable points for two steps using a Dubins Car [17]

Dubins car is further detailed in section 4.11 regarding the implications on the path planning algorithm.

CHAPTER III

PATH PLANNING

Path planning is a challenging task with many different aspects which must be carefully understood and taken into consideration depending on the scope of the application.

Planning has been traditionally considered an offline preparation step before the real action takes place. The sequence of actions is previously computed without a tight time constraint. The execution step attempts to follow the pre-established plan. This works well for theoretical problems, puzzles and most of virtual applications, including simulators, games, and other static defined computer problems.

In real situations, however, there are other important factors to be considered. The controller does not guarantee the exact motion execution due to environment variations, unpredictable situations and overall uncertainty. Thus, it is important to be aware that a perfect planning may result in an unfeasible path due to these real external influencing factors. In such cases re-plan might be needed. For real applications, it is more important to have a good faster planner than a complete and optimal planner, so that, if necessary, the planning step could be reassessed without severe impact on the task performance.

Resuming, there are two typical situations: a static environment and a dynamic environment. On a static environment, obstacles are considered to be static and all relevant information about the environment is completely known *a priori*. Dynamic situations arise whenever only partial information is available for planning. Obstacle location may change and/or may vary along motion.

Dealing with dynamic environments is far more challenging and requires interactive re-planning and re-adjusting.

In the current case-study, for simplification reasons, it is assumed that obstacles are fixed and their location is known. Nevertheless, the algorithm design was carried on considering that high probability exists for the need of re-planning. Note that unexpected obstacles may eventually appear in real situations, and also, the uncertainty[3] intrinsic to sensors, controllers and actuators,

[3] Uncertainty is the lack of exact sensing or actuation. Sensing uncertainty originates on the lack of information about the environment. Actuation uncertainty originates in the lack of accuracy of motors or other actuators. Uncertainty can be further increased by the influence of external "noise".

may cause the agent to deviate from the pre-established planner. Thus, a faster planner is a wiser choice when compared to an optimal planner.

Regarding the algorithm, there are generally two different approaches, based on the criteria of the desired outcome: feasibility and optimality. Feasibility is the property of the resulting plan that makes it suitable to be performed by a machine having intrinsic kinematic, dynamic, or actuation limitations. Optimality refers to the best solution amongst all possible solutions for a given problem. In the case of planning algorithms it is usually carried out by minimising a cost function. It is fairly understandable that the computational time required to retrieve a plan, exponentially increases as the number of possible solutions increases. There are also cases where the number of possible solutions is infinite, so optimality becomes only a delusion.

Feasibility is already a very challenging aspect; achieving optimality is considerably harder for most problems [17].

A feasible path is the one in which efficiency is not taken into consideration. A planner based on the feasibility criteria only computes a path from an initial state to the goal state and frequently does not improve the solution as soon as it finds one. Optimality concerns about efficiency. The output is optimized for some given parameter, besides reaching a goal destination. The most common implementation is to minimise the distance travelled from start to goal states.

"In many applications, it is difficult to even formulate the right criterion to optimize. Even if a desirable criterion can be formulated, it may be impossible to obtain a practical algorithm that computes optimal plans." [17].

This chapter covers the essential concepts about path planning, including how to formulate the problem, what features does a good planner have, what the difference between complete and heuristic planners is and what are the advantages of sampling methods.

3.1. Path planning related concepts

Before starting to present all the other algorithm properties and the planner framework it is important to mention some basic background about general path planning problems.

Next, some important definitions are presented:

Workspace: refers to the environment where the agent operates (W). It could be a planar ($\mathbb{R}^2$) or a three-dimensional space ($\mathbb{R}^3$). The workspace area containing obstacles is represented by WO, being the WO_i the ith obstacle. The free space is given by the set $W_{free}=W\setminus\cup_i WO_i$ (note \ is the subtraction operator);

Configuration: is a possible given state of the agent. The configuration is represented by the relevant information for path planning, namely the value for each of the degrees of freedom. For example, a configuration q ϵ $\mathbb{R}^n$ is written in coordinates as q=(q1, q2, ..., qn);

Configuration Space: is the set of all possible agent configurations Q. (Note: the configuration space is conceptually very different from workspace W.)

Obstacle: is a region in the workspace that must be avoided to prevent collisions. The agent configurations which intersect these regions should not be included in the planner solution. An obstacle in the workspace that intersects a given configuration belongs to the set $QO_i = \{q|R(q) \cap WO_i \neq 0\}$, being R(q) the set of points of the workspace occupied by the agent at configuration q;

Plan: Is a sequence of actions to be taken in order to reach a specific goal configuration q(1) starting from a given initial configuration q(0). For a more complete definition refer to LaValle[4];

Path: is a curve in the free configuration space C_{free} connecting q(0) to q(1). It is a continuous curve on the free configuration space (see Fig. 14). The solution to the path planning problem is a continuous function σ that:

$$\sigma: [0:1] \rightarrow Q \ where \ \sigma(0) = q_{start}, \sigma(1) = q_{goal} \ and \ \sigma(s) \in Q_{free} \ \forall_{s\in[0,1]}$$

[4] *"a plan imposes a specific strategy or behaviour on a decision maker. A plan may simply specify a sequence of actions to be taken; however, it could be more complicated. If it is impossible to predict future states, then the plan can specify actions as a function of state. In this case, regardless of the future states, the appropriate action is determined. Using terminology from other fields, this enables feedback or reactive plans. It might even be the case that the state cannot be measured. In this case, the appropriate action must be determined from whatever information is available up to the current time. This will generally be referred to as an information state, on which the actions of a plan are conditioned."17. LaValle, S.M., Planning algorithms. 2006: Cambridge university press.*

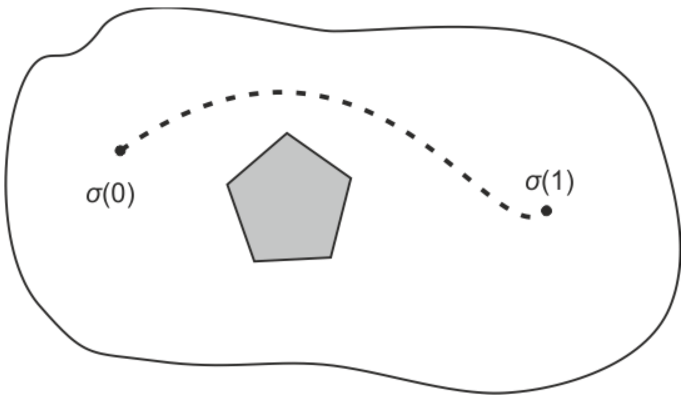


Fig. 14 – Representation of a path from σ (0) to σ (1) by a dashed line.

3.2. Path planning for practical applications

In real situations there are unpredictable external factors (usually called Nature factors) which influence the robot (or other physical agent) behaviour, by changing its speed, direction, balance or any other parameter. Robots acting in real environments, especially in uncontrolled external ones, suffer these influences. These are commonly referred as dynamic environments which are intrinsically uncertain, immediate, and complex.

It would be logical to think that any kind of planning would fail in practice because of the above mentioned unknown factors. Nevertheless, it is important to have a previous plan that can be altered and adjusted as the robot faces real adversities. Such planning, targeting real situations should be fast, meaning that it should deliver a plan within a reasonable short amount of time depending on the application. For the present case study, it would be considered very long a planning time of an hour, and it would be considered very fast a planning time of less than one or two minutes.

The planning phase is absolutely necessary to guarantee high performance of the task. In case the robot succeeds implementing the plan, the mission is perfectly accomplished, but in case the real robot deviates from the plan, then a fast re-plan may be invoked to accomplish the task by finding an alternative way which optimizes the output under the new circumstances.

For the purpose of field robotics (robots operating in external dynamic environments), researchers argue that the planning and the execution phases should run in parallel, to enable real-time adjustments, opposite to traditional AI approach [35].

As previously stated, dynamic environments are characterized by the fact that preceding information about obstacles is absent and/or may vary along time. The goal is usually known or at least recognizable, but one may also consider a moving goal. To overcome that additional difficulty, path planning in dynamic environments usually employ a parallel real time planning and executing phase strategy.

Concerning the present case-study, it is assumed that obstacles are known *a priori* and do not move, as well as the goals. However it cannot be considered a static environment, because the plan may have to be re-built as the robot deviates from the pre-established plan due to uncertainty or in cases where an unpredictable obstacle shows up on the field. It is in fact a dynamic environment, that can change but it is not expected to change frequently. By consequence, it is reasonable to accept a planning strategy that works in serial mode with the execution phase, but still produces fast results whenever necessary.

"Dynamic path planning is problematic because the path needs to be continually recomputed as new information becomes available. Continuous navigation is possible if navigational directions are provided during the computation of the plan, or the computation is faster than the rate of change in the environment." [35]

3.3. Path planning problem formulation

The motion planning objective is to find a feasible sequence of motions which drives the robot from the initial to the final configuration, while avoiding obstacles. Usually there is a cost function (or reward function) associated with that sequence that should be minimised (or maximised).

According to Canny and Reif, a motion planning problem falls in one of three categories: shortest path, dynamic motion planning, and motion planning in the presence of uncertainty [36].

It is out of the scope of this thesis to address uncertainty but it should be clear that it is absolutely necessary to take uncertainty into consideration when real field robots are to be controlled. For the purpose of simplicity let us consider uncertainty as a sub-problem that does not interfere with the planning phase, in this application.

The problem which is being addressed is a type of planning problem and has all the ingredients of a motion planning problem adding the kinematic constraints. There are two significant differences: first, there is not one single goal (singular or region) but a large set of non-mandatory goals and second, the initial and final configurations are exactly the same. These two important differences make every algorithm currently available in literature either unfeasible, or not optimal, or at least not adequate for the current application. As presented later on this document (see section 4.2), significant adaptations of current state-of-the-art algorithms were performed to enable a feasible solution for this problem and to originate the TWIN-RRT* algorithm. The general motion planning problem definition given bellow still applies for the current case, where the initial and final configurations are the same, although special care should be taken on the definition of the cost function and restrictions.

Problem Definition:

Given a compact set $X \subseteq \mathbb{R}^n$, refered to as configuration space, comprising all possible configurations x, let the open sets X_{obs} be the obstacle region, and X_{goal} the goal region.

The set X_{free} is called the obstacle-free region and is defined as $X_{free}=X \setminus X_{obs}$.

A path is a continuous function in X given by $\sigma: [0,1] \rightarrow X$. The path σ is collision-free if $\sigma(\tau) \in X_{free}\ for\ all\ \tau \in [0,1]$. The set of all collision-free paths is denoted by $\sum_{free}$.

Let x_{init} be the initial configuration and x_{goal} the final configuration. These configurations may be the same or different, but they must rely on the free-space X_{free} for the plan to be feasible.

The planning problem for the current case-study is to find a collision-free path $\sigma: [0,1] \rightarrow X_{free}$ that starts at the initial configuration $\sigma(0) = x_{init}$ and reaches the goal configuration $\sigma(1) = x_{goal}$. It is also admissible to have a region of possible goal configurations, thus instead of a unique goal configuration x_{goal}, an admissible goal region $\sigma(1) = X_{goal}$ will be used.

Considering a cost function $c: \sum_{free} \rightarrow \mathbb{R}_{\geq 0}$ to be a non-negative cost relative to a given collision-free trajectory, the optimal solution for the motion planning problem is given by the collision-free path $\sigma^*: [0,1] \rightarrow X_{free}$ that minimises the cost function $c(\sigma^*) = inf_{\sigma' \in \sum_{free}} c(\sigma')$.

Let us assume that the cost function is the total path distance. This should work fine except for the case where the initial and final configurations are exactly the same. In such case, the robot does not even have to move. Since initial and final configurations are the same, then the cost function has to be redefined to accomplish the task of reaching multiple targets (or waypoints) before reaching the final goal configuration.

Redefining the cost as efficiency (see equation 2), the objective changes from minimising the distance to maximising the efficiency. But this redefinition by itself does not solve all the problems, because in the case where no distance is travelled and no target is reached, the efficiency will become indeterminable (see equation 11).

$$\boldsymbol{Efficiency} = \frac{\mathbf{0}}{\mathbf{0}} = \boldsymbol{ind.} \tag{9}$$

It is necessary to add a condition, which might be, for example, to travel at least 1m.

Other restrictions may apply, which are presented on chapter IV.

3.4. Planning towards multiple targets

Planning towards multiple targets is not quite a motion planning problem. Instead, it resembles more an optimization problem. Nevertheless, when real dynamic constraints are to be considered, as well as the need for real-time re-planning in real environments, the problem of planning towards multiple targets inherits most of the motion planning properties.

Optimization problems are usually addressed by operations researchers [18, 37, 38] and deal with static environments, where information about the agent, obstacles, goal and environment is known *a priori* and does not change.

Motion planning researchers have mostly developed solutions for point to point applications [21, 39-45]. In this field there are hundreds of different approaches. Several algorithms and variations have been developed to address different requirements.

Researchers did not pay much attention to multi-target applications for many years. Recently there has been some interest in designing solutions to multiple targets mainly for UAV (unmanned aerial vehicle) applications [46, 47]. Multi-target planners have successfully been adapted from D* algorithm [48, 49]. Efficient motion planners for sequencing targets based on Lin-Kernighan (LK) algorithm for industrial robots applications have recently been developed [50].

"Most previous robot motion-planning approaches assume that a simple, specific origin-to-destination configuration is given to each agent, so that their main focus is usually on the optimal path generation between two predefined positions" [46]

A common example of an optimization problem is the Travelling Salesman Problem (TSP). It concerns a starting location and a set of markets. The salesman should visit every market by travelling the shortest path possible. A very large number of variations exist for this type of problem, some of them with strong similarities with the current case-study problem, like the TSP with profits.

TSP related problems have successfully been solved by GA (Genetic Algorithms), ACO (Ant Colony Optimization), heuristic algorithms, simulation annealing algorithm, tabu search algorithm, etc. Optimal solutions for TSP related problems work well for a relatively low amount of targets but are very time-consuming for high number of targets [51, 52].

Planning for hundreds or thousands of targets is unfeasible using graph based algorithms. Optimality in such cases is also impracticable because optimal algorithms will never reach a solution within reasonable time for a real application like the case-study presented here.

On the other hand, these types of solutions do not usually take into account the kinematic restriction, which makes them unsuitable for the present case-study. A motion planning related strategy is therefore the most promising approach.

3.5. Planning towards non-compulsory targets

Planning for a set of targets is a challenging task [53]. The case-study refers to a situation where there are multiple targets but none of them is considered indispensable. The fact that targets are not compulsory may provide more feasible solutions but in another way it adds complexity to the algorithm design. When there is no obligation for the agent to meet a specific target or a set of targets, then the main question is how to choose the best target or set of targets. Defining in advance a target or set of targets will definitely bias the process and will not guarantee optimality convergence. Moreover, the interference of obstacles and the agent kinematic constraints should also be included in that process, what makes it even more difficult to solve. The approach followed in this thesis is to not choose any targets at all. The strategy adopted follows an inverse approach. The algorithm should have the ability to find a feasible path and incorporate as many targets as possible. Feasible paths are compared and the most efficient one is selected throughout the process.

Before describing in detail what the solution is, a brief overview of the most significant existing algorithms is presented next. They solve somewhat related problems and inspired the development of the new algorithm.

3.6. Existing methods

Problem solving algorithms include classical graph search algorithms (Leiserson et al., 2001) like "Breath first" and "Depth First", "Dijkstra's algorithm"(Dijkstra, 1959), A* (Fikes and Nilsson, 1971), "Best first", "Iterative deepening" and others (Pearl, 1984).

These algorithms were originally developed to address problems confined to virtual, static, well known environments. Later, came up the, so called, motion planning algorithms, which address real, unknown environments, kinematic and dynamic constraints, uncertainty and other factors.

Bug Algorithms [54] were inspired after Tom's work [55] and consist of simple sensor-based planners that consider the robot a single point in a planar 2D space. The basic idea is to follow a straight line towards the target and whenever an obstacle is found on the way, the robot circumvents it. The tangent bug algorithm [56, 57] is an extension that considers a 360 degree sensor for robot orientation.

Bug algorithms are limited to 2D applications. To address problems where multidimensional and non-Euclidean configuration spaces are involved, other methods were developed. These include potential functions, such as gradient descent algorithms. The main problem with this kind of algorithms is that they "suffer" from the local minima "syndrome". This means that these algorithms will never find the goal configuration if they become trapped in a local minima configuration. Gradient or attractive/repulsive techniques are for that reason not complete, and therefore reaching the goal is not guaranteed.

The Randomized Path Planner (RPP) proposed by Barraquand and Latombe [58] is an algorithm that usually succeeds to escape from local minima by using random walks.

Roadmaps

Canny presented for the first time the roadmap approach applied to motion planning [59]. He presented later the Opportunistic Path Planner (OPP) which generalizes his original work [60].

However, roadmaps based on Canny work rely on the explicit representation of the free configuration space which is in many cases unfeasible, especially when the robot has several degrees of freedom [16].

Sampling based algorithms were developed to solve path planning problems within reasonable time and without the need of explicitly representing the free configuration space.

Sampling Based algorithms

The first sampling-based planners [18, 61-63] opened the way to the well-known Probabilistic Roadmap planner (PRM) [19]. The PRM algorithm creates a roadmap in the free configuration space. It is a two queries process. First, it builds a roadmap and then it returns the path from any point to another by computing the smallest distance using the previous roadmap established.

PRM proved to be effective in solving several problems [18, 38, 64-67] by ensuring also the probabilistic completeness [66, 67].

Expansive-Spaces Tree planner (EST) has proven to be appropriate for single-query planning and shows very good experimental performance [68, 69].

Rapidly-exploring Random Tree (RRT) planner is able to find fast solutions even for many degrees of freedom and different constraints [6, 70].

Sampling-Based Roadmap of Trees (SRT) planner combines both EST and RRT to construct a PRM like roadmap of single query-planner trees [71, 72].

The above mentioned sampling-based planners are just some of many recently developed planners. Many variations exist for each of these planners in order to deal with kinematic constraints [6, 64, 73-76], dynamic environments [77, 78], real-time planning [79], terrain constraints [80], the presence of uncertainty [81], cooperative tasks [82, 83] and others.

One common problem of the above mentioned sampling-based planners is that after they succeed finding a feasible path, they do not further improve it.

Recently RRT*, an asymptotically optimal planning algorithm [2, 5, 84] was proposed to overcome that limitation.

Despite the multitude of variations, none of the known algorithms provide a solution on how to deal with high number of multiple targets and how to do it in an efficient way considering in particular that all targets are non-mandatory targets. The additional requirement that the final configuration should match the initial configuration, by itself rends most of the above stated algorithms inapplicable unless significant alterations are provided.

3.7. Features of a good planner

Completeness, optimality and reduced computational complexity are the three major desired features of a planner.

Completeness

Completeness refers to the capability of a planner to return a solution in a finite amount of time whether there is one, given any input. In other words, a planner is complete if, in case there is a feasible solution it will retrieve it sooner or later. Usually a complete path is the one that explores all the possible combinations until if finds a solution. The methods that explore the different combinations are called combinatorial.

Combinatorial methods usually are considered to be complete, but sampling-based methods do not. In this case, a weaker notion of completeness is adopted: resolution completeness or probabilistic completeness.

A resolution complete algorithm will find a solution in finite time whether there is one, but it may run forever if there is not.

Most of sampling-based algorithms use random sampling, thus they are dense (see section 3.10 for a definition of denseness) with probability one. The probability of such algorithms to find a solution converges to one as the number of samples increases, thus are called probabilistic complete[5] [17].

---

[5] An algorithm ALG is probabilistically complete if (2. Karaman, S. and E. Frazzoli, *Sampling-based algorithms for optimal motion planning.* The International Journal of Robotics Research, 2011. **30**(7): p. 846-894.):

$$\lim_{n\to\infty} inf\ P\left(\left\{\exists x_{goal} \in V_n^{ALG} \cap X_{goal}\ such\ that\ x_{init}\ is\ connected\ to\ x_{goal}\ in\ G_n^{ALG}\right\}\right) = 1$$

where a graph G=(V,E) on X is composed of a vertex set V and an edge set E.

Optimality

Optimality is the feature that allows an algorithm to progressively adjust the solution by converging to the optimal solution. Optimality guarantee is only given by complete algorithms. When an algorithm is not complete, then usually the expression *asymptotically optimal* [6] applies, meaning that although it is not possible to test all the possibilities, the solution will always tend to the optimal solution as the number of iterations increase [2].

Computational complexity

Computational complexity refers, in a simplistic manner, to the time or space necessary to find a solution for the problem. Usually, time is not used because the processing time changes as the computing systems evolve. Instead, the number of operations, *O*, is used as a metric for time complexity and is indexed to the number of inputs. For example, the class P categorizes all the problems that are solvable in polynomial-time, i.e., the algorithm runs in $O(n^k)$ operations for some integer k, being n the number of inputs.

Similarly, considering space complexity, the class PSPACE refers to problems that require a polynomial amount of space during the execution of the algorithm.

$$P \subset NP \subset PSPACE \subset EXPTIME$$

When only the lower bound for a problem is known it can be referred as X-hard, being X any of the classes, for example NP-hard. When the upper bound is known, then it is called X-complete, for example, NP-complete.

The general motion planning problem was shown to be PSPACE-complete [59].

In AI, an efficient algorithm is the one that solves a particular problem in polynomial time, thus belonging to class P.

The way a problem is described defines, most of the times, its complexity. Weakening requirements and tolerating completeness (probabilistic completeness for example) are ways to craft more efficient algorithms [17].

[6] An algorithm ALG is asymptotically optimal if: $P(\{\lim_{n\to\infty} \sup Y_n^{ALG} = c^*\}) = 1$
Where c* is the robustly optimal solution with finite cost, for the cost function Y (2. Ibid.).

3.8. Complete planners

Complete planners are of little use when dealing with more than 3 DOFs (degrees of freedom) because of their combinatorial complexity. Sampling-based planners can be used with many DOFs but they are not complete. Instead, they are probabilistically complete when the sampling is random, i.e., they will eventually find a solution if one exists, but will run forever otherwise. If sampling is deterministic, then it is called resolution complete depending on the sampling resolution.

The Randomized Path planner – RPP was shown to be probabilistically complete [85, 86] as well as the Probabilistic Roadmap - PRM planner [66, 67, 73, 87, 88].

Rapidly Exploring Random Trees – RRT were also proven to be probabilistically complete even in the presence of kinodynamic constraints [6].

Path planning research usually foregoes obtaining complete solutions and privileges feasible solutions that satisfy imposed constraints.

3.9. Heuristics in path planning

A heuristic is a way of biasing an algorithm towards a faster solution, but without guaranteeing optimality. Heuristics are commonly used in path planning in order to promote faster or better results in some way. These have proven to be very valuable in combinatorial path planning. A good example is the comparison between the A* (a heuristic driven algorithm) and Dijkstra algorithms (from which A* was derived).

In sampling-based planners heuristics have proven to be also valuable in achieving faster results. It is possible to bias the sampling in RRT towards the goal and increase the rate of convergence of the planner [89, 90]. The trade-off of this strategy, in case of RRT for example, is the loss of some of the exploration feature of the algorithm.

3.10. Advantages of sampling methods

Sampling-based methods are extremely successful in dealing with many DOFs and do not require explicit representation of the configuration space or obstacles in the workspace.

The motion planning problem is very hard from the computational perspective. It has been shown to be PSPACE complete which makes the use of complete algorithms (which are combinatorial) unfeasible in practice for many DOFs and/or other constraints.

Sampling-based algorithms check for collision every time a new sample is taken, instead of using an explicit representation of the environment. Samples, which are allowed configurations of the agent in the environment, are then connected to construct a roadmap.

To further understand the importance of sampling it is important to understand the background properties: denseness, uniformity, regularity and dispersion.

Denseness

Let U and V be any subsets of a topological space. The set U is said to be dense in V if $cl(U) = V$ (where cl means closure of a set). This means adding the boundary points to U produces V. A simple example is that $(0, 1) \subset \mathbb{R}$ is dense in $[0, 1] \subset \mathbb{R}$. Another example is that the set $\mathbb{Q}$ of rational numbers is both countable and dense in $\mathbb{R}$ [17].

A random sequence is said to be probably dense.

Uniformity

A sampling strategy is said uniform if it generates evenly spaced samples in the configuration space. An example of uniform sampling is the van der Corput sequence [91].

Regularity

Regularity is a property somewhat related to uniformity but has a stronger meaning. A regular sampling is obtained deterministically. Random sampling generates uniform samples but cannot generate regular samples. That would contradict the random property by definition.

In fact, the nature of the random numbers generation is per se an important concern of sampling methods. Usually a pseudo-random generation is used in computer programs. One has to be careful about the regularity of the samples. There are methods that test the quality of the random procedure such as the chi-square test. The irregularity of samples is inherent to random sampling methods and is well represented as Voronoi diagrams [63].

Dispersion

Dispersion refers to the idea of having samples such that the largest uncovered area would be as small as possible [17].

This definition is backward intuitive. A lower dispersion value means that points are better dispersed. It should be understood as, the lower the value, the better the dispersion.

Resuming, a sampling method should produce a dense and uniform but not a regular set of samples. The dispersion should therefore be as low as possible.

3.11. The RRT approach

The most significant sampling-based algorithms in literature are PRM and RRT. What distinguishes them is the fact that PRM is a multi-query while RRT is a single query planner. The idea of multi-query is useful when several paths (using possibly different starting and goal configurations) are to be computed using the same immutable environment (constant targets and obstacles locations). Hence, most of the information gathered to compute a first path can be used to find a second path on the same known environment. The multiple-query approach requires a pre-computational time to establish a roadmap. Queries at a second stage may then be answered much faster [18, 19, 65].

The PRM has been widely used for problem solving on different applications. These include amongst others, planning for multiple car-like robots [92, 93], planning in high-dimensional configuration spaces [18], planning for linkages with closed kinematic chains [75], narrow passages [94] and changing environments [38].

Although it performs very well in many cases, multiple-query algorithms require, in some cases, a fair amount of pre computational time. Also, they do not suit the needs of real dynamic environments with constant changing of obstacles for example. To satisfy those needs many single-query methods were developed [7, 61, 69, 95, 96].

From these, RRT emerged as the most interesting one, being applied in real time applications [3, 97], dynamic environments [98], manipulation robots [5], autonomous driving [21], kinodynamic and non-holonomic planning [7, 99-101] and others [76, 102, 103].

3.12. Optimality in sampling based planners

Developing an efficient algorithm implies to be concerned about optimality. Computing optimal motion plans is a very challenging problem even in simple cases [104]. Sampling based algorithms have proven to be a very intelligent choice in terms of computational expenditure although compromising the completeness [17, 33, 105, 106]. Recently Karaman and Frazolli [107] presented improved versions of two of the most known sampling based algorithms, PRM (Probabilistic Roadmaps) [106] and RRT (Rapidly Exploring Random Trees) [108]. They presented asymptotically optimal incremental sampling algorithms, which means, that they approach the best solution as the number of iterations grows up. The RRT has the advantage over PRM that does not need a learning phase [17]. The RRT*[7] has been reported to be very effective even when applied to robots with several DOF (Degrees of Freedom) [3, 109].

RRT* was chosen as a starting point for its asymptotical optimal feature and also for the absence of a pre-computing phase. It does not require prior tuning or any other preparation step.

In the next chapter a detailed explanation on how the RRT* was modified to retrieve a solution for the problem of the case-study described in section 1.1 is presented.

