UNIVERSITY OF MACEDONIA
DEPARTMENT OF APPLIED INFORMATICS
MSc IN ARTIFICIAL INTELLIGENCE AND DATA ANALYTICS

Computing trajectories in 3D space to avoid obstacles using systematic and stochastic search algorithms

A dissertation
by

Vasileios Markou

Thessaloniki, September 2021

Computing trajectories in 3D space to avoid obstacles using systematic and stochastic search algorithms

Vasileios Markou

Bachelor in Applied Informatics, University of Macedonia, 2019

Dissertation

Submitted in partial fulfilment of the requirements for
MSc IN ARTIFICIAL INTELLIGENCE AND DATA ANALYTICS

Supervisor
Ioannis Refanidis, Professor

Approved by the three-member Examination Committee on 21/09/2021

Ioannis Refanidis, Professor	Dimitrios Hristu-Varsakelis, Professor	Ilias Sakellariou, Assistant Professor
.....	.....	.....

Vasileios Markou

....

Abstract

Recent years have been characterized by great technological advances. When it became an option to exploit in industrial work, the tasks were automated with the use of robots. In this dissertation, we present a custom-made virtual robotic arm capable of constructing Jenga towers. In contrast with previous work, we do not use force sensors for placement detection. Furthermore, to demonstrate the robot controller 's extensibility in design, we run two different simulations, ending with two different structures. Simplicity and efficiency have been considered to achieve the desired results in the experiments above, raising the key points where computational costs cannot be reduced, while providing simple alternatives for the rest of some common issues in robotics. We also compare and present results concerning various topics regarding commonly used algorithms including A*, after implementing them for the purposes of this dissertation.

Keywords: robotics, robot, arm, manipulator, jenga, structure, A*, Dijkstra, RRT

Foreword – Special thanks

I am grateful to several individuals and the University of Macedonia for supporting me throughout this study. First, I wish to express my sincere gratitude to my supervisor, Professor Ioannis Refanidis, for his patience, insightful comments, helpful information, and support that guided me throughout the writing of this thesis. I also wish to appreciate the University of Macedonia for accepting me and providing me with the necessary theoretical and practical background to overcome this project' s difficulties.

Table of contents

List of Figures

List of tables

Notation

1. **n**; the next available node (in this research this will be used to mark the next available motion in terms of action performed)
2. **f(n)**; heuristic function used to determine the next most appropriate node to visit
3. **g(n)**; distance, or cost, from the start node to the current node
4. **h(n)**; distance, or cost, from the current node to the goal
5. **b(n)**; greatest value that can be used for an action of the robot to be performed without colliding with any obstacles in case they are provided

1 Introduction

1.1. Problem – Importance of the topic

Over the years, technology has played a leading role in shaping the world, assisting mankind in the development of goods and services necessary for survival. At some point in human history, there was an increasing need for providing greater quantities of various products. As times changed, so did the needs and demands of modern-day life, including consumers and scientists. However, this has always been restricted by the limitations that human nature implied. That was a critical point that turned mankind towards seeking automated ways of implementing various tasks. Albeit in the initial stages, robots used to silently take over the heaviest of tasks, replacing workers in the different industries, while providing quicker and more efficient ways to create products. This was one of the first goals to be reached by a robot. As the years passed by, the more robots started to appear in industrial use, the faster they tended to become irreplaceable in their work. This also led to robot involvement in many other aspects of life, making it a matter of time until they could find their way in all kinds of scientific fields, e.g., the process of manufacturing a car is to provide the necessary parts, while a robot could take care of the actions to apply in order for it to be properly assembled.

What is more, it is quite common in robotics to try to imitate certain behavior, including animal movement in certain terrains, as a necessity in analyzing it, before we can proceed with mimicking it and applying it to different problems. In the previous example, this could assist us in providing environment-specific robots that can move with greater ease and study their surroundings. Multi-sensor behavior, namely the combination of input gathered from various different sensors, for example a vision sensor and a force sensor, is deemed of significant importance in common uses, e.g., vehicle-control, where complexity is detrimental. What appears as common behavior turns into a surprisingly difficult problem, when applied to a robot. This usually happens, since the robot is tasked with the following to make even the slightest move;

- data processing
- strategy selection
- world impact estimation
- proper move calculation

All the above combined can sometimes lead to information overload for the robot, for example in cases where a vision sensor is constantly feeding the robot with new images that need to be processed, making it unable to respond in real-time, while trying to tackle

various and complex tasks. Our research aims to provide some suggestions reducing the computational cost of handling complex tasks on the problem of constructing a Jenga tower and similar structures. We have chosen the Webots open-source robotic simulator. In order to demonstrate our ideas, we are using a distance sensor and a camera sensor to reach one of our tiles, in contrast with previous implementations on similar problems that featured a camera sensor and a force sensor. The controller was written in Python and features a state-machine algorithm designed for portability and simplicity. The robotic arm is equipped with a fork-like gripper to be able to move the tiles in our simulated the world. As mentioned previously, we also demonstrate the use of the same robot with minor-to-no changes to the controller, to create an entirely different shaped structure, that of a "Π". The simulations were able to complete the tasks described before, demonstrating the importance of simplicity and efficiency in robot designing.

1.2. **Aim - Objectives**

The main objectives of this study include covering and exploration of several topics regarding robotics in modern days, such as progress reflected by software that has been developed so far or implemented algorithms for motion planning and path planning. We also sought to create a controller able to lead a robotic arm to construct a Jenga tower and similar structures. Before that, the controller is responsible for following the shortest path as it is found by implementing 3 (three) different algorithms, namely Rapidly exploring Random Tree (RRT), Dijkstra and A* (a-star). These algorithms will be presented shortly later, along with details on their implementation. As a side-note, the reader should keep in mind that to be able to provide solutions on various scenarios including the presence of obstacles around the targets (tiles), we had to make a few adjustments and hypotheses, the most important of them being the fact that in A* the heuristic function used takes into account the step that each motion is able to perform to pick the next least expensive action, which appeared to be an extremely important in situations where the robotic arm was faced with multiple obstacles on its path. The rest will be presented in detail in the corresponding chapter.

1.3. **Questions – Research hypotheses**

Robotics is a well-known subject and common cause of scientific interest when it comes to its application to human life. It was therefore expected to be questioned as both a scientific field and an actual means to provide better quality of life for everyone. However, in order to embrace technological advance we first need to understand its effects,

advantages, and applications to our lives. This call cannot be answered unless we delve into researching and implementing on our own, which has been the main reason for choosing this subject to author this dissertation. We need to understand, after experimentation, that simplicity does not always lead to reduced costs, e.g., financial costs and also, when it comes to algorithm implementation, complete algorithms tend to offer greater accuracy while performing smaller number of actions but need to keep all the explored space in memory, proved through experimentation. To tackle the above we can explore several ways of efficient memory storage, including but not limited to R-Trees, as the reader will gradually delve more into their importance.

1.4. Contribution

At first, we had to consider the latest trends in robotics to be able to provide readers with an interesting topic that is tackled by researchers at present. The topic was selected as constructing Jenga-like structures and building a controller that could handle the motion plan behind it. To be able to achieve the above, there is this need of effective algorithms, with effective being used to describe goal, or task completion and speed of providing such a solution, meaning that it must be delivered within logical time limitations and computational resources. Experiments had to be carried out after successful implementation of A*, Dijkstra and RRT algorithms. The reader should note at this point that RRT is specifically used in motion planning, unlike the other two algorithms that are used in graph research. Finally, while achieving the above, similar studies and projects will be presented along with algorithms that have proved to be of importance in generic, or relative ideas and implementations. We need to understand that simplicity does not always lead to reduced costs, e.g., financial costs and also, when it comes to algorithm implementation, complete algorithms tend to offer greater accuracy while performing smaller number of actions, but need to keep all the explored space in memory, proved through experimentation. These algorithms are strongly discretized by the completeness in their logic and thus, when using them we expect that the quality of solutions will vary in different ways, i.e., the execution time to reach their goals.

1.5. Basic terminology

1. World refers to a 3D simulated environment, that for simplicity consists of a horizontal plane, serving as a floor-like base
2. Robot is a custom-made robotic arm with 6 DOF
3. Tile is a rectangle-shaped object that will be used as a Jenga tile

4. Obstacle is a rectangle-shaped object that servers the purpose of interrupting the way from our robot to a tile
5. Controller refers to the controller written in python 3.8 to provide the logic implemented by the robotic arm
6. Webots refers to the software used for running the simulations
7. RRT refers to Rapidly-exploring Random Tree algorithm
8. Dijkstra refers to Dijkstra algorithm
9. A* refers to the a-star algorithm
10. PRM refers to Probabilistic Roadmaps algorithm
11. APF refers to Artificial Potential Fields algorithm
12. DFS refers to Depth-first search algorithm
13. BFS refers to Breadth-first search algorithm

1.6. Structure of the study

We will be going through the various aspects of what it involves to study and develop solutions as plans in the robotics world in an order that aims to present key parts and make the reader familiar with concepts not previously encountered, while maintaining simplicity, yet completeness of information presented. Chapter 1has already discussed what it means to explore the various applications of robotics and their impact on our lives. In Chapter 2 we will be focusing on literature studied in order to be able to provide the reader with useful information in order to give the necessary context of robotics. On the contrary, Chapter 3 describes various approaches in algorithm implementation and proceeds to applying three of them, which will be studied further. Consequently, in Chapter 4 we included some of the results that experimentation has led to after applying the principles above and finally, Chapter 5 will conclude to what we need to keep after having read this dissertation, summarizing the most critical parts we need to remember.

2 Literature review – Theoretical background

Several scientific articles have been studied to provide the theoretical background and perspective of this thesis, along with published books widely acknowledged for their authenticity and hard evidence in informing the reader from the most basic to advanced issues that a researcher is tasked with solving when encountering robotics. They will be presented below to also familiarize readers with the perspectives that inspired us and led to the results presented later. In a simple manner, we will be exploring different approaches for various problems, given that each algorithm can have a unique impact on every problem faced.

2.1. Multi-sensor information gathering in robotics

One of the works studied is used to discuss the various aspects that integrate different sensors, aiming to achieve a common goal [1]. It features the vast necessity that engulfs robotics in industrial development, along with collaboration with humans as main scientific goal is to explore of new ways to provide robots with more sophisticated and rich information. In the main years of human evolution, it has been an integrated part of human nature the ability to be able to receive different stimuli and process them at the same time as electrical currents are parsed through billions of neurons. Therefore, it would be safe to assume that all five senses, namely sight, smell, touch, hearing, and taste have played a more than significant role in human brain evolution. Nowadays, mankind seems to be interested in focusing on increasingly complex tasks to be able to deliver more sophisticated results in many aspects of life. One of these sectors could not be the industrial use and application of technological advances that tend to cover for today' s needs. Of course, during mankind' s tremulous history of progression, we admittedly have to point out the importance of creating machines that could replace labor and automate processes including the creation of various goods.

In modern times, there is an ever-growing lust for knowledge and constant development of robotics since it represents the higher form of machine application to simplify everyday life and represents the greatest example effort of automating different processes. However, after decades of research and studying it has become clear that there is a tendency to create human-like robots to simulate human behavior in the best way. Ideally, various scientists aim to achieve a general-task controller capable of achieving a wide range of unsimilar tasks, providing the middleware that would exploit the different dynamics of a system and its sensor capabilities.

What could be a suitable place to start with when it comes to performing different tasks? It appears that a remarkably interesting environment where such application must be applied is no other than modern-day industry, at its various levels, from assembling the necessary components to delivering the end products to the customers. The most interesting part however, appears to be the description of so far discovered and applied controllers as close-sourced when it comes to sharing among different problems. In fact, naming the most common sensor issues, it usually comes down to force calculation, distance estimation, visual representation, and scalability of manipulators, which seem to have individually been solved as of today. Unfortunately, it is easy to say that similar problems tend to get a feel for exponential growth in difficulty in solving when they get to be combined, since previous implementations are not made publicly available. From the preliminary stages of applying robotics into task automation for industrial applications the focus has been on providing quick and accurate manipulators on regards to trajectory-following problems, for example the discrete position that can be computed by position controllers.

In this paper, it is especially referred to as an example of continuous growth and interest in the research and application over force control, which has led to a static phase in further development when it comes down to this scientific sector. Moreover, additional great load of work has been put to visual servoing, mainly on the theoretical aspect of its utilization, as it describes the motivation to apply visual-based sensing towards controlling the robot [2]. To be more specific, we should underline and understand the necessity of fast image processing algorithms that need to be developed in order to have a better chance of approaching better results and make more complicated robots that include an interface that connects the various established motion commands to be sent to the controller, along with the order and process that they need to undergo with. At this point, it should be profound that due to the dynamic nature of programming background to exploit a system' s sensors, it is more than likely that we will have to put a lot more effort into designing the overall programming architecture of the controller, since it will serve as the mind able to serve the ever-increasing demands and requests in delivering and answering the various aspects of simple to complex commands, responsible for the un-breaking functionality and flexibility of the whole robotic system. As mentioned in the paper, two well-known examples that are able to perform the desired operations with the minimum necessary effort, providing a more understandable, easily scalable, flexible and maintainable system of robot manipulation system, both following and fulfilling the primitive requirements mentioned and analyzed above, could be no other than the projects that are known in the

global scientific society by their abbreviations, OROCOS, an abbreviation that stands for Open RObot COntrol Software and OSACA, an acronym used to refer to Open System Architecture for Controls within Automation systems [3].

To focus on the current project, however, we will be shortly describing the basic design and architecture of both software and hardware implementation. More specifically, the manipulator used is a Staeubli RX60, with its controller having been replaced similar to the ones mentioned above, while its power electronics have been maintained, since they can be savored to serve different purposes no matter the scope within which they have been initially scheduled to operate. It is in fact a network of PCs with QNX serving as its OS (Operational System), extracting the main working processes straight out of the MATLAB/Simulink software, enabling equal ease of addition or change according to task specification during runtime. The first PC is responsible for handling the power needs for the different tasks and mainstreaming it accordingly to the rest of the hardware to fulfill the commands. It also helps with basic low-level joint control. At this point let us clarify that this functionality can also be adopted by the rest of the PCs, although they serve the different user requests. It is also obvious that despite the intention to create a scalable system, it will perform differently depending on the complexity of each purpose in terms of power consumption, e.g., distance calculation in contrast with image processing. What is more, the freedom in role-playing described above is restrained in the current implementation, namely the first PC serves for joint control, the second one handles changes regarding transform (such as position, orientation or velocity applied), the third one is responsible for responding to user commands and last but not the least, the fourth one helps with providing real-time and more sophisticated image processing through footage captured during execution.

The main middleware, implemented in all four PCs is called MiRPA, which stands for Middleware for Robotics and Process Control Applications, serving as the base of all operations. More specifically, as part of the scalability desired to achieve the logic behind it covers for it being a distributed real-time software implementation having only one communication server for each software aspect. These consist of two modules, named as MP (short for Manipulation Primitives) Interface, being responsible for receiving the requests made by the user at a more primitive form and interpreting them in a way that the MP Execution module can map them to specific predefined actions. In this way, if one would not be satisfied by the options available, it would pose as an easy task to extend its functionality by adding more sensors or actuators and having to update their drivers to be mapped with the new actions added in the MP Execution module.

As we discussed previously, this system is quite flexible and have high scalability degree. More importantly, one' s demand as the application programmer only requires the list of available devices. What comes into play next is the modification of MPs, which naturally only consist of three parts, namely the hybrid moves, which represents the task to be achieved along with a list containing the DOF (Degrees Of Freedom) that correspond to each different state during the process of achieving the desired task. Second comes the tool command part, which informs the system about the additional actuators that are addressed through adding more drivers to map to. In this case, it is only responsible for choosing between opening and closing the gripper. The final part that needs to be configured seems to be the exit condition, which is nothing but a set of different Boolean terms combined to form a longer expression that further describes that the desired motion or command has been reached, based on the different sensor values that indicate the various states.

To show the scalability of the system that has been described so far, the researchers decided to organize a little demonstration, with the robot competing against itself to create a Jenga tower. To provide context clearance and for the sake of transparency, it is worth mentioning that abstracting the aim of playing Jenga, it is no other than extracting a loose block without disturbing the tower' s stability. More importantly, the goal was similar as in the actual game, no other than creating the highest possible tower, with the best result consisting of 28 distinct levels. The robotic gripper used served two different purposes utilizing an arbor; the functionality of pushing a block outside the tower and of course, the ability to be able to place it back on top of the tower. It is also worth mentioning that two charge-coupled CCD cameras were placed away of the robot in positions that enabled them to detect observe the whole tower in a perimetric way and were programmed to process and detect tiles that had been distinguished in the following way; each block was painted black and featured two white lines and two dots, Also, the rest of the robot' s hardware configuration consists of a 6D force/torque sensor, a 6D acceleration sensor and a laser distance sensor. Let us clear at this point that generally, judging by previous similar problem solution implementations, we can speak of three well-distinguished approaches; the first one is by using impedance control, which is mainly used to identify the impedance applied from the tower' s tiles back to the gripper through calculation of both forces received from the tower and the estimated distance the gripper has away from it. Second, there has been an effort of combining the above into a single calculation matrix, instead of dealing with them individually, an approach considered as parallel control, aimed to simplify, and provide faster estimations, while sacrificing accuracy. The third and final

approach, which happens to be adopted in the current project as well, is commonly known as force/position control. As noted, this technique also focuses primarily on force sensor readings along with position estimation, while demonstrating a critical difference; these two metrics are separately considered as two dimensions that interfere with one another orthogonally. It should be underlined that this is a more generic approach and does not need to undergo any modifications despite changes originating from different coordinate systems used. It also serves well as separating the forces caused by inner collisions that occur due to passive and static contact made by objects, as compared to external environmental forces that occur after taking actions. What is more, it allows for more compliant motion control, which so far has only been approached in a more theoretical perspective as it involved a more realistic and force-guided analysis of robot and environment interaction. Jenga was picked as a case scenario due to its design to include programming organized as tasks, force manipulation and understanding including but not limited to processing continuous calculations in real time, combination of data produced by various kinds of sensors, trajectory prediction to be able to estimate among others the tower stability and climaxed control software architecture, able to handle different states of the game. The process initiates by choosing a random tile that has been marked for extraction after having been recognized by the cameras, by trying to push the desired block a few centimeters out of the tower, provided that the cameras will not pick up any over exceeded tower motion and the force sensor does not detect high repulsion when it attempts the above, otherwise the previous process is repeated with the next randomly picked block. Of course, it is necessary that the manipulator positions the gripper towards the tower while maintaining a preset distance threshold, which later begins evaluating the pose of the respected tile. To be able to correctly decide when a tile has been extracted either of the following two criteria must be met; either a setup force threshold has been exceeded, or both cameras have detected instability of the tower. It is important to note that in order to avoid false signals leading to the robot taking no action, both cameras must claim the above. The necessity of putting a block back on top of the tower was covered by a force-guided MP module to avoid exceeding a threshold.

As far as the software architecture is concerned, we need to mention that the modules regarding force/torque, distance and vision are closed-loop controllers, meaning that they are constantly being fed with new data and use old to smoothen the progressive curve of function, whereas the modules that correspond to data that deal with position and velocity are feed-forward only. Moreover, with the use of a hybrid controller described as above, we can set some discrete points that distinct the different values we need to monitor

our sensors for reaching, considering the physical impact from each degree of freedom coming by the different robot' s joints. In this way, more complex commands can be produced, that, if handled properly, can concurrently manipulate multiple actuators providing us with the option of more compact options to exploit the different DOF offered by the various aspects of the robot gripper, for example we can apply action to both position and force actuators in a single command. What is more, utilizing the hybrid controller' s capabilities, one can switch between the various modules during trajectory estimation, by any means needed, namely the transform (position, velocity, and acceleration) and the space it takes place, where that is described by the coordinates, or the tasks required to fulfill, mainly with the use of Position and Velocity controllers. At the same time, it is available for the MP to decide whether it is a viable option to take a specific action, with viable corresponding to the tower' s behavior, meaning whether the two cameras provide via the Vision controller feedback that indicates turbulence, or even a collapsed tower, keeping record of each tower tile. Finally, the Force/Torque controller is responsible for providing the MP Execution module with the target pose that it is supposed to achieve after performing a move. It is also extremely useful to understand that the previously mentioned functionality is usually limited by values we need to reach with certain sensors, which further elaborates on the necessity of closed loops.

Sequentially, it cannot be helped to further instruct research teams to study and develop command-based robots, primarily focused on the middleware, without the need to calibrate the available interface on a wide scale. Software producers should not be bothered with the more technical details of implementing their goals. On the contrary, it would be both a delight and catalyst when it comes to end products to be able to press themselves towards providing the users with task-specific controllers that can offer a more specific approach during different case scenarios without however losing their ubiquitous application through extendibility. This enables scalability in both software and hardware applications of pre-existing robotic skeletons, that provide the scientists with even higher freedom in applying different manipulator control algorithms, tasked with goals of a wide range of scientific spectrum. Furthermore, multisensory integration seems to be growing as researchers continuously aim their robots to become more sophisticated and problem agnostic. Judging even by the small non-industrial example provided above, it is safe to assume that soon, if not already, it will become extremely crucial to be able to perform what is usually referred to as sensor data fusion in modern literature, to describe the phenomenon where various kinds of sensors use their potential to create more complex information that can respond to higher levels of complexity. This could very well lead to

a new era in industrial applications of robotics, aiming to further automate production processes, while at the same time it will be able to deliver faster and more efficient time and resource management.

