Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Master's degree in Automation Engineering

IMPLEMENTATION, ANALYSIS AND COMPARISON OF PATH PLANNERS BASED ON GENERATION OF RANDOM POINT TREES

Thesis supervisor: Prof. Ruggero Carli

Thesis advisor: Ing. Roberto Polesel

Candidate: Giulia Romanello

ID: 1178805

Padua, 24 February 2020

Academic year 2019-2020

Thesis developed at Euclid Labs S.r.l..

Euclid Labs
3D Vision and Robotics

Contents

Abstract

A robot is a machine that can replace human beings in both physical activity and decision making phase during task execution. The success of a generic robot activity requires the execution of a specific movement, starting from defined instructions. In order to do this three main aspects must be considered. Modelling, which is the derivation of a mathematical models describing the I/O relationship that characterizes the robot components. The control system that guarantees the correct execution of the desired motion. Finally the planning of the movement which is the goal of this study, i.e. how to allow the robot to autonomously plan its movement. In particular this thesis focuses its study on the path planning problem.

The term path planning refers to the procedure that provides a geometric collision free route that enables the robot to execute the desired task without colliding with the objects around it.

The purpose of this work is to implement, analyze and compare two different path planning algorithms in three different static environments which include: avoiding a single major obstacle, solving a navigation problem and finally going through narrow passages. In particular this project studies the sampling based path planners, a family of path planning algorithms which represents the connectivity of the free space with different data structures, like trees (RRT). Two different bidirectional RRTs and many sampling strategies were proposed and investigated, in order to understand reasons of failure and success of the implementations.

From the measurements made it was possible to observe that the construction of a simple tree without a large number of nodes and the use of a combination of different sampling strategies speed up the planner's search for a valid path.

In conclusion all tests were done on a thin long rod which has the capability to autonomously plan it movement and no robots are taken into account. So in the future it might be interesting to connect a robotic arm to the rod in order to understand if working with inverse kinematics could bring advantages with large payloads.

This thesis has been developed at 'Euclid Labs S.r.l' in Treviso, which designs and develops hi-tech solutions for robotics and industrial automation.

Chapter 1

Path planning techniques

1.1 Introduction

Nowadays robotics applications are infinite. Robots are in every aspect of the life, such as autonomous car, surveillance operations, agricultural robots, planetary and space exploration missions, Unmanned Autonomous Vehicle (UAV), and robotic manipulators. These are only some of the numerous examples and many times it is difficult to realize that we are working with one of them, given how much they are present among us.

It is possible to differentiate robots in: robots under control of an operator, robots set to do some specific job/movement and others that work autonomously with total independence on the operator, interacting by themselves with the environment.

Always more often the last two typologies are required in industrial applications. A robot is a machine that can replace human beings in the execution of a task, as regards both physical activity and decision making.

The success of a generic activity requires the execution of a specific movement prescribed to the robot. The correct execution of such motion is guaranteed by the control system which should provide the robot's actuators with the commands consistent with the desired motion. Motion control demands an accurate analysis of the characteristics of the mechanical structure, actuators, and sensors. The goal of such analysis is the derivation of the mathematical models describing the input/output relationship characterizing the robot components. Modeling a robotic manipulator is therefore a necessary premise for finding motion control strategies.

So it is possible to say that there exist three main topics in robotics, all strictly necessary for robot realization: modelling, planning and control.

Fig. 1.1: Robotic arms in an assembly line.

One of the most interesting issue is how to teach the robot how to move, i.e. the planning issue. The challenge of this new generation of engineers is to build a "brain" within a robot. For this reason path planning is a very important aspect. Allow the robot to have the ability to independently plan its movements based on the environment in which it lives is a not trivial problem.

The term path planning refers to the procedure that provides a geometric collision free route that enables the robot to execute the desired task without colliding with the objects around it. Path planning is the first step in dealing with motion planning problem and in presence of obstacles it is the most delicate step. Many studies have been conducted on this hard topic, but no one of these managed to have success in all situations.

The aim of this thesis is to analyze, implement and compare two different path planners in three different static environments. The reasons of failure are going to be investigated and some possible improvements are going to be proposed. Creating a "general" path generator, which manages to work properly in multiple situations, is still a complex and attractive problem today. Especially when there are manipulators that have to move large and heavy payloads, the problem becomes even more difficult, due to collisions of the payload. This work is intended to be a preliminary study to discover a new approach to manage the handling of large payloads.

The structure of this thesis is the following:
in this Chapter the theoretical background, used in the project, is reported. It is explained the problem definition, a classification of different types of path planners and some important instruments to be considered for their realization. However all methods and materials adopted are explained and developed in Chapter 2. It is possible to see the algorithms used and their implementations, the space representation employed and finally the sampling strategies applied. All tests and measurements are shown and discussed in Chapter 3, where three different environments were selected to test the algorithms and the sampling strategies in terms of trees' size, computational time and final path length. A recap of the most relevant results and possible future improvements are presented in the Conclusion chapter.

1.2 Motion Planning Problem Definition

Robotic systems are expected to perform tasks in a workspace that is often populated by physical objects, which represent an obstacle to their motion. What it is desired it is to endow the robot with the capability of autonomously planning its motion, starting from a high-level description of the task provided by the user and a geometric characterization of the workspace. However, developing automatic methods for motion planning is a very difficult endeavour. Replicate the spatial reasoning, which humans do instinctively, it is very complex. This is why path planning is still an open problem. To fix the ideas, the *Canonical problem* of motion planning is reported, referring to [1]:

Consider a robot $\mathcal{B}$, which may consist of a single rigid body (mobile robot) or of a kinematic chain whose base is either fixed (standard manipulator) or mobile (mobile robot with trailers or mobile manipulator). The robot moves in a Euclidean space $\mathcal{W} = \mathbb{R}^N$, with $N = 2$ or 3, called workspace. Let $\mathcal{O}_1, ..., \mathcal{O}_p$ be the obstacles, i.e., fixed rigid objects in $\mathcal{W}$. It is assumed that both the geometry of $\mathcal{B}, \mathcal{O}_1, ..., \mathcal{O}_p$ and the pose of $\mathcal{O}_1, ..., \mathcal{O}_p$ in $\mathcal{W}$ are *known. Moreover, it is supposed that $\mathcal{B}$ is free-flying, that is, the robot is not subject to any kinematic constraint. The motion planning problem is the following: given an initial and a final posture of $\mathcal{B}$ in $\mathcal{W}$, find if exists a path, i.e., a continuous sequence of postures, that drives the robot between the two postures while avoiding collisions (including contacts) between $\mathcal{B}$ and the obstacles $\mathcal{O}_1, ..., \mathcal{O}_p$; report a failure if such a path does not exist.*

The proposed problem definition focus on the geometrical studies, it tries to find a sequence of postures (position and orientation) of the robot that lets it to perform the task, if possible.

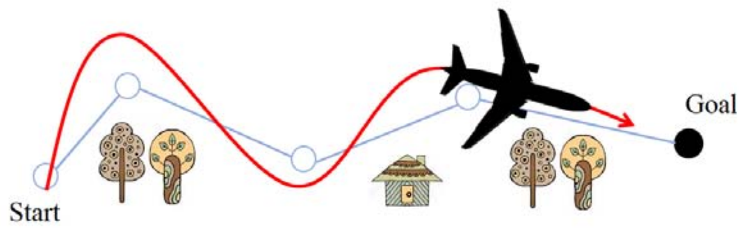


Fig. 1.2: Path planning procedure.

However the motion planning problem has to consider also the kinematic constraints of the robot. Every robot has physical limitations due to actuator limitations, i.e. how fast they can change velocity. The most sophisticated planners consider this limits and a common approach to solve the entire motion planning problem is to decompose it. First a geometric path is found (path planning problem), then it is smooth and finally it is time-parameterized, in order to satisfy actuator constraints (trajectory planning problem). An example of this implementation is in [2].

It is not mandatory to separate the problem in this two part, they could be solved together simultaneously. Nevertheless this is the strategy that this thesis follows. In particular in this thesis only the first step is performed, the creation of the free geometric path and a simple smoothing. Before starting the discussion on the path planning algorithms, let's make some clarifications.

1.2.1 Difference Between Path and Trajectory Planning

It is very important not to confuse the path planning problem with the trajectory planning problem. Even if these two terms are often used as synonymous, in this specific case they represent two different issues.

A *path* denotes the locus of points in the space, which the robot has to follow in the execution of a assigned motion; a path is a pure geometric description of motion.

On the other hand, a *trajectory* is a path on which a timing law is specified, for instance in terms of velocities and/or accelerations at each point. The trajectory planner receives as inputs: the geometric path, all the constraints due to the physical limitations of the motor's actuators and other constraints specified by the user.

In the following is reported the trajectory planning problem [2] in order to understand the difference with the path planning problem reported in section 1.2:

Given an initial state and a set of final states, the goal of this problem is to find a valid trajectory between these two states, where the term state indicates the vector of positions and velocities $[\mathbf{p}, \mathbf{v}]$*. The final trajectory must satisfy constraints on position, velocity and acceleration and it must be collision free.*

1.2.2 Online and Offline Planners

The robot's ability to autonomously plan its movement is an element that provides a classification in different types of planners.

If a robot calculates its entire path in advance, before starting the movement, it is an *offline* planner. Instead, if it is able to decide movements in real time, while on the move, it is an *online* planner.

Online planners are able to react to changes in the environment, to moving targets, and to errors encountered during movement. However, they require a very sophisticated path planner that has to quickly deal with high-dimensional search space, moving objects, kinematic and dynamic constraints of the robot.

So even if online planners are very attractive and powerful, they are not always used because they are very difficult to create. Fortunately in many industrial applications reacting in real time to the change of the environment or having flexible behaviour are not required, i.e. a manipulator closed in an automated cell that always performs the same tasks. For all these cases it is sufficient to realize a good offline planner, which finds the path at the beginning, testing offline all the possibilities. The only thing to pay attention is to think about how to pass information about the obstacles to the planner. This delicate argument is treated in the following section.

1.2.3 Configuration Space

As reported in the *canonical problem* (see Section 1.2) the robot moves in a Euclidean space $\mathcal{W} = \mathbb{R}^N$, with $N = 2$ or 3, called *workspace*. The workspace is the operational space, where the robot moves and where there are the obstacles.

For an n-DOF manipulator it is possible to divide the workspace in two:
the *reachable workspace*, which is the portion of environment that the manipulator's end-effector can access with at least one orientation;
and the *dexterous workspace*, the region that the end-effector can reach while attaining different orientations.

The task must be done in $\mathcal{W}$, but many times it is preferable to work with a simpler space. In 1979 T. Lozano-Pérez and M. A. Wesley [3] introduced a new idea to solve path planning problem which becomes very popular.

The very effective scheme, proposed by Lonzano-Pérez and Wesley, is obtained by representing the robot as a mobile point in an appropriate space, where are reported also the obstacles. A good choice for that space is to used the configuration space C of the robot. A configuration describes the pose of the robot, and the configuration space C is the set of all possible configurations.

A configuration has to express the generalized coordinates of a robot (or an object), which are generally of two types. The first represents the position of the body and it is expressed in Cartesian space; the latter is for body orientation and it is an angular coordinate.

The angular coordinates takes values in $SO(m)$ ($m = 2$ or 3) the *Special Orthonormal group* of real ($m \times m$) matrices (see Appendix A for all details on angular coordinates representation). The configuration space is finally obtained as the cartesian product of the two spaces, the Cartesain and the $SO(m)$.

Some examples are discussed below:

- If the robot is a 2D shape that can translate and rotate in $\mathcal{W} = \mathbb{R}^2$, C is the Special Euclidean group $SE(2) = \mathbb{R}^2 \times SO(2)$ (where $SO(2)$ is the special orthogonal group of 2D rotations), and a configuration can be represented using 3 parameters (x, y, θ);

- If the robot is a solid 3D shape that can translate and rotate in $\mathcal{W} = \mathbb{R}^3$, C is the Special Euclidean group $SE(3) = \mathbb{R}^3 \times SO(3)$ (where $SO(3)$ is the special orthogonal group of the 3D rotations), and a configuration requires 6 parameters: (x, y, z) for translation, and the Euler angles (θ, ϕ, η) for rotations;

- If the robot is a fixed-base manipulator with N revolute joints (and no closed-loops), C is N-dimensional.