[7] The * stands for optimal as conventioned in AI literature.

CHAPTER IV

PLANNING TOWARDS MULTIPLE TARGETS WITH TWIN-RRT*

Planning towards multiple targets where none of them is mandatory is the problem to be solved. This chapter presents the new TWIN-RRT* algorithm which aims to accomplish the objective of retrieving an efficient path which reaches at least one target before reaching the final configuration. The target or set of targets to be reached is not defined beforehand and the algorithm searches and selects and changes the best set of targets in order to maximise the efficiency.

Before stepping into the details of the efficiency of the algorithm itself and the fundamental ideas of path planning, a more concrete explanation of the metric used for evaluating the performance of the new algorithm is presented.

4.1. Detailed problem definition

Referring back to the original problem stated in 1.4, it is important to refine it before going into more detail about the planning algorithm.

Given a set of targets located at different positions, the problem is how to navigate through at least a given amount of targets (but not a defined set, which means, the choice of the targets is not given beforehand) on a limited, well defined area, while maximising the efficiency.

Efficiency is generally related to the energy necessary to accomplish a given task. Usually the less energy spent on performing the same task, the more efficient the process will be.

The metric to use for evaluating the algorithm's performance was defined in section 1.4 as efficiency. There are several ways to define efficiency, depending on the problem and on the type of solution to achieve. In this case efficiency was defined as:

$$Efficiency = \frac{Number\ of\ reached\ targets}{Distance\ travelled} \left(\frac{targets}{m}\right) \tag{10}$$

Using this definition it is possible to evaluate the algorithm's performance, but it is necessary to use a reference or another algorithm to compare with. Since there are not any other algorithms that solve this problem, it was decided to use a standard reference. The reference is the case when a robot completely sweeps the total workspace, visiting every point only once and finding all the existing targets.

This enables the establishment of a more useful and tangible metric: the efficiency gain.

Efficiency gain: is the measure of improvement relative to the standard complete sweep strategy. It is defined as:

$$\boldsymbol{Efficiency\ gain} = \frac{best\ Path\ efficiency\ (targets/m)}{complete\ sweep\ efficiency\ (targets/m)} \tag{11}$$

The sweep strategy was selected as a reference because it is well determined in terms of efficiency. One cannot use, for example, a random strategy because sometimes it may perform better than a complete sweep and other times may perform worse.

The case defined as having $Efficiency\ gain{=}1$ corresponds to the complete sweep strategy.

Using the efficiency gain as metric it is easily observable that any value equal or below 1.0 is a bad plan. It is not better than doing a complete sweep of the entire field that does not require any complex calculation. This metric does not depend on the number of targets on the field or on the size of the field. It can be used to compare the performance of the algorithm in different cases with different number of targets and different field sizes. It would also enable comparison with other algorithms should they exist.

Restrictions may apply in the form of minimum number of targets, minimum distance travelled or maximum time spent, among others. A necessary condition is that the initial and final positions are the same. Although this is not a limiting condition for the algorithm itself, it means that the algorithm should be compatible with this requirement. In other words, the algorithm may work with initial and final configurations different from each other, but still must be able to retrieve feasible paths when initial and final configurations are exactly the same. This is an important requirement since most of path planning algorithms do not retrieve any path when initial and final configurations are the same.

It is assumed that targets location is given a priori. See the Hybrid Localisation System described in 2.4 for more details.

Restrictions may vary on value and/or type. The number of targets to reach may be restricted to a minimum, the duration of the path may be limited to a maximum and the time spent in calculation may also be restricted to a maximum value. Many other restrictions could be implemented as further detailed in chapter 5.

4.2. TWIN-RRT* in detail

The TWIN-RRT* algorithm comprises two trees growing from the same initial configuration. As the initial and final configurations are exactly the same, the TWIN part of the name reflects the fact that both trees are "born" from the same starting configuration. Each tree grows having the same characteristics of the RRT* algorithm. One is the forward tree (Ta) because the agent will follow the path starting from the root, through a set of nodes towards a given node and the other tree is the backward tree (Tb) because the agent will navigate from a specific node, through other nodes towards the root (Fig. 15). A complete path (curved loop) is formed whenever the two trees connect and the obtained path complies with the imposed restrictions.

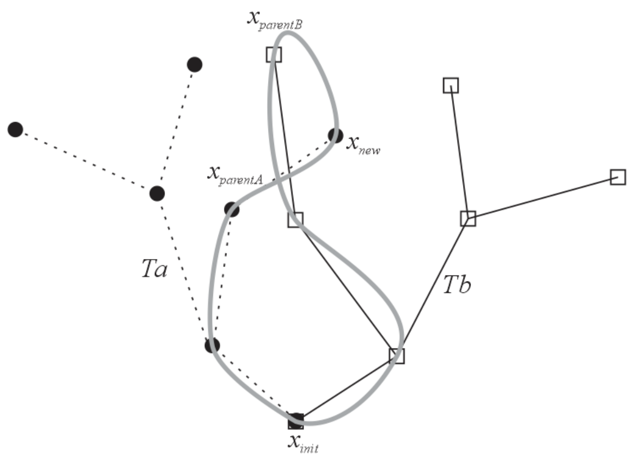


Fig. 15 – TWIN-RRT* algorithm considers two trees growing from the same initial configuration (x_{init}).

The final generated path complies with the kinematic constraints, by using Dubins Curves [34, 40, 44, 110, 111]. It is assumed for simplicity that all curves are made at a fixed turning angle, so the curvature radius is always ρ_{min}.

The Twin-RRT* efficiency may be compared to the complete sweep strategy as shown on Fig. 16. Given a set of 4 targets on a field the sweep strategy enables the agent to reach all the targets by covering every point in the workspace. The sweep strategy efficiency for the example bellow is 4 targets / 100m which equals 0.04. The TWIN-RRT* strategy may not reach all targets but is able to find a more efficient path. In this case the efficiency is 0.07 which is 50% better than the complete sweep strategy.

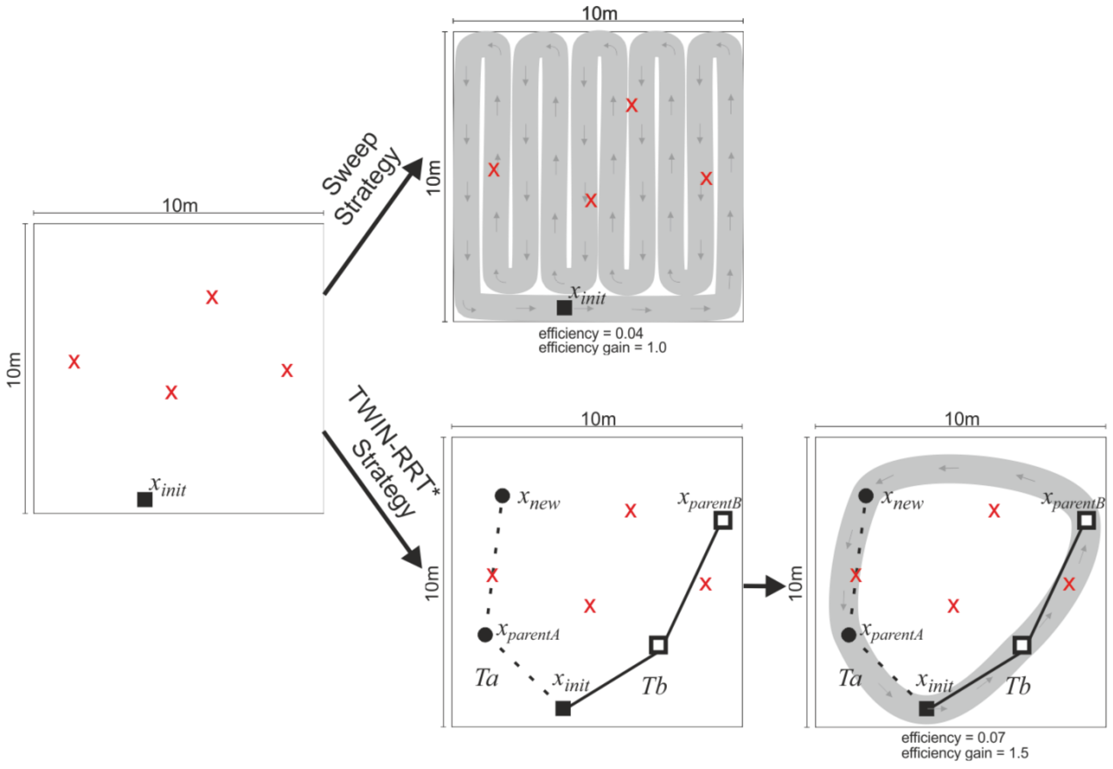


Fig. 16 – Comparison between the sweep strategy and TWIN-RRT* strategy

Fig. 17 represents the evolution of the TWIN-RRT* in more detail. Targets locations are omitted for simplification purposes. Every configuration, except the initial and final configurations are randomly sampled throughout the process. The values and paths of the drawing do not correspond to real data. They were adopted for the sole purpose of better explanation of the algorithm evolution.

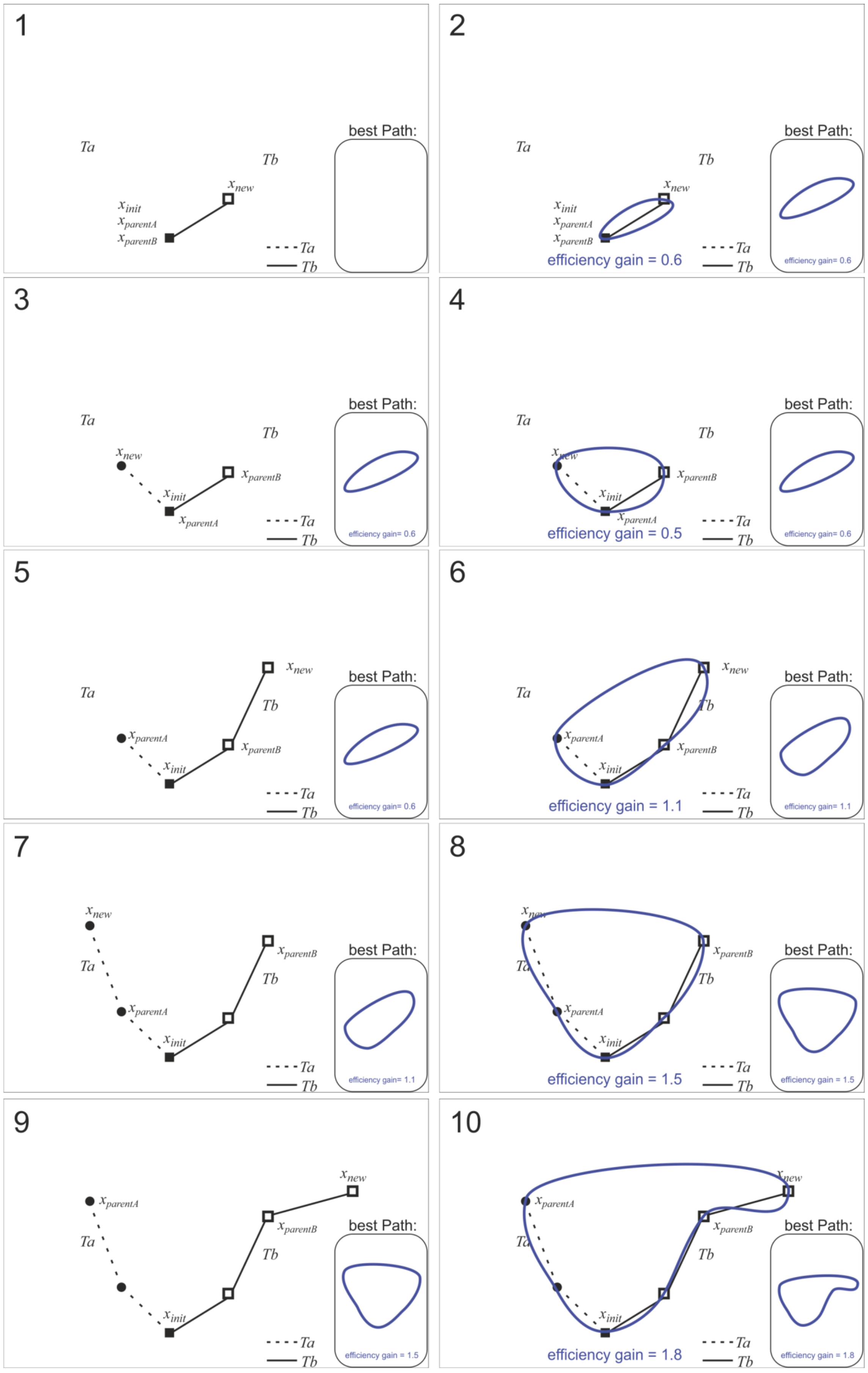


Fig. 17 – TWIN-RRT* concept representation

It starts with two root nodes, $x_{parentA}$ and $x_{parentB}$ having the same configuration x_{init}. If a sampled configuration is allowable (i.e. produces a robot pose which is free of obstacles) then a node containing that configuration is generated and added to one of the trees no matter if there are targets or not. In step 1, a new configuration x_{new} is sampled and connected to one of the trees (Tb in this case). Then the algorithm evaluates if it would also be possible to connect x_{new} to the other tree (Ta). If so, as shown in step 2, a complete closed loop path is generated and the efficiency gain is calculated for that path. Since before there was none feasible path, this path becomes the *"best Path".* Note that x_{new} is only connected to one of the trees. A given node cannot be connected simultaneously to both trees, although there could be another node with the same exact configuration connected to the other tree.

In step 3 the process repeats: a new node x_{new} is generated an added to Ta. In step 4, another closed loop path is generated by linking the two trees at x_{new} node. Its efficiency gain is calculated and compared to the previous *"best Path".* In this case the new path is not better than the previous one, so the "best Path" does not change.

In step 5 a new node x_{new} is generated an added to Tb. In step 6 the closed loop path has an efficiency gain of 1.1 which is better than the previous *"best Path"*, so the "best Path" becomes this one.

In step 7 x_{new} is added to Ta and in step 8 an even better path is found substituting the previous *"best Path".* The process goes on and every time a better path is found, it is saved as *"best Path".* In the example of Fig. 17, after steps 9 and 10, the best path has an efficiency gain of 1.8, which means that this path is 1.8 times more efficient than doing a complete sweep of the field.

The idea of having two or more trees is not new. RRT-connect algorithm [108] uses two trees to improve the performance of the RRT algorithm. TWIN-RRT*differs from RRT-connect in the way that both trees have the same root (x_{init}), and instead of the original RRT, it uses the RRT* which adds the asymptotic optimality feature. The basic idea behind the TWIN-RRT* is that whenever a new sample is added to one tree, it tries to connect to the other tree closing a complete loop. That loop constitutes a full path and it is then evaluated concerning its efficiency. The efficiency is further improved whenever a better path is found. The planner continues to search for better possible paths regarding the efficiency until one of the termination conditions occurs (for example, time limit). The new proposed algorithm, Twin-RRT*, is summarized as follows:

Algorithm 1: Twin RRT* (x_{init})

```
1  $Ta.init(x_{init});Tb.init(x_{init});$
2  for $i = 1$ to $N$ do
3      $x_{new} \leftarrow Sample(i);$
4      $X_{near} \leftarrow Near(Ta,x_{new});$
5      if $X_{near} = \emptyset$ then
6          $X_{near} \leftarrow Nearest(Ta,x_{new});$
7      $L_{near} \leftarrow PopulateSortedList(X_{near},x_{new});$
8      $x_{parent} \leftarrow FindBestParent(L_{near},x_{new});$
9      if $x_{parent} \neq NULL$ then
10         $Ta.add(x_{new});$
11         $Ta \leftarrow RewireVertices(Ta,X_{near},x_{new});$
12         $X_{near} \leftarrow Near(Tb,x_{new});$
13         if $X_{near} = \emptyset$ then
14             $X_{near} \leftarrow Nearest(Tb,x_{new});$
15         $L_{near} \leftarrow PopulateSortedList(X_{near},x_{new});$
16         $x_{parent} \leftarrow FindBestParent(L_{near},x_{new});$
17         if $x_{parent} \neq NULL$ then
18             $NewPath \leftarrow CheckPath(x_{new}, x_{parent}, Ta, Tb);$
19             $BestPath \leftarrow ChechBestPath(NewPath, BestPath);$
20     $Swap(Ta,Tb);$
21 Return $BestPath;$
```

The *PopulateSortedList, RewireVertices and FindBestParent* sub-routines may also be found elsewhere [109] but for convenient reference they are transcribed here.

Algorithm 2: PopulateSortedList (X_{near},x_{new})

```
1 $L_{near} \leftarrow 0;$
2 for $x_{near} \in X_{near}$ do
3     $\sigma_{near} \leftarrow Steer(x_{near},x_{new});$
4     $c_{near} \leftarrow Cost(x_{near})+Cost(\sigma);$
5     $L_{near}.add((c_{near},x_{near},\sigma_{near}));$
6 $L_{near}.sort();$
7 return $L_{near};$
```

Algorithm 3: FindBestParent (L_{near},x_{new})

```
1 for $(c_{near},x_{near},\sigma_{near}) \in L$ do
2     if $CollisionFree(\sigma_{near})$ then
3         return $x_{near};$
4 return NULL;
```

```
Algorithm 4:RewireVertices $(E,L_{near},x_{new})$
1 for $(c_{near}, x_{near}, \sigma_{near})$ € $L$ do
2     if $Cost(x_{new}) + c(\sigma_{near}) < Cost(x_{near})$ then
3         if CollisionFree $(\sigma_{near})$ then
4             $x_{oldparent} \leftarrow Parent(E, x_{near})$;
5             $E.remove((x_{oldparent}, x_{near}))$;
6             $E.add((x_{new}, x_{near}))$;
7 return E
```

The following code excerpt shows how the implementation of the main segment of algorithm 1 (lines 3 to 29) using C++ language looks like. The complete source code is available in the accompanying digital media support.

```
///iteration.hpp
using namespace std;
template<class State, class System>
int RRTstar::Planner<State, System>
::iteration (int TREE) {
    Vertex<State,System>* vertexParent = NULL; /// xparent
    Vertex<State,System>* vertexNew = new Vertex<State,System>;
    Vertex<State,System>* vertexNewClone = new Vertex<State,System>;
    vertexNew->state = new State;
    vertexNew->parent = NULL;
    KdTree *kdtree1,*kdtree2;

    if (TREE == TREE_FORWARD) { kdtree1  = kdtreeForward; kdtree2 = kdtreeBackward;}
    else {                      kdtree1  = kdtreeBackward; kdtree2 = kdtreeForward;}

    // 1. Sample a new state
    system->sampleState (vertexNew->getState(), rho, currentIteration, simCounter); /// Sample(i)

    // 2. Compute the set of all near vertices
    std::vector< Vertex<State,System>* > vectorNearVertices; /// Xnear
    getNearVertices (vertexNew->getState(), vectorNearVertices, kdtree1, TREE);  /// Near(Ta,xnew)
    // The vectorNearVertices contains all the elements within the ball radius

    // 3. Find the best parent and extend from that parent
    if (vectorNearVertices.size() == 0){

        //3.a Extend to the nearest
        if (getNearestVertex (vertexNew->getState(), vertexParent, kdtree1) !=SUCCESS){ /// Nearest (Ta, xnew)
            delete [] vertexParent;     //free the memory of
            return FAILURE;
        }else{
            vectorNearVertices.resize(1);
            vectorNearVertices[0]=vertexParent;
            if   (findBestParent   (vertexNew,   vectorNearVertices,   TREE,   FIRST_CONNECTION)   !=   SUCCESS){
if (DB40) cout << "ERROR after Nearest Vertex: findBestParent Failed!"<<endl;
                return FAILURE;
            }
        }
    }else {

        // 3.b Extend the best parent within the near vertices
        ///FIRST_CONNECTION
        if (findBestParent (vertexNew, vectorNearVertices, TREE, FIRST_CONNECTION) != SUCCESS){
            return FAILURE;
        }
    }

// 3.c Add the vertex to the tree
    insertIntoKdtree(*vertexNew,TREE==TREE_FORWARD ? kdtreeForward : kdtreeBackward);
    if (TREE==TREE_FORWARD){numVerticesForward++;}else numVerticesBackward++;

    // 4. Rewire the tree
    if (vectorNearVertices.size() > 0)
       rewireVertices (*vertexNew, vectorNearVertices, TREE);  /// Ta <- RewireVertices (Ta, Xnear, xnew)

    /// Extend to the other tree
    // 5. Compute the set of all near vertices in the other tree
    vectorNearVertices.clear();
    vertexNewClone->state=&(vertexNew->getState());
    getNearVertices (vertexNewClone->getState(), vectorNearVertices, kdtree2, TREE == TREE_FORWARD ?
TREE_BACKWARD : TREE_FORWARD);
    vertexParent = NULL; // reset the vertexParent because a new Parent is to be calculated for the other tree
    if (vectorNearVertices.size() == 0) {

        // 5.a Extend the nearest
        if (getNearestVertex (vertexNewClone->getState(), vertexParent, kdtree2) !=SUCCESS){
            delete [] vertexParent;     //free the memory of
            return FAILURE;
        }
        vectorNearVertices.resize(1);
        vectorNearVertices[0]=vertexParent;
        if (findBestParent (vertexNewClone, vectorNearVertices, TREE, SECOND_CONNECTION) != SUCCESS){
            return FAILURE;
        }
    }else{

        // 5.b Extend the best parent within the near vertices
        /// xparent<-FindBestParent
        if (findBestParent (vertexNewClone, vectorNearVertices, TREE, SECOND_CONNECTION) != SUCCESS){
            return FAILURE;
        }
    }
    /// vertexNewClone won't be added to the tree. It will be used only to retrieve the path between the
vertexNewClone and vertexParent (of the second tree)

    /// At this stage it's sure to have a complete closed path
    // Check Path needs to check the number of balls along the all path (obstacles and distance are correct and
previously calculated)

    checkPath(vertexNew, vertexNewClone, TREE); /// NewPath <- CheckPath (xnew, xparent, Ta, Tb)
    return SUCCESS;
}
```

4.3. Similar planning methods

TWIN-RRT* was developed to solve a particular motion planning problem that has similarities with some optimization problems (like TSP, for example) but has many components of a motion planning problem, including kinematic constraints, for example. Conventional optimization or combinatorial algorithms cannot be applied because the number of targets is too big. Motion planning algorithms require substantial modifications in order to be applied to the current problem. So far, planning algorithms have never been applied to similar problems, so it is not feasible, for the moment, to use any other algorithm as a benchmark to the new TWIN-RRT* algorithm.

For benchmarking purposes a standard complete sweep path is used as reference (see equation 3). This means that the agent starts from the initial configuration and completely covers the free-space region in the environment, by visiting every location at least once and preferably without passing twice in the same location, and finishes at the final configuration (which is equal to the starting configuration).

4.4. Other multi-target methods

Recently some work has been carried out concerning multi-target applications [46, 47, 50, 112]. The common ground is that all of them use a small number of targets. Dealing with a relatively low or fixed amount of targets is a kind of problem significantly different from the current case-study. The planning approach in those problems is first to find the optimal sequence and then to apply a motion planner. In the first step the best sequence is determined by an optimization or combinatorial algorithm. In the second step a motion planning algorithm is applied for each consecutive pair of points in the entire sequence. The final path is then retrieved by adding all the intermediate paths one after another. As previously stated, this approach is feasible for a relatively small number of targets, but impossible to use otherwise, due to the computational complexity.

4.5. Multiple target algorithms applications

Multiple target algorithms are becoming available for many other applications including military [47], industrial production [50], environment exploration [48, 49, 113], sports and agriculture [103].

It is reasonable to assume these algorithms will play an important role in the next generation of planning algorithms.

In particular, TWIN-RRT* was designed to maintain RRT* properties such as high DOFs suitability, probabilistically completeness, asymptotical optimality, low computational cost and kinodynamic constraints compatibility.

The RRT* inherits the probabilistically completeness of RRT [2, 6]. The TWIN-RRT* uses the same random sampling techniques, and no node is rejected until a solution is found, so it does not lose the probabilistically completeness property.

TWIN-RRT* also conserves the probabilistically optimality of RRT*[2] since the same “Rewire” feature is conserved and only samples in the collision space are rejected.

Although TWIN-RRT* has the overhead of maintaining two trees relatively to RRT*, it is still applicable to high DOFs situations. A small increase in computational time/space is expected when compared to RRT* in those cases.

Besides keeping the RRT* most interesting properties, TWIN-RRT* adds a function cost which can be adapted to many other applications. Two trees are growing from the same root. Every sample is evaluated as a possible connection point, but it only effectively connects to one tree. Closed path loops are retrieved. The trees can be biased according to the function cost, but generally they should maintain the original growth structure of RRT.

These features enable the construction of closed loop paths for many different applications, including high number of non-mandatory targets as the present case-study. Other possible applications involving single robot operation include real-time applications, surveillance, agriculture, military, space, and other situations where kinodynamic constraints may also exist.

4.6. TWIN-RRT* main features

TWIN-RRT* produces feasible, “low cost”, asymptotically optimal and probabilistically complete paths for one agent with kinodynamic constrains towards multiple non-mandatory targets.

The purpose of this algorithm is to solve the case-study problem but keeping it as general as possible, such as compatibility with higher DOF agents and applications.

The new algorithm provides a practical implementation of feasible, “low cost” planning especially where a closed loop is required. Initial and final configurations are allowed to be exactly the same.

Planning for multiple targets has generally been a task for multiple robots [46, 112, 114]. This algorithm computes an efficient path for one sole agent towards multiple targets where none of them is mandatory. It inherits the low computational cost, probabilistic completeness and asymptotical optimality from RRT*. It uses efficiency (see equation 2) as cost function, which can be re-defined for different applications.

Besides being compliant with different cost functions, the TWIN-RRT* is also compliant with kinodynamic constraints.

4.7. TWIN-RRT* as an evolution of RRT*

TWIN-RRT* is based on RRT*, which is itself considered an evolution of the basic RRT algorithm. RRT* is an asymptotically optimal incremental sampling based method and therefore, TWIN-RRT* inherits these same characteristics. The asymptotical optimality feature of RRT* which is absent in RRT is accomplished by growing a tree that can change its structure along the time: in any iteration there is the possibility for branches to change from one parent to another. Both trees of TWIN-RRT* have each one the same inherited feature. Also, the kinodynamic constraints compatibility of RRT* is also maintained in the TWIN-RRT* algorithm.

RRT* builds only one tree, whereas TWIN-RRT* maintains 2. TWIN-RRT* further differs from RRT* because both initial and final configurations are the same. RRT* does not retrieve any feasible path for this particular case. RRT* like most of motion planning algorithms does not compute a path when start and goal configurations are the same. TWIN-RRT* allows to find a path even when some other metric different than distance is to be maximised or minimised. RRT* finds an optimal path from one start configuration to only one goal configuration (as long as it belongs to the pre-established configuration goal region). TWIN-RRT* constructs a path where several target configurations are reached. Note that in TWIN-RRT* targets are considered different from goal configuration, whereas in RRT* the target coincides with the goal configuration. TWIN-RRT* distinguishes between intermediate targets and final goal configuration. Intermediate targets are not mandatory while final goal configuration is. Using RRT* the best path is retrieved using the structure of the tree at the final stage. In TWIN-RRT* the best path is not always retrievable from the tree structure at the last stage. It may have been found on a previous stage, so it is important to save the best path in memory every time a new one is found, by replacing the previous best one. Since the trees structure changes all the time it is impossible to retrieve previously found paths. Note that in TWIN-RRT* algorithm the two trees never join. They may have similar nodes but trees do not "glue" together. They are always treated separately.