The next part of our literature bears the weight of dealing with fusion of diverse sources of input combined to achieve a common goal [4] and was published at Science Robotics in 2019. It demonstrates that physics can have their own latent knowledge that robots can exploit to mimic human senses to achieve environment awareness, using PPO & OpenAI Gym, Gaussian Mixture model, Dirichlet process, Bayesian neural network as main tools. Humans have forever longed the ability to combine and put to actual use the data they could receive from using their senses by interpreting it into valuable information in real time. In a terribly comparable way, animals have also shown to possess the ability to claim the above as well in a dynamic environment, resulting in their growing sense of understanding it, exploring it, surviving and eventually, thriving in it. However, humans have gone beyond that. being able to complete and even sometimes compete in challenging and more complex tasks that require higher intellect. On the other hand, little to none progress has been made in robotics, since it has been difficult to create the ability to sense and acquire information through use of common intuition, let alone combine it all to produce higher forms knowledge representation. For example, there has been quite some progress in relation to simulating sight, with machine learning playing the leading role in what we modern days call computer vision. It is based on being able to identify certain patterns for recognizing objects, after having been trained in a variety of different images representing them. The latest, is usually a resource-commanding process, which in many ways could lead to either what is usually known as ‘overfitting’, or just fail, being unable to capture the objects appearing on the image properly. Despite all that has been told, it remains a decent simulation of the way that human sight works and thus, it has known quite some acknowledgement among scientists in the past years, showing marks of everlasting and growing tendency.

All the above cannot but justify the work that has been done by researchers all over the world. One strong example of such work could be the attempt to combine both computer science and real-time processing of data to be able to make a robot learn how to play the incredibly famous game of Jenga. This board game was chosen due to its demanding skill of physical understanding, strategic play and interpreting of visual data, along with the ability to be able to create responsive actions that will not affect the stability of the tower, resulting in it being unstable and eventually collapsing. For that reason, a rather common approach of implementing the game’ s mechanics was via Bayesian model,

that could sequentially create a chain of events that could resemble a cycle of understanding and interacting with both the environment and the objects in it. Two important questions seem to emerge and tackle with the scientists' skills; as far as the sense of touch is concerned, what could one do with contact information and the force applied to them, in align with object recognition of referencing, to computer vision

The researchers had to deal with an interesting and quite huge amount of yet physics limitations, that would in fact slow them down. In contrast with the later, they were able to produce effective methods that could successfully tackle with those. The main logic behind the game' s methods has been a completely hard-coded state machine, that will be able to choose the best available next action to be taken, based on several measurements coming from different sensors embedded on the robot' s body producing a massive number of signals, ready to be processed. Also, the robot used to consider the number of successive successful extractions operating in different randomly created towers. It is worth mentioning that the design of the problem is such that does not account for the presence of an opponent. Moreover, as expected the robot uses two different sensors, an Intel RealSense D415 RGB camera to be able to record RGB images showing the pose of the tiles that correspond to a 6-DOF system of movement (regarding its placement and rotation) and an ATI Gamma 6-axis Force-Torque sensor mounted at the wrist, to be able to measure the applied force on the tip of the robot hand' s finger, so that they could calibrate the impact it had accordingly. In a general manner, the image representations made by the camera were inaccurate many times, which led to misfire of fail signals, based on criteria such as after a tile is extracted the tower rotation caused would not exceed 15 degrees and in terms of placement it would not appear to have moved more than 10mms. Noteworthy, to be able to simulate learning process and experience. all the measurements were entering a buffer of poses/observations including the tower condition, so that the robot could process them in a way that would simplify the task for choosing the best suggested action.

To provide some more clearance of mind which also serves well for explaining why the scientists produced choosing Jenga for those not familiar with the game, it consists of several tiles placed on top of one another to seem as if they represent a structural admiration, intuition, and smooth movement. Therefore, the tower will not become unstable, which could eventually result in it falling. What is more, it requires a proper extraction technique or algorithm, to have the best decision of tile extraction off the tower of till Furthermore, it is obvious that in order to provide the robot with the information

needed to be able to run the simulation and guess the ultimate number of possibilities, it is obvious that a loop is needed.

The results bore fruit, proving the researchers correct regarding the state machine, as it would turn out to be an entirely different level of abstraction to expect the robot to be able to achieve rule extraction based on the two senses they gave to it, vision, and contact. In fact, they compared it to different well-known techniques used in machine learning, to be able to evaluate its performance. More specifically, the first one was reinforcement learning using PPO implementation from the OpenAI gym framework. As a reward they set it to be the positive displacement of a tile, from minor to major turbulence caused to the tower' s position that counted for negative reward. Second, they used Gaussian Mixture Model with a Dirichlet Process prior over the number of clusters to include image abstraction representations. They discovered that values for DP < 0.7 were not able to detect clusters in the image samples given, including 'move', 'tile' which stood for detecting presence of tile in the image, or 'no block' to express the absence of tiles in the image sample given. At this point it is worth mentioning that the scientists after different and random feed of samples, they concluded that the best number of representations in terms of clusters is 4, without providing more clearance. Finally, they also compared their model to the one they produced after using a Bayesian Neural Network, modeling the state transitions and force applications to the tower blocks, so that they could maintain history and simulate learning in a way that could map the tower interactions mapped to the physics conditions established after performing an action on some block.

After theoretical testing, the researchers created a physical world representation of their simulation to be able to test their lab results in real world scenarios, where robot and tower enactments could not be fully predicted. As expected, compared to the three techniques mentioned above, the implementation was able to yield better-suited results and thus, leading to more delicate application of force and block choice extraction decisions. A major impact seemed to have made the fact that the state machine they used was hand coded, which actually led to a much cleaner version of the pick-and-push process, along with properly handling the pre-calculations along with the post-manipulation and physics simulation of the tower' s block positions and rotations.

2.2. **Robotic framework for structures and their importance in our lives**

Another piece of scientific work that has provided additional background includes a topic similar to the one discussed in this dissertation, with various points being quite similar and involving the need for solving very similar problems, entitled *Greedy Stone*

Tower Creations with a Robotic Arm and introduces ideas and implementations, mainly exploiting various third-party created software to explore the possibility and dynamics of stacking different shaped stones to form tower-like structures [5]. The key points of this research are that of a robot is tasked with stacking stones of unknown shape on top of the other, while utilizing PCL framework and clustering algorithms for processing data fed by a RGB camera and a force/torque sensor. Since its birth, mankind has always sought shelter from external weather conditions threatening to harm their health and outdoor potential enemies, including other humans. This led to the creation of houses that could serve their purpose to be reliable and sturdy enough to withstand outcoming damage of all sorts. That is how the idea of having structures housing humans was created at first serving as their property and aiming to provide stability and safety. Speaking of creation, if we look into humankind' s evolution, we will also notice the dramatic change in both size, design and material requirements. Eventually, humans started organizing into communities and societies, sharing their ideas of hospitality and human shelter as part of their civilization. As time passes, one cannot but notice that the evolution of humankind, among other things including but not limited to work places has been drastically altered, since teams of people with common goals started appearing, with needs to organize themselves under the aegis of the same organizations. In modern day societies, the ever growing need to provide companies along with other forms of organized common life has been driven by and sometimes even strongly affected by the quality and quantity of the constructive materials available.

After centuries of evolution, humans had finally the intellectual capacity to understand and seek ways of automating the creation of such structures, an effort that was kickstarted with the first Industrial Revolution, taking part during the 18th century. That was the first official attempt by humankind to apply formerly acquired knowledge in many sectors of science through machines. Since peeking into the very beginning of machines helping humans to achieve their goals, let us now investigate modern applications along with their impact on a primal need that has driven humankind since its origins. The production of quality-level buildings that would serve as shelters from the outside world.

Trying to provide a clear understanding of the line of evolution so far, we will briefly mention the presence of huge machines such as bulldozers to be able to cover up for the need to apply great workforce into creating large structures. The next idea that included industry, seemed to be the automation of the production process in many goods and services. The final and most interesting part has been the exploitation of robotics, that would actually come to put an end to human supervision and need to overlook many parts

of automation processes, since it has been unavoidable that at some point machines would have to decide for themselves as to what would actually be the next best step, in a scenario that has not been part of the main production chain, especially when there are dire and high-cost consequences, coming as the result of automation in industry, e.g. the creation of structures.

That was a crucial point where robots could play such a key role, since there are many distinct aspects of the building process that could be automated. Admittedly, one could expect that in such an environment where most of the factors are well-measured and known at their full extent to affect the process, it would be rather needless to say that there is an increasing need for human supervision. However, the truth is, that even the most recent ways of robotics application in construction have been in indoor environments, where there is, as mentioned full knowledge of the environment and the availability of certain materials. So, what happens when we need to consider the fact that construction includes outdoor structures, without predominant knowledge of what could be used us constructive material, since the robot would have to utilize the available resources.

Imagine that at some point, which is a really close goal of humankind to be achieved soon, humans decide that the planet' s resources have become somewhat unable to sustain the growing needs that modern lifestyle is demanding. In fact, due to overgrowing population growth, that case of scenario has already decided to be provided with the solution to create house-like structures at other planets, such as Mars. Due to the needlessness and inability of humans to create the surviving habitat on such planets, robots have been tasked to create and provide us with pre-made constructed buildings that could serve as houses.

Unfortunately, considering the complexity of such a task, many researchers have concluded that it is no longer possible to carry with them the materials needed to achieve the above. This, however, has led to the creation of the idea to provide construction robots in outdoor environments with the ability to identify and use already existing resources to create such structures. Driven by this inspiration, many scientists seem to have seriously considered to put some effort into developing similar case solutions. In this paper, researchers were tasked and challenged with the need to create simulations with robots being able to identify objects that could be put to such use and later, form constructions with some structural order, namely stack them vertically, which would mean that they are able to balance, something that has always been considered extremely needed.

To be able to simulate and minimize the working process, the scientists decided to use a robotic arm empowered with an RGB camera and a force/torque sensor, to be able to

perform the following tasks; identify an object that has already been present on the scene and later use it to balance it atop other similar objects, while maintaining structural stability and thus, balance. More specifically, the force sensor was placed at the attaching point of the gripper to be able to account for the impact that would be created after the placement of each new object on top of the existing ones, since that would be the place that would provide them with the necessary feedback to proceed with the implementation. Furthermore, the camera sensor was able to provide high resolution images that were used to identify objects based on their key points, in other words frames created by discovering some important points of the seen objects, recognized with the help of PCL (Point Cloud Library), which is an implementation of the ISS (Intrinsic Shape Signatures), a framework that describes the importance of certain points of an object that help identify its shape. The last step was to rotate the object identified to a default pose, based on clustering algorithms that could verify the presence of enough points to confirm the presence of an object and of course the fact that its flatter surface is vertically pointing.

The results were not really encouraging however, showing us that there is still large room for improvement. More accurately, the experiment conducted via the effort of consecutive runs of the robot to try and stack four different objects, namely rocks in this case. The outcome was that the robot was able to successfully stack all four of the rocks on top of each other on only two out of the eleven trials, which was more than two times the cost of failure that the scientists have predicted. However, it is worth mentioning that this should not be taken lightly, as in a real case scenario, it is guaranteed that the robot will have a lot more alternative options, since there are expected to be more objects in the scene. Also, it could potentially mean, since researchers revealed for example the percentage of successful placements of rocks of a set consisting only of three of those, which was a solid seven out of the eleven different efforts, that in a general case scenario where we assume N available rocks, the probability that N-1, N-2... etc. is still a very promising expectation. In other words, what could not be achieved due to lack of experiment conduction materials at its peak, could not necessarily be a failure, if we extend the number of available objects on-scene, meaning that what appeared to be less than acceptable, namely stacking less than all four stones could be more than enough in actual life problems.

There has been and always will be a need to create structures that can house humans, or even whole teams of them. What is more, the current work could be considered only the beginning of what will be one of the most popular tasks to achieve in modern robotics in the next few years. It is unavoidable that humans will have to turn to robots in order to

provide a safer option to construction and make the most out of the available resources, without the need of human supervision.

2.3. **Robot autonomy and human-robot collaboration**

Years have passed since computers have entered our lives, but whether that was efficient has been affected by the ability to imitate and solve problems, along with the quality of solutions. It is quite common to consider the complexity of a problem in order to be able to decide the best course of actions that need to be made towards solving it. For robotics, after researchers managed to solve simple kinematic-like problems, they turned to more advanced issues that encompassed more delicate approaches for them to be able to complete certain tasks. A communal problem that has troubled scientists and roboticists among the world has been the creation and maintaining of a Jenga-like tower, as it will be described shortly, demonstrating the importance and complexity of a planner to efficiently proceed with solving a planning problem [6]. Most common reasons have been the need for a planner that can safely evaluate the best block candidate for removal, along with other reasons, such as the ability to maintain stability. In this paper, a PIONEER armed robot with 5 degrees of freedom was created, keeping the overall cost to the minimum possible, by using previously made system components (such as the PIONEER robot itself) and strategic algorithms to be able to predict system failures. The robot PIONEER by Adept, has been one of the most used robots in scientific research, since it provides flexibility, because it can rotate about 300 degrees in a second despite its low speed, autonomy due to its long-lasting batteries and of course the programmable interface to simplify control over it, 5 cross-platform tools. It also features high-precision sonar sensors, making it ideal for tasks that include obstacle avoidance and navigation in complex environments. So far, previous works have been implemented in similar problems by South and PhysX, simulating the physics that could represent such a tower along with the conditions to maintain it, but we should bear in mind that they did not go as far as the 18-level tower, but were able to partially construct it.

The current system was able to surpass that limit significantly by creating a full 18-level Jenga structure and adding another 10 levels on top of it, while, as mentioned above, the team tried to keep the cost to the bare minimum. Remarkably, the main and in fact only strategy adopted was the random selection of blocks, which was evaluated by a 3-stage planner, called BRP (short for Block Removal Planner). It is worth mentioning that despite the fact that the overall performance achieved was nowhere comparable to the one using the Kroger platform, it was able to deliver incredible results, especially if we consider that

as far as sensors are involved only 2 vision camera sensors were used, a decision that put this robotic arm implementation ahead of previous and later robots that mostly feature vision and force-torque sensors to be able to maintain tower balance and avoid collapsing due to overextending the force applied by the robot in a given moment.

The planner, as described above, only consists of three distinct stages. The first stage is the one responsible for creating a list of potential blocks to be removed, based on neighboring tiles according to the previous extraction action performed. The second step includes the selection of one such tile using heuristics to provide selection. At this point it would be important to note that due to minimizing the disturbance levels caused to neighboring tiles, the central ones were chosen with slight priority over side tiles, if possible. Also, regarding heuristics used scientists did not further specify the methods used in the paper. The third stage appears to be the most sophisticated one, since it conducts a physics simulation to compute the efficiency of the removal of this block, judging by the overall tower stability it results in. Based on proper gripper rotation it is assumed that the movement will not cause any close by movement, thus disturbing the transform (including position and rotation) of neighboring tiles. Vision-based error detection is used to evaluate the cost of this move and the option to abort it is present. More specifically, the planner uses heuristics to evaluate the remaining stability, using iterative techniques throughout the list, starting with the tiles closest to the one chosen in stage 2 if it is deemed as a bad removal option. The latter is determined by the return value of the heuristic, which compared to a threshold of overall tower stability set by the user will determine the overall quality and safety of performing this move.

Moreover, it is worth mentioning that the project was able to sustain continuous monitoring of the tower' s motion through the vision camera during the removal action to be able to launch a 'stop' signal in case of threshold exceeding. The above was achieved with limited resources, such as a single desktop computer to perform the decision-making and processing of the sensors' readings. The main logic consisted of two subsystems, with one of them being the CMOS camera mentioned before as the most critical component, which provides the necessary image readings, which are then transformed into a 2D planar version of the image to represent the player' s point of view, increase realism and reduce error. The second system is the one that handled and processed all motor commands passed to the servo component to be able to manipulate the robotic arm in a successful way. Also, the distance from the griper to the tower is considered known and standard, while it is worth mentioning that the tower shape and size, along with lighting and movement of the structure are constantly considered, resulting to a more accurate image representation. The

motion of the servo-based robotic arm on top of the PIONEER robot is only position controlled, which led to fewer actions needed to be taken, but also the necessity of initial orientation to be given on the gripper.

The Player system following the server-client architecture as implied above continuously keeps on listening for position change controller commands. Furthermore, the best outcome able to achieve was 28 levels of Jenga tiles, however in the final implementation it had only nine levels of tiles due to the length limitation of the arm used. This, however, should not be accounted for as a failure of compliance with the standards it set. Instead, it should reflect to the various difficulties that robotic systems manipulation includes when dealing with more complex problems, such as Jenga, Also, to avoid the need for further computations, whenever a tile was removed it was not placed on top of the structure as the game suggests, but instead it was dropped by the side. Again, this was raised as a decision to avoid meaningless computations, without meaning that it could not deliver the proper actions requested and reach its goals. It is noteworthy to mention that in case of tower being disturbed by a single move due to e.g., blocks being out of place, then in this scenario the system and eventually the robotic arm itself were not to perform any corrections, staying consistent to the game rules in real life scenarios. The most helpful components of the entire process were the vision-based estimation of the tower state along with the correction algorithms and measures applied to it, while the continuous monitoring of the tower' s movement was providing with real-time data on overall tower stability, being able to assist in structural integrity and sustainability.

The main inspiration behind this paper and the ones that contributed the most to the final result appear to be already known commonly used Jenga block extraction strategies, the values monitored during block removal trials through simulations and the realistic and low-cost physics evaluation considering tower stability. However, there is always room for improvement, which in this case could possibly mean better image processing techniques using better resources along with more accurate movement detection algorithms to be able to monitor and interpret the tower' s stability in order to provide the system with better and safer removal options. Furthermore, heuristics can always be improved by adding more parameters, considering several factors such as history kept to compute the next move and of course combination of both, all in favor of maintaining tower stability. Another example of future work can reflect on the validity of simulations, which could be more accurate in the future. The last, however, should be done with caution due to possible dramatic increase in both computational and physical costs, as it has been one of the key points of this research to maintain small cost.

The next step in literature review regards ways explored that involve human-robot collaboration. Admittedly, it can become quite complex for a robotic system to understand its environment, let alone operate in it and achieve its goals. Thus, having discussed increasing the intel it gathers above, it is time we consider ways to fuse motion planning and task completion with human intellect. The study that will be presented introduces a modern system that can simplify the process of robot automation, by involving humans. More specifically, it elaborates on constructing an intellectual bridge to fill the gaps that prevent robots from being fully operational, replacing complexity tasks with human interaction [7]. In modern day industry, throughout many different sections of applied sciences and trading, along with coordination and transportation, there has been monitored a direct increase in automation of fulfilling specific tasks, most of the production process. However, not all areas have been automated, which creates an ever-growing need for applying techniques where the progress of the completion of a task includes supervision and most of the time collaboration between humans and robots, thus the lust for creating a bridge to be able to combine the dynamic capabilities, or as mostly wanted, the best of both worlds.

A terrific way of achieving the above is through being able to make a robot and a human person handle and manipulate objects and actions in the same environment, at the same time if possible. Such abilities are offered when using augmented reality, also known as AR, where the user can manipulate virtual objects, actions that take place using the real environment as a setup. Of course, similar techniques such as VR and MR are also frequently used, but for the sake of this paper we will be focusing on AR. This enables one to interact with virtual objects placed at a known environment in a robot-agnostic and object-centric approach, giving the means to spend more time on determining the actual goals, rather than spending time specifying the respective actions to achieve them whatsoever.

This opens a whole new world of possibilities why the more complex the setup of a robot and the problems it aims to solve, the harder it gets for the user to be able to understand the dynamics and actual reach of each potential, other known as reachable workspace. Furthermore, the user is un-bounded by the restrictions of the use-case scenarios of a robot' s planner, along with the complexity of organizing the actions needed to be able to achieve the desired goal. Let' s just consider a problem where a user demands the robot to be able to pick and place cans of same shape and size from position A to position B. We can easily think of different scenarios for this problem already, for example cans placed with different orientation and spread across more than one location. The way

someone would come to naturally approach this problem would need a lot of man work and time spent to create a controller that would be able to create valid plans to achieve the above. By using AR, one can only focus on the representation of the objects, along with the motion plans to be able to make it feasible, with the latter not even required to be optimized, or even valid.

So, for the sake of clearance, let us repeat the plan of execution. The user can share information and communicate with a robot via AR, used as the bridge to overcome the problem of combining the best capabilities of task automation and human supervision. In this way, the user can define an initial position of the cans, as mentioned in the example above, which is the actual test-case scenario in this paper. Then, the user could place the cans in an entirely new area, which would later enable the planner to take over and produce approachable schemes of action and motion planning.

To be more specific, the robot gripper in this example is dubbed Fetch and, on its endpoint, includes sensors such as vision cameras. This enables us with feedback that could be used to create created plans, or provide us with information concerning the interference of the user during each step of execution of the planner steps. Had we taken a more traditional approach on obtaining our goal, thus called low-level motion planning, we would end up with robots being functional only under human teleoperation, or manually setting collision-free volumes in order to optimize the determined trajectories of a robot' s motion.

The intended goal is to provide the user with a framework that will be able to let the user focus on high-level operations, based on previously programmed low-level actions that make up what is a called, as mentioned above, a high-level action. The user is assisted with visual representations based on the Vicon camera feedback we get from the sensors of the robot. In this way, it is easy to group low-level actions such as planned paths, objects that are selected currently and gripper selected actions into a group of useful information to present to the user.

It is worth mentioning for once again, that the user is responsible for the recording of start and end positions of the objects they wish to move. The robot takes over from that point, executing a pick-and-place algorithm expressed in steps.

The combination of dynamics mentioned before, however, poses some serious limitations, including the lack of extendibility of the low-level available actions, that do not cover for different versions of the problem, a different environment setup, or a completely different problem. Although, the scientists decided to keep the idea of a virtual inventory, which serves as the user' s personal warehouse of objects to be used, which will

be able to combine the user' s intensions, expressing them through the robot' s selected motion plan.

With HARS (sort for High-level Augmented Reality Specifications) a user can take advantage of an AR interface, can define the transform of a virtual object coming from the virtual inventory, including position and rotation, and later let the robot take over by implementing a motion planner in a much more dynamic environment, rather than hardcoding the actual possible states, which undermines a latent pose-agnostic virtual object placer.

To achieve all the above, one would suspect that the user then struggles with a lot on their plate, considering the complexity of certain tasks. However, not the user does not need to know the actual low-level actions taken, but also it enlarges the area of goal implementations, which is introduced as a goal region.