Also the obstacles are reduced to a set of points that corresponds to the collision configurations of the robot, this subset of the C space is called C$_{obs}$. The set of configurations that does not collide is called C$_{free}$ = C\C$_{obs}$. With this new notation it is possible to redefine the path planning problem as: find a path in the C$_{free}$ space from an initial point to a final point [4], see Figure 1.3.

However still remain an hard topic how to represent the C$_{free}$, similarly the C$_{obs}$.
In the following different types of path planner will be presented. Not all the planners required the explicit computation of the C space, for example the sampling-based planner which are the subject of this study.

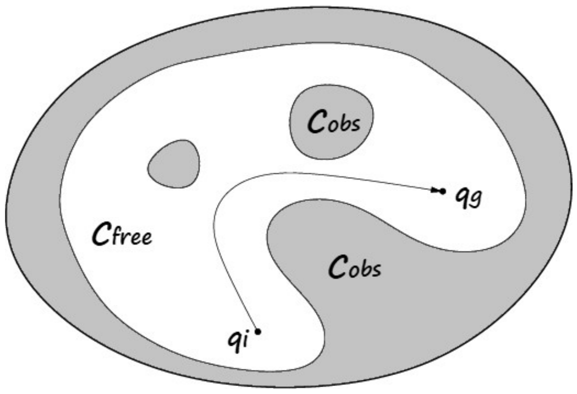


Fig. 1.3: C space description.

1.3 Different Types of Path Planners

In this section many types of path planner algorithms are going to be introduced. In 1991 Latombe [5] subdivides the path planner algorithms in three big classes, according to the methodologies used to generate the geometric path:

- *Roadmap algorithms* are also known with the name *Planning via Retraction* (section 1.3.1). The main idea of this kind of planners is to represent the C_{free} space by a roadmap. A roadmap is a particular graph which is able to express the connectivity of the C_{free} space;
- *Cell decomposition algorithms* (reported in section 1.3.2) try to subdivide (exactly or approximately) the C_{free} space in simply-shaped regions with the following characteristics: configurations in the same cell are easy to connect with a free path and also configurations in adjacent regions could be connected (easily);
- *Artificial Potential algorithms* (see section 1.3.3), they are very used for online planners, because they react quickly to environment changes. They move the robot under the influence of a potential field U.

However there is another important and efficient group not mentioned by Latombe, which is:

- the *Sampling-based* path planners ([6]), which will be discussed in section 1.3.4. A sampling-based algorithm represents the connectivity of the configuration space with a set of sampled states that build a graph. At each iteration a new sampled configuration is chosen and if it is collision free it is added to the graph. There are different ways to select the new sample and different data structures (graph) to represent $\mathcal{C}$ space, according to them there exist different types of sampling based algorithms:
  - *Probabilistic roadmap algorithms*, the new sample is created using a randomized approach and if it is collision free it is added to a roadmap;
  - *Tree based algorithms* has a new data structure: a *tree*. The construction is based on random samples added progressively to the tree, following specific rules.

Recently others types of methodologies have been developed and an overview of some of them is given in section 1.3.5.

1.3.1 Roadmap Algorithms

Algorithms that use retraction planning procedure represent the connectivity of the free space with a roadmap, which is a network of safe path. The first step for these kind of algorithms is to create the roadmap, while the second one is the retraction procedure, i.e. the connection of the start and goal state with the roadmap. Depending on the type of roadmap and on the retraction procedure adopted, this general approach leads to different planning methods.

One of the most popular of these methods is described in the following, under the simplifying assumption that C_{free} is a limited subset of C (just for simplicity $C = \mathbb{R}^2$) and is polygonal, i.e. its boundary is entirely made of line segments.

The method which is going to be described uses the Voronoi diagram to create the roadmap and to perform the retraction (see [7]). Intuitively if a body $\mathcal{B}$ has to move amidst obstacles in the space, it is better not to let it get too close to the obstacles, but if it is possible, it has to move sufficiently far from them, in order to avoid collisions. A nice criterion is to keep $\mathcal{B}$ equidistant from at least two obstacles at all time during its motion.

To carry this approach out in detail, an in-deep study of Voronoi diagram is required. An example of Voronoi diagram is depicted in Figure 1.4, the black line is equidistant from obstacles and boundaries and it represents the roadmap.

For each configuration $\mathbf{q}$ in C_{free} is defined the *clearance* by

$$\gamma(\boldsymbol{q}) = min_{\boldsymbol{s}\in\partial C_{free}} ||\boldsymbol{q} - s|| \tag{1.3.1}$$

where ∂C_{free} is the boundary of the C_{free}. The clearance $\gamma(\boldsymbol{q})$ represents the minimum Euclidean distance between the configuration $\boldsymbol{q}$ and the C_{obs} region.

Moreover, considering the set of points $N(\boldsymbol{q})$ on the boundary of C_{free} that are neighbours of $\boldsymbol{q}$:

$$N(\boldsymbol{q}) = \{s \in \partial C_{free} : ||\boldsymbol{q} - s|| = \gamma(\boldsymbol{q})\}, \tag{1.3.2}$$

it is possible to define formally the generalized Voronoi diagram of the C_{free} space as:

$$\mathcal{V}(C_{free}) = \{\boldsymbol{q} \in C_{free} : card(N(\boldsymbol{q})) > 1\}, \tag{1.3.3}$$

where $card()$ denotes the cardinality of the set. Therefore the Voronoi diagram is the locus of all the configurations that have more than one neighbor.

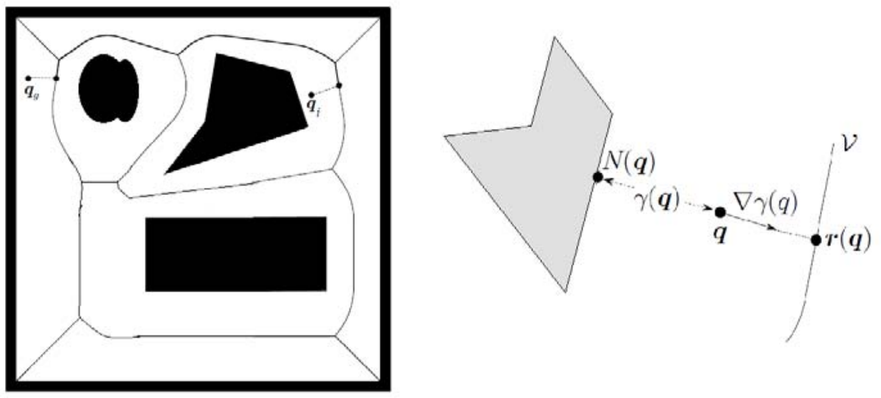


Fig. 1.4: On the left the roadmap creation with Voronoi diagram, on the left the retraction procedure in details.

If the initial and goal configurations ($\boldsymbol{q}_i$ and $\boldsymbol{q}_g$) do not belong to the Voronoi diagram, as it is shown in Figure 1.4, it is important to pay attention to the retraction procedure.

A simple retraction method is discussed here. First the clearance of $\boldsymbol{q}_i$ is derived, then the gradient of the clearance $\nabla\gamma(\boldsymbol{q}_i)$ is recovered. The gradient identifies the direction of the steepest ascent for the clearance at $\boldsymbol{q}_i$. It is directed as the half-line originating in $N(\boldsymbol{q}_i)$ and passing through $\boldsymbol{q}_i$. So the first point of intersection with $\mathcal{V}(\mathrm{C}_{free})$ is the connection point $\boldsymbol{r(q_i)}$. The same procedure is repeated for $\boldsymbol{q}_g$.

Now a search graph algorithm is required in order to find a path in the roadmap, which connects the initial configuration with the goal configuration.

Regardless of the search algorithm used, it is worth to notice that the above path planning method is *complete*. A path planner is complete if it finds a solution (if one exists) or reports a failure; it always gives an answer to the path planning problem.

The motion planning method based on retraction can then be considered *multiple-query*. In fact, once the generalized Voronoi diagram is complete, it could be easily and quickly used to solve other instances (queries) of the same motion planning problem.

However it is preferable to employ this planner in small dimensional spaces, due to the fact that the difficulty on implementation of the Voronoi diagram increases with the complexity of the spaces.

1.3.2 Cell Decomposition Algorithms

A cell decomposition algorithm (also known as grid based algorithm) is a procedure which divides the free configuration space into regions of simple shape. There exist two types of cell decomposition algorithms, depending on the type of decomposition used and they are both reported below:

- **Exact decomposition** if the union of all the regions forms exactly the entire C_{free} space.

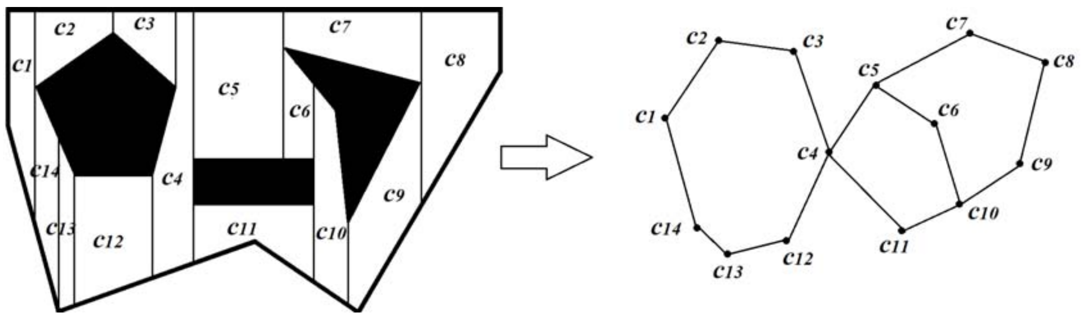


Fig. 1.5: Exact cell decomposition. On the left the space cell subdivision in trapezoids, on the right the graph found.

In Figure 1.5 it is possible to see the cell decomposition stage and the relative connectivity graph associated. The nodes of the graph represent the cells, while the edges connecting two nodes indicate that the two regions are adjacent. Adjacent regions could be easily connected. Typically the center of the region is taken as node for the graph in order to figure also the shape of the free connected space.

After the computation of the connectivity graph, the algorithm searches for the path that connects the two regions (c_i) and (c_g), which contain respectively the initial and goal configuration. The path turns out to be a *channel*, namely a sequence of adjacent cells from c_i to c_g.

- **Approximate decomposition** if the union of all the regions is contained in the C$_{free}$ but does not coincide with it.

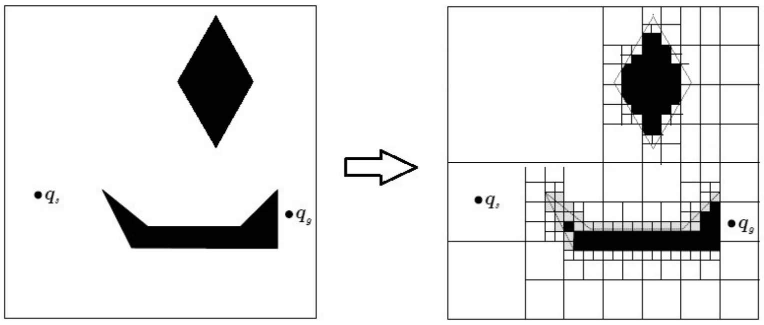


Fig. 1.6: Approximate cell decomposition. On the left the assigned problem, on the right the approximation.

The approximate decomposition works in a different manner. As it is shown in Figure 1.6 the algorithm starts dividing the C$_{free}$ space into four cells, that are labelled and classified according to the following specifications:

free cells, they have not obstacles inside;

occupied cells, they are entirely contained in the C$_{obs}$ space;

mixed cells, if they are neither free or occupied.

The algorithm continues by creating the associated graph, which has all the free and the mixed cells as nodes. Then a path is searched in this graph. If the path does not exist a failure is reported. However if the path exist and it is composed by all free cells, the procedure ends. Instead if the path exist but contains mixed cells, for each of these cells it is done the decomposition step. In turn they must be divided in four parts and classified as free, mixed and occupied. The algorithm works until an error or a complete path of the free cells is detected.