RRT* cannot retrieve a closed loop path whereas TWIN-RRT* does.

4.8. Implementation details: workspace, configuration, robot and obstacles representation

Workspace and the configuration space representation constitute major factors that influence the computational cost. Although the TWIN-RRT* does not require a pre-computational step, there are a few preparation steps which have to be addressed before running the algorithm.

The following assumptions were made regarding the current case-study and may serve as an example or starting point for other applications. Some of those tasks are related to the discretisation of the environment.

Fixed known obstacles are represented as squared blocks of 1m^2. The robot dimensions are added to the obstacle dimensions as a Minkowski sum (see Fig. 18) hence simplifying the robot representation [115]. The agent dimensions, width and length, are therefore reduced to a dimensionless point, which makes the planning much more simple. The obstacles representation should compensate for that simplification, and a predetermined value (e.g. max $(Agent_{width}, Agent_{length})/2$) is added to the dimensions of each obstacle.

In practical terms this means to add the maximum "radius of the robot" (distance from any point of the robot to its center) to each of the obstacle regions, thus enlarging the obstacle representation to avoid collisions. As a benefit, this representation eliminates expensive computational costs. It should be noticed that taking the maximum "radius of the robot" is an oversimplification which does not have much interference when the number and dimension of obstacles is small and well dispersed in the workspace. Complex environments can also benefit from this representation but care must be taken to avoid turning a feasible path into an unfeasible one due to that oversimplification.

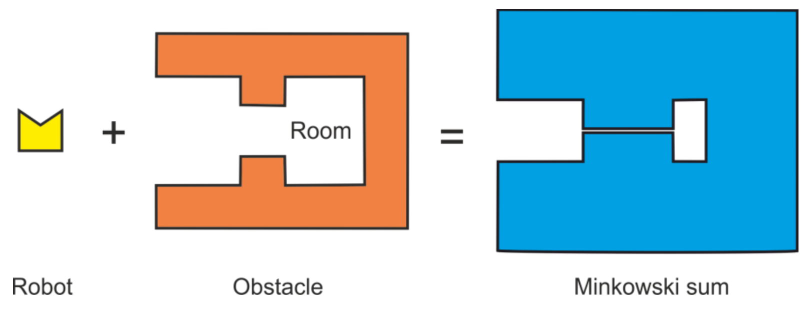


Fig. 18 - Given two 2D polygonal sets A,B∈ R^2 the Minkowski sum of these two sets is a set with the sum of all elements from A and all elements of B: $A \oplus B = \{a+b \mid a \in A, b \in B\}$

The configuration space is also discretised but not explicitly represented. The discretisation is used for sampling. In order to make it suitable and feasible for the planner to retrieve a path in a timely manner the workspace was discretised in squares of $1m^2$. Considering a typical field of 100m width and 300m length, 30 000 different positions are available. The configuration space has 3 degree of freedom, x, y and θ. x and y values assume integer values: x ranges from 0 to 100; y ranges from 0 to 300. θ values represent angles which may assume 36 possible different values ranging from 0 to 360 degrees by steps of 10 degrees.

The agent localisation is assumed to be precisely known, as well as the obstacles location and dimensions. The targets location, as presented before, are estimated with the hybrid information system (see section 2.4), but for the purpose of the planner, that process is transparent, which means that the information is used as if it was the real and valid one. Being that information statistically relevant, it is assumed that using it will also provide statistically relevant results (paths).

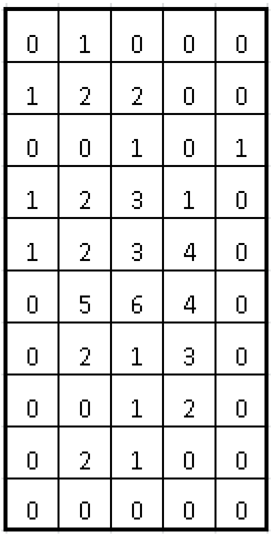
The sampling process is also discretised, so that only integer numbers are allowed. This enables to significantly improve the processing time.

As stated before, restrictions such as minimum number of targets, maximum number of targets, minimum efficiency ratio, minimum distance, maximum distance, maximum calculation time and others, can also be established in advance.

The robot is assumed to move at constant speed. Regarding the robot kinematic constraints, and for simplification purposes only, the robot movement was limited to 3 possible types: maximum turning to the right, maximum turning to the left and linear forward movement. The discretised set of allowable actions at each state is called motion primitives. Assuming this simplification, it is then possible to use the Dubins curves approach (see section 2.6) with fixed ρ (curvature radius) value to compute the shortest path from one configuration to another while observing the kinematic constraints.

Targets localisation:

The targets localisation is also discretised according to the environment discretisation. Each square (1m x 1m) has a certain number of targets. For example, let us consider a field with 5m x 10m represented in Fig. 19. Each square represents an area of 1m^2 and has a certain number of targets inside, in this case represented by the number inside each square, e.g. at column 2, line 6, there are 5 targets.



0	1	0	0	0
1	2	2	0	0
0	0	1	0	1
1	2	3	1	0
1	2	3	4	0
0	5	6	4	0
0	2	1	3	0
0	0	1	2	0
0	2	1	0	0
0	0	0	0	0

Fig. 19 - Balls localisation discretised representation

For practical reasons it is assumed that when the agent reaches a determined square all the targets inside that square are considered reached.

Initial and Final Position:

The initial position and orientation of the robot are exactly the same as the final ones. This is a significant constraint that influences the choice and tuning of the planning algorithm. The robot must perform a closed loop trajectory, returning to the starting point. This ensures that the robot, after completing a plan, is again in the same starting position and orientation and ready to start executing a new planned trajectory. Referring to Fig. 20, on stage A, the robot is on the starting configuration. Stages B and C refer to intermediate configurations along the path. At stage D, the robot reaches the final configuration upon completing the planned path, returning to its original configuration (position and orientation).

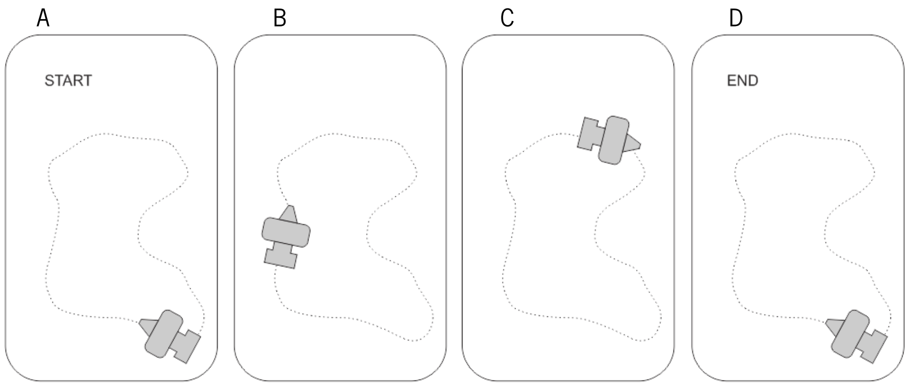


Fig. 20 – Representation of 4 stages of a complete path

Table 1 resumes the simplifications and assumptions adopted to simplify and improve the performance of the path planning.

Table 1 – Discretisation and simplifications adopted to improve performance

-	the environment is discretised in squares of 1m width;
-	the robot moves at constant speed;
-	the robot moves forward, left and right, but never backwards;
-	the robot has kinematic constraints but it always turns at maximum turning angle (θ_{max});
-	the robot is considered to be a single point with no dimensions;
-	the obstacles are assumed to have their dimensions plus half of the robot maximum diameter;
-	obstacles are fixed and their dimensions and localisation are known beforehand;
-	the number and location of the targets is known *a priori*;
-	whenever the robot passes through a given point it reaches all the targets within the correspondent area;
-	the sampled robot configurations are discrete (x and y are integers; θ angles are integers between 0 and 360 with 10 degrees interval);
-	initial and final configurations are exactly the same.

4.9. Planning for non-holonomic vehicles

For path planning purposes, non-holonomic robots pose a much more challenging problem.

To accommodate the differential kinematic constraints a steering function has to be included and the shortest path from one point to another in 2D is no longer a straight line.

Dubins Curves are the simplest and optimal solution whenever is possible and desirable to assume only 3 types of movement (straight forward, turn right, turn left). Turning is also considered to be always at maximum angle possible (θ_{max}), and consequently the turning radius is minimum (ρ_{min}).

Non-holonomic vehicles using the Dubins curves can then perform only three basic movements, which are generally referred as motion primitives: S (straight movement), L (left turn) and R (right turn).

Dubins [34] showed that the shortest path between two configurations requires only up to three motion primitives applied in sequence. The combination of these 3 primitives produces 6 possible optimal paths: {RSL, RLR, RSR, LSR, LSL, LRL}. The other combinations do not render better possible paths. The length of each segment (t, p and q respectively) is variable. Note that t, p and q letters refer to the first, second and third segment respectively and independently of their type. For each segment length, a constant action (primitive) is applied. There might even be the case where one or two segments are of length zero. Considering a LSL combination, for example, if both t and q segments are zero, then the movement will consist of a straight path.

Dubins Curves:

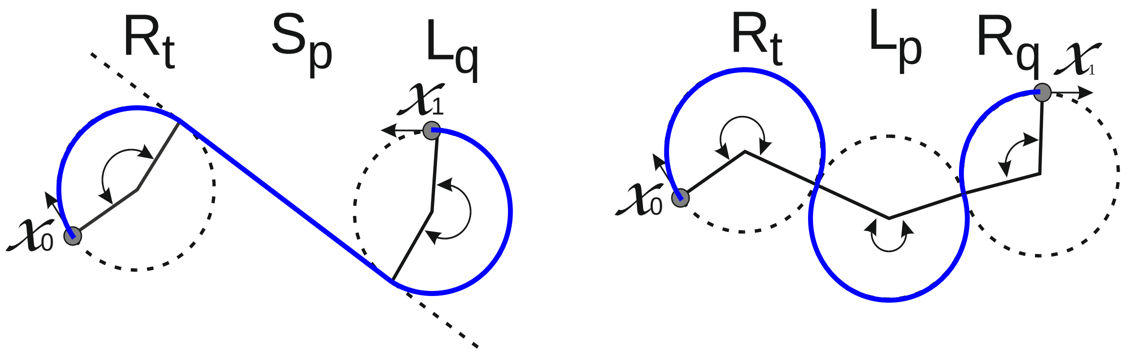


Fig. 21 – Dubins Paths representation for optimal path between two different configurations x_0 and x_1 (starting and final configuration respectively). On the left, a RSL type and on the right a RLR type of movement generated from the three possible primitives L (left turn), S (straight line) and R (right turn).

4.10. Non-holonomic agents and its implications on KD-Trees Representation

Non-holonomic agents pose an additional problem for data representation in KD-Trees.

Briefly, a KD-tree is a binary tree where each node represents a k-dimensional point in a space of k-dimensions (see Fig. 22). Considering a 2D space, each node will have the correspondent x, y Cartesian coordinates. For a 3D space, each node will have the correspondent x, y and z Cartesian coordinates. KD-trees are useful for even higher dimensional representations. The advantages of such representation is that it allows a fast retrieval of points within a given area or neighbourhood of a point. This feature makes the KD-tree representation a preferable structure for sampling motion planning implementations. In the case of RRT and RRT*, for example, it allows a set of neighbour points (of a given configuration) to be quickly and easily retrievable, and so the nearest neighbour point required for the algorithm comes at low computational cost.

The calculations within the KD-tree structure are all based on Euclidian distances. This works fine for holonomic robots because each configuration is usually represented by Cartesian coordinates, but the same cannot be applied for non-holonomic robots. The inclusion of the direction (θ) which is an angle together with the x and y coordinates makes the original KD-tree structure unfeasible for the same purposes.

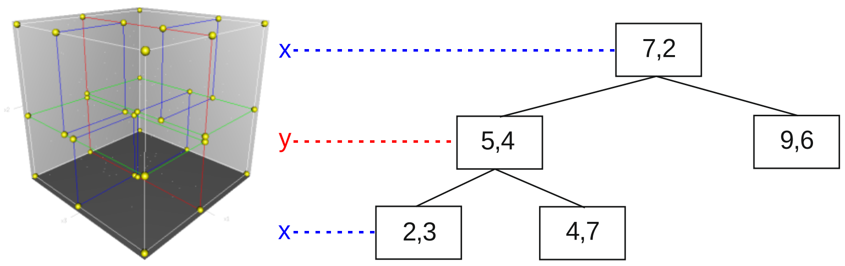


Fig. 22 – On the left a 3D-tree visual representation and on the right a 2D-tree structure representation [116].

An important modification was made to the standard KD-tree, in order to allow different configurations for the same (x, y) point, i.e. configurations including also the orientation (θ) of the agent, and at the same time maintaining the fast neighbour retrieval feature. The proposed solution consists of using for each node, a set of possible nodes sharing the same (x,y) coordinate, but having different orientations (see Fig. 23).

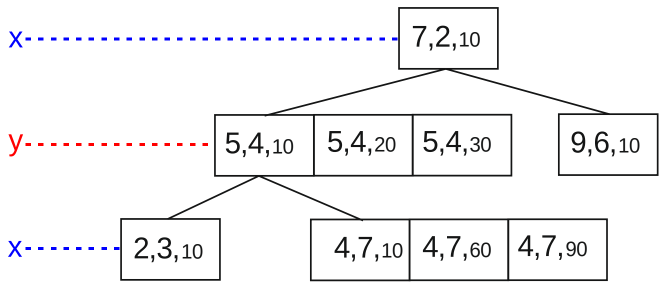


Fig. 23 – Adapted KD-tree to accommodate information about angles. Each node can be a set of nodes sharing the same (x,y) coordinates.

In a conventional KD-tree there are typically two child nodes (left branch and right branch) for each node. In the 2D KD-tree of Fig. 22, the parent node is at coordinate (7,2). It has two children, the left one at (5,4) and the right one at (9,6). Only a maximum of two children is allowed per node. Also, any node has only one parent node.

The proposed modified KD-tree of Fig. 23 accepts more than one node having the same pair of coordinates (x,y) and different values of theta (θ), e.g. there are 3 left branch children of the parent node (7,2,10) having the configurations (x,y,θ) equal to (5,4,10), (5,4,20) and (5,4,30) respectively. For convenience they are referred to as left siblings. In the same way, there can also be right siblings, see e.g. the configurations (4,7,10), (4,7,60) and (4,7,90) on the right branch of the parent node (5,4,10) at the 3rd level of the KD-tree. At the search stage, not only a given node will be retrieved, but also all his siblings, should they exist.

In chapter 5 the results of several simulations are presented in order to better understand the characteristics of the TWIN-RRT* algorithm. It is important to stress the fact that it is not possible to compare its performance to other algorithms because there are no other algorithms described in literature capable to deal with multiple targets in the same way as the proposed algorithm does.

The case-study presented in 1.2 serves as a common framework for the detailed study on the algorithms' behaviour. By the end of the next chapter it will become clearer how the TWIN-RRT* performs under different situations like varying the number and localisation of targets, using different obstacles configurations, changing the starting configuration and adjusting different technical parameters.

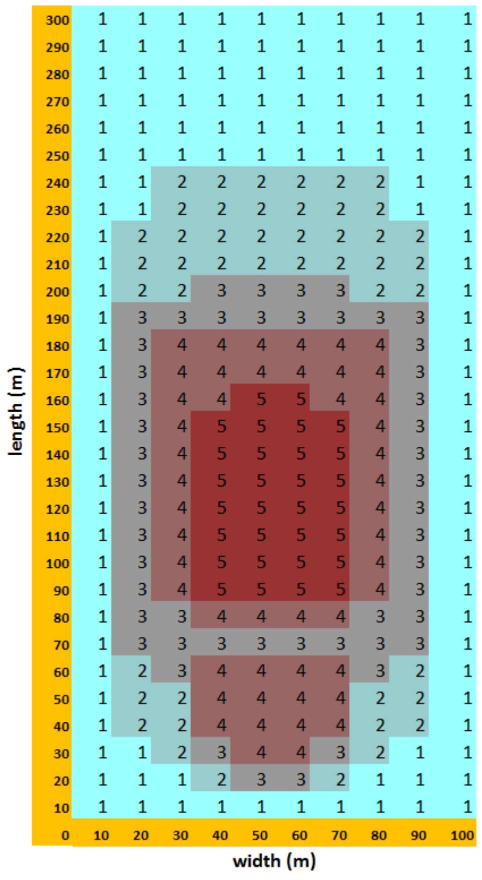
CHAPTER V

RESULTS AND DISCUSSION

Basic simulation scenario

A set of simulations were performed to study the behaviour of the TWIN-RRT* algorithm. A main standard scenario was defined to establish a reference for all experiments.

The standard scenario is composed by a rectangular field (100m x 300m) discretised in squares of 1m length and having 5 000 targets distributed in it. The target distribution follows a Monte Carlo distribution pattern. The seed matrix is called the probabilistic matrix (Fig. 24). The probabilities were empirically determined following the input given by driving range experts, since there is no reliable public data available.



length (m)										
300	1	1	1	1	1	1	1	1	1	1
290	1	1	1	1	1	1	1	1	1	1
280	1	1	1	1	1	1	1	1	1	1
270	1	1	1	1	1	1	1	1	1	1
260	1	1	1	1	1	1	1	1	1	1
250	1	1	1	1	1	1	1	1	1	1
240	1	1	2	2	2	2	2	2	1	1
230	1	1	2	2	2	2	2	2	1	1
220	1	2	2	2	2	2	2	2	2	1
210	1	2	2	2	2	2	2	2	2	1
200	1	2	2	3	3	3	3	2	2	1
190	1	3	3	3	3	3	3	3	3	1
180	1	3	4	4	4	4	4	4	3	1
170	1	3	4	4	4	4	4	4	3	1
160	1	3	4	4	5	5	4	4	3	1
150	1	3	4	5	5	5	5	4	3	1
140	1	3	4	5	5	5	5	4	3	1
130	1	3	4	5	5	5	5	4	3	1
120	1	3	4	5	5	5	5	4	3	1
110	1	3	4	5	5	5	5	4	3	1
100	1	3	4	5	5	5	5	4	3	1
90	1	3	4	5	5	5	5	4	3	1
80	1	3	3	4	4	4	4	3	3	1
70	1	3	3	3	3	3	3	3	3	1
60	1	2	3	4	4	4	4	3	2	1
50	1	2	2	4	4	4	4	2	2	1
40	1	2	2	4	4	4	4	2	2	1
30	1	1	2	3	4	4	3	2	1	1
20	1	1	1	2	3	3	2	1	1	1
10	1	1	1	1	1	1	1	1	1	1
0	10	20	30	40	50	60	70	80	90	100

width (m)

Fig. 24 - Probabilistic matrix used to seed the target distribution on the field. Note: This seed matrix was established following Golf Driving Range experts contribution.

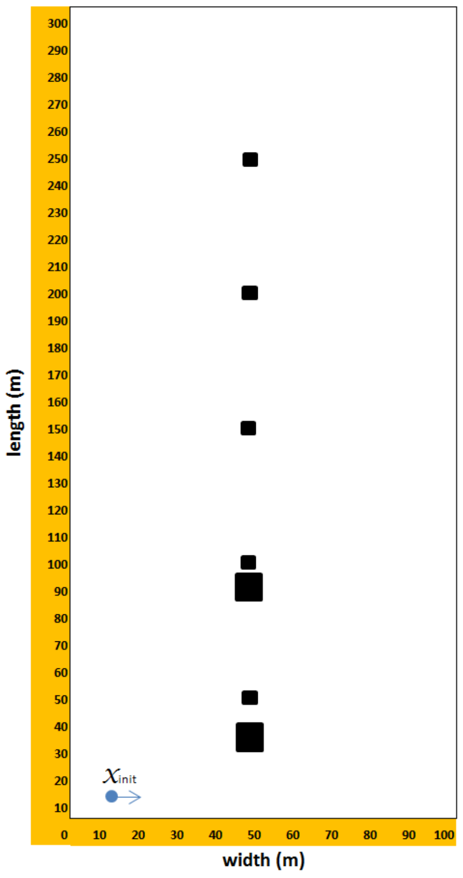
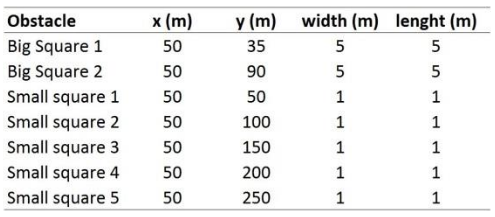


Fig. 25 – Agent starting position and obstacles representation on the field. Note: The obstacles representation is a mere indication of the approximate relative position. They are not scaled proportionally.

There are a few obstacles lying on the field (Table 2). These represent obstacles typically found in real *Driving Ranges*. All obstacles are represented as squares or rectangles for simplification purposes. The starting and final position of the robot (agent) was set at (x,y,θ)=(12,12,0).

Table 2 – Obstacle localisation and dimensions



Obstacle	x (m)	y (m)	width (m)	lenght (m)
Big Square 1	50	35	5	5
Big Square 2	50	90	5	5
Small square 1	50	50	1	1
Small square 2	50	100	1	1
Small square 3	50	150	1	1
Small square 4	50	200	1	1
Small square 5	50	250	1	1

No other conditions were predetermined. The terrain is considered to be flat and no slopes, holes or any other external factors are to be considered for simulation purposes.

For the purpose of this document a "run" should be understood as a computerised search for an efficient path. A standard run is limited to a maximum of 10 000 iterations of the TWIN-RRT* algorithm. Other restrictions may apply accordingly to the simulation purpose, e.g. maximum time to perform the search. If not otherwise stated the standard conditions of Table 3 apply.

Table 3 – Standard parameters valid for every simulation if not otherwise stated.

General Parameters	Number of targets in the workspace	ballsLaunched = 5 000
	Maximum number of iterations to perform	maxIterations = 10 000
	Minimum number of targets to reach	minBallsToCollect = 0
	Computational time limit (0 = no limit)	maxTime (s) = 100s
	Maximum path distance (0 = no restriction)	maxDistance (m) = 25 000
	Minimum ratio to achieve (0 = no restriction)	minRatio (balls/m) = 0.00
TWIN–RRT* Technical Parameters	Distance between two consecutive positions used to check for obstacles and calculate the path	discretisationStep (m) = 0.50
	Minimum curvature radius	rho (m) = 2.00
	Gamma: parameter used to calculate the region to search near neighbours.	gamma(m)= 10.00

5.1. Fast convergence towards an optimal solution

In order to evaluate the convergence of the algorithm towards an optimum solution, 100 runs with 10 000 iterations each were performed. Every improved path was registered. Fig. 26 shows the evolution of the efficiency gain along the time.

There is one difference regarding the standard conditions for this simulation: the computational time limit restriction was not applied (see Table 4).

Table 4 – Parameters for fast convergence towards an optimal solution simulation

General Parameters	Number of targets in the workspace	ballsLaunched = 5 000
	Maximum number of iterations to perform	maxIterations = 10 000
	Minimum number of targets to reach	minBallsToCollect = 0
	Computational time limit (0 = no limit)	maxTime (s) = 0s
	Maximum path distance (0 = no restriction)	maxDistance (m) = 25 000
	Minimum ratio to achieve (0 = no restriction)	minRatio (balls/m) = 0.00
TWIN–RRT* Technical Parameters	Distance between two consecutive positions used to check for obstacles and calculate the path	discretisationStep (m) = 0.50
	Minimum curvature radius	rho (m) = 2.00
	Gamma: parameter used to calculate the region to search near neighbours.	gamma(m)= 10.00

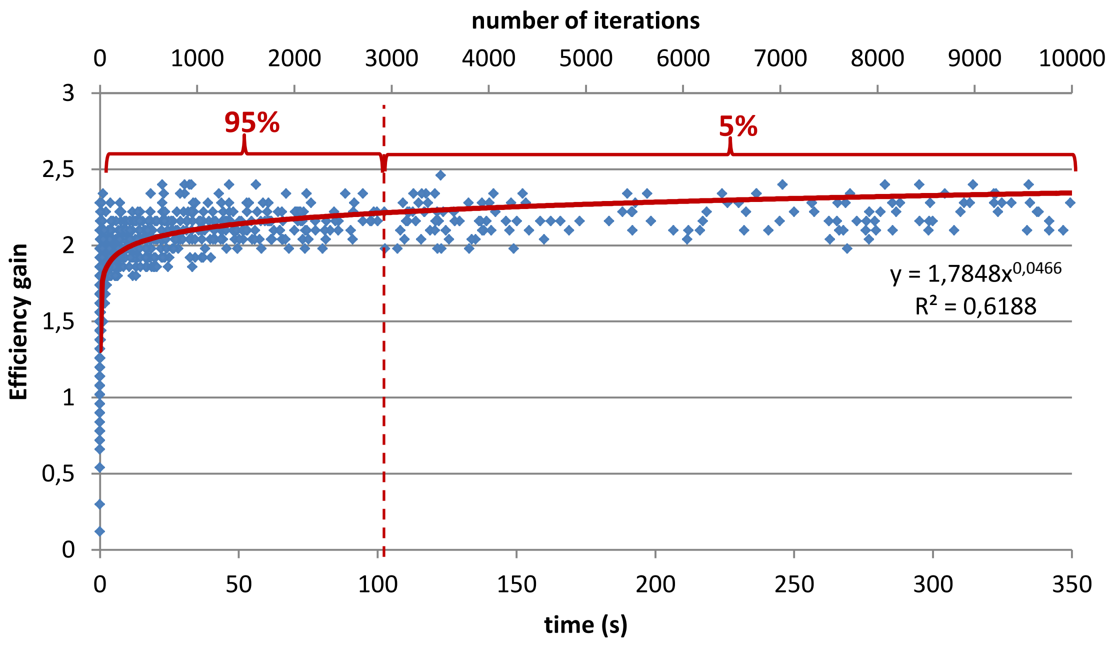


Fig. 26 – TWIN-RRT* performance evaluation considering efficiency gain evolution along the time

A total of 1361 points are plotted on this chart. These represent every improvement step of the algorithm result obtained over time for each of the 100 runs. Two important observations should be stressed. The first one is that most of the improvement happens at early stages. 95% of the points (which reflect an improvement) take place before 100 s.

The second observation is that the algorithm shows a very fast convergence. The results show that more than 50% of total improvements take place in less than one second. There were even some cases (2%) where the best result was even obtained in that period. Fig. 27 shows more detail about this. In this chart only the best result from each run is plotted. It is important to refer that the best result may actually appear very fast at the initial steps of the run. This chart shows a consistent pattern along the time, which corroborates the assumption that it inherits two main features of RRT*: it is continuously improving the result thus is asymptotically optimal, and it produces initial feasible plans at very early stages.