To be more specific, a goal region is nothing but all the valid alternatives of a motion plan to complete a desired action target. The user can define the orders of magnitude that the above will impact the plan execution. This is achieved by applying workspace constraints. These are simply distance tolerances to consider it has approached a target (either an object, or location) and can be modified by the user.

Moreover, a set of different workspace constraints can describe the general preferences of the user to consider an object as preferably placed in the desired location, while maintaining respect with a real object. This could safely be interpreted as a goal region, or even so a goal state, containing all the information that can identify that a robot has reached its target configuration.

It is worth mentioning that while in goal pose configuration, it is essential that the robot used must not be colliding with any nearby obstacle. This is both realistic and time saving, since having to perform several calculations to determine the above could have cost the robot both time and processing power. However, the real problems come into place when the robot detects a previously unreported object in the path towards the goal state. Through the AR interface, this can be easily overcome by generalizing the area that an object can be placed within. Also, the robot tends to keep track of the order of the user' s hand, which enables it to captivate a more refined idea of the appropriate path. The real dynamics of this paper' s suggested proposition comes into play with the user being able to preview the robot' s intended motion, giving them the ability to alter a mistakenly made choice by the robot. We can already see how valuable information sharing is when it comes to human-robot collaboration.

Upon initialization, or as referenced the 'boot', based on the localization system used by the robot a position frame reference is used to compare the virtual with the real object' s position and orientation, maintaining calibration, a process repeated in each session.

Returning to the example mentioned before though, in that simple scenario which represents a very practically common working problem that needs to be tackled even in modern robotics, the Fetch robot will first have to identify each own position, proceed into understanding the target' s position and finally approach it, while using distance tolerance defined by the user to determine whether or not the object is within reach. It then checks for any non-previously reported objects that interfere with its course of motion and finally creates a representing area from which it will be able to approach the target-object. After that step, it also uses a placement region which corresponds to the area that could be used to place the can, letting the robot freely choose any available spots.

With the above techniques the scientists were able to create a more generalized version of a human-robot collaborative way of achieving complex tasks, with the use of high-level motion queries, that can reflect to more open requests from the user.

2.4. **Human-like robotic arms**

Published at IEEE in 2011, with the aim to create a robot capable of playing Jenga using artificial fingers [8], it has emerged to be one of the few scientific articles that seem to be tackling with an almost identical problem. It also features a more human looking robotic arm, trying to imitate the same complexity that encompasses human armature. Main tools are an omnidirectional camera and a metric introduced for computing structural integrity of Jenga tower called degree of danger. It has been well established that as society evolves and more sectors of science turn to task automation to be able to focus on more sophisticated and complex aspects of problem solving, we need to force greater use of robotics with higher intellectual capabilities along with more developed decision making. In fact, we can already witness both the need and application of the above in various aspects of industry, increasing productivity and reducing the need for human supervision. It is even more commonly known that to achieve greatness we often need to start small. That is, researchers usually tend to game simulation, since they combine both strategic decision making and physics simulation in providing the full experience. It is widespread among scientists to usually consider two diverse groups of games that could in fact be enhanced and simulated with the use of robotics. The first group consists of games where it is essential to apply more sophisticated techniques to provide challenging solutions and

actions on an entirely virtual and simulated environment. The other group consists of games that can be interactive in real-time and in the real world, played against actual human opponents. Speaking of games, one of the old favorites has always been Jenga, having both tactical reasoning and understanding of the environment as mandatory qualities.

In summary, among other teams of scientists, this specific team focused on being able to utilize a more human-like approach of playing the famous board game. To be a little more specific, they decided the use of a multi articulated robot hand, consisting of two fingers covered with soft skin, instead of following the common approach of utilizing a gripper to be able to select and grab blocks to be extracted. In addition, they used as most researchers suggests an omni-directional camera, with the obvious difference being the fact that the camera can provide depth in the images created, instead of 2D planar-like cameras that have been used so far, ignoring the information that such commodity could provide us with.

The main problem they have been focusing on, could still be interpreted as a more enhanced version of the ones already explored, since it worth mentioning that it includes playing against a human adversary, yet the goal is still trying to stack as many blocks as possible on top of each other to create new levels of tiles while maintaining tower stability. As a result, an experiment was conducted by taking turns against the robot to show the potential of the robot' s block extraction evaluation function.

The research group was in fact trying to approach Jenga robotic simulation in a more interactive and dynamic example, involving the unpredictability of the human factor interaction with the robot, to be able to demonstrate that such an option is already available, giving more solid ground to future collaboration between humans and robots which could combine the productive scalability of machine application in industry, along with the human mind' s ability to analyze and find similarities in more complex problems. This could lead to a new industrial revolution, forever changing the way we can produce the necessary goods around the globe, covering the needs of millions of people. Although it all sounds exciting and very promising, for now we will be focusing on describing the endgame of this scientific experiment, after going through the process they produced, to overcome step by step the different difficulties their dream had to go through.

Two things have been common with previous approaches, despite the innovative design of solution they tried to provide. The first thing has been the camera they had to include in their implementation, which along with force sensors placed on finger tips were responsible for the tasks corresponding to block detection and finally the need to calculate

the force with which the robotic arm should be extracting and placing the tiles from and to the tower, keeping in mind the necessity of maintaining tower stability.

It is now vital that we describe the method of extraction and algorithm to be able to calculate the force that had to be applied. But first, let us take a brief look at the robotic arm itself. In this study, the robot arm is the PA10-7C arm, provided by Mitsubishi Heavy Industries, Ltd. and has seven DOF (degrees of freedom). It is also worth mentioning that the way its joints have been distributed in a way similar to that of human arms. First things first, we must look at the way the objects are recognized as tiles. The camera serves the purpose of providing the robot with 2D 640x480 pixelated feed of planar-like pictures of the environment taken with a VS-C450U-200-TK by Vstone Co., Ltd. camera and recognized objects that is later used to represent the 3D image of a tile. As we mentioned above, the latter seems to be in contrast with their claim of not basing the image processing into 2D images, but it still impresses that this is achieved indirectly, since the 2D versions of the objects detected are fed to a middleware that is responsible for representing the exact block. The next step in the algorithm is to evaluate the danger that extracting the block identified implies to the whole structure' s stability. There is not present a more sophisticated way of approaching the selection of a block to be removed from the tower, however the camera that is able to identify the one it recognizes first, also serves to provide the algorithm with a list of one or two alternative candidates. We also need to specify that all tiles are expected to have the same thickness and mass. In addition, the main area of focus as the tower gets taller is always with the camera focusing at its center, to maximize the blocks it should be able to detect. Noteworthy, the main logic behind computing the degree of danger is the fact that we consider the number of upper surfaces a virtual vertical line drawn from the top crosses their very center until the blocks at level that stops in the middle of the tower. In case there is just one candidate, the above metric is not used, but in case the latter is not true, we obviously must choose the one that returns the shortest value, since it means that it plays the least significant role in the tower' s stability and thus, its removal will have the least significant impact. At this point, we must make clear that the process of placing the tile back on top of the tower follows the same principle as the method of extraction, with a small difference; the metric is now considering a virtual vertical line drawn from the center of the tower all the way up to the top, however it is still important to point that the minimum value is still selected, since we aim to disturb the tower as little as possible. As far as the force sensor is considered, we must underline that small six-axis force sensors (NANO5/4 by BL. Autotec Ltd.) were placed on the tip of the fingers to provide the best feedback possible, while the soft skin covering the arm was

made by elastic gel to make the arm manipulation task more stable providing additional frictional force.

The main experiment conducted represented a match between the robot and a human competing on a six-level tower of Jenga tiles. The main task was to prove that in order to provide more accurate and precise force-based measurements that can lead to better results in terms of simulating human behavior we need to adopt a more man-like approach of robot design. Mimicking other beings' behavior has acted as a deterrent to help in humankind' s evolution, both emotionally and physically. It appears that we should follow our own example and provide robots with physical design as close as possible to humans to expect better results.

Previous mention of the importance and involvement of board games in robotics evolution has been referred to as a crucial factor to speed up the development of complexity and completion of more sophisticated, so called high-level, actions that a planner is expected to perform. Due to this significance, researchers have been motivated to work and study robotics in similar cases, which led to more concrete base for developing robotic systems that are both human-like and can prove to be worthy antagonists against humans as described in *Development of a Jenga Game Manipulator having Multi-Articulated Fingers* [9]. Past work by the same team is discussed above in this dissertation and has been extended and driven by the same factors as certain researchers have based their work on the implementation of a Jenga tower manipulator. More specifically, in contrast with former attempts they put a significant amount of effort to be able to further simulate the experience of human involvement and playing of Jenga, to provide us with more concrete examples of greater influence of robots in everyday activities and, to study the interaction between humans and robots in a more dynamic environment.

It is widely admitted that games can be an incredibly challenging, yet rewarding area to apply robotics, since one needs to tackle with many different issues to be able to deliver a working simulation. Furthermore, they serve as a working blueprint of human intellect application and study, judging by their nature to stimulate the brain into thinking complex solutions and strategies to overcome different obstacles. Jenga, the world-famous game, for example, is quite widely studied into applying robot manipulation and planning techniques, mainly since it combines environment detection and understanding, along with strategic planning and physics evaluation.

As mentioned above, the current study was heavily influenced and based on research conducted barely a year before this one was published. Most importantly, it offered some great alterations to the various aspects of different implementation parts,

while maintaining the same principles, ideas, and goals. To be a little more specific, the main course of idea was aimed towards the implementation of a robotic hand that would be able to suffice into playing competitively against humans with two articulated fingers, instead of utilizing a gripper as in former experiments. What is more, they researchers decided to cover the whole arm with soft skin tissue, to provide more stability while getting a grasp of the Jenga blocks during extraction and placement. As far as the tower state recognition is concerned, they used an RGB camera that would provide the robot with different frames of the tower, while they were fed to the planner which was responsible for choosing the best candidate tile for removal through a sophisticated intuitive method of extraction.

Let us first discuss the key differences that distinguish the current implementation compared to the already existing as it was described above. We first need to clarify that in terms of logic applied the same process was followed, so it is important to point out the fact that the scientists considered the pre-made concept as fulfilling to provide nice results, however the main differences lie within the implementation followed as it was significantly altered and improved in unusual ways. What is more, they still aimed to mimic the way a human would play the game in the most natural of ways, which justifies the claim that their goals did not shift from the original study, an important reference to elaborate for the claim we made above. Considering that partially the research team behind the newer implementation consisted of members that took part in the very first implementation this should be expected and a means to firmly insist on creating robots based on human resemblance, as it provides more stable and natural results, helping us understand the way robots should be created in order to be injected in our lives, a fact that seems unavoidable. So, to resume on enumerating the distinction between the two implementations; first, there were significant improvements to the state recognition time needed for the robot to understand the current state of the tower, which also involves the proper tile chosen to be removed. In detail, instead of an omni-directional camera, they decided to use a monocular one, which might have restrained the FOV (Field Of Vision), however it also led to fewer unrelated information to be fed to the planner, which created the necessity of providing less images for processing to understand the way the tower stood still, leaving more room for providing a more realistic approach. Second, another key improvement was the ability to make the fingers used by the robot hand thinner which provided more refined movement and successful tile extraction along with error production, eventually leading to making fewer miscalculated decisions. On the contrary, more accurate actions were able to be made on the cost of having to use commercially used Jenga tiles, since it was no longer

needed to use custom made tiles, namely strapping two normal blocks together due to the wider implementation of the fingers, which happened to be the case in the first study. In addition, another significant impact was made after altering the position from which the tiles are grabbed. This time, instead of having to grab a block from the top, the two fingers of the hand can grasp a tile from the side, mimicking the human behavior and giving the opportunity to play with higher Jenga towers, since that has been a major constraint of the previous work, as it only allowed three-level towers to be implemented based on the arm' s length.

Combining the above with the robotic arm already present, it helped the researchers create a more reliable and realistic implementation of a Jenga manipulator, which could realistically play Jenga [3]. Noteworthy, the project was separated into different tasks, judging by the differentiation of goals needed to be achieved as they were described by the scientists. The order of these tasks is the following; first, they had to be able to provide the planner and in fact the robot with the ability to recognize the tower state, to be able to have the available data to compute the best tile removal candidate, a task that was achieved through the addition of the monocular camera mentioned above. Second, the images provided created a tower state representation that was able to conclude to a specific tile choice to be removed, through a sophisticated method. It appears that the scientists, in order to maintain tower stability, they thought of a metric called "degree of danger", which simply counted the number of upper surfaces a vertical line would cross from the bottom all the way up to a level k, passing through the center of gravity of these levels, namely the center of gravity of the block placed in the middle position of each Jenga game level. In case more than candidates were present, we would have to choose the one that had the smallest degree of danger, since theoretically this would correspond to the safest available option. The next step was to decide where the grabbed block should be placed, having only three available spots on top of the tower. This led to simpler computations, but at this point it is interested to point out that the scientists thought of this process as similar to the extraction functionality and thus, the same metric was used to evaluate the position that posed the most minor degree of danger, interpreting it as the position that would have the least significant impact on regards to the stability of the tower maintaining its balance. For simplicity, we just skip the discussion of the remove/place process followed by the robotic hand, as it is implied to be mechanic and same in all different scenarios. Instead, we should focus our attention on a similar issue that came up during implementation. That issue was no other than the obvious asymmetry in the block shapes, since they were slightly different from each other, something that very well could be the product of misplaced actions. To

solve this problem, the researching team decided to slightly tilt the way the hand should be grabbing the blocks every time to be able to deal with the different placement ways that could potentially lead to false grabs and tower collapse. Moreover, the force sensors were placed on the tips of the fingers to provide more accurate calculations regarding the force needed to be applied during placement of a block back on the tower.

Experimenting, in contrast to the first implementation, as we mentioned above, it was a game changer to be able to create greater towers of Jenga tiles, providing more insight as to whether the implementation was hugely successful. There is room for improvement, since the robot did not seem to have the expected results, however there are many factors that could be altered and provide us with better results, e.g., providing the robot with a more sophisticated tile extraction method, or taking more than two images of the tower to understand its state, as the current implementation suggests. Finally, we should also mention that the game between a human and a robot is automated apart from the fact that whenever a human has completed a move, they need to signal the robot to initiate its move decision, a point that could also be altered in future implementations. However, the scientists seem to believe that a more human-like robot is expected to achieve greater performance.

2.5. **Motion planner benchmarking**

Continuous evolution of both hardware and software regarding robotics has led to a spontaneous need of performance measurement to performance assertation to ensure efficiency, separating mere solutions from implementable actions. Under the lines of this guidance, the word performance was associated to benchmarking. This has led to the development of tools and thus problems that can measure the efficiency of algorithms, problems, and software in robotics, which will be introduced in the presentation of different benchmarking problems that can safely evaluate the efficiency of different planners in a problem-agnostic way [10]. Use of robots in various aspects of modern lifestyle and culture seems to be growing with exponential rate, eventually leading to the need of a measure or metric to be able to compare different planners. Moreover, one of the recent planner trends that has received excessive attention in recent years is no other than the one of motion planning, aiming to provide automation and ease among different TAMP (TAsk and Motion Planning problems). Transportation, medical applications and industry are only some of the highly affected sections of the fundamental gears that keep today' s societies from falling apart and seem to also attract a lot of attention considering the fair fact that their role is so crucial that one could not think of a single day in their lives without

the exploitation of what technology has to offer on regards to the services mentioned above. However, as we previously underlined there is no current way of being able to provide a concrete and solid comparison metric as to which should be esteemed as the crown leader of each corresponding problem. That issue has been tried to be dealt with a team of researchers trying to provide us with the first platform independent evaluation method and format of organizing and presenting such problems, through a collection of benchmark issues trying to cover the most popular and challenging aspects of TAMP.

To begin with, we need to understand the reason such benchmarking system is more than necessary. In fact, robotics has received global recognition and interest across its application to many different scientific sections. However, that alone would not be able to provide us with a clearer image about the usefulness of the potential of planners and robot controllers in general, unless we are able to compare them effectively. Let us first have a brief look at the evolution of humankind through technological advances and its impact on society. Humans have always, for example, have struggled with the need to be able to provide themselves with the food they needed, the shelter they sought, or even the clothes that could protect them from different weather conditions. Throughout industrial revolutions we have witnessed many different changes in technology and the progress could not but be expected to eventually affect the complications and applications that the effort to achieve the above would experience some alterations. Leaping through history to avoid wasting the reader' s time, it is only the least fair to strongly claim that we, humans, have always struggled to industrialize technological advances to fit for our needs and make modern life easier. Inevitably, humankind reached a point in time and technological progress that could no longer spare us the time to handle problems of the past, as their solutions were already invented, tending to turn our thinks into automating their functionality, especially with the used of programmed robotic behavior. That has been already widely known, however, it has been discussed the fact that us societies evolve our needs tend to become more complex, meaning that we can no longer rest at previously discovered techniques of planner evaluation, since today' s planners and controllers tend to describe higher level actions and even abstractions of robot states and actions. To give a common yet still intuitive example, one could refer to the robot' s location without feeling the need to specify the coordinates of its base, rather than its relevant location to a known environment (e.g., some robot X is in the living room), which if analyzed can easily be interpreted as an abstraction of the current state of the location of the robot, where the user is informed about the relevant (or, as explained by many researchers semantic) location, preventing the user from entangling with details such as its actual coordinates. Furthermore,

we could also use a similar example to describe abstract forms of actions. For example, one could possess a robot with the skill to provide them with a glass of water. During robotics' earlier stages of development, we would naturally care to define the different sub-actions that would make up for the result, that would cover for transporting the robot to the cupboard, get a grip of a glass, fill it, and then move on towards the user, until it could finally stop. During these times it was relevantly easy to be able to compare between the different planners, as their level of implementation included the most basic of actions.

In contrast with what we have already discussed, such methods are obsolete and useless when it comes to high-level actions and states, that use one or more layers of abstraction, unavoidably hiding most sub-goals needed to be achieved along with states to be reached, becoming easier for people to understand, further develop and evolve the actions taking part in goal achievement. This benchmark collection is primarily aimed towards PDDL problems, but could easily be generalized to respond to other formats, such as ASP (Answer Set Programming), or C+. Noteworthy, their focus has been around path finding for problems issued to include the necessity of continuous actions.

Moreover, TAMP, this paper' s main product of scientific research, includes those problems that need to undergo certain modifications to meet certain requirements for the comparison to be able to take place among similar planners. First, it is crucial that we strictly define the established relationship between state representation and system configuration including the hardware and software responsible for delivering the desired outcome. Second, it is also particularly important that we specify direct relationships between actions and motion plans, in order to provide a clearer understanding of the ones responsible for contributing to the result the most. One could think of the latter as a means of heuristics, but in case of judging by how much faster a variable gets close to the target value, we could instead how much the desired outcome approaches what we mean to achieve. At this point, we should point out that TAMP planners use their own symbolic-geometric mappings to describe object states and thus, they are independent of the language implemented upon. To be able to understand the general format of such a file we provide the next definition and further elaborate; $\Sigma = (S, A, \gamma, s0, Sg)$ represents a task' s domain different domains for evaluation. More specifically, S corresponds to the finite set of states possible from the current state, namely the initial one. A, on the contrary, consists of a finite set of all the available actions that could be performed from this point forward, while γ is used as transition function which is used to give us the ability to move from one state to a new one through a specific action, given that all prerequisites are met. What is

more, ‘s0' is simply used to identify the initial state, whereas ‘Sg‘ represents the set of goal states that we wish to achieve.

Having made the above clear, it is obvious that judging by the vast area of non-explored states, which includes both the ones we can reach in one step along with the ones that can be reached later in the process, the amount of information we need to maintain often tends to get out of hand, so it is beyond doubt needed to use on of the more compact forms of representation, such as the ones mentioned to keep track of the various problems. These problems could be part of other larger problems, but to be able to be classified as TAMP, it is vital that they do not contain infeasible actions in their domain (A), also their task space is often relatively large, otherwise there is not much use of benchmark problems proposed and of course, we need to keep clear that certain metrics are also mentioned. Among such techniques, we can count the trade-off between performing an action and evaluating the progress we were able to make, the factor of monotonicity in terms of how much certain objects are used compared to the rest and the general impact on current state a planner includes, or in other words, how drastically the state space needs to be modified.

After discussing the preliminary requirements, we can now safely progress with mentioning the problems on which the benchmarking process takes place. First, we introduce the already famous problem of the Towers of Hanoi, and its extended version to be able to handle the physics complications that should avoid a disc being moved while being below another, which was successfully solved by providing the discs with greater thickness. Second, they introduce the problem benchmark dubbed Block Worlds, which involves correct placement of blocks with letters in alphabetical order. Third, we should discuss about the problem of non-Monotonic, where the main goal is to place tiles in groups based on their color, while considering their size. Moreover, they present the benchmark of Sort Clutter where the robot is trying to place in size decreasing order the block it is given. What is more, we have the Kitchen problem, which expects the robot to be able to place several tiles in the proper order to achieve a desired sequence.

As modern-day societies tend to evolve, there seems to be an increasing demand on delivering more complex services or products. It is undoubtedly part of our reality the necessity of automation in many distinct aspects of life and the application of robotics seems to be a reliable solution. However, as complexity increases, so does the level of automation and its demands do, expecting for more abstracted, higher-level actions, while the amount of information makes it crucial that we also describe the robot steps and environment states in a compact, yet understandable way. This paper proves that it is also still necessary to be able to compare different planners suggested and provides us with five

different problems to be able to understand how to correctly choose between the available motion planners.

3 Methodology

3.1. Controller and implementation details

In robotics, it is vital that we combine each robotic system with a controller that offers the necessary flexibility and efficiency when it comes to solving problems. One of the options is to create a controller that can perform and execute specific problems and actions. When this route is chosen, we soon realize that we cannot use such controllers further than a certain extent, which eventually leads to inability of controller usage. Of course, as in every situation it is wise to always measure and consider the advantages and disadvantages of each method studied or used. In this case, this type of controllers offers limited dexterity and solution to problems where fast prototyping is necessary. However, they do not perform well in terms of problem scalability and diversity. At this point, we start considering another option, that of a more generic controller that will be able to tackle with various scenarios, even when we cannot even consider them. It usually, in the world of robotics, comes down to creating motion planners, or even path planners, that will provide better outcome, depending on the problem.