The main advance of this kind of path planner is that (as the roadmap planners) they can be considered *multiple-query* path planners. Namely once the connected graph has been computed it can be used to solved multiple path planning problem in the static environment considered.

However when there are moving objects in the workspace the cell decomposition algorithms cannot be used, because they have to change their structure (graph) in real time.

Another disadvantage of both the exact and approximate algorithms is that, even if they could be used in high dimensional spaces ($\mathbb{R}^N$), the complexity of these planners is prohibitive, being exponential in the dimension of C. Their importance is therefore mainly theoretical. As a consequence, this technique is effective in practice only in configuration spaces of low dimension (typically, not larger than 4).

1.3.3 Artificial Potential Field Algorithms

In artificial potential field (APF) algorithms the robot moves under the influence of a potential field U_{tot}, which is the superposition of an attractive and a repulsive potential:

$$\mathrm{U}_{tot}(\mathbf{q}) = \mathrm{U}_{att}(\mathbf{q}) + \mathrm{U}_{rep}(\mathbf{q}). \tag{1.3.4}$$

The attractive potential U_{att} is typically a paroboloid or a cone with vertex in the desired goal state $\mathbf{q}_g$. This potential is designed in order to guide the robot to the final configuration.

Instead the repulsive potential U_{rep} has to move the robot away from obstacles. In particular under a certain range of influence the robot is rejected. The repulsive potential increases as a function of the inverse of the distance between the robot and the limits of the obstacle. Outside this range of influence the potential is null.

The total force which acts on the robot is the opposite of the gradient of the total potential:

$$\mathbf{f}_{tot}(\mathbf{q}) = -\nabla\mathrm{U}_{tot}(\mathbf{q}) = f_{att} + \sum_{i=1}^{p} f_{rep,i}(\mathbf{q}). \tag{1.3.5}$$

In Figure 1.7 there are shown two possible movements of the robot in order to avoid the obstacle and reach the target.

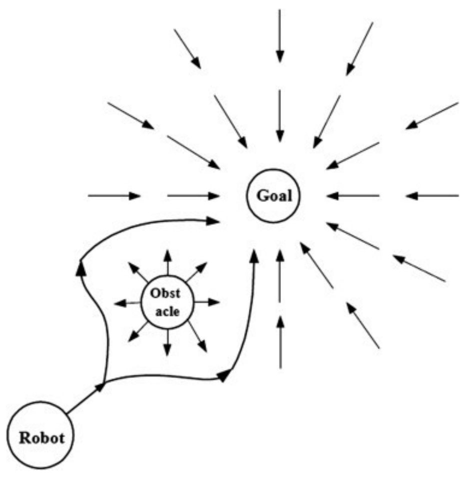


Fig. 1.7: Simple artificial potential field example with repulsive potential around the obstacle and attractive potential around the goal.

The idea of imaginary forces acting on a robot has been suggested by Andrews and Hogan (1983) [8], it is a very elegant and simple idea, which continues to be studied. In fact these algorithms are very used in real time applications, where an online planner is required, because they are very fast in computation and they are able to react to environment changes quickly.

However they are intrinsically affected by major problems. A study of Koren and Borenstein in 1991 [9] well investigates all the principal drawbacks of APF algorithms. During their experiments Koren and Borenstein identified four significant problems, typical of all the APF algorithms:

- the presence of local minima, where the robot may find itself trapped;
- no passage between closely spaced obstacles could be found;
- there are oscillations in the presence of obstacles;
- and also in narrow passages.

In Figure 1.8 it is depicted a simple example of a two-dimensional local minima problem. The local minima problem is the most discussed and famous and it is also known with the name trap-situations problem. A trap-situation may occur when the robot runs into a dead end (e.g., inside a U-shaped obstacle), but of course many other obstacles configuration can lead to the same bad situation. A heuristic or global recovery is used to escape local minimum.

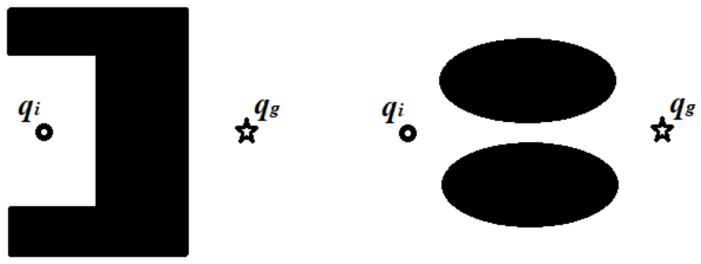


Fig. 1.8: On the right the local minima problem, on the left no passage between closely spaced obstacles.

Oscillations in presence of obstacles and moreover in narrow passages are the two most significant limitations of potential field methods. In particular in narrow passages the robot experiences repulsive forces simultaneously from opposite sides and that causes large unstable and undesired motion. In the most unfavorable case, the robot is unable to cross the small crack, because the sum of all the repulsive forces moves the robot away from the slit.

1.3.4 Sampling-Based Algorithms

Many times the explicit representation of the C space is very difficult to provide. For that reason sampling-based algorithms have been developed, because these kind of algorithms prevent the explicit construction of the C_{free} space.

The basic idea consists of determining a finite set of collision-free configurations that adequately represent the connectivity of C_{free}. These configurations are used to build a graph that can be employed for solving path planning problems. At every iteration a new configuration is sampled and tested with a collision detector, if it will be free from collisions it is added to the graph.

The two most famous sampling based algorithms are reported in the following, this subdivision refers to the structure and construction of the graph used.

- **The probabilistic roadmap algorithms (PRM)** typically start with the roadmap construction (depicted on the left of Figure 1.9).

The construction is different from the one explained in section 1.3.1. One may proceed in a deterministic way, choosing the samples by means of a regular grid that is applied to C. However, it is preferable to use a randomized approach, in which the sample configurations are chosen according to some probability distributions.

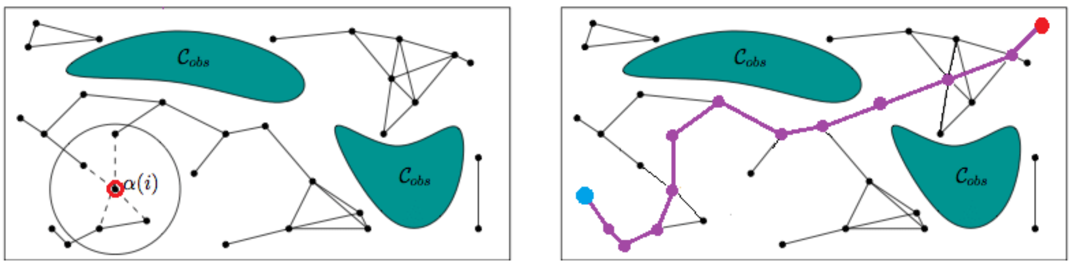


Fig. 1.9: On the left the roadmap construction phase and on the right in purple the final path.

More information about good probability distributions are given in section 1.4, instead let's see how to build a roadmap. One possible method is the following: at each iteration a new sample $\boldsymbol{q}_{rand}$ is generated from a probability distribution in C. Then $\boldsymbol{q}_{rand}$ is tested for collision, if $\boldsymbol{q}_{rand}$ does not cause collisions it becomes a node of the roadmap; otherwise the configuration is discard.
Once a configuration is added to the roadmap, it is immediately connected (if possible) through free local paths to sufficiently "near" configurations already in the roadmap. Usually the new roadmap node is connected with all the nodes within a circle of radius $\alpha(i)$ or with the closest node in terms of Euclidean distance. In most cases the edge connecting two nodes is a straight line between them.
The algorithm goes on until a maximum number of iterations is reached or the number of connected components in the roadmap becomes smaller than a given threshold. The output of the algorithm is a list of free configurations or a failure message if the path does not exist.
The PRM method is intrinsically *multiple-query*. The main result of this method is the remarkable speed in finding a solution to motion planning problems. In particular in high-dimension spaces, the time required to have a first valid solution is very reduced compared to the methods already presented in the previous sections.
The main downsize of PRM is that they are only *probabilistically complete*. An algorithm is probabilistically complete if the probability to find a solution (when one exists) tends to 1 as the computational time tend to infinity.
This means that, if one solution does not exist, the algorithm will run forever. In case of no solution a stopping criterion is enforced by the user in order to end the method. Some common shutdown criteria set a maximum threshold on the number of iterations or on the time elapsed since the beginning.

- **The tree based algorithms** are based on the concept of *Rapidly-exploring Random Tree, RRT*. In 1998 La Valle introduced this innovative and efficient procedure in [10]. In this case the space is represented with a tree, that has a root node which is incremented at each iteration through the goal. This special tree is called RRT

and it is a randomized data structure specifically designed to handle problems with nonholonomic constraints (i.e. dynamics) and high degrees of freedom.
It is a *single-query* planner and it does not rely on the generation of a graph that represents exhaustively the connectivity of the free configuration space. In fact it works in a subset of the C$_{free}$ space.
In the following will be discussed the RRT behaviour and here it is reported also the pseudocode.

```
RRT Algorithm
________________________________________

Input: $q_i$, $q_g$, $T_{max}$
Output: Tree $\mathcal{T}$

$\mathcal{V} \leftarrow \{q_i\}$;
$\mathcal{E} \leftarrow \emptyset$;
while $\mathcal{E} \cap q_g = \emptyset$ & $t < T_{max}$ do
    $q_{rand} \leftarrow$ Sample();
    $q_{near} \leftarrow$ NearestNeighbor($\mathcal{V}, q_{rand}$);
    $q_{new} \leftarrow$ NewState($q_{near}, q_{rand}$);
    if CollisionFree($q_{near}, q_{new}$) then
        $\mathcal{V} \leftarrow \mathcal{V} \cup \{q_{new}\}$;
        $\mathcal{E} \leftarrow \mathcal{E} \cup \{(q_{near}, q_{new})\}$;
return $\mathcal{T}$ = ($\mathcal{V}, \mathcal{E}$);
```

A tree data structure is defined by the set of its nodes (vertices) $\mathcal{V}$ and the set of its edges $\mathcal{E}$. The initialization is very easy, the edge set is put equal to null, instead the vertex set contains only the root node (i.e. the starting configuration or ending one). Then a new sample is generated by a probability distribution on the C space (`Sample`). If this new node is not in collision the algorithm searches for the node of the tree closest to it (q_{near}), with the function `NearestNeighbor`. It is worth to notice that the choice of the distance metric influences a lot the tree structure and the computational time.
Finally a new configuration is found with the `NewState` function and if the path between q_{near} and q_{new} is free from collisions the new node is added to the tree.
There exist many ways to implement the `NewState` function, in Figure 1.10 is depicted one simple. Every edge of the tree has the same length δ, so the new configuration is the one belonging to the straight line connecting q_{near} with q_{rand} far δ from q_{near}. Other implementations are discussed in the next chapter.
The tree grows until the goal configuration or a stopping condition is reached. A stopping condition is required in order to forced the shut down when a solution does not exist. In fact (like PRM) RRT is also *probabilistically complete.* Usually a good stop criterion adopts the computational time: if the time required for the computation exceeds a certain threshold (T_{max}), the algorithms arrests. Another possibility concerns the maximum number of iterations of the algorithm, as already said.

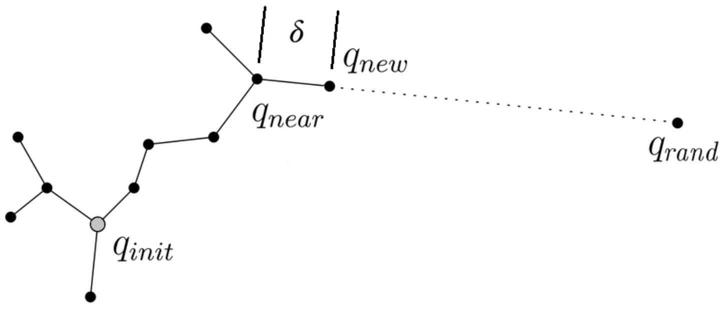


Fig. 1.10: Rapidly-exploring Random Tree (RRT) based algorithm.