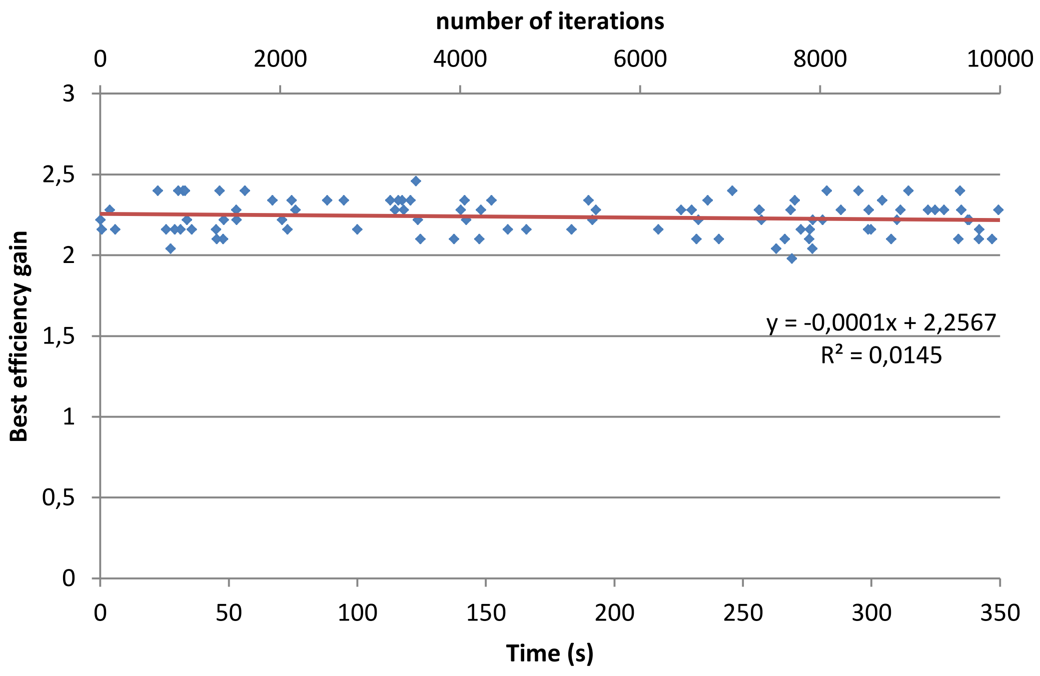


Fig. 27 – Best efficiency gain for every run. There are 100 points, one for each run, representing the best efficiency gain and the corresponding time at which it was obtained.

Using the same data, the chart of Fig. 26 was redrawn by grouping values within time intervals specified in Table 5.

Table 5 – Efficiency gain data grouped by time intervals

time (s)	**0**	**1**	**2**	**3**	**4**	**5**	**10**	**15**	**20**	**30**	**50**	**100**	**150**	**200**	**250**	**300**	**350**
Eff.gain (mean value)	1,58	1,87	1,92	1,96	1,97	2,00	2,03	2,04	2,08	2,13	2,15	2,17	2,19	2,21	2,19	**2,25**	2,23
standard deviation	0,30	0,13	0,11	0,11	0,11	0,11	0,11	0,11	0,11	0,12	0,09	0,11	0,09	0,11	0,10	0,09	0,10
percentage (of max)	70%	83%	85%	87%	88%	89%	90%	91%	93%	**95%**	96%	96%	98%	98%	97%	100%	99%

The corresponding chart is shown in Fig. 28. From this data it is possible to observe that the maximum average value obtained for the efficiency gain is 2.25. Notice that just after 10s the algorithm already returns values above 90% of the maximum value, which shows its fast early convergence. After 30s, the values are over 95% of maximum.

Combining the analysis of both charts, the time value of 100s was chosen as end condition for the next simulations for practical reasons. At 100s we have the guarantee that c.a. 95% of the improvements were attained and also that the efficiency gain is very close (less than 5%) to the "maximum" expected value.

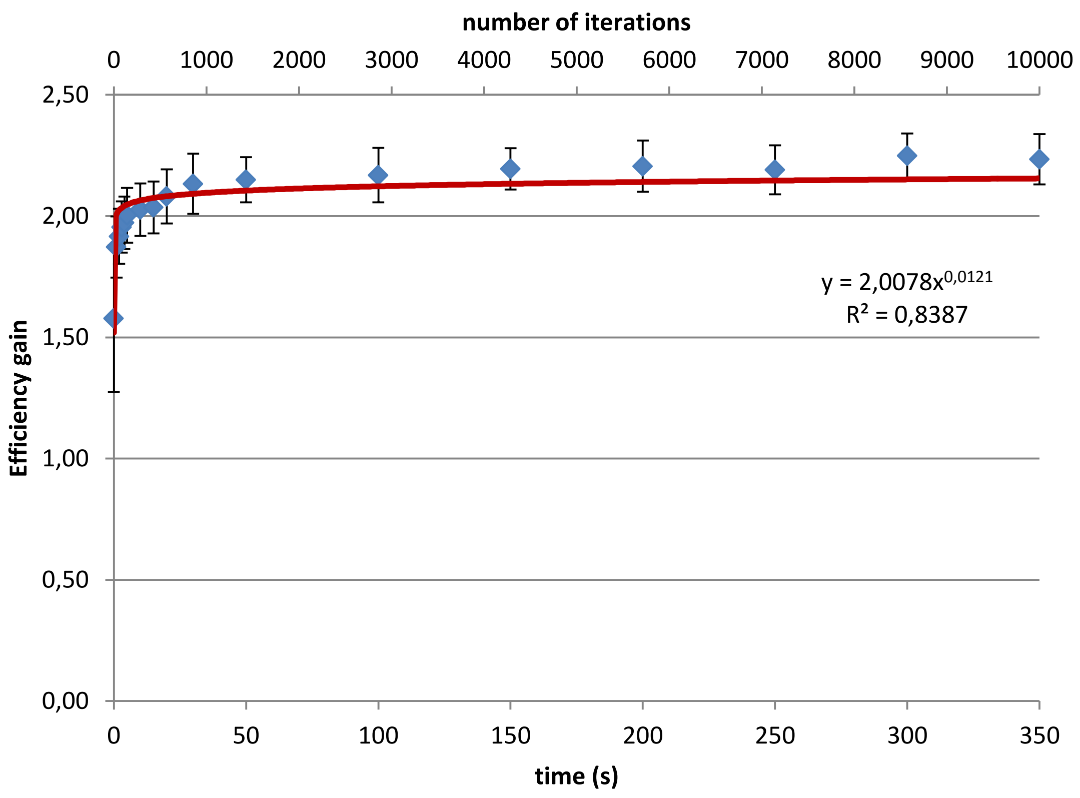


Fig. 28 – TWIN-RRT* performance evaluation considering efficiency gain evolution grouping samples along the time

Due to computer processing limitations it is unfeasible to test all possibilities, but the results show that TWIN-RRT* apparently converges to the optimal solution. This is coherent to the assumption that TWIN-RRT* inherits the asymptotically optimal feature of RRT* [2, 4].

Results showed that 100s is more than sufficient to have an efficient path. The following simulations were performed using this 100s restriction.

5.2. Density of targets and its impact on performance

To evaluate the impact on performance due to increasing number of targets a set of 19 simulation batches was performed ranging from 1 to 1 000 000 targets in the workspace. Each batch comprises 10 runs. The settings used correspond to those of the standard conditions except for the number of targets that varies depending on the batch (see Table 6). The results are summarized in Table 7, where the first row indicates the number of targets in each batch.

Table 6 – Parameters for density of targets simulation

General Parameters	Number of targets in the workspace	ballsLaunched = variable
	Maximum number of iterations to perform	maxIterations = 10 000
	Minimum number of targets to reach	minBallsToCollect = 0
	Computational time limit (0 = no limit)	maxTime (s) = 100s
	Maximum path distance (0 = no restriction)	maxDistance (m) = 25 000
	Minimum ratio to achieve (0 = no restriction)	minRatio (balls/m) = 0.00
TWIN–RRT* Technical Parameters	Distance between two consecutive positions used to check for obstacles and calculate the path	discretisationStep (m) = 0.50
	Minimum curvature radius	rho (m) = 2.00
	Gamma: parameter used to calculate the region to search near neighbours.	gamma(m)= 10.00

Table 7 – Efficiency gain evolution depending on the number of targets available

	1	2	5	10	20	50	100	200	500	1000	2000	5000	10000	20000	50000	100000	200000	500000	1000000
AVG	111,45	86,26	49,55	29,61	24,21	11,97	7,90	5,30	3,53	3,13	2,60	2,20	2,00	1,92	1,87	1,84	1,85	1,81	1,84
STD	66,70	43,95	21,26	8,71	8,46	1,58	1,74	0,71	0,30	0,55	0,30	0,06	0,07	0,07	0,06	0,03	0,03	0,04	0,03
dev %	60%	51%	43%	29%	35%	13%	22%	13%	9%	18%	12%	3%	3%	3%	3%	2%	2%	2%	2%

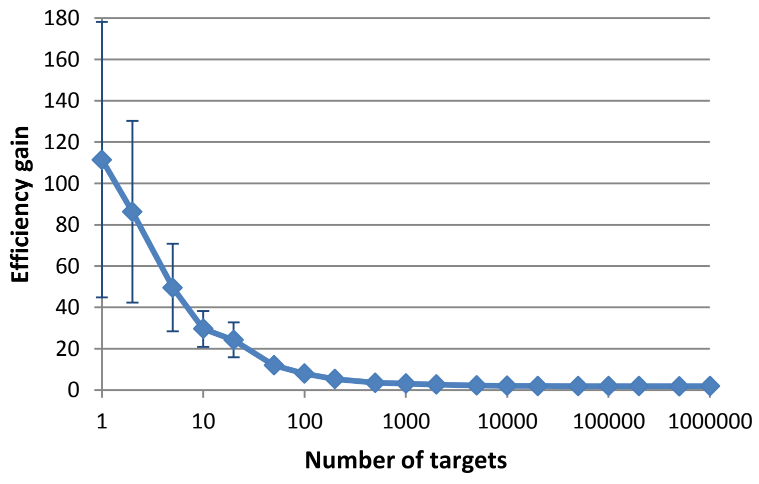


Fig. 29 – Dependence of the efficiency gain on the number of targets available.

As the number of targets increases the efficiency gain decreases. Roughly, from 1 to 100 the decrease is sharp (see Fig. 29). For more than 100 targets the decrease is smoother. Note that the standard variation is very high for a small number of targets. When using 5 000 targets or more the efficiency gain standard variation is within reasonable values, i.e., values deviate under 5% from the efficiency gain mean value, which means that the performance is stable and predictable above 5 000 available targets. Nevertheless it should be emphasised that these results are only valid to the established framework, but the algorithm behaviour is expected to be similar in other environments/applications.

Furthermore, the efficiency gain tends to stabilise around 1.8 for higher values, which is a clear indication that it will always be better than a sweep strategy, no matter how big is the number of targets.

Two conclusions may be derived from these results: first, the algorithm performance is steady and reliable for higher number of available targets making the TWIN-RRT* suitable for addressing a large number of targets; second, it is clearly a much better approach than a complete sweep strategy (efficiency gain = 1.0). The efficiency gain ranges from a minimum of 1.8 (for high number of targets) to more than 100 (for small number of targets), depending on the number of targets.

5.3. Restriction on the minimum number of targets

In order to evaluate how the algorithm deals with restrictions on the minimum number of targets, 13 simulation batches were performed with 100 runs per batch.

The standard conditions were applied except for the minimum number of targets to reach (see Table 8), which varies from batch to batch according to the following sequence: {0, 1, 2, 5, 10, 20, 50, 100, 200, 500, 1 000, 2 000, 5 000}.

Table 8 – Parameters for restriction on the minimum number of targets

General Parameters	Number of targets in the workspace	ballsLaunched = 5 000
	Maximum number of iterations to perform	maxIterations = 10 000
	Minimum number of targets to reach	minBallsToCollect = variable
	Computational time limit (0 = no limit)	maxTime (s) = 100s
	Maximum path distance (0 = no restriction)	maxDistance (m) = 25 000
	Minimum ratio to achieve (0 = no restriction)	minRatio (balls/m) = 0.00
TWIN-RRT* Technical Parameters	Distance between two consecutive positions used to check for obstacles and calculate the path	discretisationStep (m)= 0.50
	Minimum curvature radius	rho (m) = 2.00
	Gamma: parameter used to calculate the region to search near neighbours.	gamma(m)= 10.00

Table 9 - Efficiency gain evolution depending on the minimum number of targets to reach

Min. targets	0	1	2	5	10	20	50	100	200	500	1000	2000	5000
Failures (%)	0	0	0	0	0	0	0	0	92	100	100	100	100
Eff.gain	2,153	2,183	2,1667	2,15	2,168	2,14	2,148	2,146	1,51	0	0	0	0
STDEV	0,109	0,107	0,10832	0,106	0,116	0,112	0,107	0,115	0,146	0	0	0	0

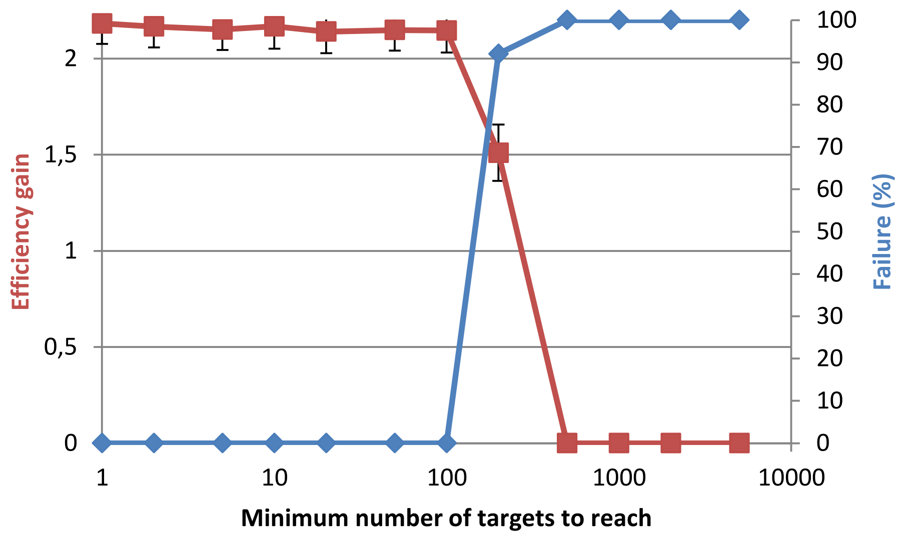


Fig. 30 - Dependence of the efficiency gain on the minimum number of targets to reach.

The performance of the TWIN-RRT* algorithm is steady to about 100 minimum targets to reach, which represents 2% of total targets on the field. When the restriction to reach 200 targets (4%) is imposed, then the algorithm fails to find a feasible solution in more than 90% of the cases and for the cases where it finds feasible solutions, the efficiency gain is only about 1.5, which corresponds to about 2/3 of the previous value (which was around 2.1). For 500 targets (10%) and above, the algorithm fails to retrieve a feasible path.

Two remarks emerge from these results. First, the TWIN-RRT* performance is very steady up to 2% on the minimum number of targets to reach restriction. Moreover the performance is always above 2.0, which means that it is at least twice as better as the sweeping strategy. Second, there is a notorious breaking point when the requirement is about 4% of the total number of targets. From this point onwards, the algorithm fails to find a path compliant with the restriction. It is possible to overcome this limitation by tuning some technical parameters, or adding heuristics. Later on this chapter it will be shown how to increase the number of reached targets (see section 5.9), but that comes with the cost of losing efficiency.

5.4. Target distribution patterns influence on the algorithm's performance.

A simulation was performed to check the evolution of efficiency gain varying the probabilistic matrix (see Fig. 24) for target distribution pattern.

It was performed without obstacles on the field, so that they do not interfere with the algorithm's performance considering different target distribution patterns.

16 simulation batches were performed with 100 runs per batch. All the 1 600 runs used the same standard settings (see Table 10).

Table 10 - Parameters for target distribution patterns simulation

General Parameters	Number of targets in the workspace	ballsLaunched = 5 000
	Maximum number of iterations to perform	maxIterations = 10 000
	Minimum number of targets to reach	minBallsToCollect = 0
	Computational time limit (0 = no limit)	maxTime (s) = 100s
	Maximum path distance (0 = no restriction)	maxDistance (m) = 25 000
	Minimum ratio to achieve (0 = no restriction)	minRatio (balls/m) = 0.00
TWIN–RRT* Technical Parameters	Distance between two consecutive positions used to check for obstacles and calculate the path	discretisationStep (m) = 0.50
	Minimum curvature radius	rho (m) = 2.00
	Gamma: parameter used to calculate the region to search near neighbours.	gamma(m)= 10.00

At every 100 runs the probabilistic matrix changed, in order to have 16 different patterns (see Fig. 31).

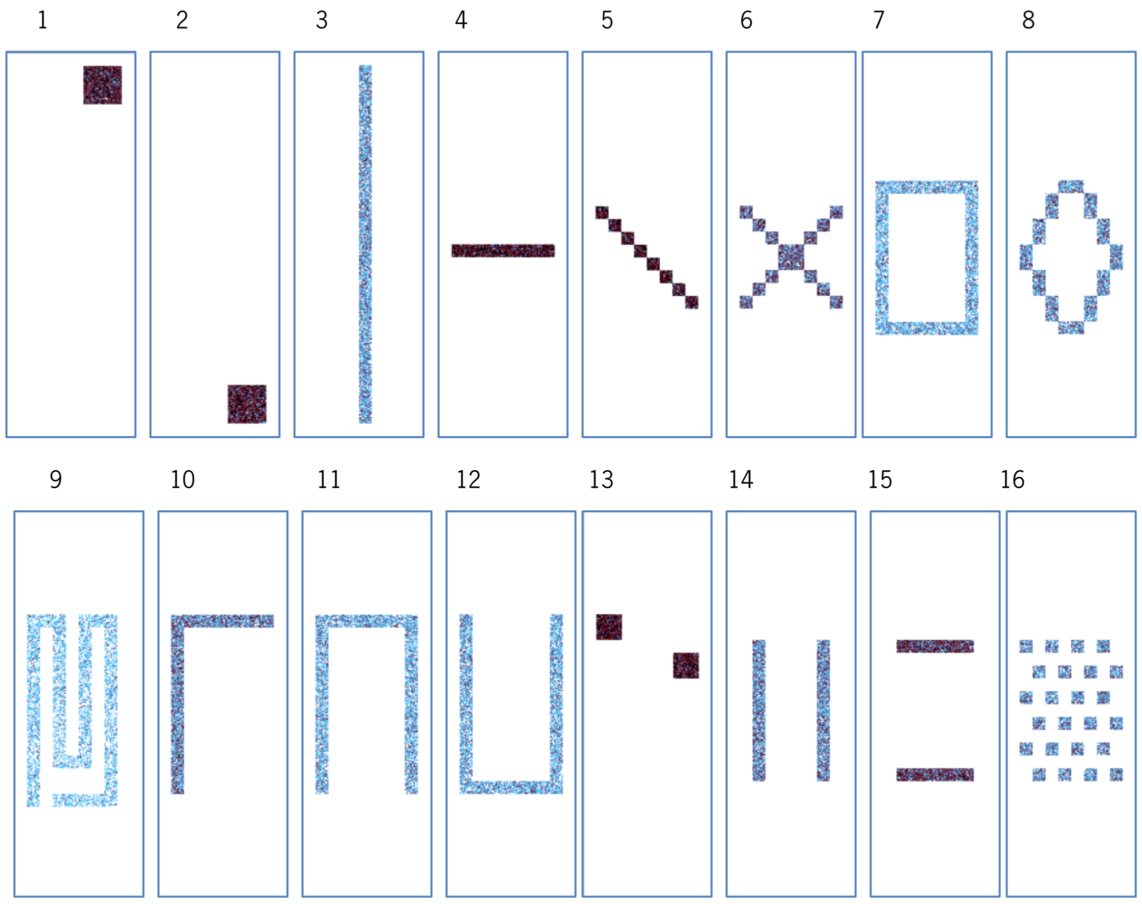


Fig. 31 – Different target distribution patterns used in each batch.

The first remark considering the results of Table 11 is the fact that every value obtained for the efficiency gain is above the average value (2.1) found in previous simulations. The average value (6.14) considering all the batches is about 3 times higher than the previous average value, and the minimum value found (3.35) is still more than 1.5 times higher. This means that the algorithm performs much better in cases where targets are more concentrated.

Table 11 – Numerical results of the 16 batches of runs varying the target distribution pattern.

Pattern	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	**AVG**
Efficiency gain	5,15	**15,88**	4,92	5,92	5,66	5,22	5,12	5,19	**3,35**	8,77	5,48	5,45	5,85	6,94	5,69	3,64	6,14
STDEV	0,41	0,92	0,61	0,65	0,94	0,61	0,41	0,40	0,28	0,79	0,50	0,46	0,47	0,74	0,66	0,25	0,57
Max. eff. gain	6,93	**19,60**	6,45	8,03	11,29	7,01	6,48	6,11	**4,00**	9,91	6,57	6,21	7,40	8,50	9,26	4,23	8,00

The efficiency gain varies significantly according to the target distribution pattern (Fig. 32). It is higher on patterns where targets are more concentrated and near the initial position, as is the case of pattern number 2. Conversely whereas the target distribution is less concentrated the efficiency gain is lower, as in the case of distribution pattern number 9.

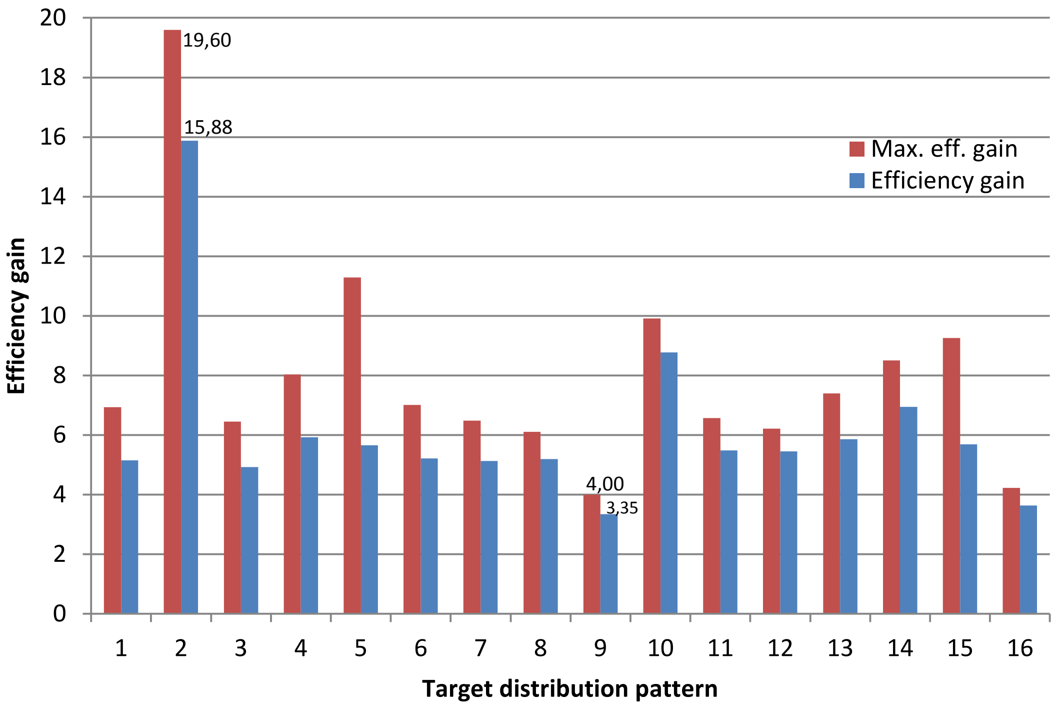


Fig. 32 – Chart representation of the efficiency gain depending on the target distribution pattern.

Conclusions about the pattern itself may be misleading.

Nevertheless it is important to verify that the path generated by the algorithm never completes the whole pattern of the distribution. It selects the part which yields the best efficiency gain, thus prioritising efficiency (see some examples in Fig. 33).

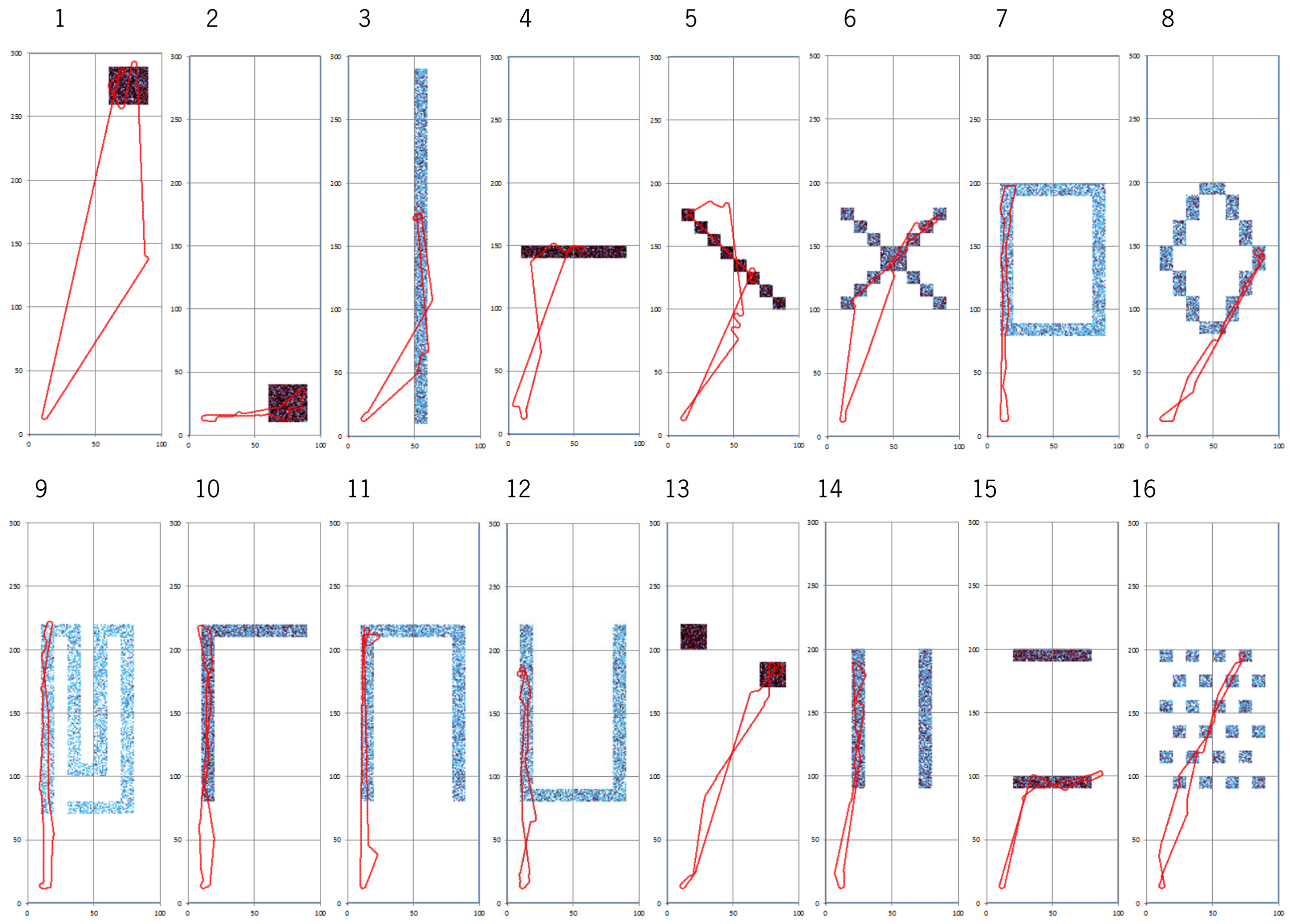


Fig. 33 – Examples of final path results for each simulation batch.