Motion planners are general purpose and offer a wider scope to providing our robots with solutions, even when they are challenged with extreme situations (Table 1), but that depends on the programming of the controller. It is obvious that motion planners cannot be easily created for every problem and even if they require a lot of testing throughout experiments, that will push them to their limits and therefore provide us with feedback concerning their problem-solving ability. Various tests are then performed and scenarios that offer the planners the chance to calculate complex solutions and provide us with a chain of commands (or motions, as it is usually referred to) that will make up the solution. We should always keep in mind that this type of approach is always guaranteed to offer better quality of results, but that in fact depends on the use cases (Table 2).

Table 1: Advantages comparison between simple and planning-based controllers

SIMPLE CONTROLLERS	PLANNERS
Faster implementation	Greater dexterity based on implementation
Cover basic required functionality	Covers fully required functionality
Easily transfer between static and dynamic environments when feedback is offered, usually through sensors	Offer better understanding of the environment, the obstacles present and the required goals to achieve

No delay at execution	Easily scalable to cover more scenarios

Table 2: Disadvantages comparison between simple and planning-based controllers

SIMPLE CONTROLLERS	PLANNERS
Limited problem-solving capabilities	Often there is delay before execution
Difficult to scale up	Time-consuming implementation
Requires programming down to the last detail	Requires vast experimenting and case scenarios
	More difficult to traverse to dynamic environments

3.2. **Algorithm solution space exploration**

Since the beginning of planners, algorithms have been at the heart of their implementation, offering and thus controlling the flow and use of information available, along with deciding the best course of action given any available context. Nowadays, algorithms have evolved and reached a level where we have a plethora of options from which we need to choose to be able to best adapt to any given problem. In this part, we will be discussing the process of different algorithms, based on their implementation.

In general, in planning we have two distinct types of algorithms; the first type uses information available and tries to find the best available route of motions that fulfill its requirements, in this case this is a direct synonym to finding the set of actions that will lead a robotic system from a starting state towards reaching an end state, defined as goal state. To do so, the algorithm will need to seek out the best available option in every repetition of its steps, which eventually leads to exploring the whole available configuration space. The other category is based on algorithms that feature a more selective process of goal reach algorithm execution, where they need to extract a sample of the available configuration space. In this way, speed is increased dramatically, which usually comes at the cost of not finding the optimal solution that will eventually lead to the goal (Table 4). However, this approach offers other advantages, that vary upon implementation and algorithm selection along with faced problem situation, such as more efficient use of resources, lower memory consumption, or even fewer state search, minimizing the required time of implementation (Table 3).

Table 3: Advantages comparison between complete and sampling-based algorithms

SAMPLE-BASED	COMPLETE
Smaller resource usage	Fully utilize a system's resources
Speed of execution	Completeness of configuration space search
Offer a satisfactory solution	Offer the optimal solution
	Answers the question about whether there is a solution

Table 4: Disadvantages comparison between complete and sampling-based algorithms

SAMPLE-BASED	COMPLETE
Solution quality varies from good to bad	Resource usage to the maximum
Usually does not fully exploit a system's capabilities	Takes considerable amount of time to execute
Cannot answer if a solution does not exist	Usually there is considerable delay till execution

3.3. **Algorithms presented**

In this dissertation, we focus on the presentation of complete algorithms and their implementation, namely A* and Dijkstra. In terms of sampling-based algorithms, we present RRT, which performs actions based on random steps. Before that, we present a list of algorithms that are quite commonly used based on their category in motion planning (Table 5) and provide some more context over their uses, basic principles, case studies, advantages and of course their disadvantages. Then we present and emphasize on the ones used and implemented in this dissertation, while providing more edible background on their implementation along with their application in robotics [11].

Table 5: Most common algorithms used for each category in motion planning

SAMPLE-BASED	COMPLETE
RRT	A* (A-star)
RRT*	Dijkstra
Probabilistic roadmaps	

Artificial potential fields	
Monte Carlo algorithm	

3.4. **Rapidly-exploring Random Trees (RRT)**

Based on sampling, randomly picking candidates for performing a next action, RRTs have widely been used in robotic motion planning [12]. RRT aims to explore the configuration space, regardless of the dimensionality. As the name implies, tree structure is maintained, aimed to offer great efficiency in solution finding.

Two main factors affect the algorithm's effectiveness and quality of solution. First, random-based sampling is limited by a maximum distance growth in the new region the branch extends to. In this way, we can avoid great differences in tree size which makes the process smoother in exploring new areas. More importantly, we need to clarify that a random step towards a feasible part of the configuration space is indeed chosen, however we use the maximum value to limit it and eventually use this value as a substitute for the originally chosen [13]. In different case scenarios, this might have different effects in performance of the algorithm depending on the problem. Usually, however, it enables us with a smoother progress of the algorithms state expanding tree, so we can avoid sudden differences between steps of execution. The second most important thing is that we set a probability for the algorithm to try between a certain number of attempts to reach for the goal state itself. This has proved to be of significant interest, since depending on the progress that has been covered in exploring the configuration space and trying on different routes and approaches towards forming a solution, it can dramatically decrease the number of steps and sum of time needed to solve each problem. A mild drawback would be to state that this effort is irrelevant to the maximum distance step we have previously selected, otherwise we would end up with no difference whatsoever in the algorithm execution. Another key factor is the number of iterations we choose to perform before we decide to end algorithm execution. In more complex environments it is usually ignored in the essence of choosing a significantly substantial number that fits the problem description and will not be able to stop the algorithm mid-execution, before it reaches a considerably and workable solution, close the goal state.

Having highlighted the algorithm's trickiest and with the most impact in its performance aspects, we will now proceed with providing a step-by-step execution that the algorithm states as the process in solution finding. Initially, we assume a configuration space, a list of obstacles if there are any, the initial state and of course the maximum

allowed distance to cover in a single step. The algorithm starts by picking a random action of a given distance less than or equal to the maximum distance allowed and the proceeds with checking whether it is feasible, given the configuration space and the obstacles we might have included. If indeed we can perform this motion, then the algorithm tries to add this action to the nearest explored so far, maintaining the tree structure. The new motion is then added to the already explored states of the tree and becomes its latest addition to the branch it was added, becoming a leaf. The above process is continued for as many times as the number of iterations describes. Given that the algorithm was designed having efficiency in many aspects, including speed of execution, there might be different conditions to indicate its end. Some researchers might consider end the algorithm if there has been a solution found, or in terms of motion planning the given tree features a state that meets the goal state fully, or based on the problem needs. However, the general idea of the algorithm imposes that execution continues till the number of iterations is met, by adding more actions to the given tree and therefore improving the algorithms state. Below we will demonstrate the main aspects of the algorithm's philosophy and implementation (Figure 1), providing more insight to the ideas of utilizing the values we chose and discussed above that lie into the heart of the algorithm, driving its execution till the end. As a last reminder, we need to consider that this algorithm aims to provide faster solutions to problems (Figure 2) and emphasizes on efficiency over quality of solution. In this way, even after the algorithm ends its execution, we cannot in any way claim that it has found the optimal solution, but this of course has to do with the dynamics of each problem at hand, as it is the case in many planning problems.

Path Planning with RRT

RRT Algorithm

```
g = Initialize algorithm
for  i = 1 to K
     next = random_move()
     g.add(next)

return solution
```

Figure 1: RRT pseudocode

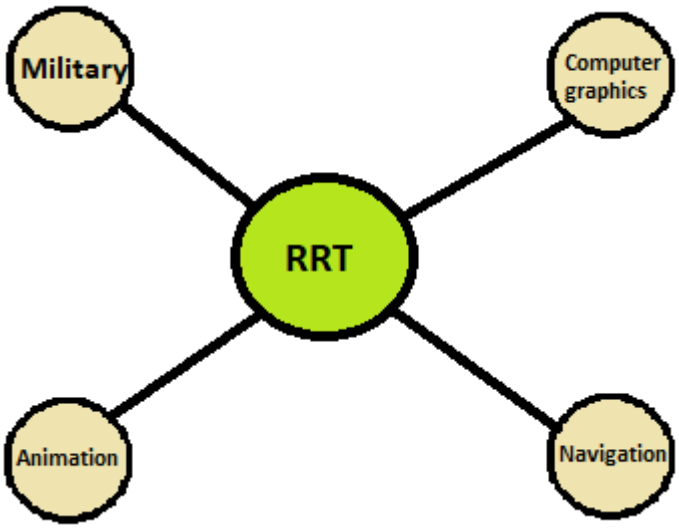


Figure 2: Some of the use cases of RRT

3.5. Rapidly-exploring Random Trees * (RRT-star)

Using the RRT algorithm for finding solutions in planning problems had massive impact to scientific community, with many researchers implementing new patents based on the existing philosophy of simplicity and resource efficiency. One of the most commonly known variations, which is a more generic term for referring to providing a more guided way of searching throughout the configuration space and find a solution for the problem at hand, is that of RRT*, originally introduced by Dr. Karaman and Dr. Frazzoli [13]. The algorithm still fits the same criteria as the original RRT; however, it now features mechanism that enables the algorithm to converge towards an optimal solution while exploring. Below we will see some other variations to RRT and some of their use case;

1. RRT* FND - extension to RRT* for dynamic environments [14]
2. CERRT - variation of RRT to include uncertainty [15]
3. TB-RRT - variation of RRT to include time in constraints [16]
4. RRT*-AR - variation of RRT* dealing with alternate routes [17]
5. RRG - (Rapidly-exploring Random Graph) variation of RRT for optimal solution convergence [18]

As we can see from the list above, RRT was able to introduce the importance of simplicity in planning, since it is well-known as impossible to understand and include all the parameters in a problem solver, which is the reason RRT is usually described and referred to as naive implementation of a motion planner. However, the actual impact of RRT and its contribution to the scientific society is recognized by the number of variations

and extensions that were based on RRT as shown above. RRT* is one of the most commonly known and used, so we will be taking a closer look at its implementation, along with the benefits it offers in contrast with the algorithm mentioned above, namely RRT and some of their differences and similarities.

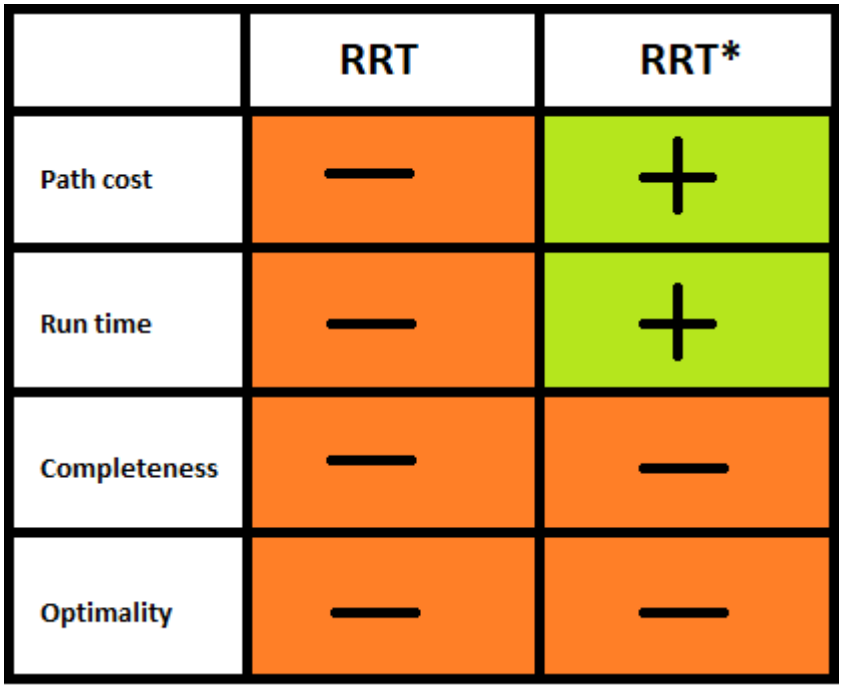
Quite obviously the algorithm's implementation that utilizes tree structure has not been affected, to be able to hold onto the benefits in performance we get by adopting a similar structure. For simplicity of this demonstration, we will be considering a 2D graph-based implementation of RRT*, to be able to highlight its key features and the ones that distinct it from its predecessor. For this reason, every time we refer to the word vertex, we will be now referring to what we called motion or action previously. The algorithm input and output are the same in terms of objectively seeking a route that will lead us from a starting state to a goal state however it is described as. Now, we will look at the actual implementation differences between RRT and RRT*.

Two main aspects of RRT* make it different from what we already described, know, and understand as RRT [19]. The first of them is the maintenance and utilization of a cost function that can provide us with the toll it takes for a new vertex to be added to our tree. In RRT, we only consider which is the closest vertex to the new feasible one and then we add it to the tree. In RRT* we no longer do that, yet we do repeat this process in every addition. More specifically, we tend to use the cost function to be able to tell between several vertices in each radius, that will give us a few neighboring vertices when we add it to the tree. In this way, if we compute that the newly added vertex should instead be added as leaf to a different branch, since we can reach this state having smaller cost than reaching it through the pre-determined branch, as a leaf to a vertex stated by RRT. The obvious gain we get by performing the above is a reduction in terms of cost-related affects to the result. In terms of quality, in other words, we get a better fitting for reaching a new state, however in real terms we still aim and reach for the same state. An obvious drawback that someone could conclude to is that this restricts searching towards more promising areas in the configuration space, which makes the search biased. Some people might consider that this is an improvement and in terms of quality of solutions it indeed provides cheaper paths to a new state, however it also tends to steer the search toward certain paths which in some cases could lead to ignorance of important paths that might demonstrate dramatic improvement as the algorithm continues its execution. Thus, the main purpose of RRT* is not to provide the shortest available path, but instead give us better quality paths to choose from.

The second significant difference between RRT and RRT* is what is mentioned in literature as “re-wiring” a vertex. In fact, it describes the process of having another run of the algorithm’s explored states to find out if the newly added vertex has been added to the best, again in terms of cost however it is measured, vertex. To make things clearer, this aims to provide us with a smoother path instead of simply providing the path. Further elaboration on this topic would suggest that re-examining the whole tree would be careless and resource-consuming, so a key part in this process is the fact that instead of running again through the whole tree, we focus on the neighbors of the vertex, as they were discovered in the previous step. Again, we need to underline the fact that this does not have a direct effect in the vertex itself in terms of changing each core values, or performing any sort of operations that will make it reach closer to the goal state. It simply describes a process that will genuinely lead to a shorter path in contrast with the one suggested by RRT and when this happens it will increase the smoothness of this traversing from one vertex to the other.

In contrast with the benefits of providing a more straight-forward solution to a problem, that will avoid unnecessary steps as much as possible, RRT* had to undergo serious performance decreases when operating against obstacles. Main reason the above is taking place has been the cost of re-searching the neighboring vertices where we can add a vertex. This affects the outcome when obstacle avoidance is added, especially because of the additional calculations that need to be performed to check whether, or not the path connecting the new vertex to any of the old ones do not traverse through obstacles [20]. Below we will show a brief overview of differences in performance aspects between RRT and RRT* (Table 6). In general, we can see that the cost to reach the goal state is obviously decreased compared to the original cost we get by running RRT, however the execution time increases dramatically. The number of nodes, or vertices, in each tree suggests that RRT* was able to explore the same number of vertices as RRT.

Table 6: Some of the performance differences between RRT and RRT*, however as we can see they both do not perform well in certain aspects



3.6. **Probabilistic Roadmaps (PRM)**

Continuing with the next sampling-based algorithm, Probabilistic Roadmaps, or PRM [21], are widely known and used in modern day robotics, due to their efficiency in path planning. Like we saw previously with RRT, this is another algorithm that belongs to the family of randomized planners, aiming to provide solutions within given time limits, without going over the whole configuration space. In other words, this algorithm avoids exhaustive search and proceeds with sampling over time, which in theory will return a solution if it exists.

First, we need to clarify that this algorithm can be fully used in circumstances where a robotic system is tasked with finding a path, or set of actions to reach a goal state, despite any given obstacles. More specifically, the algorithm will indeed perform a random move, but before adding it to the chain of actions it will check if it is feasible. It is designed to answer queries regarding graphs, which implies that before we proceed with applying the algorithm's principles, we need to provide it with an initial state, along with some sort of discretion of the configuration space.

Again, there are many different approaches for implementing this algorithm, especially due to the variety in problems that robotics researchers are facing nowadays. However, three are the main aspects of the algorithm that can be parameterized and an affect both the quality and efficiency during execution. The first one is regarding the presence of obstacles. Admittedly, it will make the process of discovering a solution more

computationally and time consuming, but it will also add some more calculations to ensure the algorithm's stability. For this reason, it is crucial that we provide the algorithm with a good distance calculating method, e.g., the Euclidean distance appears to be the most used, so we need to pick one based on the problem at hand. As mentioned before, when dealing with coordinate systems, Euclidean distance can prove to be very efficient, however we should keep in mind that the essence of distance in this context exists to describe the distance from a given state to another, so depending on the robotic system we use this might refer to more than just coordinates, increasing in complexity along with dimensionality. Another particularly important variable we need to estimate is the number of vertices (again we assume that our problem is based on graph theory, due to the nature of this algorithm's use) to consider that we have fully explored our configuration space. This number should not be taken lightly, as a higher number of vertices might be able to provide us with a more accurate solution, however it will cause the algorithm to become unresponsive in terms of delivering solutions that are not bound by time increases in search due to overly complex environments. To be able to find the proper number of vertices we need to have a proper understanding of the problem we are solving. Finally, as in RRT, we need to connect a newly found vertex into the existing graph, which means that we need to use the distance metric to estimate up to *k* different vertices that are supposed to be the closest to the new vertex. In this way, the algorithm can provide a smoother transition between different states and motions. Also, we minimize the probability that the algorithm will hit an obstacle along the way as it connects the two vertices, given the fact that obstacle collision has been used before.

Let us now take a more in-depth look of the algorithm and explore the drawback, along with some quite common difficulties one might face when tasked with implementing PRM. At first, the parameters discussed before need to be rightly calibrated, otherwise we might end up with a planner that will provide no solutions whatsoever. This requires in most cases a lot of experimentation and research, categorized based on the problem that one is facing. Also, when we choose some random point from our configuration we need to remember two things; the first is that we need to choose a point from the space provided that all points stand equal chances which will minimize the bias toward certain given directions and we also need to take into account that points, or more likely whole areas around objects that have been set to represent obstacles are more difficult to be chosen either due to poor sampling, or rounding performed to decrease computational complexity. In general, we should also consider using different approaches based on problem complexity, given the fact that this algorithm will not perform well in crowded spaces.

Again, we point out the general efficiency it provides, but let us not be taken away and forget that experimentation is required. In this way, randomized sampling might become slightly different than what we proposed above. Instead, it should be more possible for the algorithm to pick random points in parts of the configuration space not covered by obstacles. At this point it extremely helpful to also consider splitting the whole space into distinct parts. We should also notice that this is an offline algorithm, since it requires absolute knowledge of the space it operates in and hence, it cannot be used for online problems, at least not without changing the setup in terms of providing the algorithm with new input to be able to cover for missing information, or intel that has changed in regards to position, orientation, shape or size of objects, which is usually referred to as the geometry of the configuration space, making this algorithm require greater amount of information when performing in real-time. We will now proceed with explaining the algorithm's process of execution (Figure 3), along with some of the parts we need to give more insight to regarding the algorithm's inner functionality.

```
PRM algorithm pseudocode

V = Empty
E = Empty
k = the number of closest neighbors

while |V| < n do
  do
    q = pick random move
  while (q under collision)
  V.add(q)
end
for all q in V
  N = find k close to q
  for all q' in N
    if q' not in E then
      E.add(q,q')
    end
  end
end
```

Figure 3: Pseudocode for PRM

At first, we provide the algorithm with an initial graph. Remember that we need to provide the geometry and positions of the obstacles as well if we have any, since the algorithm requires total knowledge of the configuration space. Of course, we also need to provide it with an initial and a goal state, so that the algorithm can understand when the search should be terminated, if we decide to add these parameters to the search performed. All the above are in par with the various setups of the algorithm and its configuration we decide pre-execution. At this point it is common that the graph we provide is considered empty in terms of not having any actual vertices that can lead from the initial state to the end state. We then proceed with picking a completely random vertex that has not been explored yet and check to see if it is indeed collision-free. This is the part where our distance metric begins to be utilized. We repeat the above until we find a non-colliding vertex that we can add to our graph. When we do find it, we proceed estimating its *k* nearest neighbors, as we suggested above. Again, we need to use our distance metric for that usually along with a planner. When this part is also over, we can then repeat the process of systematically picking random vertices till we meet the number of vertices we set during initialization of the algorithm. The algorithm will terminate execution either when it explores the number of vertices we decided, as the general implementation of it implies. However, it is up to us to decide that by altering this condition and check if in a way we have reached our goal state, depending on the problem configuration we chose as it is mentioned above, providing a little more guidance to the algorithm.

3.7. **Artificial Potential Fields (APF)**

When it comes to robotics, most of the times we are tasked with solving problems that have their roots into modern day situations, as in autonomous driving and the automation of processes that we face in our lives. As mentioned, a vast range of these problems falls under the category of mobile vehicles and our effort towards making them fully autonomous.

Artificial Potential Fields [22] help us tackle similar case scenarios, where we cannot afford the time or resource implications of fully exploring our configuration space. When this is the case, we are forced to search for new methods and strategies that provide us with decent quality solutions in a more efficient way, whether this requires fastest execution, or computationally restricted planning. In most cases, we need to solve a problem using an online strategy, as in the case of Artificial Potential Fields. An online algorithm, for clarity, will provide us with a plan that improves and changes in real time.

What is more, in many cases, depending the implementation it might help us with more efficient information storing.