To speed up the search for a free path going from q_i to q_g, the *Bidirectional Rapidly-exploring Random Tree (B-RRT)* method uses two trees T_i and T_g, respectively rooted at q_i and q_g. A simple implementation is reported in section 2.3.1 and an upgrade version in section 2.3.2.

So the major drawback of probabilistic planners is that they are probabilistically complete. Both the sampling based algorithm presented do not known anything of the workspace. The collider, the instrument which does the collision detection and knows where obstacles are, is like a black box for the algorithms. They work purely at random, for that reason in some environments they are not successful and spend much time. One example is ambient with narrow passages and holes, which are very difficult to individuate with a randomized approach.

Over years, many different variations of these kinds of algorithms have been suggested. Many papers have been written and it is not trivial to compare them, because the algorithms and the test environments are different.

1.3.5 Evolutionary Algorithms

In recent years new methods have been developed and one of the most interesting takes inspiration from nature: the *Evolutionary algorithms.*

Evolutionary algorithms use mechanisms inspired by biological evolution, such as reproduction, mutation, recombination and selection to solve the path planning problems [11]. The solutions of the algorithm are considered as individuals within a population. There is a “fitness function” which determines the relations between individuals and the level of aptitude that a particular individual has to solve the given optimization problem.

The most popular type of evolutionary algorithm is the *Genetic algorithm (GA).* John Holland introduced genetic algorithms in 1960 based on the concept of Darwin’s theory of evolution, then one of his student extends them. A genetic algorithm is an evolutionary optimization method used to solve, in theory “any” possible optimization problem.

For a genetic algorithm each individual can be encoded in a finite set of parameters. These parameters are the gens (genetic information) that make up the *chromosome.* The chromosome is the real structure of the individual and the solution of the planning problem.

Let’s see more in details a simple GA. The GA is an evolutionary method, this means that each iteration of the algorithm is used to evolve a population S of p individuals to find the “fittest” individual to solve a particular problem.

Every iteration starts with a generation. In this step a new set of individuals is created using genetic operators (i.e *Crossover* and *Mutation*). These operators make new chromosomes with the intention of bettering the overall fitness of the population. The individuals, that have to perform a genetic operation, are chosen with a *Selection method.* Finally the new chromosomes replace some old chromosomes according to a *Replacement method.* The Simple GA [12] is known to have the next set of common characteristics:

- Constant number of p individuals in the genetic search population;
- Constant length binary string representation for the chromosome;
- One or two point crossover operator and single bit mutation operator, with constant values for μ (*mutation rate*) and γ (*crossover rate*);
- Roulette Wheel (SSR) Selection method;
- Complete or Generational Replacement method or Generational combined with an Elitist strategy.

An example of simple GA implementation is reported in [13] and in the following is reported the flowchart associated.

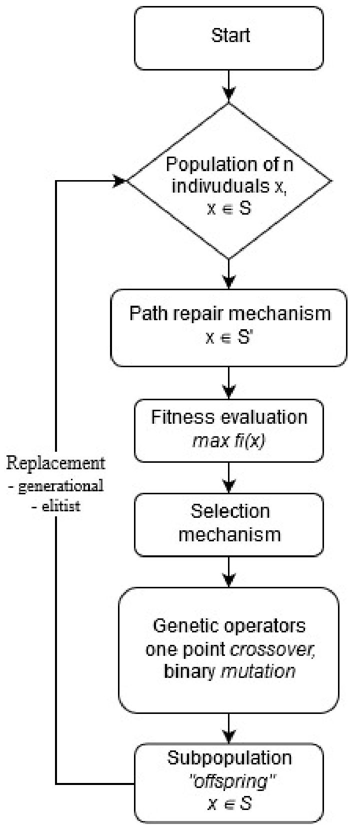

Start
Population of n indivuduals x, x ∈ S
Path repair mechanism x ∈ S'
Fitness evaluation max fi(x)
Selection mechanism
Genetic operators one point crossover, binary mutation
Subpopulation "offspring" x ∈ S
Replacement - generational - elitist

Fig. 1.11: Conventional genetic algorithm flowchart with one optimization objective.

The population S, used in the genetic search, is initialized with p total individuals. In the path repair mechanism each path is classified inside of the population S, as valid or

non-valid. The path are non-valid if:
- it contains cells with obstacles;
- it contains cells out of bounds;
- the final cell is not the destination.
The subpopulation S' is made up by entirely non-valid paths in S. The path repair procedure tries to establish if the non-valid path in S' could be saved and if they could become valid.

Paths which are still non-valid after the repair step are bad evaluated at the fitness evaluation step, all the valid path however are accurately analyzed. Usually the fitness evaluation function optimizes only one objective, which is the path length. Then we define fitness $f_i(x)$ as given by:

$$f_i(x) = n^2 - c, \tag{1.3.6}$$

where c is the number of cells in a given path x and n is the number of columns and rows in the grid representing the map of a given terrain.

For the termination criteria of the GA, a fixed upper limit k gives the maximum number of generations.

1.4 Generation of Random Points

The planners already described in section 1.3 can require:

- A distance metric to express relations between samples, which will be discussed in section 1.5;
- A distribution function to generate samples in the space. This function is really important and necessary for *Sampling-based* algorithms, because it is the core for their growth techniques.

In many applications a uniform sampling distribution guarantees success and it becomes one of the most used probability distribution. Computing a good uniform distribution it is useful for search algorithms because it lets to avoid oversampling and undersampling of large regions.

A point in the Cartesian space can be uniformly random sampled, in a very simple way. It is sufficient to take three independent random values, one for each coordinate (x, y, z).

Instead all these requirements are not trivial for rotation in $SO(3)$, in particular having a basic uniform sampling distribution for rotation it is not so immediate.

Before explaining how to obtain a uniform distribution let's consider what uniformity means for rotations. A standard approach to give the definition of uniformity uses the Haar measure. The Haar measure says that: "*if X is a random rotation with uniform distribution, then for any fixed but arbitrary rotation R, $R \cdot X$ and $X \cdot R$ must have the same distribution as X.*"

Unfortunately it is not possible to create a uniform distribution of rotations taking three independent uniformly distributed Euler angles between $[-\pi, +\pi]$. Three independent random Euler angles (sampled between $[-\pi, +\pi)$) provides a distribution that is heavy biased towards polar regions, as it is depicted in Figure 1.12.

A proof of this statement could be easily provided. As it is explained in the following subsection, uniformly distributed unit quaternions correspond to uniformly distributed rotations. The average magnitude of the w component for uniformly distributed unit quaternions is give by:

$$\frac{1}{\pi^2}\int_0^{\pi/2} 4\pi \cos^2\theta \sin\theta d\theta = \frac{4}{3\pi} \simeq 0.4244. \tag{1.4.1}$$

Now it is possible to prove that a uniformly distributed spatial rotation does not have a uniformly distributed angle. In fact the w component of a unit quaternion is the cosine of half the angle of rotation. When the angle is uniformly distributed between 0 and 2π, the average magnitude of w will be $2/\pi \gg 0.6366$, which exceeds the correct value for a uniform rotation by a factor of $\frac{2}{3}$.

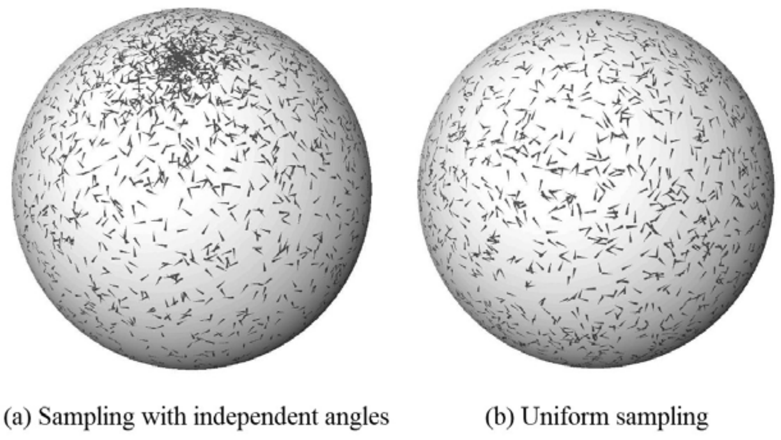


Fig. 1.12: Not uniform (a) and uniform sampling (b) of $SO(3)$, using Euler angles. There are 5000 rotation samples, represented with their orientation, depicted on the surface of a sphere.

In [14] and in [15] are proposed some methods that guarantee a uniform sampling also for rotations in $SO(3)$. There are two possible approaches to obtain a good sampling distribution, working with Euler angles or with quaternions. Both methods are reported in the following.

1.4.1 Uniform Sampling with Euler Angles

As said in the previous section, independent Euler angles do not provide uniform distribution of rotations. If all the angles can vary between $[-\pi, +\pi]$ the distribution is biased toward polar regions. Fortunately there exist simple ways to create a uniform distribution, which is explained in [14] for Roll-Pitch-Yaw Euler angles. The idea behind Algorithm 1 is the following: two Euler angles can vary between $[-\pi, +\pi]$ (θ and η), instead the third (ψ) must be chosen in the range $[-\frac{\pi}{2}, +\frac{\pi}{2})$. In this way it is possible to avoid oversampling in polar region and also a double coverage of the space of rotations which is obtained when all the angles can vary between $[-\pi, +\pi]$.

The ψ angle is calculated from the inverse of the cosine plus a correction factor. The term $rand()$ expresses a random number between $[0, 1]$.

```
ALGORITHM 1:  Uniform Sampling with Euler Angles

Input: none
Output: uniform random Euler angles (Roll-Pith-Yaw) $(\theta, \phi, \eta)$

$\theta = 2\pi * rand() - \pi$;
$\phi = \arccos 1 - 2 * rand() + \pi/2$;
if ($rand() < 1/2$) then
    if ($\phi < \pi$) then $\phi = \phi + \pi$
    else $\phi = \phi - \pi$
end
$\eta = 2\pi * rand() - \pi$;
return $(\theta, \phi, \eta)$;
```

1.4.2 Uniform Sampling with Unit Quaternions

In [15] it is reported the simplest method for creating a uniform distribution of $SO(3)$. This method involves the use of quaternions.

The geometric problem of generate uniform rotations, in the quaternion representation, turns out to be one of generating a point uniformly distributed in a sphere of size four, the quaternion unit sphere $w^2 + x^2 + y^2 + z^2 = 1$.

Composition of rotation matrices is equivalent to the product of the corresponding quaternions. So uniformly distributed unit quaternions correspond to uniformly distributed rotations. As in the previous algorithm, the term $rand()$ indicates a random number between $[0, 1]$. The proof of the correctness of the Algorithm 2 is explained in [15].

```
ALGORITHM 2:  Uniform Sampling with Quaternions

Input: none
Output: uniform random quaternion (w,x,y,z)

$s = rand()$;
$\sigma_1 = \sqrt{1 - s}$;
$\sigma_2 = \sqrt{s}$;
$\theta_1 = 2\pi * rand()$;
$\theta_2 = 2\pi * rand()$;
w = $\cos\theta_2 * \sigma_2$;
x = $\sin\theta_1 * \sigma_1$;
y = $\cos\theta_1 * \sigma_1$;
z = $\sin\theta_2 * \sigma_2$;
return (w,x,y,z);
```

1.5 Distance Metrics

In some path planning algorithms it is useful to have a distance metric defined on the $\mathcal{C}$-space in order to give a measure of the "closeness" between two points.

In this subsection it will be presented some possible distance metrics for the configuration space.

The purpose is to derive a scalar weighted function (d) which considers the distance between two configurations in terms of translation (ρ_t) and rotation (ρ_{rot}):

$$d = w_t * \rho_t + w_r * \rho_{rot}. \tag{1.5.1}$$

The choice of the weights w_t and w_r is very crucial for the success of the algorithm.

In the following there are described some metrics appropriate for the Euclidean space $\mathbb{R}^3$ and others for the special orthogonal group $SO(3)$.