Two additional notes should be made regarding the examples of Fig. 33. Patterns 3, 7, 9, 10, 11, 12 and 14 produce a path which is mainly vertical along the y axis and using the most left part of the pattern, because the starting point (x,y,θ)=(12,12,0) is nearer. Another remark is the fact that inside the target areas the path is more irregular, having more curves and changes on direction, whereas outside those areas, the path tends to be a straight line (see examples 1,2 and 13 of Fig. 33).

5.5. Impact of obstacles location on efficiency gain

A simulation was performed to check the dependence of efficiency gain varying the number and configuration of obstacles in the workspace. 15 simulation batches in total were performed with 100 runs per batch. All the 1 500 runs used the same standard settings (see Table 12). The first batch was performed without obstacles on the field, so that it could be used as reference.

Table 12 - Parameters for impact of obstacles location on efficiency gain simulation

General Parameters	Number of targets in the workspace	ballsLaunched = 5 000
	Maximum number of iterations to perform	maxIterations = 10 000
	Minimum number of targets to reach	minBallsToCollect = 0
	Computational time limit (0 = no limit)	maxTime (s) = 100s
	Maximum path distance (0 = no restriction)	maxDistance (m) = 25 000
	Minimum ratio to achieve (0 = no restriction)	minRatio (balls/m) = 0.00
TWIN—RRT* Technical Parameters	Distance between two consecutive positions used to check for obstacles and calculate the path	discretisationStep (m) = 0.50
	Minimum curvature radius	rho (m) = 2.00
	Gamma: parameter used to calculate the region to search near neighbours.	gamma(m)= 10.00

At every 100 runs the obstacle configuration changes, so there are 15 different situations (Fig. 34). For each obstacle configuration, 100 runs were performed. Obstacles are represented in black and the violet squares represent the targets location. The probabilistic matrix was therefore different from the standard conditions in order to test if the planner follows the targets location while avoiding obstacles along the way.

Regarding Fig. 34, where the different situations are represented, a brief description follows: situation 1 is the reference case where no obstacles are added, situations 2 and 3 represent a scenario with obstacles aligned both vertically and horizontally respectively; situation 4 presents a big obstacle in front of the targets area; in situations 5 and 6 the obstacles partially surround the targets area; situation 7 is a special case where the targets are completely obstructed by obstacles, what makes the targets area completely unreachable; situations 8, 9 and 10 are narrow passages with increasing complexity from 8 to 10; three special cases of narrow passages are presented in situations 11, 12 and 13 which are usually called bug trap configurations (double bug trap in the case of situation 13). Situation 14 is a labyrinth containing dead ends and where the targets are far away from the starting point. Situation 15 presents a scenario with high density of small obstacles in the way.

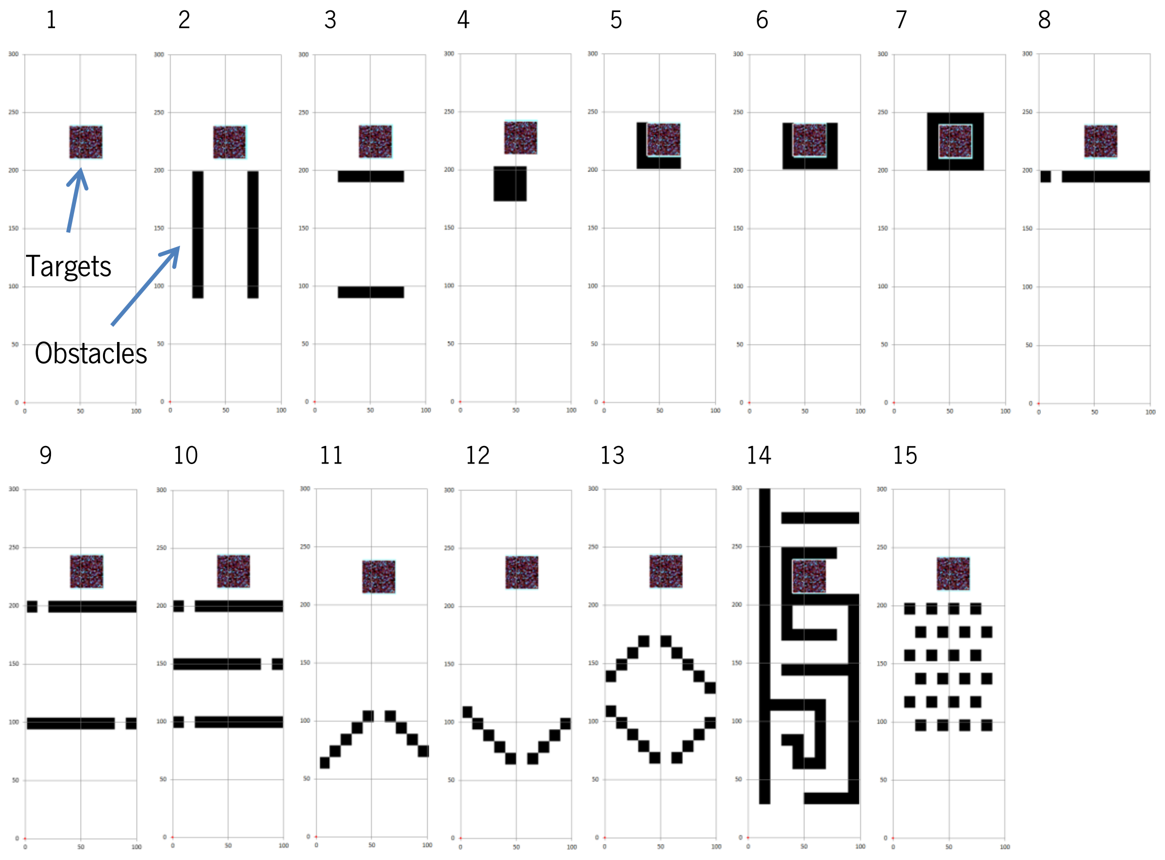


Fig. 34 – Different configurations of obstacles (black) used in each batch. The constant region (violet) having a square shape refers to the ball distribution probability, which was the same for every run in the current simulation.

Table 13 presents the results obtained for every batch. The chart of Fig. 24 shows the same results in a more perceptible way.

Table 13 – Summarized results for each batch having different obstacles configurations

Pattern	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Eff. Gain	**6,14**	6,06	5,51	5,85	5,93	5,00	0,00	5,95	4,89	4,42	5,74	5,76	5,90	**2,97**	**6,14**
St.dev	0,42	0,41	0,49	0,38	0,35	0,48	0,00	0,34	0,24	0,23	0,35	0,33	0,42	0,24	0,53
Max eff. gain	7,67	**7,97**	7,66	6,81	6,97	6,19	0,00	6,97	5,72	5,05	6,91	6,87	7,62	**3,75**	7,85

Comparing the situations 2 to 15 with the reference situation (batch 1), only in 5 out of 13 situations (batches 3, 6, 9, 10 and 14) there was a significant decrease in efficiency gain. For this purpose, batch 7 is not taken into account because it is a special unfeasible situation. So it means that roughly 60% of the evaluated scenarios do not influence the efficiency gain. In the remaining situations (batches 3, 6, 9, 10 and 14) the low efficiency gain, ranging from 10%

(batch 3) to about 50% (batch 14), was mainly due to the fact that the target location was hidden "behind" a large obstacle (batches 3 and 6), or the obstacle configuration requires a longer path to reach the targets (batches 9, 10 and 14).

The situation 15 is a very important situation because it establishes a scenario densely covered with obstacles which are dispersed in the workspace. Nevertheless, the results show a completely immune response of the TWIN-RRT* algorithm to the presence of such obstacles. The average efficiency gain result is exactly the same (6.14) as the reference case without obstacles.

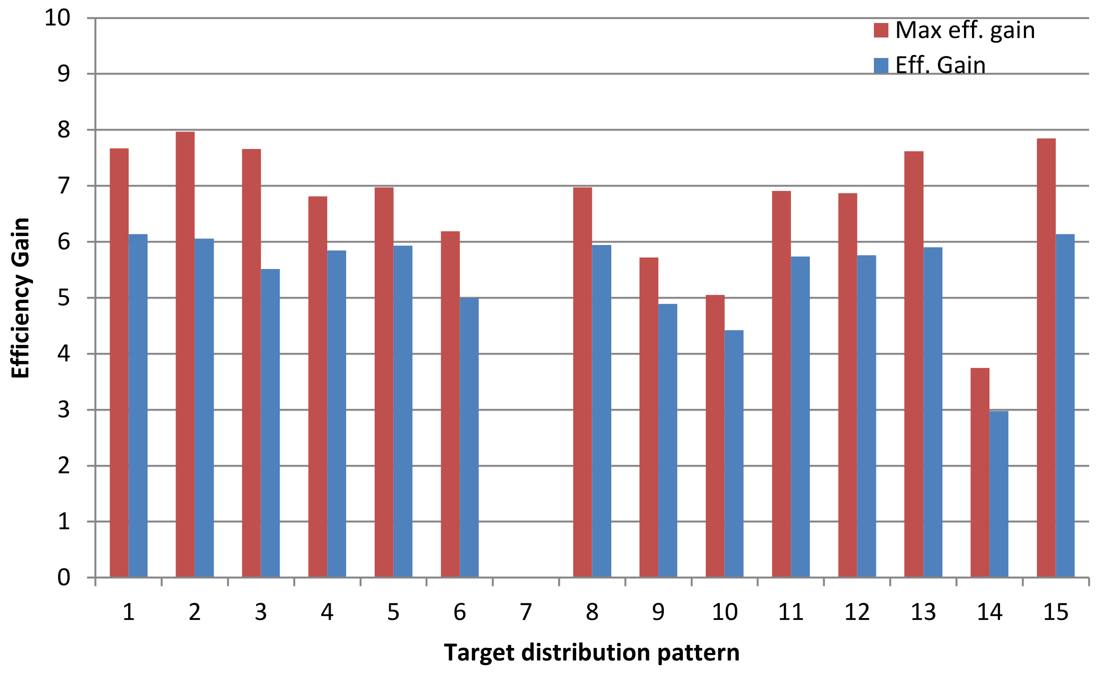


Fig. 35 – Efficiency gain according to different target distribution pattern

Fig. 35 depicts typical path results for each obstacle configuration. The TWIN-RRT* algorithm finds a feasible path regardless of the obstacle alignment (horizontal - batch 2 or vertical – batch 3) and regardless of its dimension (batch 4). The situation 5 is similar to situations 3 and 4 but it shows the ability to contour the obstacle in the most efficient way. The obstacle configuration 5 biases the best path to the right side.

Situation 6 is a difficult situation because the only way in and out is facing to the opposite side of initial and final destination. As the example shows the TWIN-RRT* solves it in an efficient manner. Situation 7 does not have a feasible solution, so the algorithm fails to find a solution and returns failure. Narrow passages are easily overcome by TWIN-RRT* as the experiments 8, 9 and

10 show.

The efficiency gain varies depending on the obstacle configuration. Bulky environments where the path is forced to go around several obstacles have lower efficiency gains (ex: 9, 10 and 14). The special cases of narrow passages, including a bug trap and double bug trap configuration in situations 11, 12 and 13 pose no extra problems when compared to situations 9 and 10, for example.

The proposed algorithm also solves the labyrinth situation 14. Surprisingly it even found a solution in 100% of the 100 runs performed. The bulky obstacle configuration in situation 15 is also feasible.

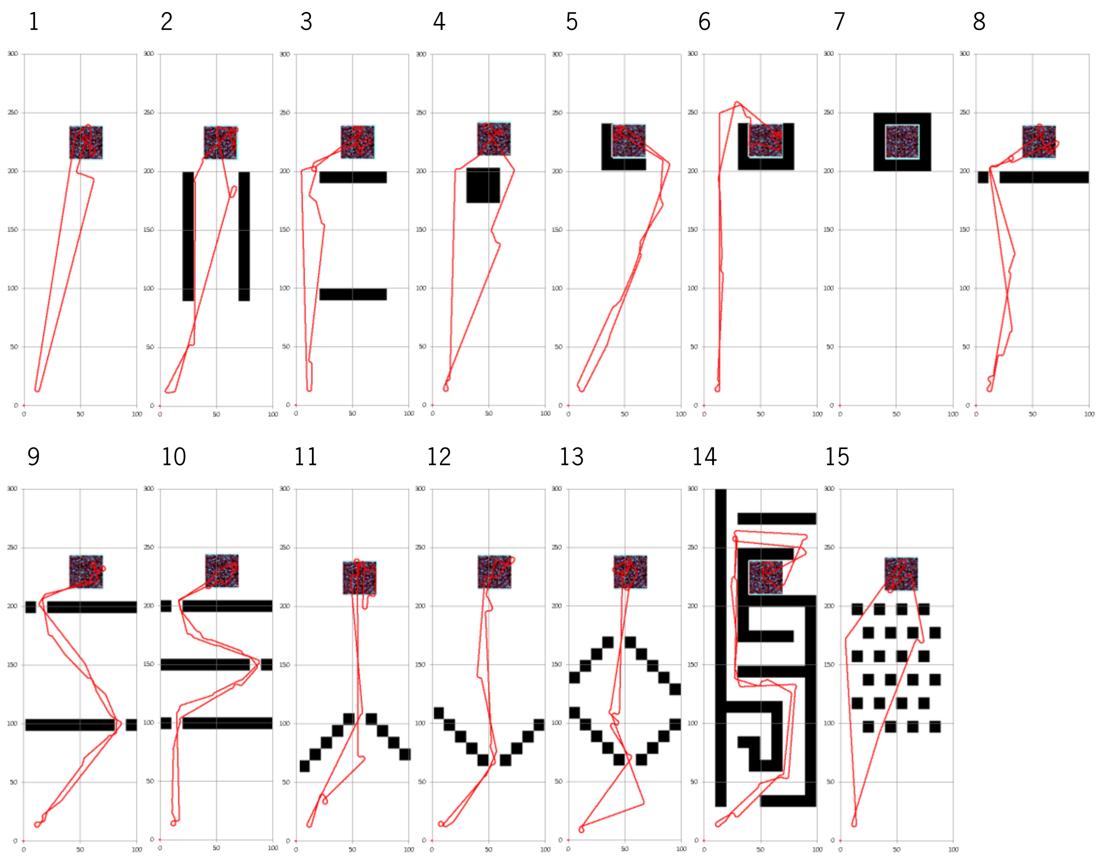


Fig. 36 – Examples of paths obtained for each batch.

Finally and perhaps the most important general remark is that the TWIN-RRT* found feasible results in every attempt, whenever a feasible path was possible. Except for the 100 runs in situation 7 (unfeasible situation), the TWIN-RRT* retrieved feasible paths for every run, 1 400 runs in total, which means 100% successful rate, no matter how many obstacles are in the working space or how are they distributed. This fact corroborates the assumption that the TWIN-

RRT* algorithm inherits the probabilistic completeness feature of RRT*, i.e., it is able to find a feasible solution if there exists one.

5.6. Influence of minimum curvature radius on the performance

Non-holonomic robots have a minimum curvature radius that limits the robot movement in the working space. The larger the curvature radius, the more difficult is for the robot to make sharp curves and consequently more difficult it is for the planner to find feasible and efficient paths. To confirm this assumption a set of 16 simulation batches were performed with 100 runs per batch. All the 1 600 runs used the same standard settings except for the minimum curvature radius ($\rho_{\min}$) value (see Table 14). The first batch uses ρ=0 which corresponds to the situation of a holonomic robot.

Table 14 - Parameters for influence of minimum curvature radius on the performance simulation

General Parameters	Number of targets in the workspace	ballsLaunched = 5 000
	Maximum number of iterations to perform	maxIterations = 10 000
	Minimum number of targets to reach	minBallsToCollect = 0
	Computational time limit (0 = no limit)	maxTime (s) = 100s
	Maximum path distance (0 = no restriction)	maxDistance (m) = 25 000
	Minimum ratio to achieve (0 = no restriction)	minRatio (balls/m) = 0.00
TWIN–RRT* Technical Parameters	Distance between two consecutive positions used to check for obstacles and calculate the path	discretisationStep (m) = 0.50
	Minimum curvature radius	rho (m) = variable
	Gamma: parameter used to calculate the region to search near neighbours.	gamma(m)= 10.00

The results are shown in Table 15 and Fig. 37. It is notorious that the TWIN-RRT* maintains a very stable behaviour from ρ=0m to ρ=10m, range where the differences observed are not statistically relevant. From ρ=15m and over, the algorithm fails to return a valid path. That is reasonably explained by the fact that the starting configuration is at coordinates (x,y,θ)=(12,12,0), which means that ρ values up to 12m will be accepted because it is the maximum distance to the field boundary. Values of ρ above 12m will certainly fail to produce feasible paths, unless the initial configuration is placed on another position of the working space.

The situation in which ρ=0 corresponds to a holonomic vehicle, thus without kinematic restrictions. It should be noticed that the algorithm accommodates this situation because it is a sub-problem or simplification of the kinematic constrained version of the TWIN-RRT* algorithm.

This clearly means that, for the established framework, the minimum curvature radius (ρ_{min}) does not influence the performance of the algorithm. Nevertheless, caution should be taken in different circumstances, for example in cases where there are more or bigger obstacles in the working space and/or the targets are more concentrated. As pointed out in the beginning, all the conclusions are only valid for the current framework, thus extending the same conclusions to other circumstances should be carefully done.

Table 15 – Efficiency gain results depending on the minimum curvature radius

rho Value	0	1	2	3	4	5	6	7	8	9	10	15	20	30	40	50
Eff. Gain	**2,34**	2,19	2,17	2,16	2,16	2,14	2,14	2,16	2,16	2,13	2,12	**0,00**	0,00	0,00	0,00	0,00
STDEV	0,82	0,11	0,10	0,11	0,11	0,13	0,12	0,11	0,14	0,13	0,14	0,00	0,00	0,00	0,00	0,00
Max. eff. Gain	**9,00**	2,54	2,48	2,49	2,49	2,51	2,42	2,48	2,58	2,42	2,38	0,00	0,00	0,00	0,00	0,00

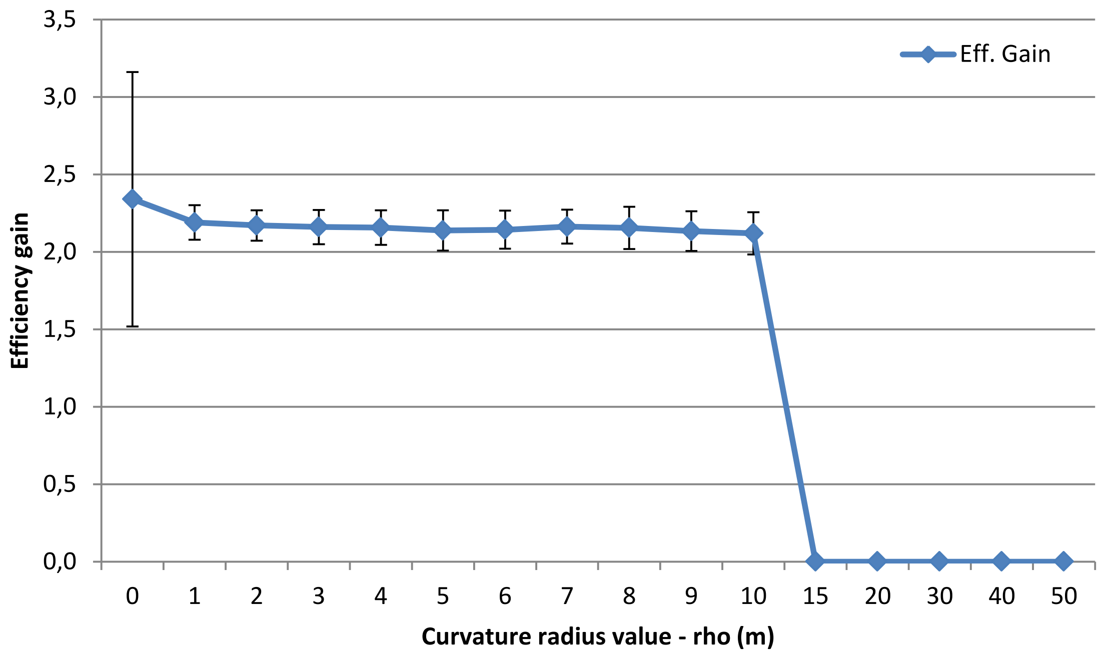


Fig. 37 – Efficiency gain dependence on the curvature radius

In Fig. 38 there are some examples of paths obtained for the cases where the TWIN-RRT* found a feasible solution. The shapes of the paths are not pertinent. The important fact to mention is the common tendency for the path to extend towards the central area where the targets are more concentrated.

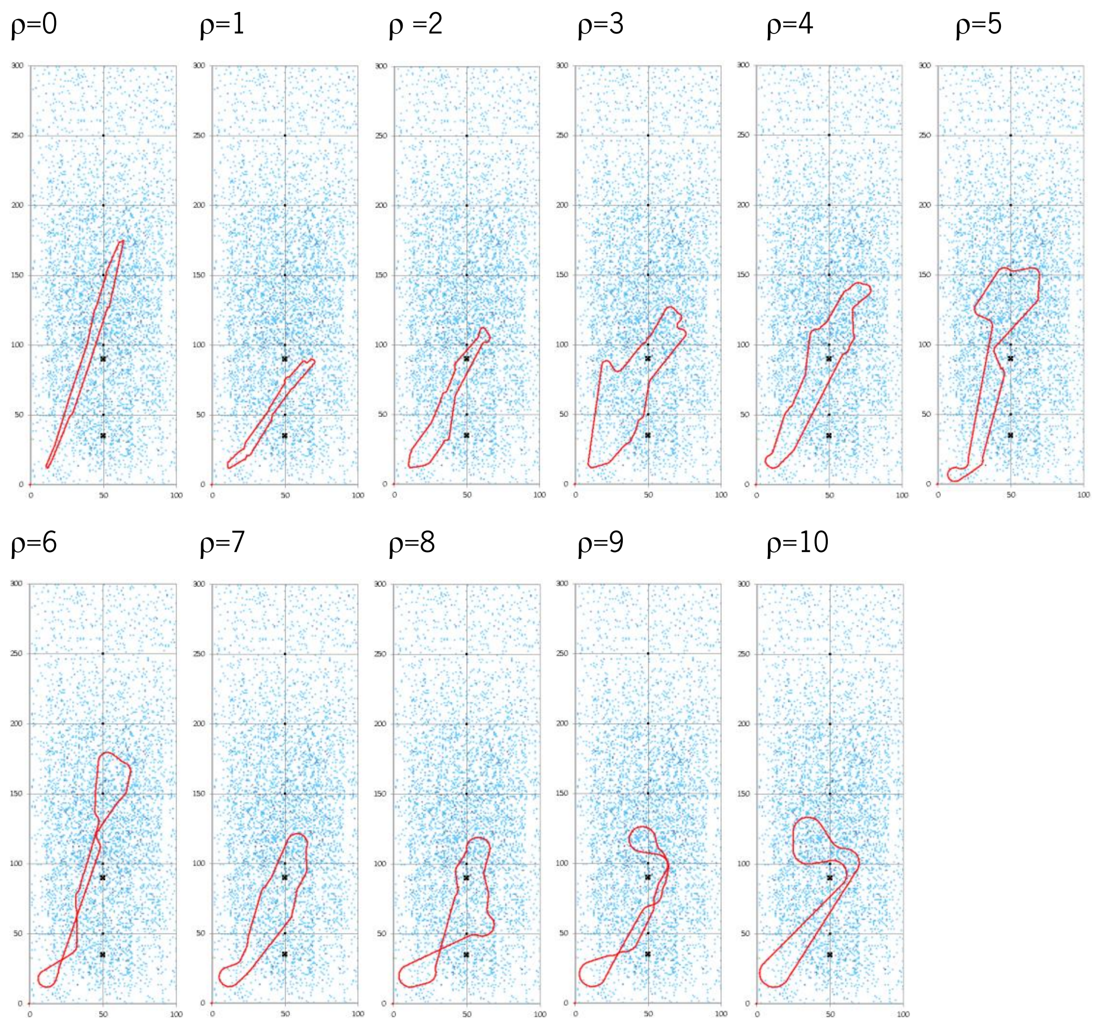


Fig. 38 – Representation of some examples with different minimum curvature radius (ρ).

5.7. Different starting configurations

In order to test the influence of the starting position and orientation of the robot on the efficiency gain, a simulation was performed with 10 different initial configurations corresponding to 10 batches of 100 runs each. All the 1000 runs used the same standard settings (see Table 16).

Table 16 - Parameters for different starting configurations simulation

General Parameters	Number of targets in the workspace	ballsLaunched = 5 000
	Maximum number of iterations to perform	maxIterations = 10 000
	Minimum number of targets to reach	minBallsToCollect = 0
	Computational time limit (0 = no limit)	maxTime (s) = 100s
	Maximum path distance (0 = no restriction)	maxDistance (m) = 25 000
	Minimum ratio to achieve (0 = no restriction)	minRatio (balls/m) = 0.00
TWIN–RRT* Technical Parameters	Distance between two consecutive positions used to check for obstacles and calculate the path	discretisationStep (m) = 0.50
	Minimum curvature radius	rho (m) = 2.00
	Gamma: parameter used to calculate the region to search near neighbours.	gamma(m)= 10.00

For each batch a different configuration is used according to Table 17. These configurations were empirically predetermined in order to have starting configurations at reasonably relevant and different positions of the workspace (see Table 17).

Table 17 – Initial configuration per batch

	x	**y**	**θ**
1.	12	12	0
2.	12	150	0
3.	12	288	0
4.	50	12	0
5.	50	140	0
6.	50	160	0
7.	50	288	0
8.	88	12	0
9.	88	150	0
10.	88	288	0

The results are summarized on

Table 18 and graphically represented in Fig. 39.

Table 18 – Efficiency gain results for each batch.

Batch	1	2	3	4	5	6	7	8	9	10
Position	(12,12)	(12,150)	(12,288)	(50,12)	(50,140)	(50,160)	(50,288)	(88,12)	(88,150)	(88,288)
Eff.gain	2,16	2,86	**1,78**	2,61	**3,60**	3,37	1,78	2,20	2,86	1,80
STDEV	0,10	0,21	0,09	0,21	0,40	0,35	0,08	0,11	0,17	0,07
Max eff.gain	2,41	3,40	2,04	3,13	4,85	5,34	2,14	2,47	3,40	1,98

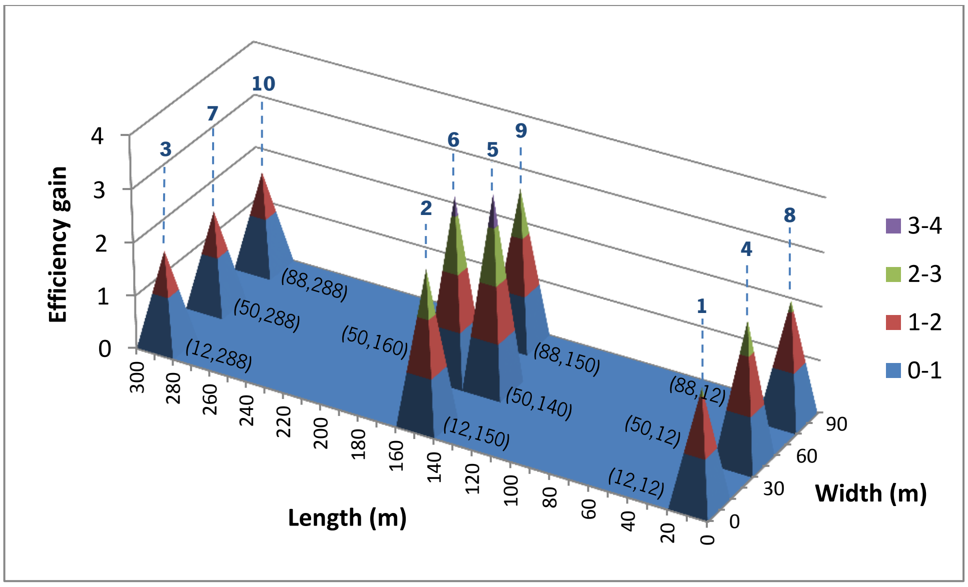


Fig. 39 – Representation of efficiency gain results for the selected points in the workspace.