What is described above is usually referred to in scientific literature as “online planning”, as a response to more advanced and complex configuration spaces and environments in general. When we implement an online planner, we often use partial environment knowledge, in regards to not exploring the whole configuration space and considering it in motion choice selection. This improves speed of execution, but might also come with the drawback of incomplete knowledge that will give inferior quality range of motions, which in the end might instead turn to greater time consumption which defies the very reason it was used and chosen for in the first place. As the algorithm proceeds, our system will collect more information, which can be stored as a map to increase efficiency in exploration, comparison, and selection. Instead of searching through all our known states we can efficiently iterate through the closest regions to what we are looking for in our planners. Furthermore, in contrast with what we just presented, we might consider using a stateless planner, in terms of not keeping any states in memory. More specifically, our planners will be able to quickly respond to given situations and provide us with actions, however this presents the drawback of completely ignoring the given states, which might lead to difficult to break loops and again result to more time wasted than saved, as it was originally intended. The previous includes sensor readings, which implies that we will need to process information gathered during execution and even respond to dynamic environments, or even unknown. Thus, there is no placebo when it comes to planners, especially when our problem includes real-time navigation and decision making. We need to consider, study, and understand our environment and carefully choose the planner that will be able to properly respond to our problems. It is also a combination of computational resources availability and sensor readings quality and access to, since we should not forget that there is always noise that might and will affect the outcome, including the accuracy of our computations, which could vary from no nuisance in our readings to completely mistaken conclusions due to incorrect, or incomplete information and thus, it is also a matter affected highly by the available hardware.

The main idea of Artificial Potential Fields is to provide the robot with more promising regions in terms of successfully reaching the goal state and, provide it with areas that should be avoided as they represent obstacles. In this way, we use the terms *affected* and *repelled* by given areas of our configuration space, which in our case are called as *fields* (Figure 4). If we want to refer to the general dynamic of the system, we refer to its *U*, which represents the general part of the configuration space that is feasibly traversable,

in terms of being both accessible and collision-free with obstacles, in other words we use a metric to answer the question of whether a point in our configuration space can be reached without violating any obstacle-avoidance constraints. In each execution of the algorithm the gradient of our feasible areas, after subtracting those that are occupied by obstacles in negated and represents the most promising course of direction for our algorithm to choose from. To describe the general potential to attract the robot towards a field, we can consider it as either a conical, or a parabolical function. The later results in more distance-aware approach that enables us to take more precisely into account the metric we use to discriminate between different states and their distance from the goal state. When we use a parabolical approach, in contrast with a more informed process of search, we are also burdened by what could appear as a more biased strategy of searching our space, since the affective power of each field tends to increase in our algorithm, so this leads to a more directed search, which in some cases might prove to be quite useful for the algorithm, but in most cases in deprives our planner from many promising areas of the configuration space that might be overseen. On the other hand, when using a conical approach, we have the advantage of a more balanced attractive power over our robot, since it increases in a constant manner, saving us from ignoring states and parts in our space. More frequently, when using parabolical computations we are granted more accurate results when in need of more accurate results, but it will at the cost of biasing them after a while. For this reason, it is quite common to combine both techniques to in the following manner; we tend to adopt a conical approach for determining the repulsive power of any field over our robot and tend to use parabolical computations for estimating the attractiveness of a field in terms of reaching towards our goal. To further elaborate on the above, we achieve a more informed idea of understanding the quality of a next chosen action, over the reason it should not be selected, which again is highly affected by some good metric, e.g., Euclidean distance. As it has been mentioned before, metrics will have the most crucial impact on our algorithms in general, since they provide the most useful parts of information that our algorithms will use. When we compute the same attraction or repulsion effects for our obstacles, we tend to do this by providing surrounding boundaries for each obstacle, to be able to describe its effects in our robot's course. In other words, we tend to reversibly increase the likeness of a field that contains an obstacle by dividing by its distance to our obstacle. Again, it is important to choose an incredibly good metric, that will fit for our problem at hand. Without stepping into too many details, the algorithm is designed in a fashion that provides a range away from which the algorithm does not consider the obstacle any more favorable to traverse to. In this way the repulsive forces are

increased and tend to reach infinity when a point is on the verge of colliding with an obstacle's boundaries, in contrast with attractive forces that are decreased to 0. The algorithm can now secure a collision-free choice of motion.

At this point, let us recap the most important pieces in the algorithm's process; first, it is used for online planning, but this in no way means that it cannot be used for offline planning, since we can very well provide the algorithm with a discretized version of our configuration space. Second, we can utilize our robots to provide actual feedback that we can use to estimate our movements, avoiding already explored states without necessary storing them. Finally, this is an incomplete algorithm, which means that the solution we get is most probably not optimal, but we should also remember that being a sampling algorithm, we need to consider checking for local minima before implementing its plans, which sometimes includes the addition of best-first implementation when trapped.

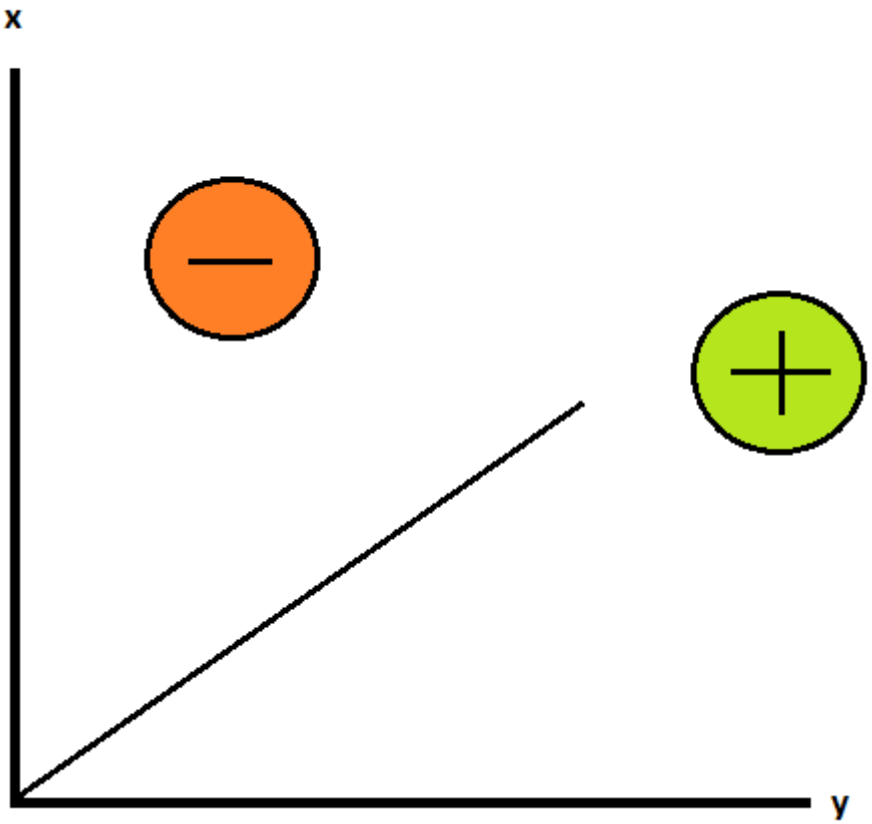


Figure 4: Sample of a step in APF execution showing how the algorithm draws the robot away from obstacles (the red circle) and closer to an un-explored area (the green circle)

3.8. **Monte Carlo Algorithm**

Another example of random-based algorithms is Monte Carlo [23]. Originating at a same name site, birthplace to gambling and very frequent example of casino, this algorithm is based on statistics in order to provide us with a point in the configuration space that will not be colliding with any obstacles. It generically refers to a family of randomized algorithms, where the probability to select from a range of options, varies according to a given statistical function, with randomness being fair in terms of chance, stacking up when it comes to certain scenarios, which differs upon implementation.

As we mentioned earlier, this type of algorithms cannot guarantee that we will be granted the optimal solution in a problem, provided there actually is one. However, for this case we should also point out the fact that between consecutive runs of the algorithm, we might get completely different answers, whether it be the quality, or even the existence of a path towards our goal [24]. This is caused by the randomized process that lies into the heart of the algorithm and is considered crucial during execution. When we complete an execution of such algorithm, it is quite frequent that we also include a test run to prove whether this algorithm is truly randomized, in terms of equally allocating chances of appearance of all outcomes between the runs. At this point let us clear that in computational theory it is considered that algorithms that belong to this family will converge to finding a solution if we repeat the process *k* number of times, with *k* approaching infinity.

One of the drawbacks that many can encounter when are first tasked with producing results using these algorithms is the fact that in many times, they tend to provide us with biased outcome due to the widely-known difficulty of providing truly random decisions, hence the tests are need as described above. Usually, what one would expect is the strong attitude towards strict feasibility checks and constraint respecting when it comes to making decisions despite the problem and its nature. As we previously mentioned this algorithm is not guaranteed to even provide us with the same chain of actions between consecutive runs, which implies the freedom of execution and randomness in the algorithm's nature. What is more, most of the times this is a pursuit goal, however we should keep in mind that it in no way removes the chance that our algorithm might end up as biased.

There are two different cases, that in the most general cases might occur when implementing such algorithms; the first one is called as *true-biased* and *false-biased*, with both implying the correctness of the provided answers to each problem as responses to general queries. The first case tends to occur in cases where the algorithm claims the righteousness of its answer and it in fact turns out to be correct. While this might sound as an ideal scenario, we should consider what happens when we know and can distinct right

from wrong, in a fashion that enables us to claim that a fact is right via our algorithm's estimations and it is true, however we cannot really say something similar for all these cases where our algorithm guesses wrong. As it has been silently implied, the other case we might encounter during implementation, revolves around the probability that the algorithm can effectively recognize an outcome or situation as mistaken, however we cannot do the same when it comes to acknowledging the validity of a given context. Whereas it is quite common to meet the above conditions in actual problem implementations as we discussed above, we should at this point underline the strong possibility that we encounter both cases. When faced with such situations, it is wise to remember that we can reduce our falsely categorized decisions by incrementing the number of times we run the algorithm, which as said earlier might be able to provide us with better quality and more trustworthy results, since this type of algorithm has a higher success ratio as it approaches an infinite number of tries. On the contrary, given that this is a randomized algorithm, one might argue that such actions work toward increasing the time spent while processing the running environment and might lead to tough time handle and intense resource consumption.

Many different applications of this algorithm tend to provide a vast and chaotic context that behaves differently depending the problem it is tasked with solving (Table 7). For example, when in tight operating spaces and within exceedingly small margin of error, it would not be advisable to use and exploit such an algorithm's traits, meaning that it might be more suited to provide a solution after even an increased period that will, however, give an answer to our problem at hand. When we adopt similar case approaches, we aim to provide efficient good in general decisions in a brief period, especially since this algorithm is usually executed in polynomial time. Remember that we can always change the setup, a couple of parameters or even the general context and provide different, both in terms of quality and variety, series of answers.

At this point, it is a demand that we present an overview of the algorithm's execution process, such that we provide a skeleton that can be used in problem solving. Previously we referred to this algorithm as a generic algorithm that in theory can make right decisions in a brief period. That said the algorithm can be implemented in many different cases, but let us take a closer view to its actual implementation when it comes to planning problems, given the nature of our study. At first, as in many other algorithms, for the most cases that includes all the offline algorithms and some of the online planners, we need to provide the algorithm with a fully described configuration space, which our algorithm can sample from. Of course, this involves all the areas that can act as obstacles,

which should be considered since it is vital that the algorithm is provided with as accurate as representations of its environment. Next, we need to generate random sequence of points in this configuration space, favorably equally scattered as options in the decision-making process of execution. At this point we should mention that we use a probability distribution to provide these points, which can dramatically affect the outcome. In most cases and driven by the motive that we seek equal chances of picking between our options, we choose to use a normal, or Gaussian distribution, pursuing true randomness. Depending on the problem, we might want to boost certain areas of our search space, so in some cases we might even consider using a different distribution, which results to the fact that it is not mandatory. In contrast, it is one of the parameters that we can change to directly affect the outcome, since the algorithm's nature is based on statistics. As the follow-up action we process the results to determine whether they obey our constraints, given that we provided the problem with any constraints in the first place, storing or discarding the chosen action. Finally, we need to sum all the results in order to construct the final plan, which forms our solution. The final part sometimes might include more processing, since it might occur that we need to refine some of the motions to achieve our goal, but this is not an original part of the algorithm and thus it will not be discussed. In general, these are the steps that combined will give a solution to a wide range of problems, though we emphasized on planning problems, which we usually need to tackle with in robotics.

Table 7: Demonstration of some of the algorithm's use cases

Weather forecasting
Signal processing
Statistical hypothesis evaluation
Commercial video game character logic
Financial consulting
Better random number generation (RNG)
Light tracing in CAD software
Chemical reaction predictions

3.9. **A-star (A*)**

After our thorough presentation of algorithms that share the principle to not search the entire configuration space in pursuit of a solution to make sure they provide low cost in terms of computational power and effectiveness of execution we will now refer to the other great family of algorithms that in contrast with what we have seen so far debates on

providing with the best possible solution they can produce. For this part we will be presenting A* [25], which is one of the algorithms that has also been implemented for the purposes of this dissertation, an algorithm that is frequently used to address large spaces and complex environments through well-guided execution.

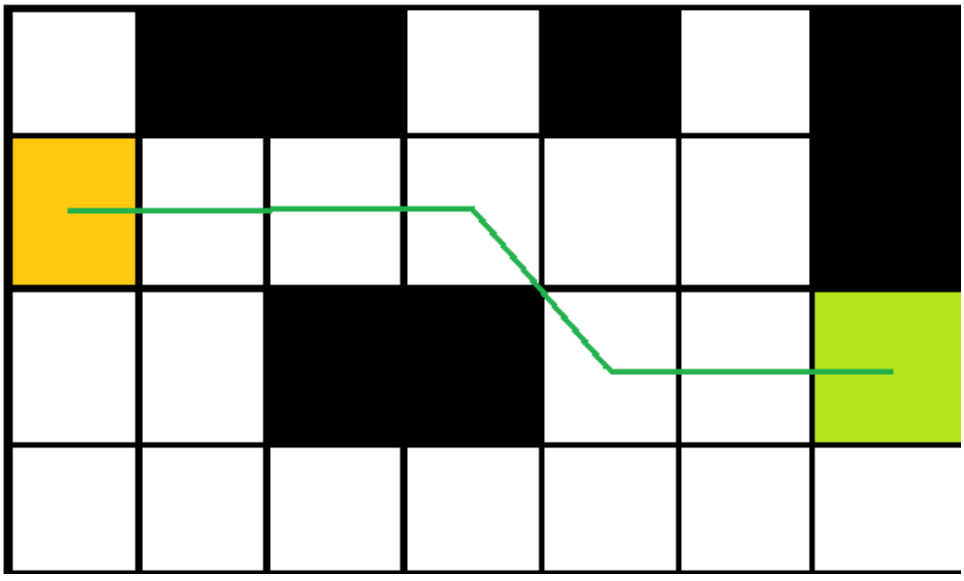
At first, let us discuss the differences that this approach makes us face and the many reasons why some might argue and debate on the effectiveness of a method. First, this algorithm is part of a variety of algorithms called complete, in terms of searching the entire configuration space in order to answer to our problems. An incredible first advantage that we can understand is the wholeness that these algorithms respond with, given enough time. In other words, provided that all paths and alternatives have been discovered and searched at their full extent, we can claim the full reassurance of our actions, in the sequence they are provided, since it means that there is no more guessing, or randomized selection. On the contrary, we have successfully searched throughout the configuration space and managed to create a plan that enables us to achieve our goals. What is more, adding a little bit more insight to what we already explained, one cannot but state another obvious advantage of the completeness with which we can answer whether a solution in fact exists. This is a profoundly significant issue and of the strongest arguments debating against the use of sampling-based algorithms. On the other hand, we should not forget that whereas this is one of the greatest advantages of these type of algorithms, we also need the necessary time and resources to decide that. More specifically, whenever we choose to execute such an algorithm, it will most probably guarantee, with a slight chance of doubt depending on any altercations we do to the algorithm to make it less resource consumptive, that if there is a solution given a problem setup, it is only expected that the algorithm will require to be given the necessary time to complete its execution searching amid the configuration space's corners. The disadvantage is that in order to do so, there are no cuts to its execution, meaning that it will give us an answer after it completes running, no matter the cost this might have. In this way, its worthy benefit becomes a burden, because depending on the problem at hand we might not be able to provide an answer in brief period. This is one of the reasons why sampling-based algorithms could be used instead since we intuitively know if a solution exists. This is not true for more complex environments, however, which also implies that given the structure of the space we might never get a solution using such an algorithm, which in no ways mean that it does not exist. Another great benefit of utilizing more systematic search algorithms is the guided structure we meet in its process. More importantly, these algorithms implement some logic which they apply into their search, which can save us time and resources (Figure 5). To elaborate more on the last, a

more guided search is not naive as the randomized algorithms are, but instead is targeted to reach for a solution exploring more of the configuration space based on the logic described for each algorithm. This provides us with more accurate and smooth results, which also gives us a better understanding over the course of actions that the path or solution consists of, which in the end results to better suited chains of commands, especially for planners.

We will now look at the algorithm's process, in order to gain more insight on its mechanics and the ways all these benefits that we talked about take place and affect its execution. First, during this presentation, let us state that this algorithm is used in graphs and path finding, with application to many different scientific fields, which make it a placebo in problem solving due to its optimality, as it offers us the best-case solution and efficiency through its guided search in the configuration space. The main process of the algorithm is as follows; first, we need to provide the algorithm with a list of options which it needs to explore later as it proceeds. These options include all the available actions that can be performed, in other words they must be feasible, which is usually checked by the algorithm. Second A* will try and expand these available actions based on their feasibility as we discussed before, providing us with their descendants to lead into the next set of available actions. Let us distinguish at this point that these actions and states are not discarded, instead we keep them in memory in case we happen to get across them in the future while exploring the configuration space to avoid re-processing these states. As time passes and the algorithm's tree structure, which is also one of its benefits given the efficiency in search time terms, is expanded by adding increased descendants, or leaves, we can already understand that one of the algorithm's main disadvantages is the necessity to keep all the explored states in memory to be able to bring them up in the future and use them. This also implies the need to be able to store them in an orderly fashion, which will make searching and iteration through them as efficient as possible. We should also keep in mind, that in order to choose make the next decision is part of the algorithm's search criterion which is able to guide the search towards the most promising direction. To be able to do that, we estimate the best next action, which is stored as the one with the lowest cost. This is the result of systematic search and application of the algorithm's heuristic function to be able to calculate the next successive move that will be as close to the goal state as possible, while maintaining the cost to the bare minimum [25]. This function is often described in scientific literature as $f(n) = g(n) + h(n)$; where the n represents the next node to be processed, while $f(n)$ refers to the quality of this node, which is the combination of the distance that has been covered from the start all the way to the next node, plus the

cost as it is usually referred to, which is coming out of the heuristic function we use and provides an estimate of the distance between the next node and the goal. The most critical part of the algorithm is its heuristic function, since the better equipped it is, the more accurate predictions it will make, given the fact that being an estimate means that depending on the problem it might refer to different uses. For example, when performing path planning, the Euclidean distance is quite often considered a useful metric to calculate an approach to the goal state, by providing us with the cost of the path that a direct line forms from the node to the goal. The algorithm will terminate after the goal state has been reached, or in case there are no other areas of the configuration space that have not been explore when it comes to planning, but in general it means that when we can no longer get any new descendants then the algorithm has reached a stopping point. That is the case that we described earlier and which will indicate that there is no solution for a problem, in case the algorithm terminates before it reaches the requested goal. However, let us for once again state the importance of choosing a proper metric to be able to rightly evaluate the cost. Different problems can and will require different metrics with which they can understand the best course of action in any given case.

Figure 5: A simple example of A* application on a grid-based problem (the yellow grid is the start and the green is the goal, whereas the dark green line represents the path)



3.10. **Dijkstra**

As we delve more into the depths of completeness of search, we notice that their benefits come along a series of issues which we need to keep in mind when implementing

these algorithms. Namely, the resource allocation necessity becomes even more crucial to attaining our goal and the speed of their execution sometimes cannot prove to be as important as it should be given the delay even for simple problems where a vast number of parameters are included. In this part we will be exploring another complete algorithm, the Dijkstra [26] algorithm.

This algorithm is used in graph theory and traversal to provide accurate paths with the minimum possible cost while navigating from one node to the next. As we further explain the process followed certain similarities will emerge and be discussed in a way that will justify what a great deal of the scientific society claims; that the algorithm we presented earlier, Dijkstra poses as a special case of A*, without using a cost evaluation function. At this point we would like to state that this algorithm is supposed to estimate the best in terms of cost efficiency route that will lead from one starting point, or node as in graph theory to a concluding or ending point, or node. Furthermore, it is an algorithm that requires certain information to be available, regarding the weights for each route, which need to be different than negative values. When such a scenario is encountered, we tend to adopt the use of different algorithms, such as the *Bellman-Ford* algorithm [27]. It is quite important, in order to understand the capacity of this algorithm's problem solving capabilities, to refer to the fact that this is the algorithm that provides us with the desired functionality concerning the use and effectiveness of everyday usage of the internet, since Dijkstra is used behind the logic of OSPF protocol, which stands for Open Shortest Path First and involves the execution of a functionality that enables us to direct the traffic between the different routes and nodes, in this case the nodes are the available gates to connect to and establish information exchange.