- **Euclidean Space** The classical method is the Euclidean norm, it is very easy and used in Cartesian space:
$$\rho_t = \|V_1 - V_2\|. \tag{1.5.2}$$

- **Special Orthogonal Group** The metric has to measure the distance between two rotations and it has to be a positive scalar function. There are many solutions to that issue;

  - Working with Euler angles, in particular with their Roll-Pith-Yaw representation it is possible to express the difference of rotations as a difference between angles. The sum of the squares o the differences between two triples of Euler angles can provide an estimate of distance:
  $$\rho_{rot} = \sqrt{(\theta_1 - \theta_2)^2 + (\phi_1 - \phi_2)^2 + (\eta_1 - \eta_2)^2}. \tag{1.5.3}$$
  However this kind of measure is very poor and does not respect efficiently what really happens between rotations. It could happen that big difference in angles does not cause a big rotation;

  - Quaternions helps to be more accurate, doing
  $$\rho_{rot} = (1 - \|Q_1 \cdot Q_2\|), \tag{1.5.4}$$
  where $Q_1 \cdot Q_2$ is the inner product of the quaternions.

  - Also the direct manipulation of the rotation matrix guarantees good estimate with
  $$\rho_{rot} = (R_1)^{-1} * R_2. \tag{1.5.5}$$

1.6 Smoothing Path Techniques

Most of the planning algorithms generate a path consisting of a set of straight lines and sharp turnings. It is preferable to recover from the broken lines a smooth and continuous path.

There exist many techniques in literature that studies how to smooth, however this problem becomes critical and not trivial in the presence of obstacles. It is not guaranteed that interpolating the broken points will provide a smooth path free from collisions.

However a simple improvement of the path can be done *pruning*.

This strategy is efficient with path obtained by sampling based algorithms in which points are random. It starts taking the list of all the vertices of the sequence of straight lines. Then the algorithm continues iteratively with the following considerations.

The first configuration considered and saved in the final path is the initial.
If there exist a collision free path between the configuration in position (i) of the list and the configuration in position $(i + k)$, the configuration in position $(i + k - 1)$ is deleted, otherwise the $(i + k - 1)$-configuration is saved in the final path and the algorithm starts again considering this position as start. The parameter k has not to be greater of the number of configurations of the current path.
The algorithm goes on until the final pose is reached and saved in the final path.

The final path could be shorter than the initial sequence of configurations, bu it is not guaranteed. It depends on the initial structure of the path. However, most of the sharp turnings are deleted and it provides a more linear path if possible.

Chapter 2

Methods and Materials

2.1 Problem Formulation

The objective of this work has already been briefly presented in the introduction of Chapter 1, without a formal explanation. However, after the brief theoretical background on different types of path planners and common tools generally used to create them, it is now possible to have an adequate formulation for this thesis problem.

The aim of this thesis is to analyze, implement and compare two different tree-based path planners, which are tested in different static environments.

In this chapter all methods and material employed in this thesis are presented in details. In particular in the following there is the explanation, implementation and analysis of the two bidirectional RRTs used, the sampling strategies adopted and the collision detector behavior.

The target to move is a thin, long rod that is able to autonomously plan its movements, like a UAV and it is described in the next paragraph.

2.2 The Choice of the Moving Object

Typically, path planner algorithms work on a robot that can move around a certain portion of space. However, in this thesis, the element to be moved is an object, a thin and long rod, which is very common as payload, for a manipulator or for a UAV.

The rod is supposed to be a free-flying object, which can translate and rotate at will. So the configuration represents the position $\boldsymbol{p}$ in Cartesian coordinates of the center of mass of the rod and the orientation in terms of rotation matrix $\boldsymbol{R}$. An homogeneous matrix is sufficient to describe its pose:

$$M = \begin{bmatrix} \boldsymbol{R}, & \boldsymbol{p} \\ \boldsymbol{0}, & 1 \end{bmatrix}. \tag{2.2.1}$$

The rod has 6-DOFs: (x, y, z) for position and (θ, ϕ, η) Euler angles for rotation.

2.3 Algorithms Used

In this thesis the sampling based algorithms will be investigated, in particular the tree based algorithms.

The choice of that algorithm is justified by its numerous benefits. One of them is that the RRT is an efficient data structure and sampling scheme that quickly searches for solutions in high dimensional spaces.

In particular a special improvement of the RRT is going to be used in this thesis: the Bidirectional RRT, also known with the name *RRT-Connect*. This planner is tailored to problems in which there are no differential constraints and it is faster than RRT.

Two implementations of this algorithm, used in the experiments, will be presented below.

2.3.1 Bidirectional RRT

The first algorithm is a simple bidirectional RRT, inspired by [16]. The implementation is reported and explained in the following.

A bidirectional RRT builds two trees, one rooted in the initial configuration and the other in the final configuration. These two trees ($\mathcal{T}_i$ and $\mathcal{T}_g$ depicted respectively in blue and red in Figure 2.1) are maintained at all times until they become linked and a path is found. In each iteration, one tree is extended and an attempt is made to connect the nearest node of the other tree to the new vertex. Then the roles are reversed by swapping the two trees. In this way both the trees grow in the same manner and they have more or less the same size, namely number of nodes.

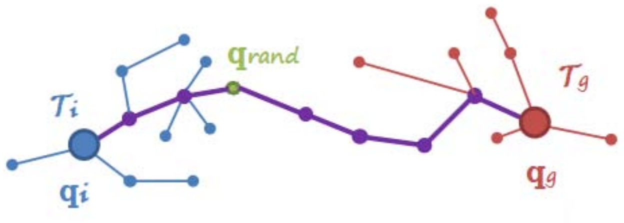


Fig. 2.1: Bidirectional tree based algorithm.

In the following is reported the pseudocode of the implemented simple bidirectional algorithm used in the experiments. Let's see in details all the steps.

First of all the initialization is necessary. Here the two trees are initialized by inserting the root node of the trees. In these phase also other data structures, explained in the next sections, are initialized. The novelty of this algorithm compared to canonical bidirectional RRTs is that for the same problem it is possible to derive different solutions. The initial loop of the algorithm allowed to find out `N` different solutions.

The difference between the various solutions is guaranteed by eliminating the junction point between the two trees after finding a path. This is done by the function `Delete()`.

For each of the `N` desired solution the algorithm goes on calculating until the boolean parameter `foundPath` becomes true. This parameter becomes true only when a path is finally found.

The core of the algorithm is inside the `while` cycle (line 10 of the pseudo code), where there are the tree expansion and the connection.

At the beginning of the cycle a new sample is achieved from the `Sample()` function. This function gives a new candidate configuration of the moving object according to different types of probability distributions (all of them are discussed in section 2.5).

Then, if the new configuration lies in free space, the `NearestNeighbor` function looks for the closest node of the tree (which is going to be extended) using Equation 1.5.1 as distance metric; otherwise a new configuration is sampled by `Sample()`. After some tests the values for the weight coefficients are chosen equal to $w_t = 0.2$ and $w_{rot} = 0.8$. In this ways the ρ_{rot} metric is taken more into account, i.e. among nodes at the same Euclidean distances you choose the one that differs least from the new point in terms of rotations. The function ρ_{rot} considered for the tests is the one written in Equation 1.5.4.

At first the `NearestNeighbor` function evaluated the closest node inside a sphere centered in q_{rand} and radius 1000. Then, if the sphere does not contain nodes, the research is extended to all the tree. This strategy is intended to speed up the search for the candidate parent. The value 1000 for the radius was suggested by the distances between the initial and final configurations of two of the three experiments, it is half the distance.

To conclude the tree extension part is sufficient to say that if the straight line connecting q_{near} with q_{rand} is free (function `IsCollisionFree()`), q_{rand} is added to the growing tree. Otherwise the configuration is discard and the algorithm starts again from line 10.

When the growth of the current tree has success, the connection is attempted. The `CONNECTION` function searches for all the points of the other tree within a sphere centered in q_{rand} and then it proves to connect them with q_{rand}. The connection has success if a free path exists between q_{rand} and one of these configurations. If the sphere is empty the nearest configuration of the other tree is found and tested. If no connection runs, the algorithm swaps the trees and extends the other repeating the same procedure.

Otherwise if there is connection, the q_{rand} configuration is added in both trees. This is very important in order to recover the final path, which links together q_i and q_g (`PATH` function). Starting from the q_{rand} configuration and asking each node for its parent in both trees, you can easily get a list with all the configurations of the desired path.

```
Bidirectional RRT Algorithm

Input: $q_i$, $q_g$, N
Output: Trees $\mathcal{T}_i$, $\mathcal{T}_g$ and PATH($\mathcal{T}_i$, $\mathcal{T}_g$)

$\mathcal{T}_i \leftarrow$ Initialize{$q_i$};
$\mathcal{T}_g \leftarrow$ Initialize{$q_g$};
for n = 0 to N do
    while (!foundPath)
        $q_{rand} \leftarrow$ Sample();
        $q_{near} \leftarrow$ NearestNeighbor($\mathcal{T}_i, q_{rand}$);
        if IsCollisionFree($q_{near}, q_{rand}$) then
            $\mathcal{T}_i$.Insert($q_{rand}$);
            if (CONNECTION($\mathcal{T}_g, q_{rand}$))
                return PATH($\mathcal{T}_i$, $\mathcal{T}_g$);
        SWAP($\mathcal{T}_i, \mathcal{T}_g$);
    Delete($q_{rand}$);
```

2.3.2 Augmented Bidirectional RRT

In this paragraph a different Bidirectional RRT is introduced. The variations of this implementation compared to the one of the previous section derive from two simple observations:

- Whenever the simple B-RRT detects a collision, checking the reachability of a new sample q_{rand}, it discards the configuration and all information on where the path is free before the collision;
- The free edges that connect the tree nodes are not taken into account when looking for new connections of adjacent configurations.

So considering these remarks, two changes were made.

During the edges collision checking, if a collision occurs, the q_{rand} is discard but the configuration q_s, which stands just before the collision, is added to the tree. In this way the part of the free branch is not thrown. These techniques is depicted on the left of Figure 2.2.

However on the right there is the second upgrade. When a new node is added to the tree also other nodes belonging to the connection branch are added to it. How many configurations to take in the link is a big issue. Taking too close configurations implies a large increase in number of nodes and the computation time, on the contrary, taking few of them does not bring an advantage in finding near configurations. After some tests the distance between nodes belonging to same link is put equal to 50 cm.

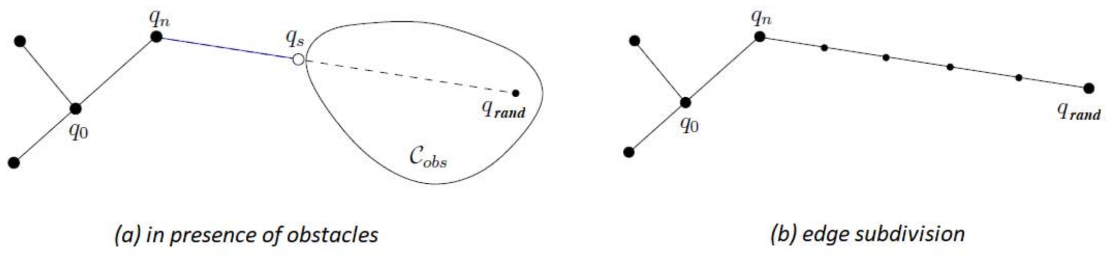


Fig. 2.2: Augmented bidirectional RRT algorithm.

These modifications require some changes to the code written in the previous section. In particular the second change involves some tricks in the `Delete` method. It is no more sufficient to eliminate only the connection node, but it is necessary to delete all its children too. Otherwise there are nodes which have a null parent that are not the root node. This fact could be very dangerous for the successive iterations of the algorithm.

The `CONNECTION` function changes too. The connection is attempted within a sphere centered in the last valid node added to the tree (i.e. q_{rand} or q_s). All nodes in the sphere are tested for connection not only q_{rand}. This was done in order to increase the probability of having valid connections and because there could be more new points in the tree.