The efficiency gain is higher when the starting point is near a region where the targets concentration is higher (ex: situation 5 and 6, corresponding to starting points (50,140) and (50,160) respectively). Inversely, the worse efficiency gains are obtained when the starting point is on lower targets concentration regions (ex: batches 3, 7 and 10, corresponding to starting points (12,288), (50,288) and (88,288) respectively). In fact, the efficiency gain of the best case (situation 5, corresponding to starting point (50,140)) is twice the value of the worst case (situation 3, corresponding to starting point (12,288)).

Fig. 40 illustrates a typical path for each situation. It is possible to observe that for the cases where the starting point is located in the most extreme regions (batches 1, 3, 8 and 10) of the workspace, the algorithm retrieves longer paths that extend towards the centre of the field where targets are more concentrated.

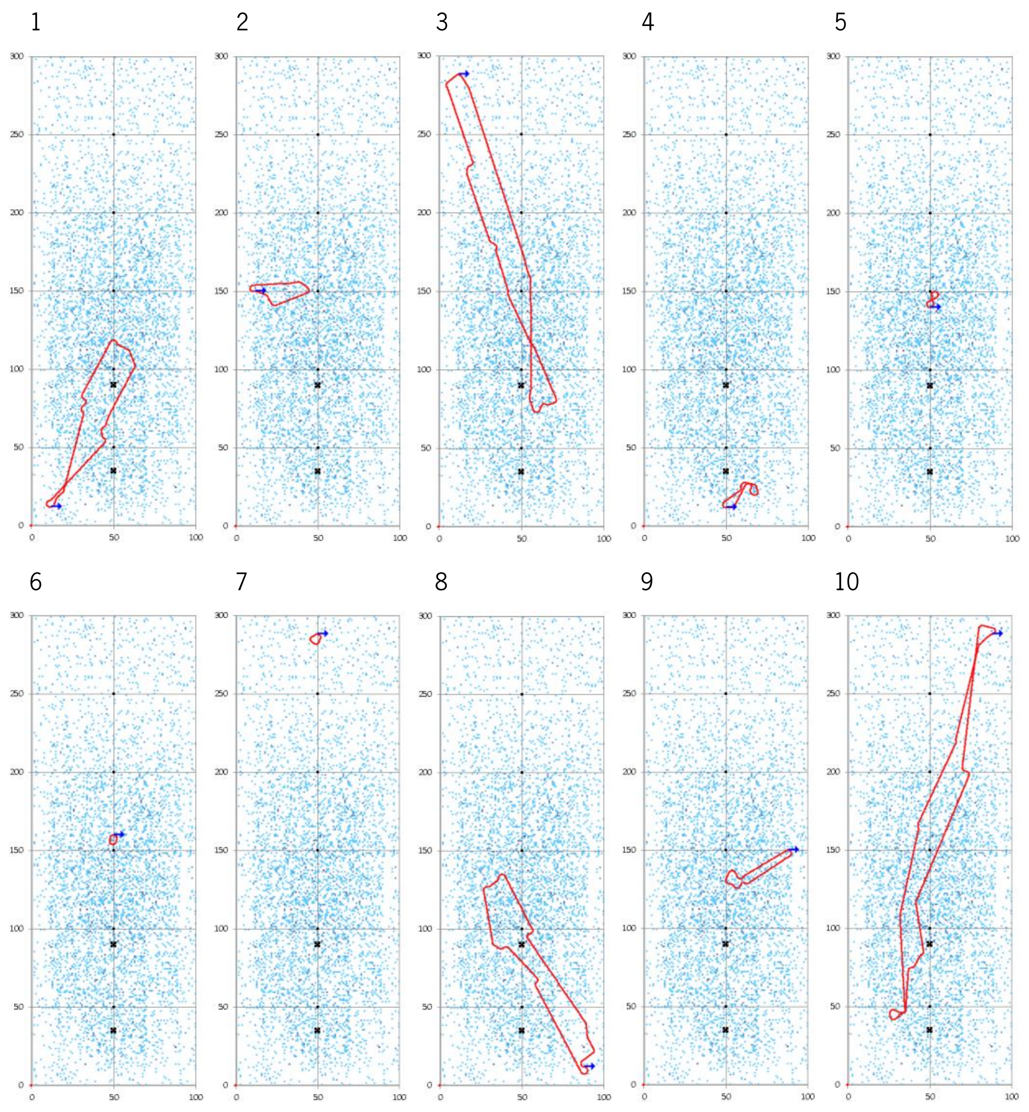


Fig. 40 – Path examples for each of the batches with different starting configurations

5.8. Different platforms with different CPUs

To evaluate the impact on performance by using different processors, 3 simulation batches were performed. Each batch comprises 100 runs. Each run comprises 10 000 iterations of the algorithm. The total running time was registered. The settings used correspond to those of the standard conditions except for the computational time limit which was removed (see Table 19).

Table 19 - Parameters for different platforms with different CPUs simulation

General Parameters	Number of targets in the workspace	ballsLaunched = 5 000
	Maximum number of iterations to perform	maxIterations = 10 000
	Minimum number of targets to reach	minBallsToCollect = 0
	Computational time limit (0 = no limit)	maxTime (s) = (no limit)
	Maximum path distance (0 = no restriction)	maxDistance (m) = 25 000
	Minimum ratio to achieve (0 = no restriction)	minRatio (balls/m) = 0.00
TWIN–RRT* Technical Parameters	Distance between two consecutive positions used to check for obstacles and calculate the path	discretisationStep (m) = 0.50
	Minimum curvature radius	rho (m) = 2.00
	Gamma: parameter used to calculate the region to search near neighbours.	gamma(m)= 10.00

This experiment is not to be considered as a proper benchmark test. It serves as a mere indication of the possible impact that different CPUs may have on a practical implementation.

Three different platforms were used, further designated as UM, PC, BB, which details are described in

Table 20.

Table 20 – Platform features

Platform	**1** (UM)	**2** (PC)	**3** (BB)
Processor	Intel® Xeon® Processor X3430, 2.40 GHz	Intel Core 2 Duo P7450, 2.13 GHz	Sitara AM3358BZCZ100, 1 GHz
RAM	8 GB	4 GB	512 MB
OS	64 bits (Linux)	64 bits (Windows)	32 bits (Linux)

Concerning the efficiency gain, there is no significant difference among the results obtained for the three different platforms (see Table 21 and Fig. 41), which means that the quality of TWIN-RRT* results is independent of the platform. This is an important assessment that can be of special interest for those who intend to apply TWIN-RRT* algorithm in real-time applications.

Table 21 – Average efficiency gain obtained for each platform

Processor	1 (UM)	2 (PC)	3 (BB)
Efficiency gain	2,24	2,24	2,23
STDEV	0,09	0,11	0,11

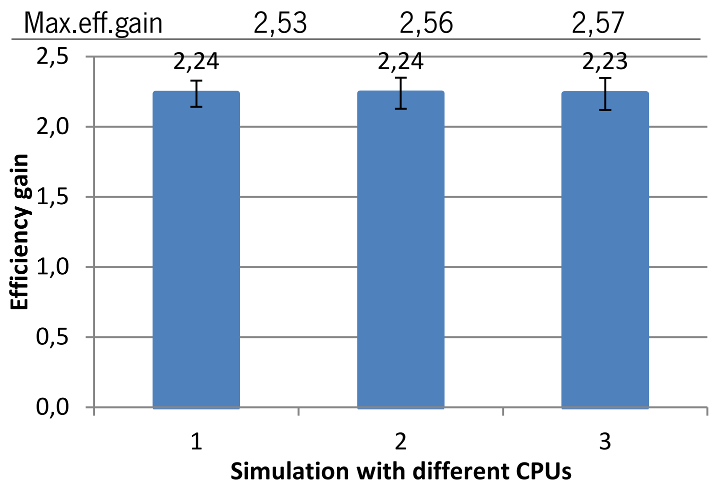


Fig. 41 – Average results for efficiency gain with different processors

Concerning the time required to compute 10 000 iterations, there is a huge difference among the results obtained for the three different platforms (see Table 22 and Fig. 42). The PC is about 4 times slower than the UM platform. The BB is about 23 times slower than the UM platform and almost 6 times slower than the PC platform. Once again these results show the importance of proper choice of the platform depending on the application. Unfortunately it is not possible to compare the TWIN-RRT* performance to other algorithms, because they are not directly applicable to the stated problem. That would be an interesting topic to study in future.

Table 22 – Average time required for different processors to compute 10 000 iterations.

Processor	1 (UM)	2 (PC)	3 (BB)
Time (min)	5	20	116
STDEV	0	5	1
MAX (min)	5	53	118

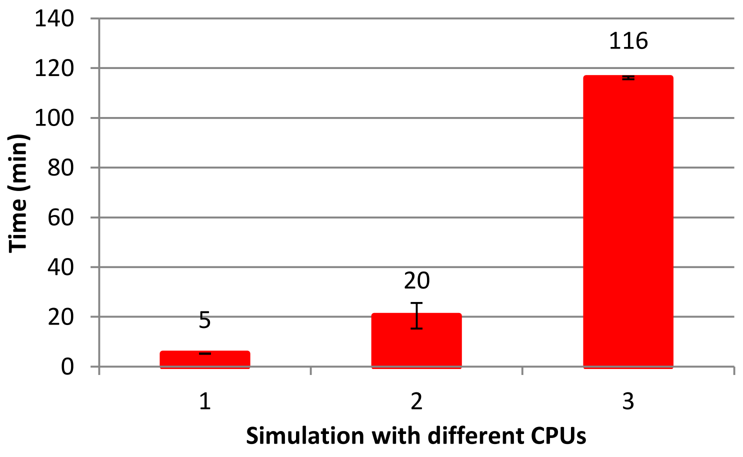


Fig. 42 - Average time required to complete 10 000 iterations with different processors.

5.9. Improving the number of reached targets

Improving the number of reached targets is not the main objective of TWIN-RRT* algorithm. Nevertheless, the typical 2% of total targets (found in experiment 3) can be improved by different ways at the expense of losing some efficiency. One of those ways is explained next.

In this experiment the algorithm runs repeatedly to improve the number of reached targets. After finding a feasible solution, reached targets are taken out of the space representation and another run starts over. Unlike the previous experiments, the target distribution on the workspace is not reset, which means that the initial target distribution is only set once at the beginning of the first run. Every subsequent run inherits the remaining targets not collected on previous runs. This strategy continues over time to increase the accumulated total number of reached targets.

Fig. 43 shows a representation of the paths retrieved by the TWIN-RRT* algorithm during the first 6 consecutive runs. It is not feasible to represent all the 100 runs in the same chart, but the most relevant factor is already derivable from this representation, which is the tendency to vary the region covered by the path along successive runs. On the first run, after the first path being computed, the targets along the retrieved path are removed from the workspace. This prevents that subsequent runs add the same targets to other paths. Thus, the second run starts with the remaining targets on the field, after subtracting the ones along the path of the first run.

The simulation goes on, and further passages on the same location cannot add the targets that were already taken into account by a path on a previous run. From Fig. 43 it is possible to observe that paths retrieved by runs 4, 5 and 6 are already more spread towards more peripheral areas than the paths retrieved by runs 1, 2 and 3. The path resulting from run 6 is also more elongated, having a longer distance than previous paths.

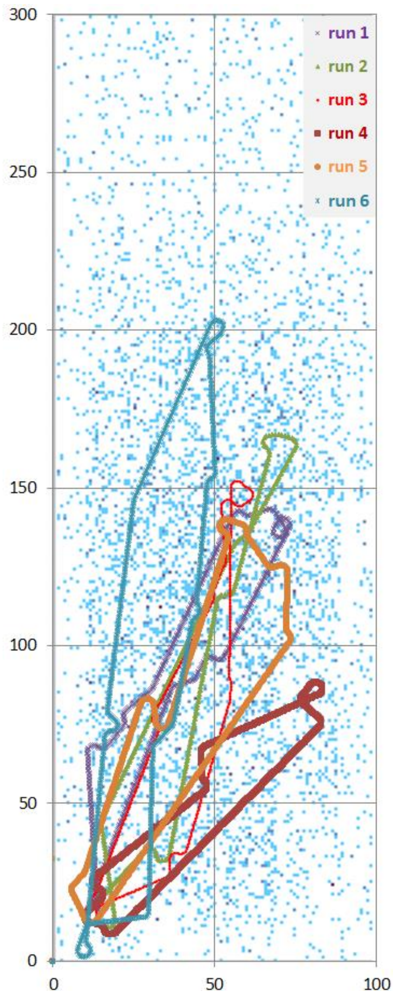


Fig. 43 – Examples of the paths resulting from the first 6 consecutive runs.

It is possible to reach about 40% of total targets without dropping below the 1.0 efficiency gain barrier (limit below which a full sweeping strategy will perform better). In a practical situation the robot would perform the runs sequentially. Moreover in the case study situation new targets are continuously being added to the field, so better results are to be expected with this strategy.

The chart of Fig. 44 also shows that the efficiency gain decays along the number of runs. After 25 runs the critical point (1.0 efficiency gain) is reached. It would be extremely inefficient to use this strategy with more than 25 runs because a sweeping strategy (efficiency gain = 1.0) would perform better.

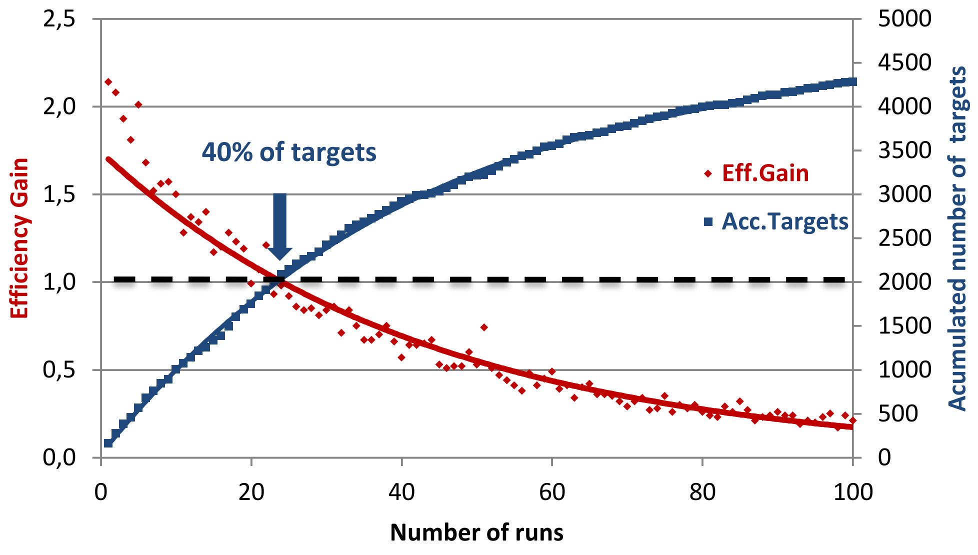


Fig. 44 – Efficiency gain for each run and accumulated number of targets over increasing number of runs.

The chart on Fig. 45 complements this information. As the number of runs increase, the number of targets reached per run decreases and the distance travelled increases. Both factors contribute for the fast decay of the efficiency gain observed in the previous chart.

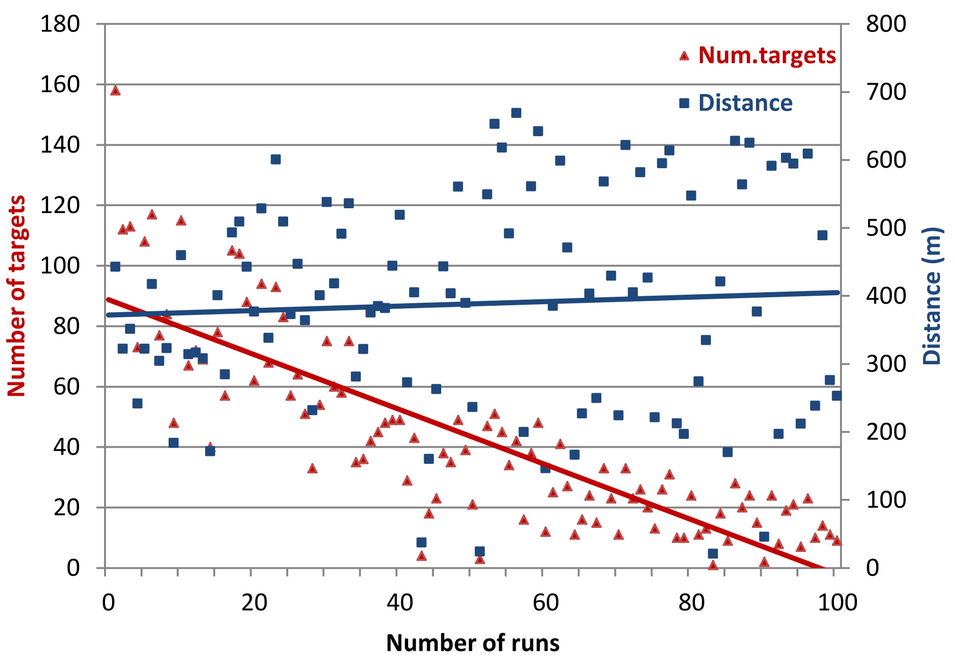


Fig. 45 – Evolution of the number of targets reached and evolution of average distance travelled over increasing number of runs.

5.10. Influence of the technical parameter "ball radius"

The TWIN-RRT* uses a parameter, inherited from the RRT*, called "ball radius", that decays as the number of vertices (or nodes) in the tree increases. The "ball radius" is a measure used to search for neighbours in the vicinity of a given node. As the name suggests, the vicinity is calculated by a circle centred in the given node and extending to a maximum length determined by the "ball radius". All the nodes contained inside that circle are considered neighbour nodes.

The ball radius is a parameter that influences the performance. It should be relatively low because it imposes an overhead on processing time. This "ball radius" is used to find nearest neighbours at least in two computational expensive subroutines of TWIN-RRT*, namely NEAR and REWIRE functions (see algorithm 4 on section 4.2), so high values for the "ball radius" parameter will significantly affect the algorithm performance.

The ball radius is configured to decrease as the number of vertices in the tree increases to avoid the iteration processing time from becoming unfeasible (see Fig. 46). The equation for this behaviour is expressed in equation 12.

$$Ball\ radius = 50\ \times GAMMA \times \left(\frac{\log(numVertices + 1)}{numVertices + 1}\right)^{\frac{1}{numDimensions}} \tag{12}$$

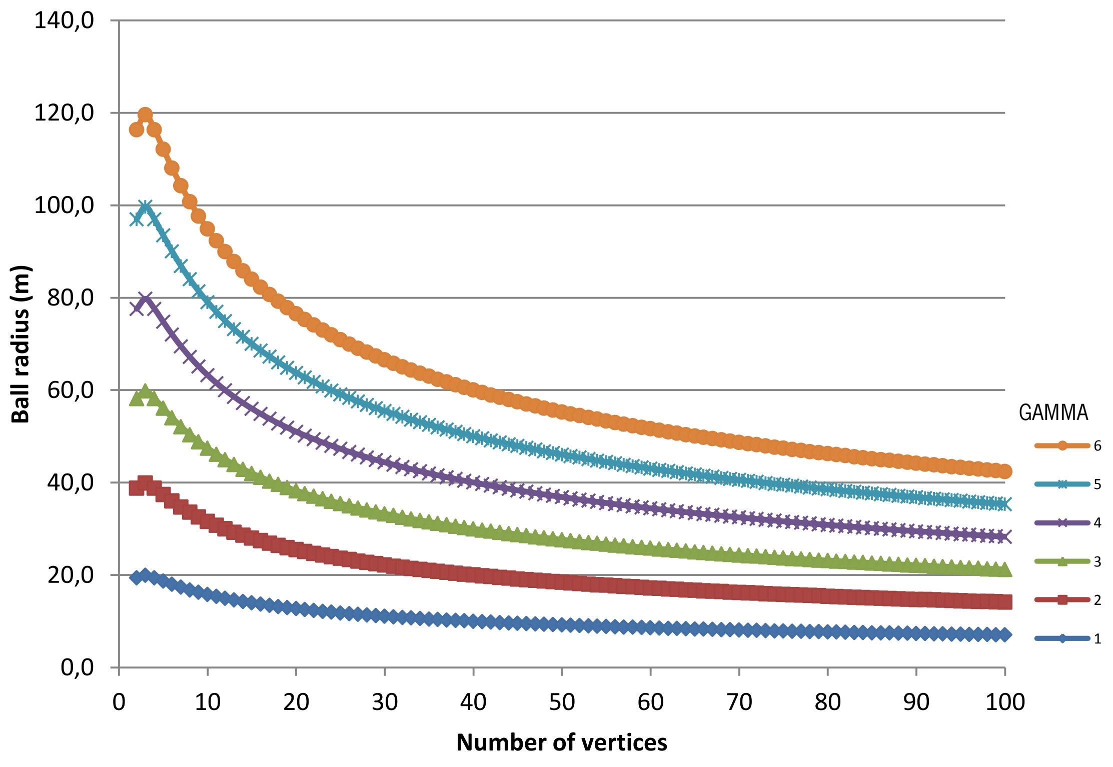


Fig. 46 – Decrease of the "ball radius" parameter as the number of vertices increases, for different GAMMA values.

The efficiency gain depends on the "ball radius", which in turns, depends on the GAMMA value. To evaluate the impact on the efficiency gain by using different GAMMA values, 17 simulation batches were performed. Each batch comprises 100 runs. The settings used correspond to those of the standard conditions except for the GAMMA value that changes from batch to batch according to the following sequence: 0.0001, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 30, 40, 50, 100 (see Table 23).

Table 23 - Parameters for influence of the technical parameter "ball radius" simulation

General Parameters	Number of targets in the workspace	ballsLaunched = 5 000
	Maximum number of iterations to perform	maxIterations = 10 000
	Minimum number of targets to reach	minBallsToCollect = 0
	Computational time limit (0 = no limit)	maxTime (s) = 100s
	Maximum path distance (0 = no restriction)	maxDistance (m) = 25 000
	Minimum ratio to achieve (0 = no restriction)	minRatio (balls/m) = 0.00

TWIN–RRT* Technical Parameters	Distance between two consecutive positions used to check for obstacles and calculate the path	discretisationStep (m) = 0.50
	Minimum curvature radius	rho (m) = 2.00
	Gamma: parameter used to calculate the region to search near neighbours.	gamma(m)= varies

Following the experiments the results presented on

Table 24 and Fig. 47 were obtained.

Table 24 – Results of efficiency gain depending on the "ball radius" value

Gamma	0	1	2	3	4	5	6	7	8	9	10	15	20	30	40	50	100
Ball radius	0	1,13	2,27	3,4	4,53	5,66	**6,8**	7,93	9,06	10,2	11,3	17	22,7	34	45,3	56,6	113
Eff.gain	1,79	1,86	2,07	2,16	2,16	2,18	**2,20**	2,19	2,19	2,17	2,18	2,15	2,13	2,11	2,11	2,11	2,11
STDEV	0,18	0,16	0,13	0,14	0,10	0,12	0,12	0,13	0,12	0,11	0,12	0,10	0,12	0,11	0,10	0,11	0,11
Max.eff.gain	2,33	2,21	2,45	2,55	2,37	2,55	2,59	2,85	2,68	2,56	2,55	2,40	2,54	2,42	2,42	2,46	2,40

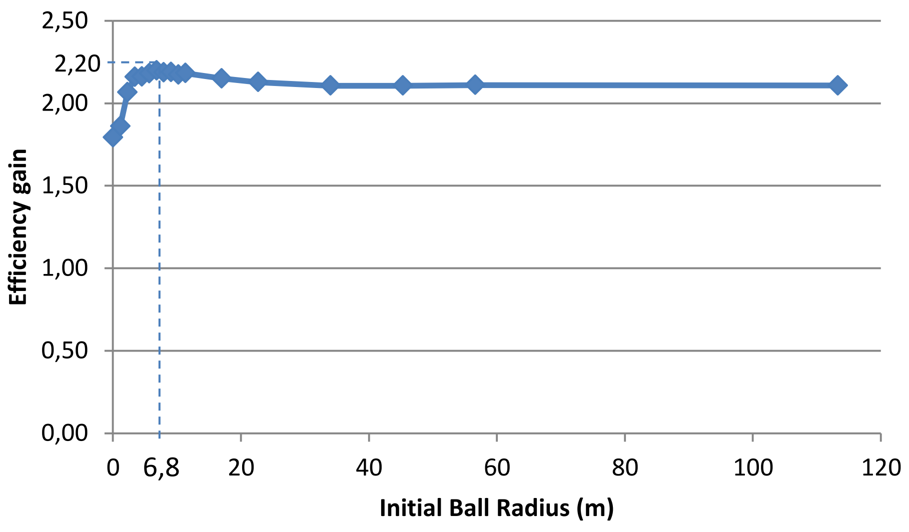


Fig. 47 – Representation of the evolution of efficiency gain varying the ball radius

The best initial "ball radius" value encountered for the present framework is 6.8 m which means setting GAMMA to 6.

It should be noticed that it may vary for different frameworks and for that reason, an experiment like this should be performed for different frameworks in order to choose the most suitable GAMMA value.

5.11. Different sampling methods

As briefly discussed in section 3.10, sampling is by itself an important and non-trivial problem. The most relevant properties of sampling are related to denseness, uniformity, regularity and dispersion. It was mentioned before that random sampling strategy used by PRM and RRT has proved to return good results in terms of the quality of sampling which by consequence provide also good quality path results.

It is also known that by biasing the sampling method it is possible to improve the results in some particular cases.

It is not within the scope of this document to exhaustively test heuristic methods to bias the sampling procedure, therefore, just to have a representative case of the importance of this topic, an experiment was designed to test a biased sampling method that privileges the locations where the probability of finding more targets is higher.

Biasing the sampling method towards the most probable target location can be an alternative to improve the efficiency gain. To test this hypothesis, a set of 2 simulation batches were performed with 100 runs per batch. All the 200 runs used the same standard settings (see Table 25).

Table 25 - Parameters for different sampling methods simulation

General Parameters	Number of targets in the workspace	ballsLaunched = 5 000
	Maximum number of iterations to perform	maxIterations = 10 000
	Minimum number of targets to reach	minBallsToCollect = 0
	Computational time limit (0 = no limit)	maxTime (s) = 100s
	Maximum path distance (0 = no restriction)	maxDistance (m) = 25 000
	Minimum ratio to achieve (0 = no restriction)	minRatio (balls/m) = 0.00
TWIN–RRT* Technical Parameters	Distance between two consecutive positions used to check for obstacles and calculate the path	discretisationStep (m) = 0.50
	Minimum curvature radius	rho (m) = 2.0
	Gamma: parameter used to calculate the region to search near neighbours.	gamma(m)= 10.00

On the first batch, the sampling was random as all the previous experiments. On the second batch the sampling was biased by using the same seeding matrix (probabilistic matrix) used to generate the targets distribution. In this way, the sampling method follows the Monte Carlo distribution corresponding to the same probability of the target distribution.