Dijkstra is a greedy algorithmic approach since it is bound to search for and provide us with the best available solution, or path. However, let us not forget that during execution and being part of its process, it is searching for the local available path and when added information makes it no longer the shortest path it tends to update and thus provides us with complete information to help us tackle with the problem's needs. Another important note to keep in mind is that this algorithm can distinct and work specifically for directed and non-directed graphs (Figure 6). When used in motion planning, we need to first discretize our actions and proceed with a certain modeling of the environment in which the algorithm will be asked to operate. What is more, as we mentioned before this algorithm among its execution keeps searching for a better fit solution till there are no more available answers or part of the space, or graph in general, to search in. In this case there might be some misunderstanding over the greedy nature of the algorithm. More importantly, as we

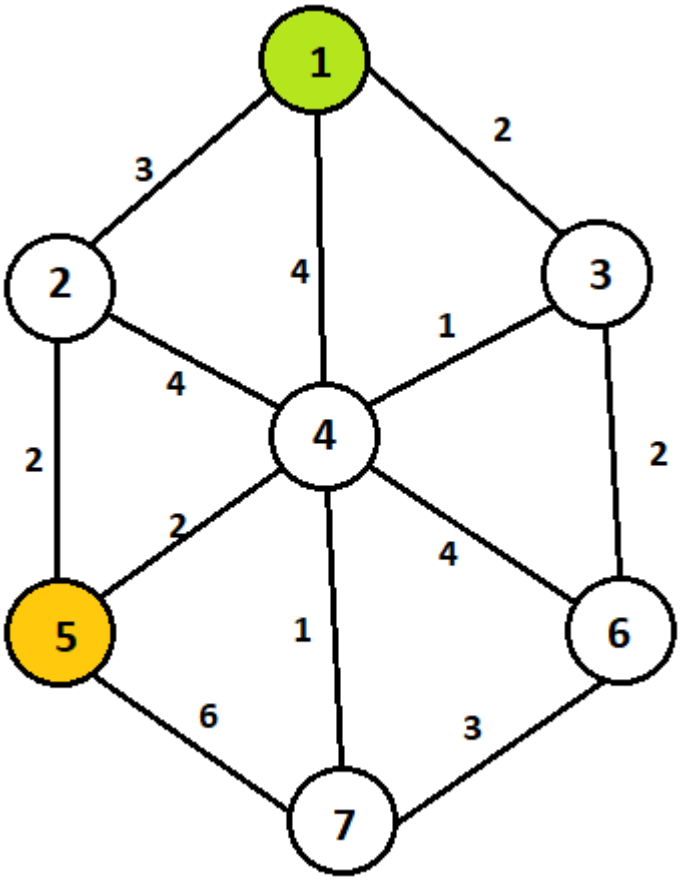
mentioned it will continue to search until it gets a more cost-efficient path leading to the goal node, if there is still available search space. As it occurs this is the terminating condition that will mark the algorithm as completed. When this happens, it is crucial to understand that the algorithm will produce the best and most appropriate in terms of cost path. In theory this alone should be enough to prove that the answer to a problem produced by Dijkstra is the best available, however we need to prove this, or more precisely explain it given the fact that the algorithm still explores locally when committing to a new path.

First, let us consider that we have a graph that only consists of one node. In this oversimplified case, it is obvious that the cost is the minimum available, which is 0 since the node has 0 units distance to itself. The main course through which we will try and evaluate the algorithm's efficiency is the fact that we will claim that for any given number of nodes we can calculate the shortest path and any other path therefore will have the same or even greater cost to be followed. As in many theories in the mathematical world, we will first claim that we already know that for k different nodes this algorithm has provided us with the best available course, with $k > 1$ for obvious reasons. To be able to claim that we are executing an algorithm that will result into the shortest path we only need to prove that by adding new nodes we still get the shortest route, in terms of the algorithm being able to identify them. For example, for $k + m$ number of nodes we should be able to still provide the best route. Let us examine the even simpler scenario where we only add one more node, namely we now have $k + 1$ nodes in our graph. Since the algorithm has already run, we have all the known nodes explored with the single exception of this newly added node. In this case, we will proceed by searching through our nodes and comparing the neighbors of each new vertex to decide whether it should update the shortest path table. In this case, when we add the new node, it makes sense that this node is already connected to some already explored node in our graph. Given this case we can assume that the previous node was connected to the recently added node since it had the shortest cost, which was estimated as the local optimum which in turn when increasing the algorithm's number of unexplored nodes tend to give us the explanation needed to back the claim about the shortest route it can provide us. Simply put, since the algorithm re-evaluates the paths in every new explored node that is added to the set of visited nodes, we can always progressively get the best available path.

Let us now describe the algorithm's execution, having proven its authenticity and effectiveness and hence its importance. First, as we already mentioned, we need to provide it with an empty set that will be storing all the explored nodes' previous nodes and the distance from each node to the next, which in the beginning is marked us infinite. We then

begin the process by setting the distance to the starting node as 0, which we described earlier. While we have not explored all available nodes, the execution will then seek the node that will be reached with the smallest available cost from the current node. In this way, we need to set mark the new node as explored and add it to the set of nodes we talked about earlier. The next step before we update our algorithm's status is to check the neighboring nodes of the newly picked node and see whether connecting them to the new node is more efficient than connecting it to the current node and if this is the case, we set that node as the middle node between the two and connect them as a sequence of shortest path algorithms considering the node weights for each one of the neighbors. This strategic extraction of nodes and setting them as explored will provide us with the shortest path possible when the execution terminates, in other words when we do not have any unvisited nodes. As it is obvious this algorithm will take a considerable amount of time before it is available to give us an answer and should be optimized through careful decisions and better-informed weight estimate when initializing the graph.

Figure 6: An example graph as input to the Dijkstra algorithm



3.11. **Storing information - RTree**

Before we proceed with offering our implementation of RRT, Dijkstra and A* we need to make a brief introduction to one of the concepts that was used and will be presented as well, at least for the case of A*. As we have described and it is very profound throughout the whole presentation of algorithms so far, it is crucial that we store the necessary information as to what states of the configuration space we have already discovered, which creates the need for an effective way to go back and of forth throughout our states for us to be able to determine the nodes that lie the closest to the next node processing. In fact, this is a need that will emerge when we need to search and iterate through 3D vectors that represent the coordinates that our robot has been to so far, as is the case in this dissertation. Like we mentioned above, we decided that as the search tree tends to expand and grow beyond our ability to effectively store and iterate through, the need has emerged and was covered by the presence of a data structure implementing an R-Tree.

R-Tree as a concept was discovered in 1984 by Antonin Guttman [28] and the main purpose it had to serve was to provide us with proper process of stored spatial information. More specifically, multi-dimensional indexes are used in order to describe and model various aspects in our lives, but for the most part and as we did in this dissertation, they are used to be able to give us a list of options answering the question regarding the closest points to a given location. While this is the main purpose, it is demanded to provide some more information as to the general concepts that there might be the need for them, such as navigation instructions, finding nearest neighbors when we need to deal with multiple dimensions and even filter through different provided shapes and structures that are used to mark areas in a geographical map.

When we decide to use an R-Tree we need to keep in mind the structure to be kept in order to represent all the available points and shapes. In the general case, every time we need to store information within an R-Tree it will be broken down to different segments of data. First, we need a unique identifier to be able to refer to each node within the tree. This is usually referred to as the *id* of the node, be it a leaf or a parent node. We then need to be able to define a minimum bounding rectangle, or MBR, which will represent the radius of this spatial data. At this point let us explain that when we want to define a single point, we can choose this course of action by ensuring that the point coordinates are the same marking the start and end of it. Another important thing is to note that we store every point with rectangle-shaped bounding box, hence the name. This might not be appropriate for every case, but it helps discretize the search space and makes it fair for each newly stored piece of information to be selected when compared to the rest. As it is implied, the more information we store the more accurate the representation will be, so we should keep in

mind that R-Tree data structures will deal with the efficiency in searching through the given states, however we will need to provide them with accurate information. What is more, when using R-Trees we need to familiarize ourselves with the idea that every node consists of other nodes, or objects, that in their turn also contain more objects. Furthermore, despite what the reader might conclude to with our previous statement, the expected height that each node is expected to have, is *log(n)* with *n* being the number of objects it contains. In this way and given the fact that R-Tree structure uses pagination to store the data, let us think of this particular concept; the MBR is used to refer to a rectangle that is bound to cover all of the objects that it consists of (Figure 7) and thus, it will then descend and search each one of the objects in order to more specifically answer the query which is usually a question similar to finding the closest points, or shapes, to given coordinates and if can find a more detailed closer object it will resort to the MBR we mentioned above. At this point let us describe the most asked questions among querying an R-Tree which will give us the number of operations and type of questions this form of data structure can answer. A basic question is whether an object provided belongs to a specified area. This is used in many applications of localization and can answer whether it is true based of inter-lapping MBRs. Another question we might need to answer is whether some object is part of an area, which is treated in an equivalent way as above. While we are describing these objects, we can also provide a list of all the objects that an area consists of, or in other words the rectangles that are present within an MBR. Another set of particularly useful operations we can perform regards to giving the answer as to whether certain objects are close enough to a certain other object, or all the objects that have distance minimum or equal to the one we provide. In our implementation of A* we have relied on the last operation in order to successfully decide whether an area has been explored before within a given margin of distance from this point, in order to avoid adding to the tree unnecessary information. Of course, some people might argue as to whether distance should be considered an acceptable metric, or we should specifically seek the exact coordinates in our configuration space's explored states. The answer depends on the problem we are dealing with and it should be in the middle; we need to reduce the acceptable distance radius as much as possible in order to provide more accurate answers, however we should also keep in mind that this might result into storing a lot of unnecessary information. As a final piece of required data, we should further elaborate as to why we use page-indexed search and the benefits from it. First, by using pages we can better organize our space, whether it is discrete or continuous, which also enables us to divide it based on the areas it consists of, or MBR of each object within it. This can lead to more accurate search and can

provide us with more effective solutions. Moreover, each page cannot contain more than a certain number of objects, named *M*, which is an indication that we need to split a page into two new ones. This is one of the parameters we control and should be carefully taken into consideration, as it has been proved that 30-40% fill is enough to give us the best theoretical performance. In general usage, R-Tree structure enables us to easily iterate through our entries effectively, however we should keep in mind that deletion can sometimes be tricky, in terms of coming together with the need to delete and update all the parent page MBR coordinates, which given the leaf depth might prove to be resource-consumptive.

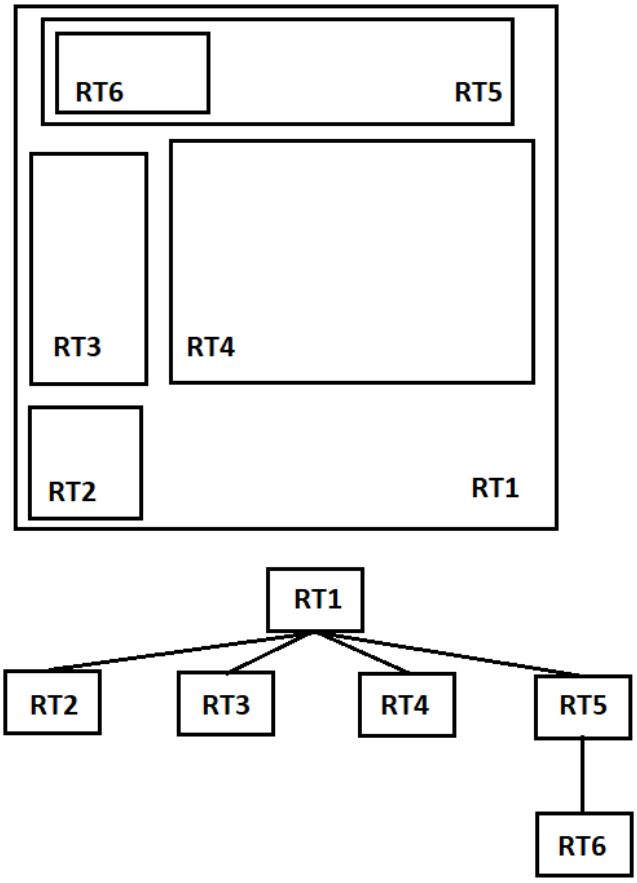


Figure 7: R-Tree storing process demonstration for a 2D example where we can see the indexes for each R-Tree stored object along with its MBR

3.12. Implementing RRT

Since we have finished with all the necessary theoretical background concerning the use and mechanisms of certain algorithms, we will now proceed to explaining our own implementation of some of the algorithms that have been mentioned and presented before. The first algorithm that was implemented is part of the random-based algorithms, namely

RRT, short for Rapidly-exploring Random Trees. The algorithm was implemented in a way that the actual input are these; the starting position, which is used to refer to the coordinates of the point of the claw used to grab each tile in our environment, the end position which is basically every position expressed in 3D coordinates of every given Jenga tile that we want to grasp with our robotic arm, the obstacles as a list of vectors with 3 values to represent the X, Y and Z coordinates of our configuration space and the number of iterations that the algorithm will run in order to simulate the similar parameter that exists in the actual RRT. We also can provide a radius which is the minimum distance that will determine whether two objects are colliding, especially important to determine the above and can directly affect the outcome of the algorithm. This parameter can make the execution slow, but much more accurate if set too low, or it can make the execution a lot faster and provide entirely inaccurate results compared to what we would expect. That is the reason we should keep in mind that it may vary among different problems depending on the environment the objects are inside. The following parameter to be set is called *stepSize* and refers to the incrementing step that will be used in order to make the algorithm increase faster the chosen values or in a slower pace that can be more precise for our computations. We can then also provide a certain number of the available different branches we can keep in terms of selecting entirely different, random-based movements. AS the last parameter we need to provide the threshold with which we calibrate the goal reach checks and more precisely the necessary distance the algorithm needs to see that the claw reaches the target before considering that the actual goal has been reached.

We begin executing the algorithm by setting all available branches to the start position of the claw and we also properly set the covered rotation and distance so far by initializing them. At this point we should clear that in the actual implementation we initialize the forward covered motion to 0 as expected, however we set the rotation that has been performed to 2.0 since we can in this way define any pre-existing rotation for our claw. This is practical and could be ignored provided that our experiments include that, or in other words if we indeed choose to not have initial rotation, we can do just that. The next step is to randomly choose the next motion for the algorithm which happens through randomly choosing one of the 4 available motions, move forward in a straight line, or backwards and the same holds for rotating clockwise and counter-clockwise. We also choose a random step to perform the motion with which ranges from a range of 0.0 to 1.0 multiplied with the *stepSize* that we mentioned earlier. The next necessary step is to check if the next motion chosen can in fact be performed since it is likely that it collides with any obstacles provided that the list with the available obstacles is not empty. The next part is

critical and requires further explanation. To perform the motion that is feasible, meaning that it does not collide with any obstacles otherwise the algorithm continues execution and goes straight to the next iteration, we need to simulate and, in a way, predict what the coordinates for the claw will be after performing this move. For this reason, we have implemented two distinct functions with just that specific behavior, which also justifies the reason we measure the overall covered rotation and distance, since all our algorithms are implemented in an offline manner, they will find a solution first if it exists and then execute it, meaning that we could not test for the actual coordinates so we had to simulate and make an estimate for them. There is no actual need for details, however it involves basic principles of geometrical shape transition and trigonometry. After we get the new coordinates, meaning the coordinates where we expect the robotic grasp to stand in the environment, we create a new state and update the specific branch. The next iteration will aim to pick the movement from this spot in the space and continue from there for each of the different branches. At this point let us explain that we use branch in order to describe the different motions linked in a path formation as alternatives to a common starting position. It is also picked us a name in order to resemble the tree expand behavior of RRT. After all the iterations have been completed, we will seek to get the one path that features the shortest available cost among those that managed to reach within a given radius close to the goal position. We will proceed with presenting the main process discussed above in a form of pseudocode. As the reader may notice for the sake of clearance and further comparison, we will provide both the original (Figure 8) and the implemented (Figure 9), for the purposes of this dissertation. The main purpose is to spot similarities and differences that were product of the modeling the problem had to undergo for the algorithm to be successfully implemented.

Figure 8: Original RRT

```
Algorithm RRT
    Input: Initial configuration Cinit,
           Desired number of nodes in RRT N,
           Maximum incremental distance Md
    Output: RRT form of graph to provide a path G

    G.initialize(Cinit)
    for n = 1 to N
        nRandom = random_move()
        nClose = nearest_neighbor(G, nRandom)
        nChosen = limit_distance(G, Md, nClose)
        G.add(nClose, nChosen)
    return G
```

Figure 9: Implemented RRT

```
Algorithm RRT
    Input: Initial configuration Cinit,
            Desired number of nodes in RRT N,
            Maximum incremental distance Md,
            List of obstacle coordinates Os,
            Radius for collision checks r,
            Number of branches B,
            Target coordinates,
            Minimum distance from target threshold
    Output: RRT graph G

for   j = 1 to B
    G(j).initialize(Cinit)

for n = 1 to N
    for   b = 1 to B
        do
            nRandom = random_move()
        while nRandom collides with any o in Os

        nClose = nearest_neighbor(G, nRandom)
        nChosen = limit_distance(G, Md, nClose)

        G(b).add(nClose, nChosen)

        if distance(nChosen, target) < threshold
            break

return G
```

3.13. **Implementing Dijkstra**

As we previously mentioned, as part of the research of this dissertation we implemented 3 different algorithms, giving more bias on complete algorithms. The next one that will be presented is the Dijkstra algorithm which will also set the basis for later discussing the implementation of A*. First, let us discuss the different parameters that we included in this case. As we previously mentioned we need to provide the algorithm we a starting position in order to know the coordinates of the point in our configuration space where the algorithm will begin its search. The coordinates are given as a list consisting of three different numbers that refer to the X, Y and Z coordinates respectively in the order they are provided. As a sidenote, in webots we consider X and Z for movement parallel to the level, whereas the Y coordinate is used to define the height of a particular point in

space. Moreover, in order to have a sound implementation and the actual termination condition we need to also provide the algorithm we the coordinates of the target's position, which as we have already mentioned is the positions of each tile, since the main problem we are trying to tackle is make our robotic arm move around and reach a tile, grab it, and then successfully place it to construct a Jenga tower. Like the need to provide a radius for obstacle tracing, we also give this opportunity in the implementation of Dijkstra as well in order to let the user decide how the algorithm should treat its obstacle. This means that when we try to avoid obstacles, we should keep in mind that we use Euclidean distance as a metric to decide the approximate distance between two points in our space. Using this strategy, we can simulate and avoid obstacles in our way and successfully reach the goal state provided that the route is feasible. As a last parameter we need to provide the algorithm with a threshold that is used to decide when a certain motion has been thoroughly examined. To further elaborate on the latter, based on the Dijkstra process we need at some point in the definition of this algorithm to decide to move on after marking a node as visited. In our implementation, in order to do that we decided to add some counter as to whether a motion based on one of the four available actions that include straight and back movement along with rotational and backwards rotation can no longer contribute towards creating a path. This requires more explanation and will be examined further at a later moment in this presentation as part of the presentation of the algorithm's main functionality and strategy in order to mimic the expected behavior.

First, the algorithm begins its execution by creating a single branch since Dijkstra does not feature multiple different paths. On the contrary the algorithm constantly works towards providing a certain path after exploring the whole graph. For this algorithm, another function was created that simulates the best available step for each motion at any given state during the algorithm's execution. We also set a dictionary that represents the cost of every available motion in each turn to determine the next shortest move and in order to avoid repetition we will be decreasing its values later. Second, we need to check if there are any available motions left so that the algorithm can continue its execution. This is a crucial step and is used in order to resemble Dijkstra's check in every loop of whether there are any nodes that have not been explored. Consequently, the problem is modeled in a way that every time we iterate through the algorithm, we consider every motion and the best possible step for it, which also checks for collision avoidance, as the actual nodes left to check. For the next move, we also consider the best available step which we calculated before and we proceed with decreasing the value of this move by one in the dictionary we mentioned earlier so that in the next loop the algorithm will only consider the rest of the

motions available so that we can avoid repetition as much as possible. At this point we need to state that we calculate the coordinates based on the type of motion it is we perform and its step using the same prediction functions as in RRT. WE should also clear that the algorithm updates the states of the branch object we created in the beginning, which again for Dijkstra is unique. Finally, we need to mention that in order to decide that the algorithm does not have any unexplored motions we will proceed with checking if the step every time we loop into the algorithm is lower than the threshold in order to instruct the algorithm that this motion has been thoroughly searched and therefore, we should proceed with removing it from our list of available motions and hence our unexplored nodes. It summarizes that if the best available step does not give us any actual improvement given the state the algorithm is in, we should no longer seek answers towards this direction and should consequently remove it as unnecessary. Quite similarly as we did before we will now present two different figures showing the original implementation of the Dijkstra algorithm (Figure 10) in contrast with the implementation that we presented (Figure 11).

Figure 10: Original Dijkstra

```
Algorithm Dijkstra(G, start):
    Q.initialize(all_nodes)

    for every node g in G
        distance[g] = +infinity
        previous[g] = empty
        add g to Q
    distance[start] = 0

    while Q is not empty
        t = min_distance(Q, distance)
        Q.remove(t)

        for every neighbor n of t
            newPath = distance[t] + length(t, n)
            if newPath < distance[n]
                distance[n] = newPath
                previous[n] = t

    return (distance, previous)
```

Figure 11: Implemented Dijkstra

```
Algorithm Dijkstra(G, start, obstacles, threshold):
    Q.initialize(all_motions)
    for every node g in G
        distance[g] = +infinity
        previous[g] = empty
        add g to Q

     distance[start] = 0

     while Q is not empty
        for every node n in Q
                best_step.add(compute_best_step(n))
        t = min_distance(Q, distance, obstacles, best_step)
        if step for motion < threshold
            Q.remove(t)

        for every neighbor n of t
            newPath = distance[t] + length(t, n)
            if newPath < distance[n]
                distance[n] = newPath
                previous[n] = t

     return (distance, previous)
```

3.14. **Implementing A***

The third and final algorithm we developed for this dissertation is the A* algorithm that has been previously presented. As we mentioned it is a complete algorithm, in terms of guaranteeing that it will find a solution provided it exists. Another trait of this algorithm is the feature of a heuristic function to navigate through the various available paths in a way that will always consider the most promising path, which sums the Euclidean distance that has been covered so far and the Euclidean distance after performing an action. We will now proceed with presenting the main context that adds up to the implementation along with some general principles that were followed in order to model the algorithm's most important aspects and map them to the problem at hand; providing a path for our robotic grasp to be able to reach the tile which as described above has always been our primary goal.