The two spheres, the one used for connection and the other used for the `NearestNeighbor` function, have radius equal to 500 mm. It is half the radius of the simple bidirectional algorithm because here there could be more nodes to test and the algorithm could become slower.

2.4 Space Representation

Even if the B-RRTs adopted in this thesis do not need an explicit representation of the space, some data structures that give a space representation where used in order to improve the performances.

In the following two data structures that represent the 3D-space in which the object is moving, are explained and analyzed.

The first data structure is a *data grid*, it is useful to increase in terms of velocity and efficiency the performances of the algorithms.

Instead the second, the *octree* where used to derive a new sampling techniques in order to investigate portions of space not already visited. Indeed the B-RRT discovers the world moving in a random way. This approach in some case is very bad and takes a lot of time. One example is Environment 3, presented in section 3.3. When there are holes and narrow passages the B-RRT algorithm struggles to find a solution. So to help him move, other sampling strategies need to be used and one of them requires more information about the workspace, which are located within the octree.

2.4.1 Data Grid

A data grid is a two-dimensional matrix in which each element is a list of arrays containing two elements: a triplet and a payload. The triplet contains the Cartesian coordinates (x, y, z) which identifies the position of the payload in the workspace, the payload is what it is desired to save in such position. In this case the payload is the homogeneous matrix representing the object.

In Figure 2.3 is shown in green the principal matrix, instead in yellow and red are reported two triplets of the array list. With the columns and rows are identified the x and y components translated of half of the size of the grid. Instead the z axis is saved in the array list.

The final data grid results to seem a three-dimensional matrix, if all the arrays are considered as depth of the two-dimensional matrix. However this data structure was adopted because it is faster and leaner to implement and use than the three-dimensional matrix.

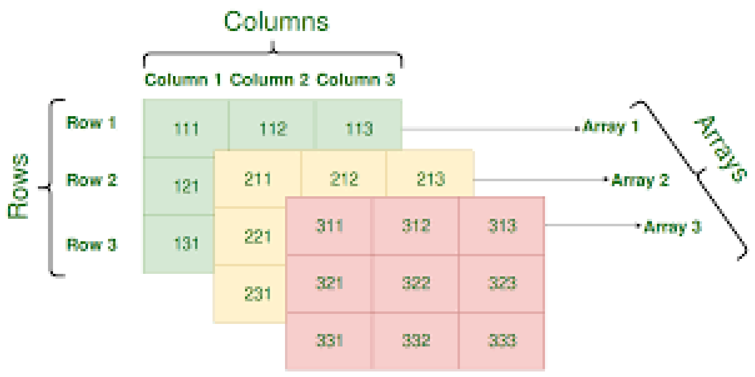


Fig. 2.3: Data grid 3D-space partitioning procedure.

The data grid represents only a portion of the 3D space, a cube of size $6m \times 6m \times 6m$. This choice was done to limit the sampling in a space sufficiently big but not infinite.

The data grid initialization is easy, it corresponds to the two-dimensional matrix initialization. The elements needed are:

the size of the cell of the data grid, which is a small cube of side 1 mm. It represents the maximum resolution of the data grid;

the origin of the data grid, which is the smaller corner (-3 m, -3 m);

the grid size (6 meters).

This data structure is used to search for the nearest node of a new node configuration, within the `NearestNeighbor` function. At first, this function looks for the nearest node inside a sphere and the data grid is able to quickly answer the question that asks which points are inside the sphere.

2.4.2 Octree

The octree is a data structure in which each internal node has exactly eight children. This data structure is used to represent the 3D-space by recursively subdividing it in eight octants, as it is depicted in Figure 2.4.

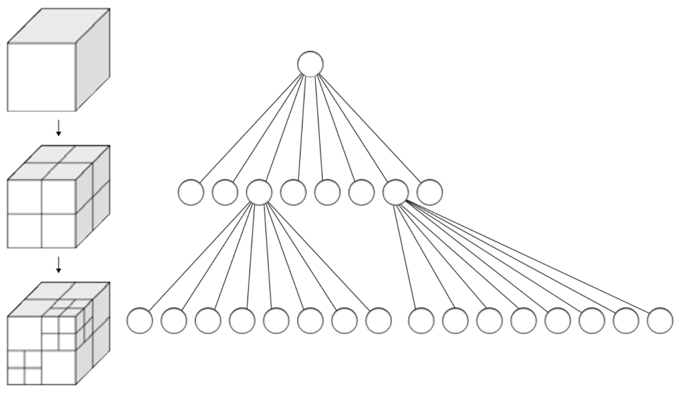


Fig. 2.4: Octree 3D-space partitioning procedure.

Each node contains information on the region it represents (storing the smallest and largest corner) and the list of all the points that belong to its region.

An octants splits in its eight children when there is more than one point inside its region. However it is not possible to divide the space into infinity. For this reason, an octant can divide itself only if the depth of the tree is lower than a specified threshold or the side of its region is greater than another specified threshold. Once the lower or upper limit is achieved the octant can contain more than one point.

This data structure was introduced in order to realize a sampling distribution which will be discussed in section 2.5.

The octree initialization must consider the size of the space that it has to represent. As for the data grid, the space is a cube of size $6m \times 6m \times 6m$, so the root of the octree must have the lower edge equal to (-3m, -3m, -3m) while the upper one (3m, 3m, 3m).

2.5 Sampling Strategies

The efficiency of the algorithms proposed are strictly related with the metrics, but also with the sampling strategies. Four sampling strategies have been adopted and mixed together in this work in order to increase the performances. They are all discussed in the following:

- A uniform distribution;
- A polar distribution;
- An "informed" distribution;
- An Octree based distribution.

The first two have been already introduced in section 1.4. They have both a uniform distribution of the Cartesian coordinates, which are sampled uniformly within a cube of size $6m \times 6m \times 6m$. Instead they differ in sampling of the rotations.

The uniform strategy chosen is the one reported in Algorithm 2, which uses the quaternion representation for rotations. However in the polar distribution the rotations have been realized with independent Euler angles sampled between $[-\pi, +\pi]$. As discussed in section 1.4 it is a distribution heavy biased through polar regions.

The last two distributions need more explanations. They could have uniform or polar distribution of rotations, as preferred, instead they change the Cartesian coordinates sampling.

However before talking about them, it is worth dwelling on how a PC can generate random points.

Creating a random sequence is not trivial, a PC can only have a deterministic behavior, therefore it can only give a pseudo-random number. All the work presented was done in the *C sharp* language and one of the fundamental classes used is the *Random* class.

This class represents a pseudo-random number generator, i.e. a device that produces a sequence of numbers that satisfy certain statistical requirements of randomness.

Pseudo-random numbers are chosen with equal probability from a finite set of numbers. The numbers chosen are not completely random because a mathematical algorithm is used to select them, but they are random enough for practical purposes.

The current implementation of the Random class is based on a modified version of Donald E. Knuth's subtractive random number generator algorithm. More details are in [17].

2.5.1 "Informed" Distribution

As said in the previous paragraph this distribution could have two possibilities for rotations distribution (polar or uniform) and it has not uniform Cartesian distribution. The idea behind this distribution is the subsequent: instead of searching for the Cartesian coordinates within the big cube of size $6m \times 6m \times 6m$, the coordinates are sampled within a cube of small size.

In details the algorithm follows these steps:

- A node of the growing tree is selected from a uniform distribution over all nodes of the tree;
- then the distance between the selected node and the root of the other tree is calculated. If this distance is bigger than the distance between the two roots it is discard and the initial distance (from one root to the other) is set as cube's side. Otherwise the calculated one becomes the cube's side;
- the center of the cube is located in the center of the segment passing through the two points considered (the sampled and the root);
- the three Cartesian coordinates are uniformly sampled within this new cube.

In this way the growing tree tends to reach the other tree and does not explore too far regions from the goal.

The name informed recalls a sampling technique present in the literature, which tries to concentrate the sampling in a specific area between the initial and final state.

2.5.2 Octree Based Distribution

The octree, already introduced in section 2.4.2, was used in this work to proposed a new sampling technique for the Cartesian coordinates.

This new procedure aims to search for new nodes in regions where the algorithm has not yet gone. To do this, it is very important to take the memory of where the nodes were sampled and the octree data structure can help with its representation of the 3D-space.

Every time a new configuration is inserted in the tree structure, it is also inserted in the octree. In this way small regions of the octree identifies a huge sampling in the area considered. Instead big region provides poor sampling.

It is simple to understand that if the aim is to search in non-sampled regions, the sampling must be performed in large octants. Between the largest regions the one selected is chosen randomly.

It is worth noting that no connection points are removed from the octree, as smaller regions are preferable for the route already found. In this way the new route searches for other regions and becomes different from the previous one.

2.6 Collision Detection

Collision detection is the operation in which the algorithm takes the most time. It is a very delicate step to achieve the goal of having a free path.

Although the sampling-based path planners proposed to use a collision detector like a black box, which says whether the current configuration is colliding or not. However it is worth understanding how this *collider* works.

Having a good and efficient collider greatly increases the performance of the whole algorithm.

In the initialization all the objects in the space were analyzed, the bounding boxes and the mesh regions are determined for each of them. The mesh used is a triangle one, so the faces are small triangles determined according to the shape of the object.

In Figure 2.5 it is represented an example of triangular mesh regions decomposition and in Appendix B there are some method for 3D object representation.

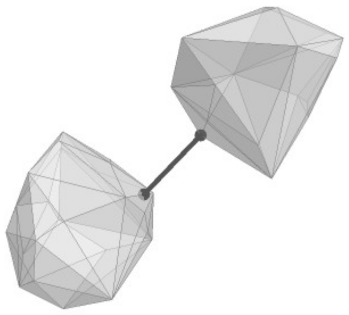


Fig. 2.5: Triangular mesh regions.

After the mesh regions storage the collider checks the collisions. In the beginning all the objects were tested in pairs, then during the algorithm only the moving objects are taken into account.

If two objects intersect then there are two possibilities: the collision checking could stop when the first colliding triangle is found or it could continue looking for all the triangles in collision. In this thesis the first solution was done, it is not necessary to find all the triangles in collision, it is a waste of time.

However testing all the triangles to find the first in collision could be a very long operation, i.e. there exist objects that have millions of triangles. For that reason some optimization techniques were adopted.

A simple and effective approach is to group triangles into small regions, which divide the object into several parts. Then the collision test is done between these areas. This method speeds up the collision checking procedure.

Chapter 3

Experimental Results

In this chapter all the experimental results, obtained in three different environments, will be presented:

- A simple situation with one big obstacle;
- A navigation problem, in which the moving object must cross many objects scattered in space;
- And in the end the most difficult environment, where there are narrow passages that the rod must go through.

For each environment the same tests were performed, that involves the two tree-based algorithms (discussed in section 2.3.1 and 2.3.2) and the different sampling strategies (reported in 2.5).

The sampling strategies already discussed can be easily summarized in the following table:

	Uniform	Not Uniform
Uniform	Real Uniform	Polar
Octree	Octree-Uniform	Octree-Polar
Informed	Informed-Uniform	Informed-Polar

Tab. 3.1: Sampling Distributions.

On the first row all rotational sampling techniques proposed are reported, instead on the first column all the Cartesian space coordinates' sampling strategies used.

The Cartesian sampling strategies are used separately but also mixed together in three different manners:

- *Separated Generation* searches for the first solution with Real Uniform or with Polar, instead the second one comes from the Octree-based distributions;
- *Mixed Generation* uses the Octree-based sampling techniques together with the Uniform and Polar distribution;
- *Very Mixed Generation* mixes all the Cartesian sampling distributions together.

The Mixed and Very Mixed generations randomly select at each iteration which sampling strategy to adopt among their candidates.

Both the algorithms look for two solutions ($N = 2$) and what you want to observe is: the time required to compute both the solutions, the numbers of nodes of the two trees and the length of path before and after the simple smoothing procedure explained in section 1.6.

The following sections have the same structure: firstly there is the explanation of the current environment, then all experiments are reported in two subsections, one with uniform rotational sampling and the other with polar rotational sampling. The sampling strategies used are the three above and the *Informed sampling* used alone. Four different sampling strategies applied to the two RRT algorithms for a total number of tests equal to 16 for each environment. One hundred samples were collected and analyzed for each test. Some additional graphical results are reported in Appendix D.