As consequence, the random samples are likely to be near the targets but not exactly coincident. Note that it is assumed that the targets location are not known in advance to the planner, only the statistically relevant data which is translated to a given probability for each location on the workspace, and for practical purposes represented by a probabilistic matrix in the simulation system. The results showing how the efficiency gain is affected by these different sampling strategies are shown in Fig. 48.

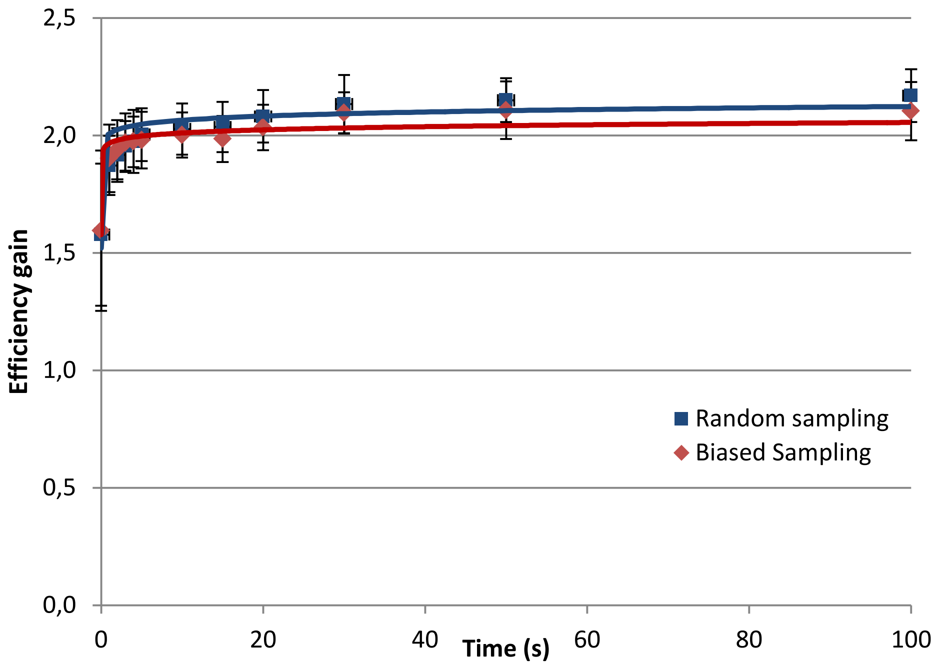


Fig. 48 – Influence of different sampling methods on the efficiency gain.

The biased sampling method retrieved results slightly worse but not significantly different than the normal random method (of experiment 1). These results corroborate the idea defended by many researchers that a normal random sampling is the best strategy for planning methods.

Using other sampling methods or heuristics may in fact provide better results, but is out of the scope of this document to further explore this subject. Future work is needed to explore better sampling strategies for TWIN-RRT*, which may also depend on the workspace, namely on the targets distribution and number and location of the obstacles.

5.12. *Extend* and *Connect* versions

The original RRT algorithm, on which TWIN-RRT* is based, has many variations, one of them being related to how the tree expands. There are two main approaches: the *Connect* and the *Extend* (see Fig. 49). The *Connect* approach in a 2D workspace consists of directly adding to the tree the sampling point having coordinates (x,y) whenever it is possible. The *Extend* approach instead of simply adding the sampled point, it first verifies if it is reached by any of the points belonging to the tree within a predefined range (step). If not, then a point is generated on a straight line linking the closest point on the tree and the sampled one and at the predefined distance (step) from the closest point on the tree (see Fig. 49).

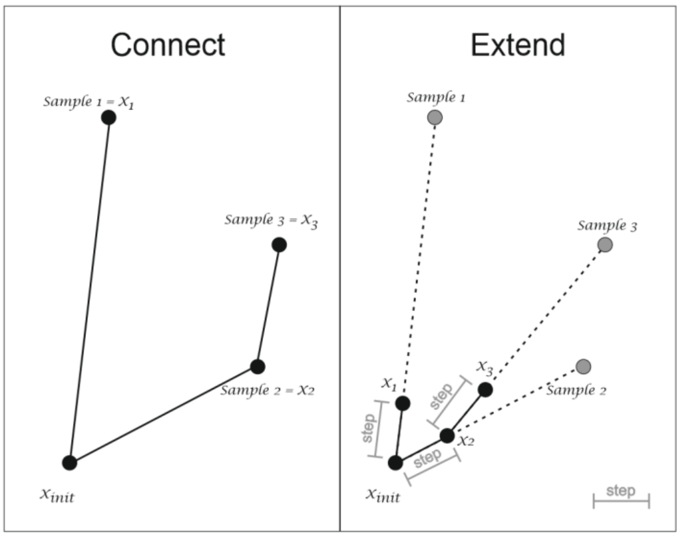


Fig. 49 – Representation of the distinction between the *Connect* and *Extend* approaches in RRT.

All the experiments presented before used the *Connect* approach. The next experiment was designed to assess the difference in terms of performance (impact on efficiency gain), by using the *Extend* version, and varying the length of the step.

16 batches using different step lengths changing from batch to batch according to the following sequence: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 30, 40, 50, 100.

The settings used correspond to those of the standard conditions (see Table 26).

Table 26 - Parameters for *Extend* and *Connect* versions simulation

General Parameters	Number of targets in the workspace	ballsLaunched = 5 000
	Maximum number of iterations to perform	maxIterations = 10 000
	Minimum number of targets to reach	minBallsToCollect = 0
	Computational time limit (0 = no limit)	maxTime (s) = 100s
	Maximum path distance (0 = no restriction)	maxDistance (m) = 25 000
	Minimum ratio to achieve (0 = no restriction)	minRatio (balls/m) = 0.00
TWIN–RRT* Technical Parameters	Distance between two consecutive positions used to check for obstacles and calculate the path	discretisationStep (m) = 0.50
	Minimum curvature radius	rho (m) = 2.00
	Gamma: parameter used to calculate the region to search near neighbours.	gamma(m)= 10.00

The length of the *Extended* Step influences dramatically the performance of the algorithm (see Fig. 50). The best extended step for the current framework is 15m. Using this step an average value of 2.3 was found for the efficiency gain. For step length lower than 15m the efficiency gain drops very quickly. Actually, step lengths below 3m render the path very inefficient (efficiency gain < 1.0). For values above 15m there is gradually a slightly decrease in performance, but it never reaches the critical value (efficiency = 1.0), being the lowest value obtained equal to 2.0 which is still twice as better as the sweeping strategy (efficiency = 1.0). Another remark which is worth stressing refers to the fact that the discretisation decisions (described in section 4.8) used for the current framework influence the results. In other words, it means that these results are only valid for the current framework and other applications may require different parameterization after testing the influence of the step length on the efficiency gain.

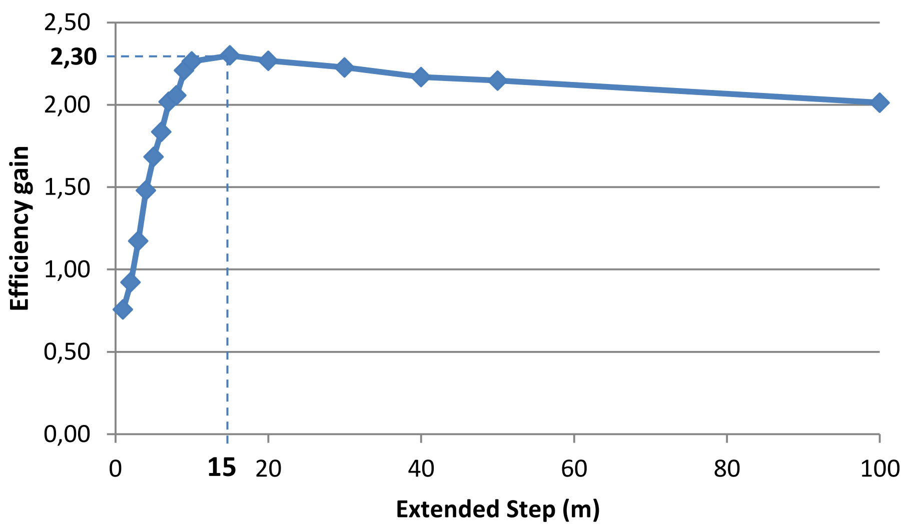


Fig. 50 – Efficiency gain depending on the *Extended Step* length.

The *Extended* version was further compared to the *Connect* version. The results of the best *Extended* case (extended step = 15m) were plotted against the values of the *Connect* version (Fig. 51). The *Extended* method retrieved results slightly better but not significantly different than the *Connect* method (of experiment 1).

By observing the results of Fig. 51 it is obvious that both versions have a very similar behaviour. The problem is that the *Extended* version really depends on the correct choice of the step value. If the best efficiency gain value (2,3 in this case) is not carefully determined by carrying out an experiment like the one plotted in Fig. 50, then it is not advisable to use the *Extended* version with a step value randomly chosen. In such cases, where it is not feasible or practical to perform such experiment, than the most reasonable choice is to use the *Connect* version. The *Connect* version does not require initial tuning of the step value and provides results not significantly different from the best result of the *Extended* version. This proves that the initial strategy of adopting the *Connect* version was the wisest choice.

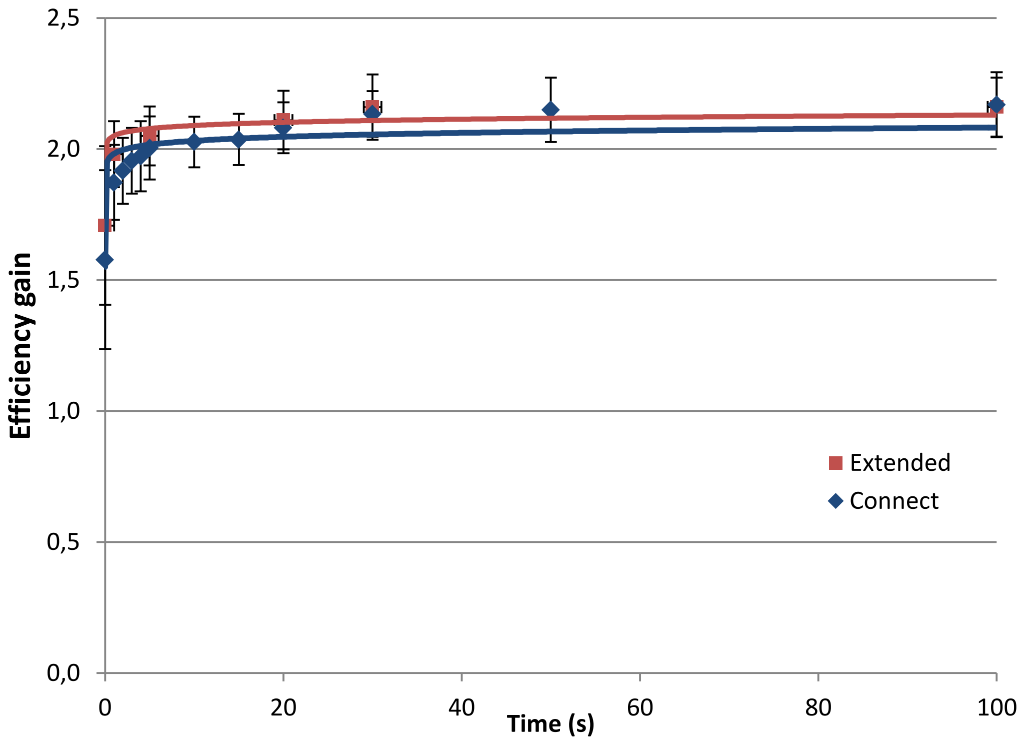


Fig. 51 – Comparison between the Extended and Connect versions of TWIN-RRT* algorithm.

5.13. Tuning *FindBestParent* and *Rewire* Sub-routines

The TWIN-RRT* uses the ‘ratio’ (targets/distance) as the metric to maximise (more specifically in the *CheckPath* sub-routine) to retrieve the best bath, but there are other two sub-routines where the metric ‘distance’ is maintained as in the original RRT*: those are *FindBestParent* and *Rewire* sub-routines (see Algorithms 3 and 4 in section 4.2) and both use the ‘distance’ as the metric to minimise when searching for the best parent node for a given sample and changing the parent for a given neighbour node respectively. Using the ‘ratio’ (targets/distance) as a metric has the disadvantage of inserting a computational overhead.

This simulation was performed to check if using the metric ‘ratio’ (targets/distance) instead of the ‘distance’ in those subroutines would provide better results. 2 simulation batches were performed with 100 runs per batch. All the 200 runs used the same standard settings. The first batch uses ‘distance’ as metric and the second batch uses the ‘ratio’ (targets/distance). The standard settings were used for both batches (see Table 27).

Table 27 - Parameters for tuning *FindBestParent* and *Rewire* Sub-routines simulation

General Parameters	Number of targets in the workspace	ballsLaunched = 5 000
	Maximum number of iterations to perform	maxIterations = 10 000
	Minimum number of targets to reach	minBallsToCollect = 0
	Computational time limit (0 = no limit)	maxTime (s) = 100s
	Maximum path distance (0 = no restriction)	maxDistance (m) = 25 000
	Minimum ratio to achieve (0 = no restriction)	minRatio (balls/m) = 0.00
TWIN–RRT* Technical Parameters	Distance between two consecutive positions used to check for obstacles and calculate the path	discretisationStep (m) = 0.50
	Minimum curvature radius	rho (m) = 2.00
	Gamma: parameter used to calculate the region to search near neighbours.	gamma(m)= 10.00

The final results are shown on Fig. 52. The batch using the 'ratio' (targets/distance) as metric instead of 'distance' retrieved slightly better results, but not statistically relevant to be considered different.

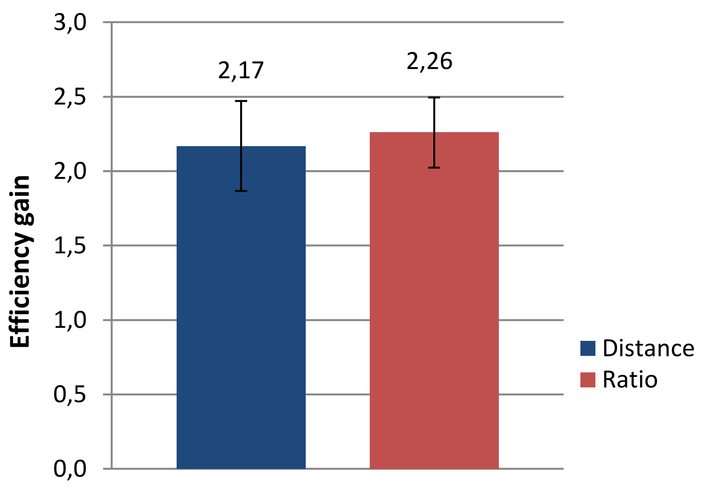


Fig. 52 – Influence of using different metrics in the sub-routines *FindBestParent* and *Rewire.*

5.14. Further increasing the number of reached targets

Besides the 'ratio' (targets/distance) or the 'distance', other interesting metric for the current application may also be used, such as maximise the number of targets while selecting the best path. As seen before, the three sub-routines where the metrics can influence the results are *FindBestParent*, *Rewire* and *CheckPath*. Another experiment was designed to test in more detail the influence of the above mentioned metric.

In order to perform this simulation the *Rewire*, *FindBestParent* and *CheckPath* functions were modified to choose the path according to the metric used.

4 batches with 100 runs each one were tested using the standard settings. Table 28 resumes the implementations details for each batch, sub-routine and metric used. Many other combinations are possible, and the ones chosen in this experiment might not even be the ones that provide the best results. Further experiments are needed to study in more detail all the possible variations.

Table 28 – Design implementation details

Sub-routine	**Batch 1**	**Batch 2**	**Batch 3**	**Batch 4**
FindBestParent	Distance	Ratio (targets/distance)	N° Targets	Distance
Rewire	Distance	Ratio (targets/distance)	N° Targets	Distance
CheckPath	Ratio (targets/distance)	Ratio (targets/distance)	N° Targets	N° Targets

Analysing the results in terms of efficiency gain (see Fig. 53), it is clear that the strategy to maximise the number of targets reached (batches 3 and 4) comes with a high cost in terms of loss of efficiency. Note that the case of batch 3 is precisely on the limit of "efficiency". The case of batch 4 is notably inefficient (efficiency gain < 1.0).

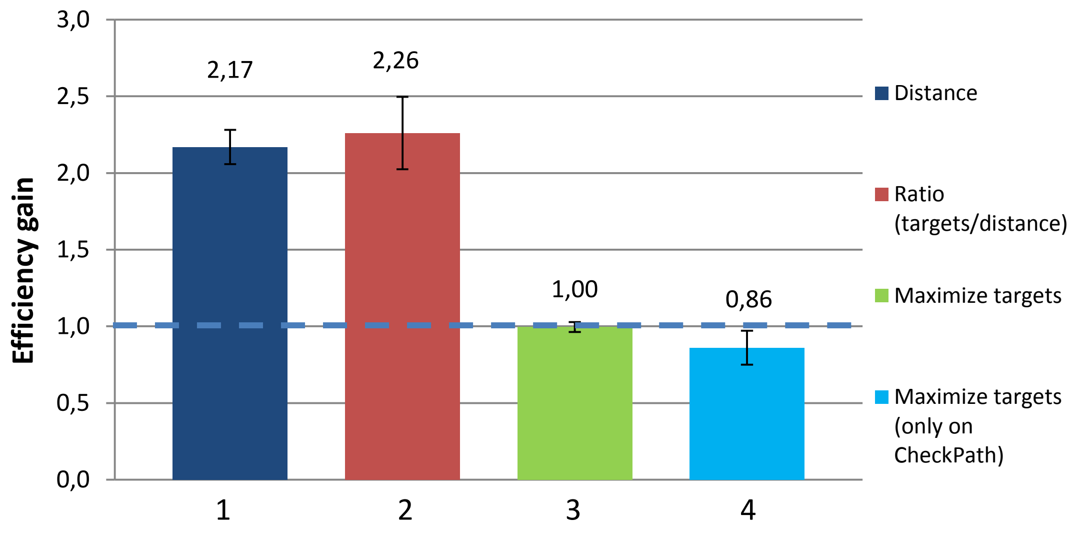


Fig. 53 – Efficiency gain depending on different metrics according to the implementation described in Table 28.

Considering the number of targets, the results are completely different. The simulation of batch 3 retrieves paths in which more than half (56%) of the targets on the field are reached (see Fig. 54). It shows that it is possible to significantly improve the number of reached targets, but that comes with the cost of losing efficiency.

The case 4 is much worse than case 3 both in terms of reached targets as well as in terms of efficiency gain, so it is not an interesting combination for the current framework.

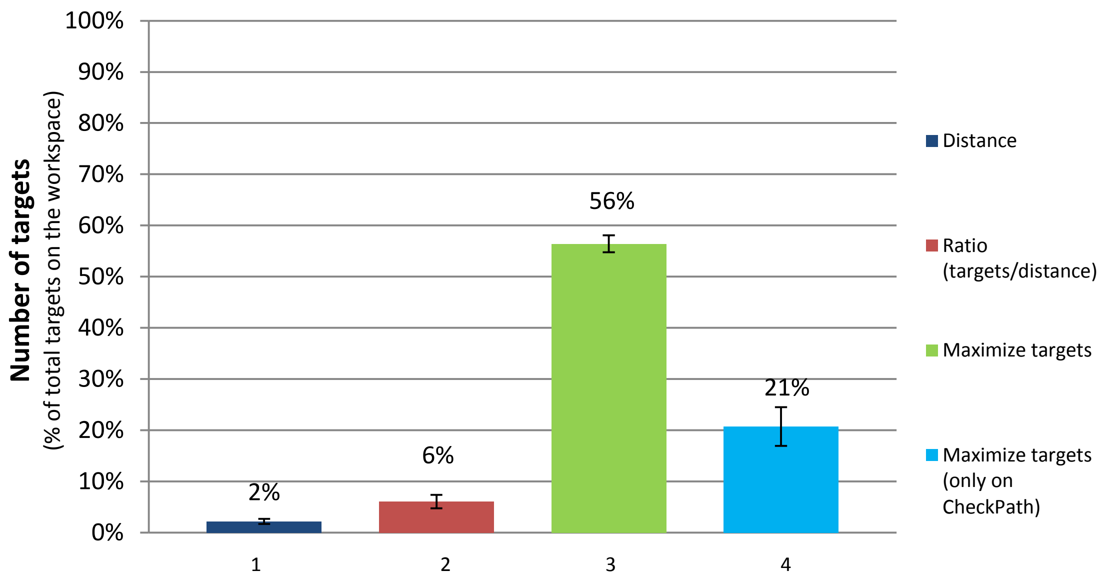


Fig. 54 – Number of targets reached (in percentage) depending on different metrics according to the implementation described in Table 28

The chart of Fig. 55 shows both the efficiency gain and the number of targets for each of the 4 situations, aggregating the results of Fig. 53 and Fig. 54 in one single chart. The situation of batch 2 where the efficiency gain is favoured in all subroutines (*FindBestParent*, *Rewire* and *CheckPath*) is the most interesting case because it has the highest efficiency and a much higher number of targets compared to situation of batch 1 where distance is used in *FindBestParent* and *Rewire*. As pointed out before, this improvement comes with an increase in terms of computational cost, but it should be considered as a potential interesting choice when a certain number of targets is imposed as a restriction.

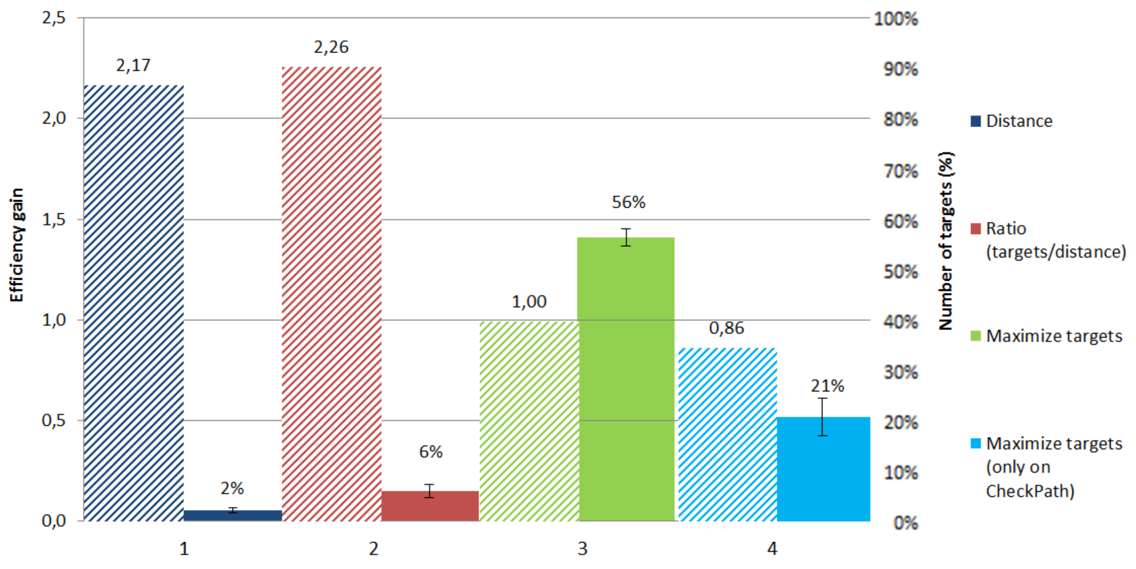


Fig. 55 – Comparison of efficiency gain relatively to the number of targets reached for the different metrics according to the implementation described in Table 28. Stripped bars correspond to the efficiency and full bars to the number of targets.

CHAPTER VI

CONCLUSIONS AND FURTHER WORK

6.1. Considerations about the problem and strategy followed

The work presented throughout this document finds its place in the artificial intelligence field, and in particular in motion planning algorithms which are suitable for autonomous mobile robot applications. As stated before, the ultimate goal is to improve the human quality of life, by having the machines doing the risky, laborious and tedious tasks.

A new random sampling algorithm - TWIN-RRT* - was presented which is able to generate feasible efficient paths towards multiple targets retrieving closed loop paths starting and finishing at the same configuration. The TWIN-RRT* maximises efficiency by maximising the ratio (targets visited)/(distance travelled) and also complies with pre-established conditions like minimum distance to travel (and others) and robot kinematic restrictions while avoiding obstacles. TWIN-RRT* was designed to work with non-holonomic robots and their kinematic limitations. Another important feature is that TWIN-RRT* is able to plan trajectories without requiring any pre-computing step like pre-sequencing of targets. The TWIN-RRT* also provides economic and environmental advantages since it produces efficient paths which by consequence leads to less energy expenditure.

The TWIN-RRT* is the only planner which is able to solve problems with a large number of targets where none of them is mandatory and having the same initial and final configuration.

An additional problem was presented in the context of the case study: the exact targets location is not known a priori, and this poses a significant additional burden to the planner. In order to minimise its effects a hybrid localisation system was developed which is able to predict the actual targets location on the workspace based on statistical analysis of real data and a virtual map generator function.

Efficiency was the metric adopted to be maximised by the function cost of TWIN-RRT*. It was defined as the number of reached targets per distance travelled (targets/m). As reference a complete sweeping strategy was used, and thus a more useful and tangible metric was defined -

the efficiency gain – which corresponds to the ratio between the previously defined efficiency and the efficiency of a complete sweep strategy.

Non-holonomic robots, like the aforementioned Golfminho, have kinematic constraints which limit the movement direction. In the particular case of Golfminho, which has an electric differential rear axle, both right and left wheel speeds account for the applied movement. It is well known that a small change in the velocity of one of the wheels causes a significant change in the direction, which by consequence causes deviations in path execution and increases uncertainty.

To limit the number of variables that influences the robot movement in real applications and also to simplify the implementation of the algorithm, the robot movement was restricted to 3 primitives, forward, turn right and turn left, known as Dubins curves. When turning, it uses always the minimum curvature radius, which corresponds to the maximum turning angle.

6.2. Design choices driven by optimality, completeness and low computational complexity

The TWIN-RRT* produces a path which is a sequence of configurations. Theoretically, an ideal controller would follow the exact path, but in reality there is uncertainty which refers to every factor that interferes with the planned movement and causes deviations. Although there are ways to address uncertainty, the strategy followed when developing the planning algorithm took in consideration the need for re-planning. That was the reason that drove the research towards a faster planner rather than a complete planner. Although optimality is generally considered to have a significant impact on the computational time, the TWIN-RRT* combines the asymptotic-optimality inherited from RRT* with a cost function in order to produce efficient paths in a reasonable low amount of time.

TWIN-RRT* retrieves not only feasible but also asymptotically optimal plans. At a given instant it retrieves the most efficient path and, if allowed to run more time, it will retrieve even better paths. It continuously searches for the most efficient path.

TWIN-RRT* was designed to be a fast planner to allow practical implementations which might need re-planning. It can run in serial or in parallel with the execution phase, but in cases where uncertainty plays an important role, the serial approach is the logical choice.

6.3. Implications of non-compulsory targets

Having non-compulsory targets is not a simplification of the problem. It could account for more admissible feasible solutions in the end which is a positive effect, but for the planning process it adds complexity because much more alternatives have to be considered. To overcome such complexity, combinatorial methods were definitely put aside. The strategy followed was to not previously select or sequence any targets at all, which is the opposite of common strategies available in literature that deals with multiple targets.