First, as we did with the previous two algorithms, we need to provide that starting position of the search as a list with a length of three in order to represent the three dimensions of our space. We also require the end position of the targets in each execution, which of course cannot but refer to each tile we are aiming to reach in an analogous way, a list that comprises of three different numbers referring to the coordinates of the tile in the environment. Next, we need to provide a list of similarly stored coordinates that refer to every obstacle we include in our environment. Of course, the list might indeed be empty and in this case, we do not need to care about the extra checks. Another thing we need to provide the algorithm is the radius that will help us determine whether our claw and one of the objects collide, in which case we cannot perform the motion we want. So far there is no actual difference in the parameters we provide compared to the two previous implementations of algorithms. However, this will change now regarding the two final parameters. We provide the algorithm with a minimum step to execute a motion along with a medium step to make this action, whether it be rotation or straight motion. These two extra parameters are one of the differences regarding the algorithm's available motions as we will see next. At this point let us also clear that in order to ensure better performance and increase the algorithm's efficiency we also decided to use R-Tree structure to store the visited nodes, which in this case it refers to parts in our environment. The benefits of the use of such data structure have been discussed above and for now we will just explain that it included speeding up various operations and greater efficiency when performing the most necessary operation for A*, the decision of whether a given state has been previously visited during execution.

Let us now provide the algorithm presentation and describe the way it was implemented. First, we initialize our R-Tree storage variable based on similarly named library we use and thus we will not be further discussing this since the implementation of R-Tree is handled by a third-party library in order to keep our states. We also need to begin keeping branches, objects of the same class mentioned above that will refer to the different available paths in order to keep in memory all our options as it is demanded by A*. Each branch contains series of states that make up for the path it consists of as it has been previously explained. The algorithm execution after we have properly initialized the above is straightforward, much as in the original algorithm implementation. Notice that in order to simulate the different states that the algorithm explores we need to initialize it with different motions for each branch. Next, we need to specify that since through experimentation and theoretical background we know that this algorithm will provide us with an answer we decided to use a different termination criterion. Instead of waiting until

we reach the goal and given that for every different tile and route needed to be performed, we cannot set a global minimum distance to be reached we decided to check all the available distances to the target in each loop and if for a specific number of iterations, we do not see any improvement then we will proceed with terminating the loop. In this way, we can gain from the algorithm's precision in execution. The follow-up thing to do is for every single branch we decide to add all the available motions taking into the account the minimum and medium values as well. Notably, we perform a simple check before adding the values of minimum and medium steps, meaning that if they are not feasible, we will override these passed parameter values. We then need to update the R-Tree with the newly calculated values and positions so that we can keep track of the visited coordinates in our space. At this point we are ready to pick the most promising motion to perform based on the data we have so far. For this to happen we also need to calculate first the Euclidean distance from the coordinates of each new candidate move to the target and then we add the distance that has been covered so far. As in A*, we will also pick the one with the smallest value and we will then try and find any nearby points in our R-Tree. If that is the case, we have a mechanism that is bound to unite the two motions in a continuous way in order to ensure that the transition is as smooth as possible. We should keep in mind that the comparison value is quite small in order to avoid having a dramatic difference among the states that are considered visited. Notice that we will simply add both motions as one in case of same movement instead of skipping it as it might have been expected. We then need to update the branch we currently are at and then proceed with the rest of the execution, until the algorithm terminates in the fashion that we described above. Let us now observe a simple generic comparison between the original (Figure 12) and the implemented version of A* (Figure 13).

Figure 12: Original A*

```
OPEN = all unexplored nodes

CLOSED = all explored nodes

CLOSED.add(start)

while OPEN is not empty
    t = min_f(OPEN)
    OPEN.remove(t)

    descendants = create list of t's next nodes to search

    for every n in descendants
        if n = goal
            terminate algorithm
        else
            n.g = t.g + distance(t, n)
            n.h = distance(n, goal)
            n.f = n.g + n.h

        for every node d in OPEN
            if d.f < n.f
                continue

        for every node d in CLOSED
            if d.f < n.f
                continue
            else
                OPEN.add(n)
    CLOSED.add(t)
```

Figure 13: Implemented A*

```
Algorithm A* (start)
OPEN = all unexplored available motions with minimum, medium and best steps

CLOSED = an R-Tree to hold spatial information of explored positions

CLOSED.add(start)

while OPEN is not empty

    t = min_f(OPEN)

    OPEN.remove(t)

    descendants = create list of t's next motions to perform with min,mid/best steps

    for every n in descendants
        if n = goal
            terminate algorithm
        else
            n.g = t.g + euclidean_distance(t, n)
            n.h = eucliedean_distance(n, goal)
            n.f = n.g + n.h
        for every node d in OPEN
            if d.f < n.f
                continue

        for every node d in CLOSED
            if d.f < n.f
                continue
            else
                OPEN.add(n)
    CLOSED.add(t)
```

4 Experiments

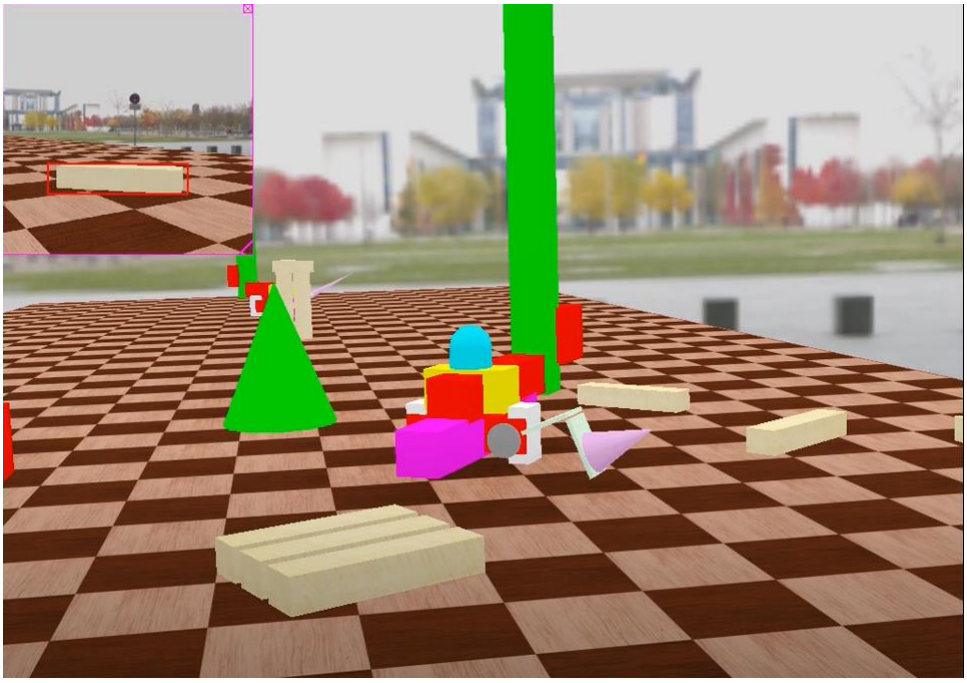
To be able to compare the algorithms we had to perform certain experiments, so that we could conclude to concrete statements as to which algorithm behaves the best. This comparison does only reflect the algorithms at hand, but is also a key part in understanding the reasons and main differences between the philosophies behind those algorithms.

4.1. Controller execution

In previous chapters we have discussed the use and effectiveness from various perspectives of the utilization of simple controllers. Before we delve more into the algorithm performance, we need to take a more in-depth look of plain controllers that are programmed to deal with a certain number of scenarios and we do not expect to use them in more complex and dynamic environments.

For this dissertation, we first created a controller written in python 3.8 and the simulations were run on Webots simulation software. The robot we created consists of 6 DOF and more specifically 2 rotary joints and 4 linear. It is designed to be able to rotate around an epicenter and as stated already our aim was to be able to give it the ability to construct a Jenga-like structure. For this reason and following similar-goal scientific literature, namely [6] and [2], we considered that modern practices insist on implementing robotic manipulators, heavily based on their ability to use visual recognition. Inspired by their various applications, our robot also features a sensor designed for providing visual feedback. More specifically, we enabled our robot to be aware of certain objects of interest, namely the tiles that will be used to create the Jenga structure (Figure 14). The camera sensor can detect those objects and distinct them from other objects that might get in our way. For instance, we decided to use geometrical-shaped obstacles that our robotic grasp needs to avoid in order to successfully reaching its destination. These obstacles are to be avoided in the simplest of scenarios, in the sense that when encountered the robot must not be directed to them. At this point, the camera sensor was also able to provide us with the relative distance to an object when it is detected and it is also capable of giving as its relative rotation as well, all in relation to the camera's transform.

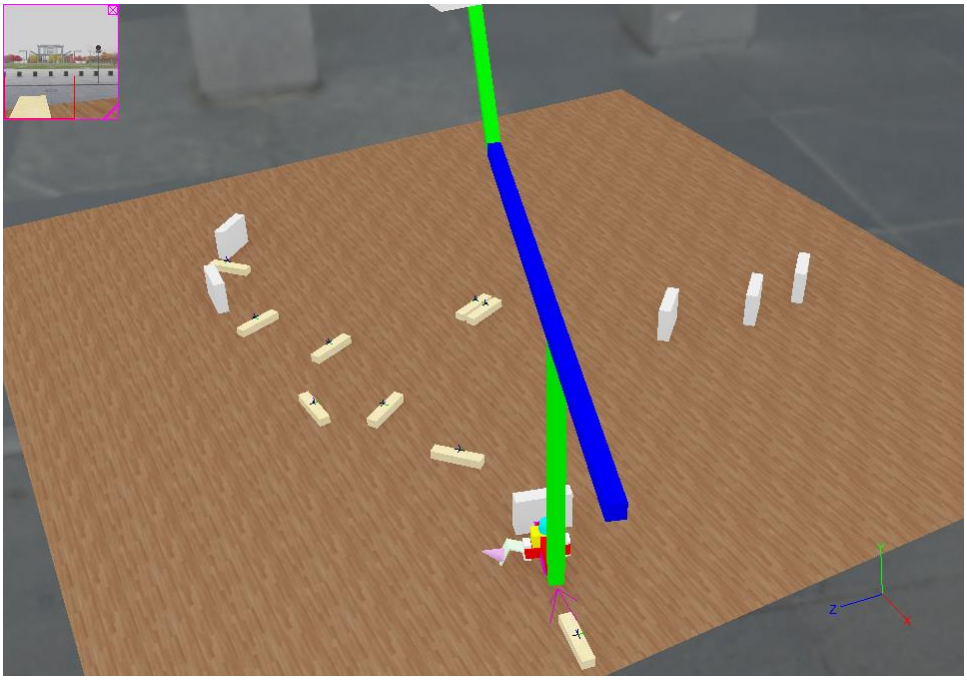
Figure 14: The camera sensor feedback (top-left corner) detecting a tile



4.2. **Sensor description**

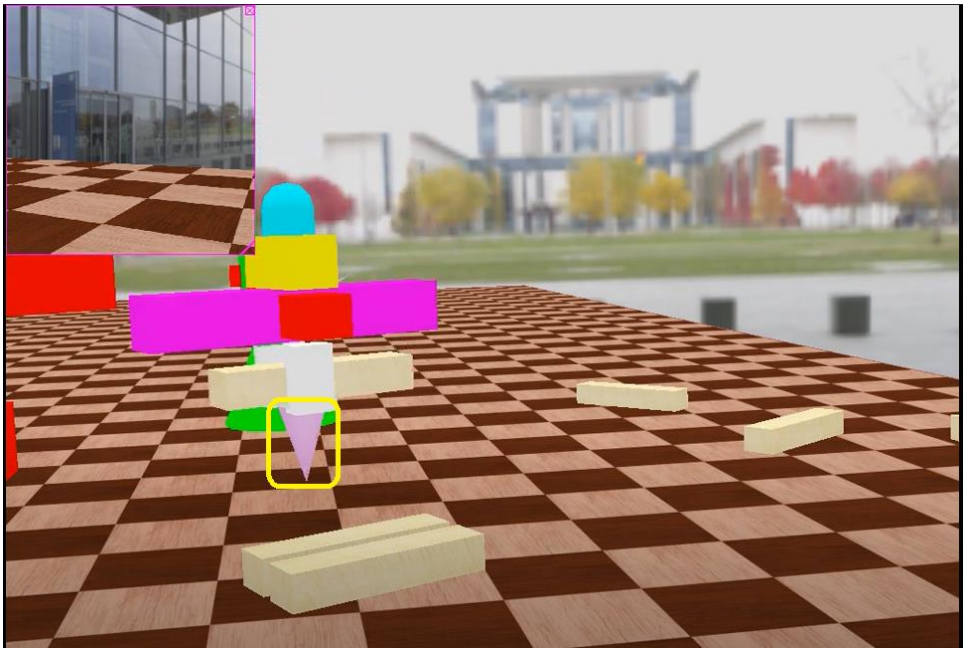
Taking visual servoing into consideration, the next step was to implement logic that would allow our robot to be able to take actions based on the visual feedback of the camera sensor. For this reason, we decided to add more obstacles, this time not just around the orbit of our robot, but also in its way to reaching the blocks, having to rely on the camera sensor to successfully detect and distinguish them from the tiles. We used grey rectangle-shaped obstacles for this case and the robot would have to perform certain chain of actions in order to avoid them and proceed with exploring the space around it, until it finds all the tiles, which always served as the final goal. As discussed, we exploit the environment' s built-in access to camera-gathered information, including the general transform of an object to successfully approach a tile, or evade an obstacle at a given time. When the latter needs to occur, we perform some basic movements that will allow the robotic arm to circle around the obstacle and then continue its course. At this point, let us state that we do not need to have previous knowledge of the obstacle positions, or the transform of the tiles. Instead, we study and extract this information through our camera sensor.

Figure 15: The robot reaching a tile after avoiding an obstacle



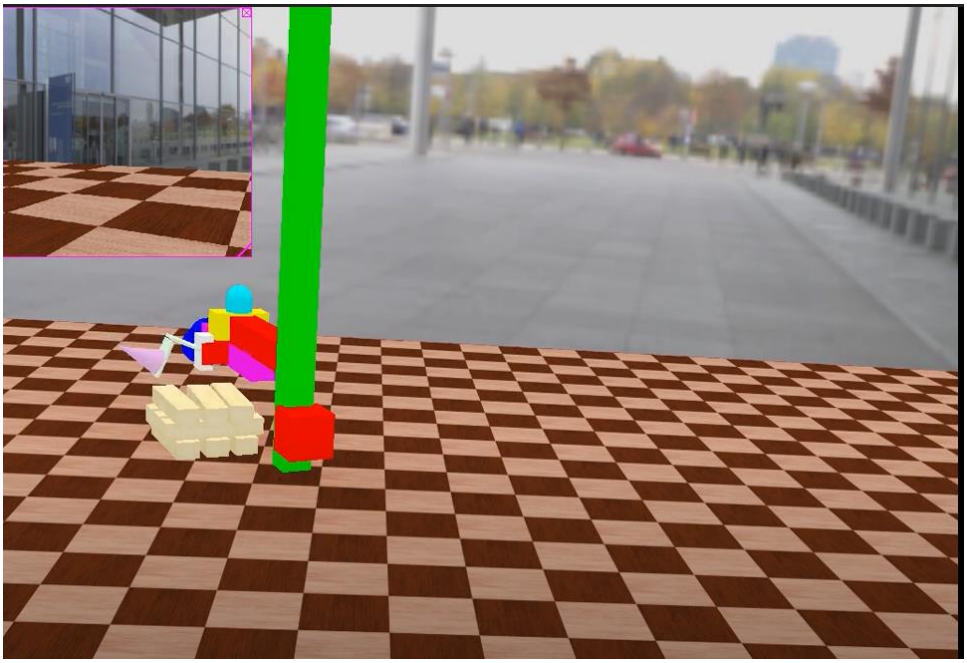
At this point, we should also discuss about the rest of the sensors used in order to create the robotic arm. More specifically, we used a distance sensor in order to be able to measure the height from which we need to descend our grasp in order to place the tile at the top of our Jenga structure (Figure 16). This of course was a result of an effort to reduce the computational cost when implementing an actual robotic arm, as scientific literature has stated that we need to utilize force sensors to be able to avoid disturbing the tower' s balance and structural integrity [6]. The sensor was able to provide us with interesting results. As expected, it performed well in terms of time complexity, however it is not suggested to be used, as that comes with a load of noise, or interference, given the environment setup. This might be enough to provide inaccurate data and the sensor itself could not be enough to deliver the results we expect. Moreover, we will state that we utilized a GPS sensor in our implementation, but this has only been the case for completeness of experiments, monitoring of the progress and basic logging, so we will not present any further of it, as it is not vital for the controller' s implementation.

Figure 16: The distance sensor range of detection



When the controller ends its execution, which happens after it successfully gathers all 12 of available tiles, something that has been hardcoded into the controller' s logic, it will complete a tower structure to resemble a Jenga tower as shown below (Figure 17).

Figure 17: The final result of all the tiles put together to form the Jenga tower

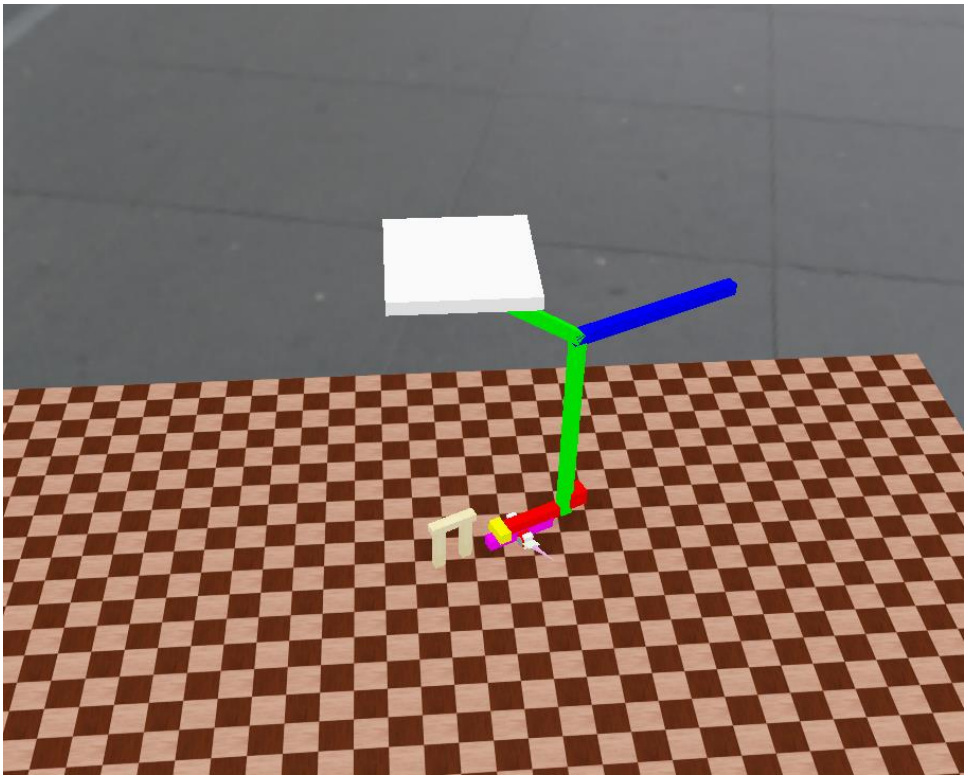


4.3. Controller extensibility

To show the extensibility of our creation, we decided to proceed with re-adjusting the controller' s programming for it to be able to construct different structures. To be more precise, we added the implementation of a controller that follows the same basic principles as the controller above, but instead can construct a "Π" with three tiles (Figure 18). The

results were rather expected and the controller was able to construct it with minor changes to the robot's configuration. These changes included the point from which the robot is supposed to grab a tile, along with omitting the use of a camera sensor, since its functionality would be the same. In the same manner, we have stripped off the environment from obstacles, since they would add nothing further to our experiments, making the robot behave in the same way as it did before with the Jenga tower.

Figure 18: The robotic arm used to create a "Π" instead of a Jenga tower



4.4. Algorithm accuracy

We will now proceed with giving certain experiment results concerning the execution time of each algorithm and we will also discuss these results. The environment at which the robots were to operate in is remarkably similar to the one presented above (Figure 15) and therefore it is omitted. Again, the grey rectangle-shaped objects pose as the obstacles that will get in the way of our robotic arm. At first, we need to understand the importance of providing fast solutions in the world of robotics. Eventually we must perform costly solutions that given the problem configuration will behave differently and provide us with different quality of results. One such measure of quality is the accuracy regarding the algorithm's findings in terms of Euclidean distance from the robot's grip to

the goal (any possible tile in the world), which in this case is represented by a path that has been established for us to reach our target, which is traversing among obstacles and finally reaching a tile in our environment to start constructing our Jenga tower. Let us now observe a table containing some of the results of given experiments. The initial position of the robotic grasp is always maintained; however, we change the configuration of the tiles in order to reach out for new end positions.

We will notice that the results will for the most part provide us with rather expected results and namely the fact that RRT in comparison to Dijkstra and A* was providing us with generally good solutions. However, each of these algorithms could reach a certain level of accuracy in its path planning process, but when it comes to accuracy and hence, quality of solutions, we need to use one of the complete algorithms (Table 8). As we have stated before this is based upon the problem we are currently facing. If the problem is genuinely complex in terms of both providing a solution along with the efficiency it needs to be delivered, if accuracy is crucial towards algorithm success, then we need to adopt one of the more complete approaches, such as in cases where the robotic arm needs to assist in surgery. If there is however a more generic conclusion here, it is that for most cases if we only measure and consider the quality of each solution, then we need to choose a complete algorithm that performs more context-aware search among the environment it explores.

Table 8: Some of the results of experiments concerning algorithm accuracy (in meters)

RRT	Dijkstra	A*
0.16	0.13	0.12
0.16	0.14	0.12
0.16	0.13	0.13
0.15	0.12	0.1
0.15	0.23	0.11

4.5. Time consumption

At this part we will discuss the performance in terms of time consumption for these algorithms to be executed. Some people might argue that in order to reduce the execution times we should decide to use heuristics-based algorithms and that could be true for many cases. However, when dealing with real-life problems the case is different. The general

truth is, as said above, that each choice we make should be problem-related. In other words, we cannot produce any generic approach for all problems. This of course has dire effects to the performance of the algorithms in terms of quality, which was discussed earlier.