In the last section 3.4 there is a comparison between all experiments and some general considerations.

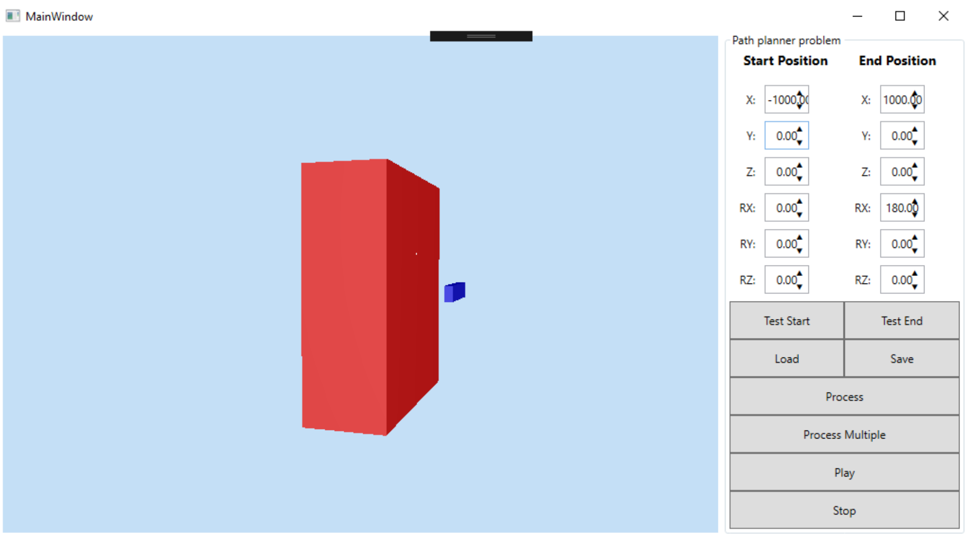
All this thesis is done in a plugin of a new software, recently developed in Euclid Labs S.r.l called *Vostok Studio*. It is a innovative software which simulates the real time behaviour of the robot. The Vostok Studio interface of the plugin is explained in Appendix C.

Measurements are done with my personal computer, an Asus PC (Operative system Windows 10) with CPU intel core i7 4510u up to 3.1 ghz and memory of 6 GB.

3.1 Environment 1: Simple Problem with One Big Obstacle

This is the simplest path planning problem, in this case the algorithms search for a path avoiding collisions with one big red obstacle shown in Figure 3.1. The obstacle is $1m \times 3m \times 3m$ and it is centered in the origin of the workspace. The moving object (depicted in blue), which dimensions are $0.1m \times 1m \times 0.2m$, has to move from the initial pose $p_i = [-1m, 0, 0]$ and $(\theta_i, \phi_i, \eta_i) = (0, 0, 0)$ to the final pose $p_g = [1m, 0, 0]$ and $(\theta_g, \phi_g, \eta_g) = (\pi, 0, 0)$.

It is worth to remember that the considered workspace is not infinite, but it is a cube of side $6m$ centered in the origin.



	Start Position		End Position
X:	-1000.00	X:	1000.00
Y:	0.00	Y:	0.00
Z:	0.00	Z:	0.00
RX:	0.00	RX:	180.00
RY:	0.00	RY:	0.00
RZ:	0.00	RZ:	0.00

Fig. 3.1: Environment 1: One big obstacle avoidance.

In the following two subsections all relevant results of computational time, path length and number of nodes are reported. The first section groups all experiments performed with non-uniform sampling of rotations, while the second section contains all the experiments with uniform sampling of rotations.

3.1.1 Uniform Rotational Sampling

The computational time required to find a solution is strictly related on how much the trees grow.

So before introducing all computational times, just have a look on the size of the two trees.

In Table 3.2 and Table 3.3 are reported the mean number of nodes and the standard deviation of all trails performed with the Simple RRT and the Augmented RRT respectively using all the sampling strategies.

	First mean	Stand. Dev.	Second mean	Stand. Dev.
Very Mixed	7.65	3.58	9.93	4.09
Mixed	7.04	3.04	9.02	3.51
Separated	6.55	2.79	8.60	3.28
Informed	8.46	4.32	12.14	4.75

Tab. 3.2: Number of the nodes of the two trees with the simple RRT algorithm.

	First mean	Stand. Dev.	Second mean	Stand. Dev.
Very Mixed	80.73	27.87	96.90	27.23
Mixed	69.66	25.27	81.53	25.63
Separated	64.95	21.72	81.9	19.75
Informed	79.11	26.11	92.65	27.35

Tab. 3.3: Number of the nodes of the two trees with the augmented RRT algorithm.

As expected, in the second table the number of nodes belonging to the trees is higher than the ones in the first table.

Although the augmented algorithm provides multiple nodes (due to the changes applied to tree growth), both algorithms have a reduced number of nodes, not more than a hundred. What it is expected to be observed is that the computational time should be small, due to the simple complexity of the problem.

To proof that statement in the following three-dimensional histograms all solutions found for each sampling method are grouped together.

The time required to find the first solution (depicted in blue) is compared with the computational time of the second one (orange histogram in Figure 3.2).

It is possible to notice that in all the histograms the time required to find the second solution is always slower than the first one. This is due to the fact that not the whole tree is deleted when the first path is found, but only one or few points are removed.

However, all tests claim that the simple path planning problem could be easily solved with all the methods presented. Therefore, on average the first solution is found in 3 or 4 seconds, while the second one is about 1 second after the first one. These results have been very satisfied.

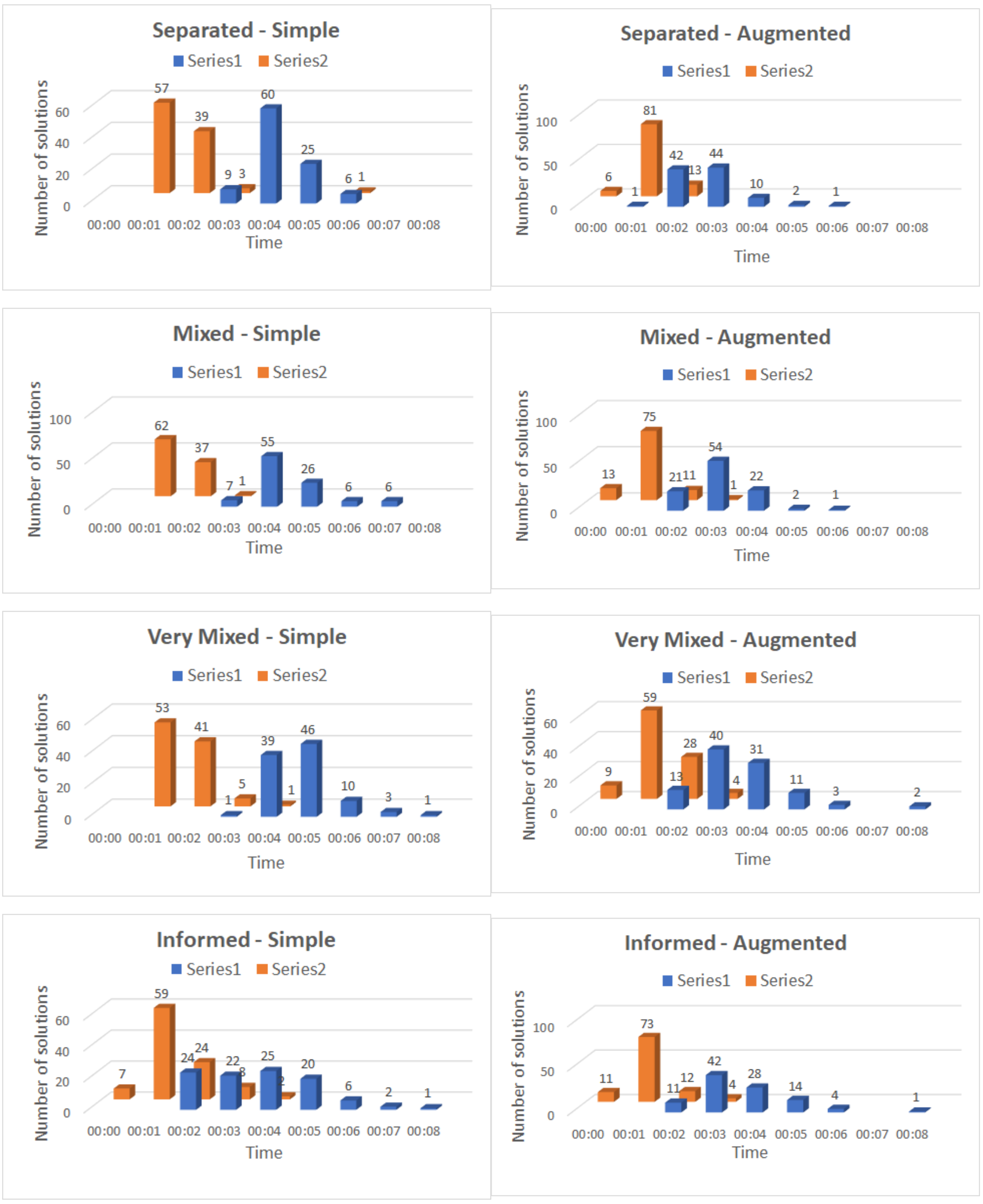


Fig. 3.2: Time required for the computation of the first (light blue) and second solution (orange) with all the sampling techniques on both the algorithms (Simple RRT on the left, Augmented RRT on the right).

Then in Figure 3.3 the four sampling techniques are directly compared. In this case only the first solution is taken into account, because this is the most complicated to find out. The four sampling strategies but also the RRT algorithms have the similar performances.

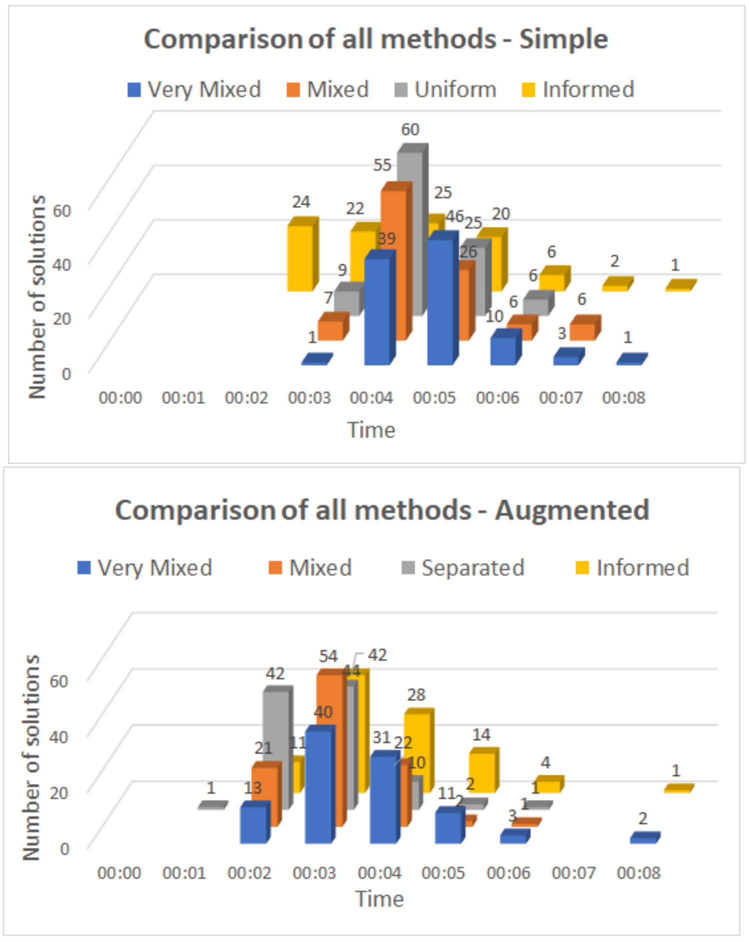


Fig. 3.3: Comparison of the results of the first solution with all sampling strategies on the top with the Simple RRT, instead on the bottom with the Augmented RRT.