A motion planning related strategy was followed because an optimal solution where there are thousands of targets is practically unfeasible using graph based algorithms with the resources available nowadays.

The development of the TWIN-RRT* was inspired in the RRT* which was the most promising algorithm known, an asymptotically optimal planning algorithm. It is a sampling-based planner which can be used with many DOFs and is probabilistically complete when the sampling is random. By adopting a sampling based strategy it avoids explicit representation of the environment and thus reduces the complexity.

TWIN-RRT* presents a solution where combinatorial algorithms fail because they cannot deal with a high number of targets. Multi-target methods described in the literature frequently use a two steps approach, in which the first one is to sequence the targets and the second one to do a point to point planning.

The TWIN-RRT* is a one-step algorithm, solving the problem by growing two trees from the same initial configuration. When a new random sample is added to one of the trees, it tries to close the loop by connecting to the other tree. Efficiency of the closed loop is evaluated. Whenever a new better closed loop is found the efficiency is improved. Dubins curves provide the solution for kinematic constraints. A feasible path is retrieved usually at early iterations of the algorithm. Every time a better path, with higher efficiency gain, is found it is saved as the best path.

6.4. Implications on KD-Trees Representation

An additional contribution related to data representation in KD-Trees was presented. KD-trees are the most efficient data structures to find near neighbours, which is a key property of RRT related algorithms.

Non-holonomic robots have for the same coordinate (x,y), several possible configurations because the orientation plays a major role. The new solution implemented provides a way of using KD-trees for non-holonomic robots having configurations with 3 DOF (x,y,θ) in a 2D workspace environment.

6.5. General conclusions from experimental data

From the experiments carried out to characterise the TWIN-RRT* algorithm, there are some general conclusions that can be taken.

The TWIN-RRT* converges towards an optimal solution. It shows most of the improvements in early stages. It also converges very quickly to the optimal solution and continuously improves the result over time, which is in accordance to the assumption that it is asymptotically optimal as its predecessor RRT*.

The new algorithm maintains a good performance even in the presence of a very large number of targets. The performance, or efficiency gain, is in every case always more than 1.8 times better than a sweep strategy. For a typical case of 10 000 targets the efficiency gain is about 2.0.

By adding restrictions, such as minimum number of targets to reach, the TWIN-RRT* shows a steady and reliable performance up to about 2% of the total number of targets available. The breaking point happens when 4% of the targets are required. Above 4% the algorithm fails to provide feasible paths.

When different target distributions were tested, the TWIN-RRT* maintained the performance and the results showed that it does not affect the efficiency and the type of results expected from the algorithm. It still prioritises efficiency without being biased by targets distribution patterns.

The simulations using different obstacles distributions pointed that TWIN-RRT* is 100% capable of finding a feasible solution whenever one exists. It overcomes narrow passages, difficult labyrinths, bug traps, double bug traps, among other challenging situations with obstacles. The experiment results corroborate the idea that TWIN-RRT* is in fact probabilistically complete as the RRT*.

It was shown that the minimum curvature radius, which is a kinematic limitation of non-holonomic robots in 2D environments (e.g.: car alike robots) does not influence the performance of the algorithm.

Varying the starting point also did not reveal any problems for the planning algorithm.

The quality of TWIN-RRT* results is also not affected by using different CPUs. The time required may vary depending on the speed of the CPU but the quality of the results remains constant.

If needed, there are ways to tune the TWIN-RRT* in order to reach more targets but that comes with the cost of losing efficiency - the limit is at about 56% of total targets, which corresponds to have an efficiency gain of 1.0.

The “ball radius” value is a technical parameter which should be tuned for each application. The optimal value for the described framework is 6.8 m which corresponds to GAMMA equal to 6.

In some cases it might be profitable to bias the sampling method to improve the results, but in the present case-study biasing the sampling method retrieved slightly worse results, which in turn reinforces the idea followed by many researchers that are in favour of random sampling for planning methods.

The *Extended* version of TWIN-RRT* depends on the choice of the step value whereas the connect version does not. Although the *Extended* version may retrieve slightly better results when properly tuned, the difference is not really significant, so for cases where the step value cannot be carefully determined it would be advisable to simply use the *Connect* version.

The subroutines *FindBestParent* and *Rewire* which are part of the TWIN-RRT* can also be modified to use a cost function such as the ratio (targets/distance) instead of the common distance metric. This modification showed slightly better results, but still not statistically relevant for the current framework, in terms of efficiency gain. The advantage lies on the average number of reached targets which is about 6% instead of 2%. The only drawback is a slightly overhead in the computational time.

6.6. Summary of contributions

This thesis provided the following main contributions:

- a new information system - Hybrid localisation system (see section 2.4) – capable of predicting the targets localisation in real-time by using both statistical real data and virtual (generated) data;
- a modified version of a KD-Tree structure which allows the representation of the direction inside the KD-Tree structure besides the common Cartesian coordinates. Points having the same Cartesian coordinates and different directions are saved together and can be retrieved when searching the KD-Tree, as a group of admissible configurations for that point;
- multiple targets path generation for one agent and a large number of targets;
- feasible closed loop path generation for one agent where the initial and final configurations are exactly the same;
- using efficiency as a cost-function on sampling-based motion planning methods, instead of distance and, by that, providing a path planning algorithm which is compliant with non-mandatory targets;
- design of a new path planning algorithm whose performance is not limited or affected by the use of a large number of targets.

6.7. Further work

The TWIN-RRT* presented in this document was tested for a non-holonomic agent moving on a 2D environment. It is believed that with simple modifications it could be expanded for use with higher DOFs agents.

The algorithm is prepared for a high number of targets which are non-mandatory. An interesting field of research is having some mandatory targets together with non-mandatory targets. Further research is still necessary.

It would be important to adapt other algorithms or develop new ones in order to make them suitable for multiple targets. Benchmarking the TWIN-RRT* with other algorithms will then be possible, and by consequence, it would also be possible to explore new ways for improving them.

The TWIN-RRT* fails to retrieve feasible paths when a restriction on a minimum number of targets to reach is imposed (see section 5.3 for details). It is possible to reach up to 56% (see section 5.14) of the total number of targets in the workspace but not more. Testing ways to overcome this limitation would be an interesting topic of research.

Another important aspect which is very relevant for any planning algorithm is the sampling method. TWIN-RRT* accepts heuristics and different sampling methods which could improve the performance or even overcome some limitations. This is an open area for research also.

Other interesting ideas to be explored are for example to add more trees to the same initial configuration and instead of growing only two trees, have more trees whose branches could serve as initial segment, final segment or intermediate segment. Also, growing additional trees from other random or pre-established locations could provide alternative waypoints which might be interesting depending on the application.

The Dubins Curves were used as the steering method because it has proved to be very efficient for non-holonomic robots in 2D environments, but other steering methods may also apply. It would be interesting to study the behaviour of other methods, and in particular ones which allow smoother changes of direction.

TWIN-RRT* has significant application potential in several areas such as surveillance, agriculture, military, space and many more. Its main features are high DOFs compliance, probabilistic completeness, asymptotical optimality, low computation cost and kinodynamic constraints compatibility. It inherits the probabilistically completeness and probabilistically optimality features from RRT*, it adds a function cost which can be adapted to different situations and produces closed paths, and it is compatible with high number of non-mandatory targets.

Planning towards multiple targets has recently gained some interest among the scientific researchers. Some real implementations using a reduced number of targets have been documented. It is expected that in the near future much more work will be carried out with a high number of targets on increasingly complex problems.

6.8. Patent applications

There were two patent applications submitted regarding the research presented on this document:

“Fully autonomous or remotely operated golf ball picking system” – Portuguese Patent n.o 103807, and United States application US-2010-0250024-A1;

“Sistema de monitorização e previsão de localização de objectos e respetivo processo” – Portuguese application n.o PT 106417.

REFERENCES

1. Nilsson, N.J., *The quest for artificial intelligence*. 2010: Cambridge University Press Cambridge.
2. Karaman, S. and E. Frazzoli, *Sampling-based algorithms for optimal motion planning.* The International Journal of Robotics Research, 2011. **30**(7): p. 846-894.
3. Karaman, S., et al. *Anytime motion planning using the RRT**. in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. 2011. IEEE.
4. Perez, A., et al. *Asymptotically-optimal path planning for manipulation using incremental sampling-based algorithms*. in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. 2011. IEEE.
5. Perez, A., et al. *Asymptotically-optimal manipulation planning using incremental sampling-based algorithms*. in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2011.
6. LaValle, S.M. and J.J. Kuffner, Jr. *Randomized kinodynamic planning*. in *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*. 1999.
7. Lavalle, S.M. and J.J. Kuffner Jr. *Rapidly-Exploring Random Trees: Progress and Prospects*. in *Algorithmic and Computational Robotics: New Directions*. 2000.
8. Pacheco, L.F.C., A.J.B.d. Oliveira, and A.F. Ribeiro, *Mobile robot for autonomous golf balls picking.* 2008.
9. Automation, S. *Golf Ball Picker Robot*. 2011 [cited 2011 2011-05-24]; Available from: http://www.softeeautomation.com/html/ballpicker.htm.
10. Bandlow, T., et al., *Fast image segmentation, object recognition and localization in a robocup scenario*, in *RoboCup-99: Robot Soccer World Cup III*. 2000, Springer. p. 174-185.
11. Lowe, D.G. *Object recognition from local scale-invariant features*. in *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*. 1999. Ieee.
12. Jian, Y., N.G. Lu, and D. Mingli. *Computer vision calibration new easy method*. in *Information Technology and Computer Science, 2009. ITCS 2009. International Conference on*. 2009. IEEE.
13. Li, Y. and S. Chen, *Automatic recalibration of an active structured light vision system.* Robotics and Automation, IEEE Transactions on, 2003. **19**(2): p. 259-268.
14. Gu, J., et al., *Removing image artifacts due to dirty camera lenses and thin occluders.* ACM Trans. Graph., 2009. **28**(5): p. 1-10.
15. Schwartz, J.T. and M. Sharir, *On the "piano movers" problem. II. General techniques for computing topological properties of real algebraic manifolds.* Advances in Applied Mathematics, 1983. **4**(3): p. 298-351.
16. Choset, H.M., *Principles of Robot Motion: Theory, Algorithms, and Implementations*. 2005: Prentice Hall of India.
17. LaValle, S.M., *Planning algorithms*. 2006: Cambridge university press.
18. Kavraki, L.E., et al., *Probabilistic roadmaps for path planning in high-dimensional configuration spaces.* Robotics and Automation, IEEE Transactions on, 1996. **12**(4): p. 566-580.
19. Kavraki, L.E. and J.-C. Latombe, *Probabilistic roadmaps for robot path planning.* 1998.
20. Suzuki, Y., S. Kagami, and J.J. Kuffner. *Path planning with steering sets for car-like robots and finding an effective set*. in *Robotics and Biomimetics, 2006. ROBIO'06. IEEE International Conference on*. 2006. IEEE.

21. Jeon, J.h., et al. *Optimal motion planning with the half-car dynamical model for autonomous high-speed driving*. in *American Control Conference (ACC), 2013*. 2013.
22. Suzuki, Y., S. Thompson, and S. Kagami. *Smooth path planning with pedestrian avoidance for wheeled robots: Implementation and evaluation*. in *Autonomous Robots and Agents, 2009. ICARA 2009. 4th International Conference on*. 2009. IEEE.
23. Katz, D., J. Kenney, and O. Brock, *How can robots succeed in unstructured environments?* 2008.
24. Katz, D., Y. Pyuro, and O. Brock. *Learning to manipulate articulated objects in unstructured environments using a grounded relational representation*. in *In Robotics: Science and Systems*. 2008. Citeseer.
25. Choset, H., *Principles of Robot Motion: Theory, Algorithms, and Implementation*. 2005: MIT Press © 2005
26. Ramesh, T., *Traveling purchaser problem.* OPSEARCH, 1981. **18**(2): p. 78-91.
27. Pearn, W.L. and R.C. Chien, *Improved solutions for the traveling purchaser problem.* Computers &amp; Operations Research, 1998. **25**(11): p. 879-885.
28. Golden, B.L., L. Levy, and R. Dahl, *Two generalizations of the traveling salesman problem. .* OMEGA, 1981. **9**(4): p. 439-445.
29. Ong, H.L., *Approximate algorithms for the traveling purchaser problem.* Operations Research Letters, 1982. **1**(5): p. 201-205.
30. Ravi, R. and F. Salman, *Approximation Algorithms for the Traveling Purchaser Problem and Its Variants in Network Design*

Algorithms - ESA' 99, J. Nešetril, Editor. 1999, Springer Berlin / Heidelberg. p. 696-696.
31. Singh, K.N. and D.L. van Oudheusden, *A branch and bound algorithm for the traveling purchaser problem.* European Journal of Operational Research, 1997. **97**(3): p. 571-579.
32. Dudek, G. and M. Jenkin, *Computational Principles of Mobile Robotics*. 2nd ed. 2010: Cambridge University Press.
33. Latombe, J.C., *Robot Motion Planning*. 1991, Boston, MA: Kluwer Academic Publishers.
34. Dubins, L.E., *On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents.* American Journal of mathematics, 1957. **79**(3): p. 497-516.
35. Zelek, J.S. *Dynamic path planning*. in *Systems, Man and Cybernetics, 1995. Intelligent Systems for the 21st Century., IEEE International Conference on*. 1995. IEEE.
36. Canny, J. and J. Reif. *New lower bound techniques for robot motion planning problems*. in *Foundations of Computer Science, 1987., 28th Annual Symposium on*. 1987.
37. Dijkstra, E.W., *A note on two problems in connexion with graphs.* Numerische Mathematik, 1959. **1**(1): p. 269-271.
38. Rantanen, M.T. and M. Juhola, *Using probabilistic roadmaps in changing environments.* Computer Animation and Virtual Worlds, 2013: p. n/a-n/a.
39. Lacaze, A., et al. *Path planning for autonomous vehicles driving over rough terrain*. in *Intelligent Control (ISIC), 1998. Held jointly with IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA), Intelligent Systems and Semiotics (ISAS), Proceedings*. 1998. IEEE.
40. Balluchi, A., et al. *Stability and robustness of optimal synthesis for route tracking by Dubins' vehicles*. in *Decision and Control, 2000. Proceedings of the 39th IEEE Conference on*. 2000.

41. Hong, W., Y. Tian, and Y. Xu. *The research of dynamic path planning for centralized vehicle navigation*. in *Automation and Logistics, 2007 IEEE International Conference on*. 2007. IEEE.
42. Berman, S., E. Schechtman, and Y. Edan, *Evaluation of automatic guided vehicle systems.* Robotics and Computer-Integrated Manufacturing, 2009. **25**(3): p. 522-528.
43. Wu, C.-J. and W.-H. Tsai, *Location estimation for indoor autonomous vehicle navigation by omni-directional vision using circular landmarks on ceilings.* Robotics and Autonomous Systems, 2009. **57**(5): p. 546-555.
44. Shkel, A.M. and V. Lumelsky, *Classification of the Dubins set.* Robotics and Autonomous Systems, 2001. **34**(4): p. 179-202.
45. Beard, R.W. and T.W. McLain, *Implementing Dubins Airplane Paths on Fixed-wing UAVs.* 2013.
46. Zhijun, T. and U. Ozguner, *Motion planning for multitarget surveillance with mobile sensor agents.* Robotics, IEEE Transactions on, 2005. **21**(5): p. 898-908.
47. Luo, Q., Z. Liu, and S.-D. Qiao. *A hybrid route planning algorithm of single aerial vehicle attacking multiple targets*. in *Machine Learning and Cybernetics (ICMLC), 2010 International Conference on*. 2010. IEEE.
48. Hongyun, L., J. Xiao, and J. Hehua, *The Multi-Goal Path Planning of Lunar Rover Based on Grid Model.* International Journal of Applied Mathematics and Statistics, 2013. **42**(12): p. 430-440.
49. Hongyun, L., J. Xiao, and J. Hehua. *Multi-goal path planning algorithm for mobile robots in grid space*. in *Control and Decision Conference (CCDC), 2013 25th Chinese*. 2013. IEEE.
50. Lattanzi, L. and C. Cristalli. *An efficient motion planning algorithm for robot multi-goal tasks*. in *Industrial Electronics (ISIE), 2013 IEEE International Symposium on*. 2013.
51. Batista-Galván, M., J. Riera-Ledesma, and J.J. Salazar-González, *The traveling purchaser problem, with multiple stacks and deliveries: A branch-and-cut approach.* Computers & Operations Research, 2013. **40**(8): p. 2103-2115.
52. Bentley, J.J., *Fast algorithms for geometric traveling salesman problems.* ORSA Journal on computing, 1992. **4**(4): p. 387-411.
53. Dorigo, M. and L.M. Gambardella, *Ant colony system: a cooperative learning approach to the traveling salesman problem.* Evolutionary Computation, IEEE Transactions on, 1997. **1**(1): p. 53-66.
54. Lumelsky, V. and A. Stepanov, *Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape.* Algorithmica, 1987. **2**(1-4): p. 403-430.
55. Tom, et al., *An algorithm for planning collision-free paths among polyhedral obstacles.* Commun. ACM, 1979. **22**(10): p. 560-570.
56. Kamon, I., E. Rivlin, and E. Rimon. *A new range-sensor based globally convergent navigation algorithm for mobile robots*. in *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*. 1996.
57. Kamon, I., E. Rimon, and E. Rivlin, *TangentBug: A Range-Sensor-Based Navigation Algorithm.* The International Journal of Robotics Research, 1998. **17**(9): p. 934-953.
58. Barraquand, J., et al., *A Random Sampling Scheme for Path Planning*, in *Robotics Research*, G. Giralt and G. Hirzinger, Editors. 2000, Springer London. p. 249-264.
59. Canny, J.F., *The complexity of robot motion planning*. 1988: MIT Press. 195.
60. Canny, J. and M. Lin, *An opportunistic global path planner.* Algorithmica, 1993. **10**(2-4): p. 102-120.
61. Mazer, E., et al., *The Ariadne's clew algorithm.* J. Artif. Int. Res., 1998. **9**(1): p. 295-316.

62. Kondo, K., *Motion planning with six degrees of freedom by multistrategic bidirectional heuristic free-space enumeration.* Robotics and Automation, IEEE Transactions on, 1991. **7**(3): p. 267-277.
63. Choset, H., *Sensor based motion planning: the hierarchical generalized Voronoi graph*. 1996, California Institute of Technology.
64. Han, L. and N.M. Amato, *A kinematics-based probabilistic roadmap method for closed chain systems.* 2000.
65. Boor, V., M.H. Overmars, and A.F. van der Stappen. *The gaussian sampling strategy for probabilistic roadmap planners*. in *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*. 1999. IEEE.
66. Kavraki, L.E., *Random Networks in Configuration Space for Fast Path Planning*. 1995, Stanford University.
67. Kavraki, L.E., et al., *Randomized query processing in robot path planning*, in *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*. 1995, ACM: Las Vegas, Nevada, USA. p. 353-362.
68. Hsu, D., *Randomized single-query motion planning in expansive spaces*. 2000, Stanford University.
69. Hsu, D., J.C. Latombe, and R. Motwani. *Path planning in expansive configuration spaces*. in *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*. 1997.
70. Kuffner Jr, J.J. and S.M. LaValle. *RRT-connect: An efficient approach to single-query path planning*. in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*. 2000. IEEE.
71. Akinc, M., et al., *Probabilistic Roadmaps of Trees for Parallel Computation of Multiple Query Roadmaps.* Robotics Research, 2005: p. 80-89.
72. Bekris, K.E., et al. *Multiple query probabilistic roadmap planning using single query planning primitives*. in *Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*. 2003.
73. Hsu, D., et al., *Randomized kinodynamic motion planning with moving obstacles.* The International Journal of Robotics Research, 2002. **21**(3): p. 233-255.
74. Cortes, J., T. Siméon, and J.-P. Laumond. *A random loop generator for planning the motions of closed kinematic chains using PRM methods*. in *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*. 2002. IEEE.
75. Yakey, J.H., S.M. LaValle, and L.E. Kavraki, *Randomized path planning for linkages with closed kinematic chains.* Robotics and Automation, IEEE Transactions on, 2001. **17**(6): p. 951-958.
76. Jijie, X., et al. *Motion planning for steerable needles in 3D environments with obstacles using rapidly-exploring Random Trees and backchaining*. in *Automation Science and Engineering, 2008. CASE 2008. IEEE International Conference on*. 2008.
77. Bodhale, D., N. Afzulpurkar, and N.T. Thanh. *Path planning for a mobile robot in a dynamic environment*. in *Robotics and Biomimetics, 2008. ROBIO 2008. IEEE International Conference on*. 2009. IEEE.
78. Ge, S.S., X.-C. Lai, and A. Al Mamun, *Sensor-based path planning for nonholonomic mobile robots subject to dynamic constraints.* Robotics and Autonomous Systems, 2007. **55**(7): p. 513-526.
79. Rose, K., *Real-time sampling-based motion planning with dynamic obstacles*. 2011, University of New Hampshire.
80. Tahirovic, A. and G. Magnani. *A roughness-based RRT for mobile robot navigation planning*. in *Proceedings of the 18th IFAC World Congress*. 2011.

81. Dadkhah, N. and B. Mettler, *Survey of Motion Planning Literature in the Presence of Uncertainty: Considerations for UAV Guidance.* Journal of Intelligent & Robotic Systems, 2012. **65**(1-4): p. 233-246.
82. Davoodi, M., et al., *An optimal algorithm for two robots path planning problem on the grid.* Robotics and Autonomous Systems, 2013.
83. Wang, L., et al. *Obstacle-avoidance path planning for soccer robots using particle swarm optimization*. in *Robotics and Biomimetics, 2006. ROBIO'06. IEEE International Conference on*. 2006. IEEE.
84. Karaman, S. and E. Frazzoli, *Incremental Sampling-based Algorithms for Optimal Motion Planning.* ArXiv e-prints, 2010.
85. Barraquand, J. and J.-C. Latombe, *Robot Motion Planning: A Distributed Representation Approach*. 1989.
86. Lamiraux, F. and J.P. Laumond. *On the expected complexity of random path planning*. in *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*. 1996.
87. Kavraki, E.E., M.N. Kolountzakis, and J.C. Latombe, *Analysis of probabilistic roadmaps for path planning.* Robotics and Automation, IEEE Transactions on, 1998. **14**(1): p. 166-171.
88. Ladd, A.M. and E.E. Kavraki, *Measure theoretic analysis of probabilistic path planning.* Robotics and Automation, IEEE Transactions on, 2004. **20**(2): p. 229-242.
89. Urmson, C. and R. Simmons. *Approaches for heuristically biasing RRT growth*. in *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*. 2003. IEEE.
90. Akgun, B. and M. Stilman. *Sampling heuristics for optimal motion planning in high dimensions*. in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. 2011. IEEE.
91. Corput, J., *Verteilungsfunktionen i, Nederl.* Akad. Wetensch. Proc. Ser. B, 1935. **38**(38): p. 813-821.
92. Svestka, P. and M.H. Overmars. *Coordinated motion planning for multiple car-like robots using probabilistic roadmaps*. in *Robotics and Automation, 1995. Proceedings., 1995 IEEE International Conference on*. 1995.
93. Sekhavat, S., et al., *Multilevel path planning for nonholonomic robots using semiholonomic subsystems.* The international journal of robotics research, 1998. **17**(8): p. 840-857.
94. Hsu, D., et al. *On finding narrow passages with probabilistic roadmap planners*. in *Proc. Int. Workshop on Algorithmic Foundations of Robotics (WAFR)*. 1998.
95. Bohlin, R. and E.E. Kavraki. *Path planning using lazy PRM*. in *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*. 2000.
96. Sánchez, G. and J.-C. Latombe, *A Single-Query Bi-Directional Probabilistic Roadmap Planner with Lazy Collision Checking*, in *Robotics Research*, R. Jarvis and A. Zelinsky, Editors. 2003, Springer Berlin Heidelberg. p. 403-417.
97. Choudhury, S., S. Scherer, and S. Singh, *RRT*-AR: Sampling-Based Alternate Routes Planning with Applications to Autonomous Emergency Landing of a Helicopter*, in *IEEE International Conference on Robotics and Automation (ICRA)*. 2013.
98. Jaillet, L., et al. *Adaptive tuning of the sampling domain for dynamic-domain RRTs*. in *Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on*. 2005.
99. Frazzoli, E., *Real-Time Motion Planning for Agile Autonomous Vehicles.* JOURNAL OF GUIDANCE, CONTROL, AND DYNAMICS, 2002. **25**(1): p. 14.

100. Jongwoo, K. and J.P. Ostrowski. *Motion planning a aerial robot using rapidly-exploring random trees with dynamic constraints*. in *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*. 2003.
101. Lamiraux, F., E. Ferre, and E. Vallee. *Kinodynamic motion planning: connecting exploration trees using trajectory optimization Methods*. in *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*. 2004.
102. Jaillet, L. and J.M. Porta, *Efficient asymptotically-optimal path planning on manifolds.* Robotics and Autonomous Systems, 2013. **61**(8): p. 797-807.
103. Pereira, N., et al., *Autonomous golf ball picking robot design and development.* Industrial Robot: An International Journal, 2012. **39**(6): p. 541-550.
104. J.Canny and J.H.Reif, *New lower bound techniques for robot motion planning problems*, in *IEEE Symposium on Foundations of Computer Science (FoCS)*. 1987: Los Angeles,CA. p. 49 -60.
105. Karaman, S. and E. Frazzoli, *Incremental Sampling-based Algorithms for Optimal Motion Planning.* CoRR, 2010. **abs/1005.0416**.
106. Kavraki, L.E., et al., *Probabilistic roadmaps for path planning in high-dimensional configuration spaces.* Robotics and Automation, IEEE Transactions on, 1996. **12**(4): p. 566-580.
107. Karaman, S. and E. Frazzoli, *Sampling-based algorithms for optimal motion planning.* Int. J. Rob. Res., 2011. **30**(7): p. 846-894.
108. Kuffner, J. and S. Lavalle. *RRT-connect: An efficient approach to single-query path planning*. in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA*. 2000.
109. Perez, A., et al., *Asymptotically-optimal Manipulation Planning using Incremental Sampling-based Algorithms*, in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2011.
110. Soueres, P. and J.P. Laumond, *Shortest paths synthesis for a car-like robot.* Automatic Control, IEEE Transactions on, 1996. **41**(5): p. 672-688.
111. Boissonnat, J.-D., A. Cérézo, and J. Leblond, *Shortest paths of bounded curvature in the plane*. 1993: Springer.
112. Bueckert, J., et al. *Neural dynamics based multiple target path planning for a mobile robot*. in *Robotics and Biomimetics, 2007. ROBIO 2007. IEEE International Conference on*. 2007. IEEE.
113. Ge, S.S., et al., *Simultaneous Path Planning and Topological Mapping (SP2 ATM) for environment exploration and goal oriented navigation.* Robotics and Autonomous Systems, 2011. **59**(3): p. 228-242.
114. Bobadilla, L., et al. *Minimalist multiple target tracking using directional sensor beams*. in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. 2011. IEEE.
115. authors, S. *Minkowski sum*. Available from: http://www.cgal.org/Manual/latest/doc_html/cgal_manual/Minkowski_sum_3/Chapter_main.html.
116. authors, S. *KD-Trees*. Available from: http://en.wikipedia.org/wiki/K-d_tree).