Using heuristics indeed makes the search well-guided towards staying on the most cost-effective path towards the end, whereas a more naïve algorithm that is based on randomization might struggle providing promising if not refined results and seeking a path that is beneficial for the result. Furthermore, we can have a more guided search of our space and thus we understand the whole problem in an entirely separate way, able to give us the necessary answers. However, this has also some drawbacks which might in fact outperform the benefits of this type of search. These kinds of comparisons have already been discussed and thoroughly both explained and explored, so we will only mention the fact that there are cases where we need to find fast solutions that will be good, meaning that in cases where we do not need to provide 100% accurate and as possible close to the target plans, then we should focus on less complete algorithms that will be able to give us a fast answer. On the other hand, we should as well decide to utilize the strengths of complete algorithms when we tackle cases where the problem is complex. In this way we can provide answers if they exist, but the complexity of operations will also lead to greater time consumption. We should at this point have a more in-depth look as to what this might mean in terms of exploring how our implementation of each one of the algorithms performed (Table 9).

Table 9: Some of the results of experiments concerning algorithm time complexity (execution time is expresses in msec)

RRT	Dijkstra	A*
13.97	3.08	1.44
13.23	3.83	3.04
15.56	8.30	6.99
12.87	4.53	4.27
15.84	5.03	4.74

4.6. States traversed

As a decisive step to comparing the three algorithms that we implemented we will be looking at the number of states that each algorithm's plan of execution included, in order to reach the goal in different scenarios (Table 10). We consider this a key step as it will show the difference between them in terms of providing what is needed to proceed.

More specifically, by understanding the way these algorithms work, as was the purpose of previous chapters we now need to see in detail the effectiveness of their strategies. Another way to do that is to search through the states that these algorithms operate and the consecutive motions they produced.

At this point let us examine a little further to understand the reason we need to perform such a comparison. At a first glance, it might seem that it is a measure of comparison that we should skip as unnecessary. If we indeed, however, proceed with it, we gain the advantage of more detailed analyze, since it can give us information that will also describe the memory needs of our algorithms for example, given that in order to perform more moves we need to store more information into memory. In this case we can get an estimate of how these algorithms perform in terms of memory usage, if they are all represented in the same way, as is our case. We store all states in the same way; they are all objects of the same class. This can make up for a fair comparison, but again this is more of a follow-up than an actual performance metric. Another advantage we gain is the fact that we can understand which algorithms can find the most effective route and of course, as expected we should expect that using heuristics is of course one of the reasons why there might be such difference in some cases. On the contrary, in many cases, however, not always, having a greater number of states means that the algorithm will transition between states in a smoother way; for example, imagine reaching the target state in only two moves, in which case the algorithm would be extremely effective, yet completely steeper than getting there without only a handful of actions. We believe that while this should be kept in mind, we should always check that given the algorithm we have a balanced result.

Table 10: Some of the results of experiments concerning algorithm motions performed

RRT	Dijkstra	A*
42	33	30
114	15	13
137	58	46
76	68	63
63	21	15

4.7. Additional experiments with A*

Delving more into complete algorithm experimentation we decided to explore the behavior of A* even further, under various scenarios and environments. Namely, we picked an

environment setup where the robotic arm had to reach a goal destination, a tile, as previously described, while rectangle-shaped obstacles were blocking its way. For A* in particular, we added a test-case of staggered difficulty. The main problem tackled is the same as described above, with the difference being that for the path the robot was able to detect that could lead to a solution, we decided to add a block in each follow-up experiment that would be rendering it un-feasible, since it was amid its course (Figure 19). The algorithm would then have to produce a new solution that would be feasible and would consider the new problem setup. The main results were that the complexity tends to increase along with execution time, however this did not seem to prevent A* from finding a path.

Figure 19: A basic experiment where the robot avoids the rectangles to reach the tile highlighted with the 3D axes

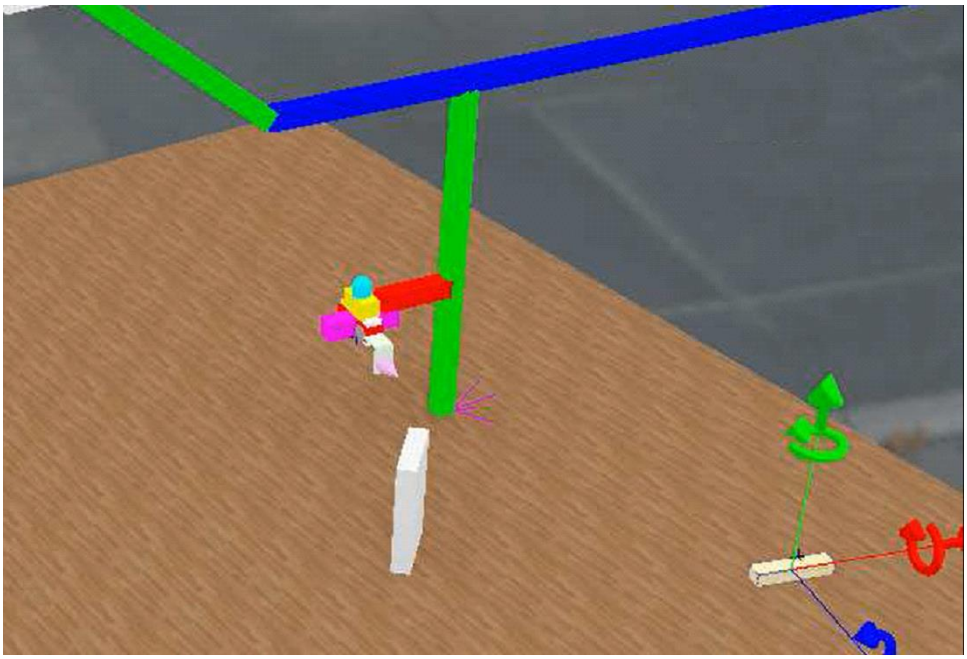


Figure 20: The basic experiment (Figure 19) whereas we now added an extra rectangle as an obstacle (highlighted by the axis) blocking the previously computed path

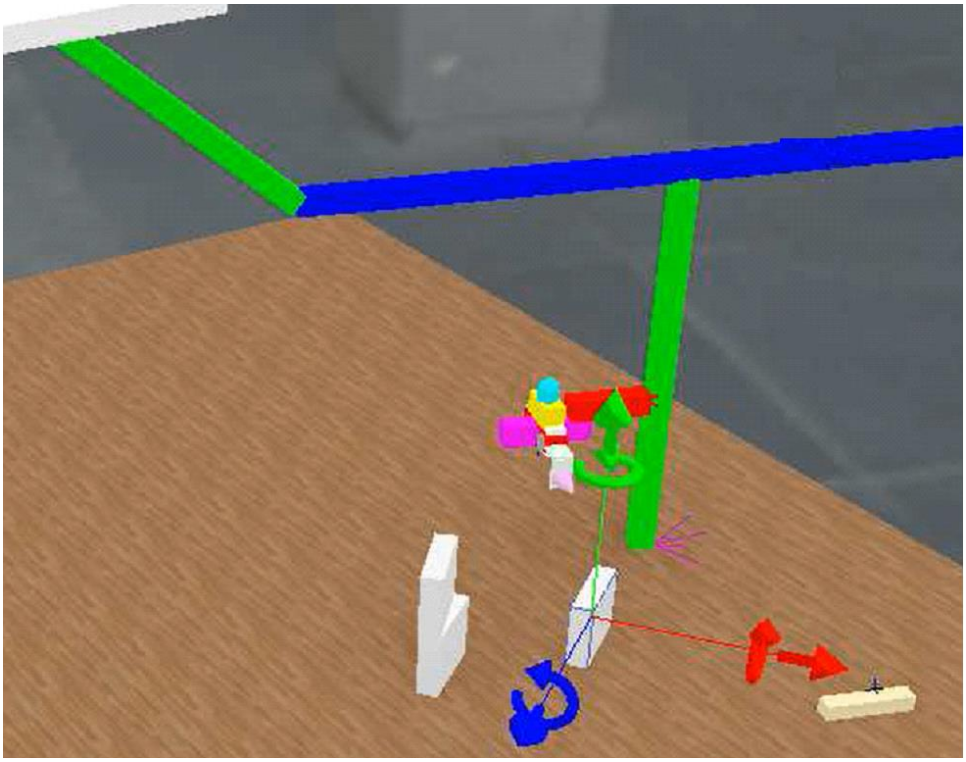
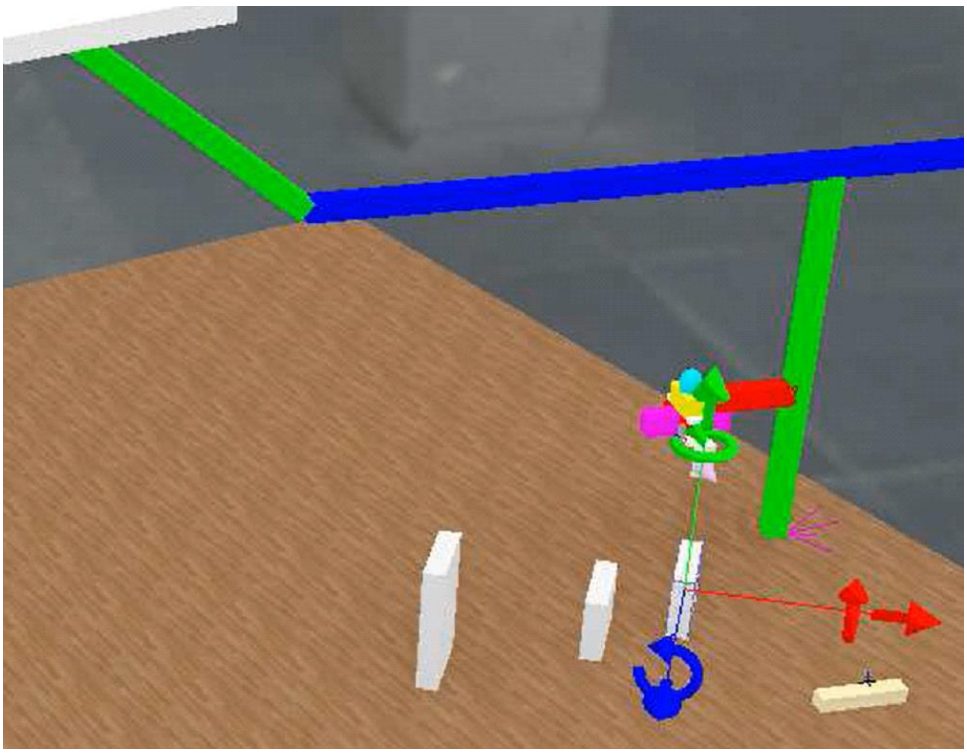


Figure 21: The experiment (Figure 20) after adding another obstacle to block the previous path highlighted by axis



In the experiments presented we can see that the algorithm is able to produce smooth paths for the robot to follow (Figure 19). After we add another obstacle capable of blocking the previous path, we can notice that the algorithm can still find its course around the obstacles while maintaining smooth transitions (Figure 20). It is important to understand that the

algorithm needs to know the position of the obstacles before the path computation begins. After we change the environment setup again while adding another obstacle, we can see that the robot guided by A* is still able to navigate through the environment avoiding the newly-added obstacle (Figure 21).

5 Conclusion

In this dissertation, we presented many cases in which robotics is used along with modern day struggles that continue to pose as serious problems. We studied various aspects of literature and ways that similar problems have been tackled. Moreover, we have studied controller organization and various purposes and means to implement. We also proceeded with exploring different algorithm philosophies in terms of better utilizing our resources, be it computational, time-like, or even complexity in terms of fully understanding a problem before we can attempt to solve it. Furthermore, we delved into attaining the benefits of certain algorithms through our attempts to simulate their behavior and logic through implementing them in a problem to reach a target in 3D space while navigating through obstacles. In this effort, we also touched the surface with exploring better spatial information management using R-Trees in our implementation of A*, which enabled us with better in terms of efficiency exploration of our states. As an aftermath, we produced different results for each algorithm, each of them leading us to different conclusions concerning the algorithm efficiency of them.

5.1. Summary and conclusions

Having reached some results, which we presented above and discussed what they mean towards the effectiveness of each algorithm, we should now proceed to summarizing outcome, for it to be perfectly clear. At a first glance, we investigated the accuracy of each algorithm, as it was described in terms of reaching the goal the closest. Complete algorithms were expected to behave better towards it and it occurred this way through experimentation. We also witnessed the simplicity of RRT in action and the surprisingly well accuracy it showed, since the values that were presented were lower than what we estimated as a general comparison value, that of 0.2 for an algorithm to be close enough to a solution, namely reaching the goal as close as it can get. We should also point out that trivial differences emerged between Dijkstra and A*, which overall gave us the most promising results. However, we should underline that more experiments could be included, that would give us better insight over this metric. Proceeding further with comparison, we need to effectively measure the time we needed to execute each algorithm through different experiments and that was again, as we expected, to outline A* as the better alternative between the three. Quite noticeably RRT in fact turned out to waste time performing random moves that did not really lead the algorithm to better states of exploration and consequently that led to many meaningless iterations, which was expressed due to the

results we presented given that the number of states it required in order to reach the goal every time were overwhelmingly greater than what the rest of the algorithms needed. In addition to what we have presented so far, in terms of speed of execution it is obvious that A* was the fastest algorithm, outperforming the other two, with a slight difference compared to Dijkstra's performance, which again could be a result of small sample of problems tackled. As a result of our previous notice, the less guided our search is, the more time it is expected to take for it to complete, which was the case with RRT. On the contrary, Dijkstra and A* behaved better and given the heuristic usage in A* it performed better compared to Dijkstra as we previously stated, something that was expected to happen.

From a macroscopic perspective, the scientist that could take this study in the next level, would benefit from the already found results and compare them to the complexity of each problem at hand. In other words, they should enrich their study with those results, especially when it comes to applying it to problems regarding planning, which is the case when it comes to robotics. If someone requires more accurate results, they should consider using complete algorithms, however, as the difficulty rises and the computational needs grow, they should keep in mind that such an approach might in fact have more drawbacks than benefits. Apart from the experiments, a researcher can also use the selected algorithm presentation to consider alternatives based on the problems they are dealing with, in order to achieve better performance from various aspects. Finally, as we stated before, it is quite wise to always take the different environment and situation of a problem at sincere consideration before proceeding with applying any logic towards solving it. As a result, different algorithms might have entirely different behavior given any problem and this in fact might sometimes lead to better results, or alternatively lead to greater exploration of how we should deal with any given problem structure.

5.2. **Research limits and limitations**

As the dissertation has concluded, we need to further explore certain aspects of it, including the various limitations that might exist. We do so in order to give more context on the potential of the current study and further motivate readers to understand the complex mechanisms that led to all the results as they were previously discussed. First, we should mention the obvious fact that only three different algorithms have been implemented. This is by no means a finite number of algorithms so that no further additions should be made. Instead, it only entraps the reader to the study and notice differences between these algorithms only. What is more, the experiments should include more observations, which does not on the other hand imply their weakness to present us with all that is produced

using each specific algorithm, or approach. Moreover, we are always restricted to certain available means, namely the computational power of our systems, so it should also be kept in mind that dissimilar needs and possibilities could emerge by performing this change. Another drawback is the difficulty of adding more available motions for our algorithms to be able to perform. As it has been stated, before we can estimate the outcome of a specific action, we first need to be able to understand where exactly in the space our grasp will be placed after performing each executed motion. In this case of scenario, we should keep in mind that extensibility is considered low due to the geometrical studies we need to consider before performing a move. Finally, in the case of A* in particular, we should include even more steps in between the execution loops, since we only provide the best, medium and a minimal step.

5.3. **Future extensions**

Having reached the end of the study, one might consider the direction to be followed as we progress through the field of robotics. In modern days, it has become quite a common need to seek navigational assistance, especially in more complex environments. Keeping the latter in mind, a scientist could extend the number of available algorithms by a considerable number, enough to apply different patterns to different problems, or even combine their potential in future work. Instead of having to abide by using only one of them, we expect many and great benefits to come after a more well-guided combination of some of these algorithms is adopted in order to solve different problems. A researcher could give more chances to random-based algorithms to solve similar problems to the ones above and, provide more problems per aspect of scientific interest, depending on the needs and necessities of tomorrow. What is more, we could include further experiments, in regards to different metrics established, some that might not even exist yet. There are many ways to measure the outcome in terms of quality when it comes to algorithmic procedures, so we let that as deliberately included in a wider range of options for every scientist to decide. In this way, we will be able to further understand the context of each algorithm and the various corners in its execution that might and in many cases should be changed to become more reliable. In addition, we need to provide a better way of storing information. As each algorithm progresses and more data will be processed as available, we should continue to enrich the analysis that takes place through the algorithm's logic with a more distilled version of all the information at hand. Another great path that remains for future exploration is to perform similar computations and problem-solving experimentations on better equipped systems that can deal with greater and vast computational needs. We could

as an even more refined approach of what has been presented consider the use of different implementations of R-Trees as they already exist and provide us with better resource management. Finally, researchers should also consider expanding the available geometric computations that provide an estimate of an object's movement and coordinates in a case that includes more accurate and multi-dimensional approaches. The last is to be taken into more consideration than the previous, since it is the main advantage and simulation means to help each algorithm understand more about the environment along with the next motion it should perform.

Literature

[1]. Kröger, Torsten, et al. "Demonstration of multi-sensor integration in industrial manipulation (poster)." Proc. IEEE Int. Conf. Robotics and Automation. 2006.

[2]. Li, Chenping, et al. "A survey on visual servoing for wheeled mobile robots." International Journal of Intelligent Robotics and Applications (2021): 1-16.

[3]. Kroger, Torsten, et al. "A manipulator plays Jenga." IEEE robotics & automation magazine 15.3 (2008): 79-84.

[4]. Fazeli, Nima, et al. "See, feel, act: Hierarchical learning for complex manipulation skills with multisensory fusion." Science Robotics 4.26 (2019).

[5]. Wermelinger, Martin, et al. "Greedy stone tower creations with a robotic arm." Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18). Lawrence Erlbaum Associates, 2018.

[6]. Wang, Jiuguang, et al. "Robot Jenga: Autonomous and strategic block extraction." 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, 2009.

[7]. Hernández, Juan David, et al. "Increasing robot autonomy via motion planning and an augmented reality interface." IEEE Robotics and Automation Letters 5.2 (2020): 1017-1023.

[8]. Yoshikawa, Tsuneo, et al. "Jenga game by a manipulator with multiarticulated fingers." 2011 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM). IEEE, 2011.

[9]. Yoshikawa, Tsuneo, Tatsuya Sugiura, and Seiji Sugiyama. "Development of a Jenga game manipulator having multi-articulated fingers." 2012 7th ACM/IEEE International Conference on Human-Robot Interaction (HRI). IEEE, 2012.

[10]. Lagriffoul, Fabien, et al. "Platform-independent benchmarks for task and motion planning." IEEE Robotics and Automation Letters 3.4 (2018): 3765-3772.

[11]. LaValle, Steven M. Planning algorithms. Cambridge university press, 2006.

[12]. Lynch, Kevin M., and Frank C. Park. Modern Robotics. Cambridge University

Press, 2017.

[13]. Karaman, Sertac, et al. "Anytime motion planning using the RRT." 2011 IEEE International Conference on Robotics and Automation. IEEE, 2011.

[14]. Adiyatov, Olzhas, and Huseyin Atakan Varol. "A novel RRT*-based algorithm for motion planning in dynamic environments." 2017 IEEE International Conference on Mechatronics and Automation (ICMA). IEEE, 2017.

[15]. Sieverling, Arne, et al. "Interleaving motion in contact and in free space for planning under uncertainty." 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2017.

[16]. Sintov, Avishai, and Amir Shapiro. "Time-based RRT algorithm for rendezvous planning of two dynamic systems." 2014 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2014.

[17]. Choudhury, Sanjiban, Sebastian Scherer, and Sanjiv Singh. "Realtime alternate routes planning: the rrt*-ar algorithm." (2012).

[18]. Kala, Rahul. "Rapidly exploring random graphs: motion planning of multiple mobile robots." Advanced Robotics 27.14 (2013): 1113-1122.

[19]. Noreen, Iram, Amna Khan, and Zulfiqar Habib. "A comparison of RRT, RRT* and RRT*-smart path planning algorithms." International Journal of Computer Science and Network Security (IJCSNS) 16.10 (2016): 20.

[20]. O. Khatib, “Real-time obstacle avoidance for manipulators and mobile robots,” in Proceedings. 1985 IEEE International Conference on Robotics and Automation, vol. 2, pp. 500-505.

[21]. Kavraki, Lydia E., et al. "Probabilistic roadmaps for path planning in high-dimensional configuration spaces." IEEE transactions on Robotics and Automation 12.4 (1996): 566-580.

[22]. Warren, Charles W. "Global path planning using artificial potential fields." 1989 IEEE International Conference on Robotics and Automation. IEEE Computer Society, 1989.

[23]. Metropolis, Nicholas, and Stanislaw Ulam. "The monte carlo method." Journal of the American statistical association 44.247 (1949): 335-341.

[24]. Eckhardt, Roger, Stan Ulam, and Jhon Von Neumann. "the Monte Carlo method." Los Alamos Science 15 (1987): 131.

[25]. Hart, Peter E., Nils J. Nilsson, and Bertram Raphael. "A formal basis for the heuristic determination of minimum cost paths." IEEE transactions on Systems Science and Cybernetics 4.2 (1968): 100-107.

[26]. Dijkstra, Edsger W. "A note on two problems in connexion with graphs." Numerische mathematik 1.1 (1959): 269-271.

[27]. Goldberg, Andrew, and Tomasz Radzik. A heuristic improvement of the Bellman-Ford algorithm. STANFORD UNIV CA DEPT OF COMPUTER SCIENCE, 1993.

[28]. Guttman, Antonin. "R-trees: A dynamic index structure for spatial searching." Proceedings of the 1984 ACM SIGMOD international conference on Management of data. 1984.