Another important aspect that will be analyzed, is the path length. Just for simplicity it is considered only the Cartesian distance function to calculate it, i.e. the Euclidean norm.

However before considering the final path length it is convenient to discuss the smoothing. In Table 3.4 and Table 3.5 it is possible to observed the first difference between the two RRT algorithms.

The Simple one after the smoothing procedure is able to reduce on average the length of the path of a 9%, instead the Augmented can reduce of a factor of 19%. Considering also the standard deviation the Simple can have a maximum reduction of 19%, the Augmented of 30%.

The structure of the initial path is the most influential parameter to decide if the smoothing is useful or not, but having a larger number of nodes to compare provides smoother solutions.

Lengths of the final path are shown in Appendix D, while the following tables (Tab.

3.6 and 3.7) show the average and the standard deviation obtained in each experiment.

	First % Reduction	Stand. Dev.	Second % Reduction	Stand. Dev.
Very Mixed	9.12 %	9.96	10.01 %	9.99
Mixed	8.22 %	11.01	10.86 %	9.89
Separated	7.23 %	9.45	10.47 %	11.28
Informed	10.91 %	11.21	14.04 %	12.14

Tab. 3.4: Percentage reduction from first to second solution with the Simple RRT algorithm.

	First % Reduction	Stand. Dev.	Second % Reduction	Stand. Dev.
Very Mixed	18.87 %	10.79	21.72 %	10.80
Mixed	18.00 %	11.87	19.77 %	10.11
Separated	19.95 %	11.19	21.00 %	11.29
Informed	19.80 %	9.75	22.62 %	10.37

Tab. 3.5: Percentage reduction from first to second solution with the Augmented RRT algorithm.

	Mean Path 1 [mm]	Stand. Dev.	Mean Path 2 [mm]	Stand. Dev.
Very Mixed	9493.40	1453.24	9480.78	1424.51
Mixed	9516.35	1381.30	9604.46	1401.33
Separated	9559.11	1380.87	9802.81	1275.40
Informed	9543.93	1420.44	9783.07	1535.43

Tab. 3.6: Final length path after the smoothing with the Simple RRT algorithm.

	Mean Path 1 [mm]	Stand. Dev.	Mean Path 2 [mm]	Stand. Dev.
Very Mixed	8074.54	1071.23	8110.41	1160.98
Mixed	8239.44	1148.76	8088.01	1169.01
Separated	8138.23	1106.80	8362.08	1153.27
Informed	7647.65	1028.00	7540.52	1068.53

Tab. 3.7: Final length path after the smoothing with the Augmented RRT algorithm.

3.1.2 Polar Rotational Sampling

The structure of this paragraph is similar to the one already read but the rotational sampling is polar now. At first the number of nodes of the two trees is considered for each experiment.

It is possible to see from Table 3.8 that the mean for all sampling in Simple RRT is around 7 nodes for the first solution. The second solution employs the construction of trees of size 9 or 10 nodes, only 3 or 4 nodes more than the first solution. This means that the second solution it is immediately found after the first.

The standard deviation is small even if it is about half the number of nodes. Instead from Table 3.9 it is easy to see that on average 70 configurations in both trees are required for the first solution, while the second one adds about 15 configurations. The standard deviation is about one third the mean in both the solutions. These results were comparable to those found in the previous subsection. The calculation time graphs of Figure 3.4 show

	First mean	Stand. Dev.	Second mean	Stand. Dev.
Very Mixed	7.82	3.76	10.38	4.37
Mixed	7.03	2.51	9.02	3.06
Separated	6.64	2.45	8.44	2.79
Informed	8.58	4.10	11.45	4.21

Tab. 3.8: Number of the nodes' tree with the simple RRT algorithm.

	First mean	Stand. Dev.	Second mean	Stand. Dev.
Very Mixed	82.33	26.68	98.09	27.64
Mixed	71.00	23.04	81.94	21.79
Separated	64.01	21.54	83.35	21.69
Informed	73.97	29.25	90.34	29.07

Tab. 3.9: Number of the nodes' tree with the augmented RRT algorithm.

how much faster the second solution is compared to the first. It takes on average 1 second to recover the second solution after the previous one. However the first is found after 2 or 3 seconds.

In Figure 3.5 the times required to find the first solutions are shown. The Simple RRT implies 2 or 3 seconds to find the solution, except in the case of Informed sampling (4 seconds are required on average).

Augmented algorithm has similar results, the mean is about 3 seconds with all the sampling strategies. Experiments confirm the validity and effectiveness of all algorithms to solve this type of problem. However as already seen in the previous section, the only difference between the Simple and Augmented algorithms consists on the path length. The smoothing acts better on tests where the Augmented algorithm is used. The percentage of reduction is shown in Table 3.10 and 3.11. To be complete, there are two tables containing

	First % Reduction	Stand. Dev.	Second % Reduction	Stand. Dev.
Very Mixed	8.63%	9.83	10.05%	9.69
Mixed	9.58%	11.78	10.11%	10.37
Separated	8.42%	10.24	11.15%	11.45
Informed	9.22%	10.62	12.63%	11.57

Tab. 3.10: Percentage reduction from first to second solution with the Simple RRT algorithm and all the sampling techniques.

	First % Reduction	Stand. Dev.	Second % Reduction	Stand. Dev.
Very Mixed	20.01%	11.30	21.73%	11.33
Mixed	19.80%	10.67	22.34%	10.83
Separated	19.26%	10.30	20.48%	11.69
Informed	16.24%	10.54	20.75%	10.01

Tab. 3.11: Percentage reduction from first to second solution with the Augmented RRT algorithm and all the sampling techniques.

the average and the standard deviation of the length of the final path. On average the solutions of Augmented RRT are about one thousand less the ones calculated with Simple RRT. In general, first and second paths in both tables (3.12 and 3.13) have similar length.

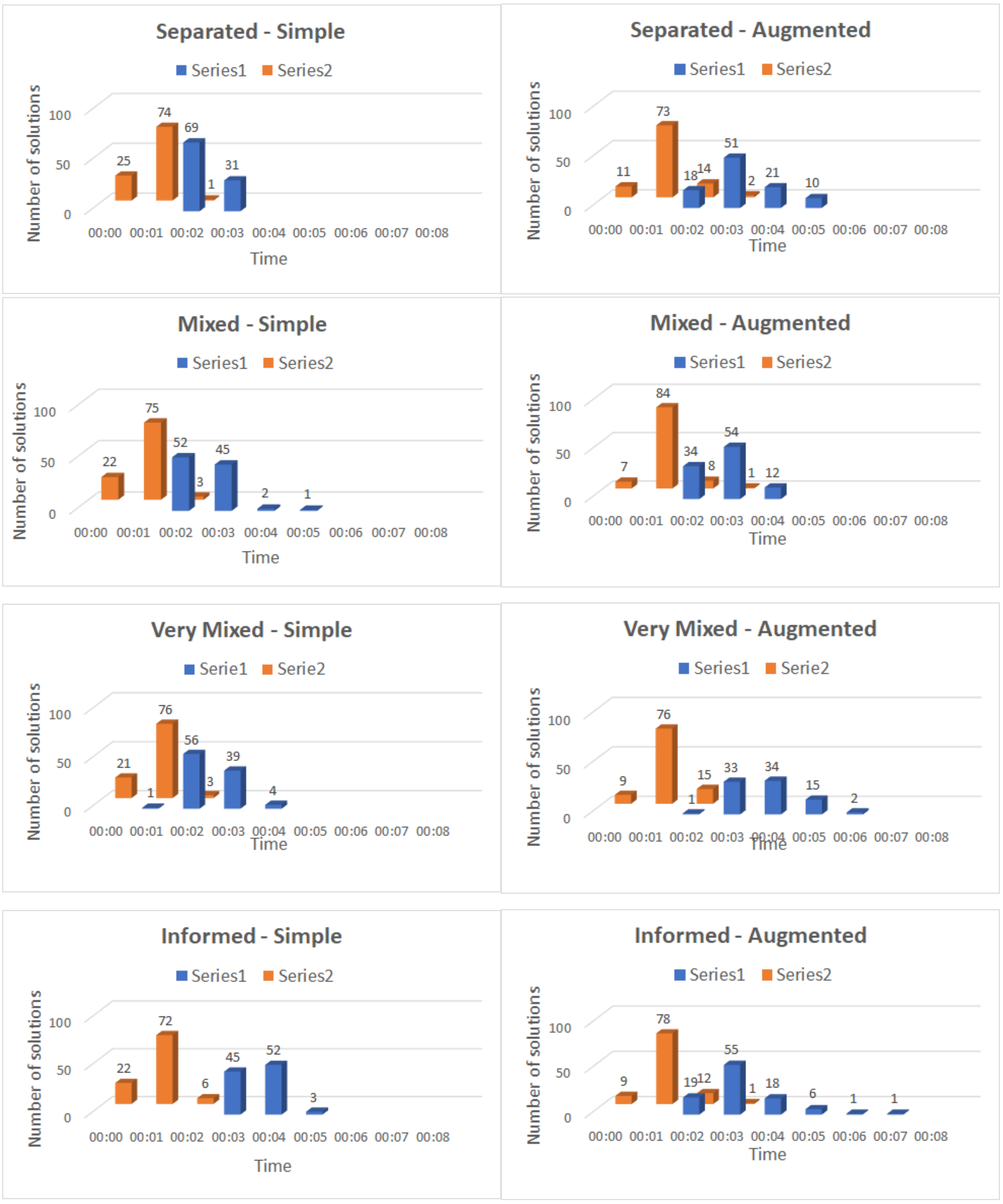


Fig. 3.4: Time required for the computation of the first (light blue) and second solution (orange) with all sampling techniques on both the algorithms (Simple RRT on the left, Augmented RRT on the right).

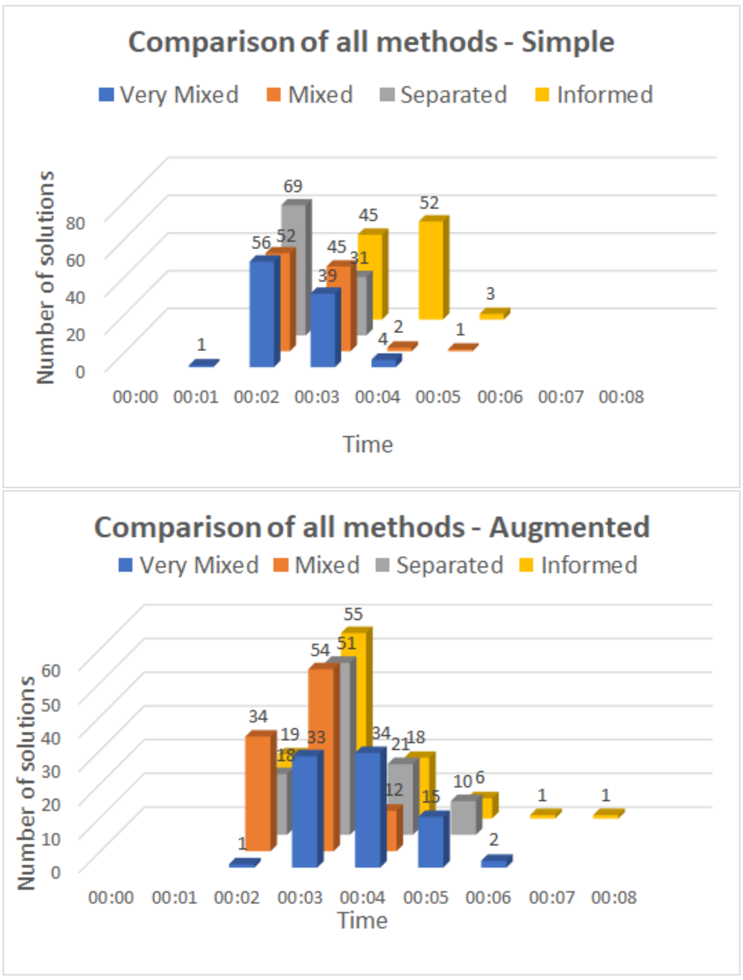


Fig. 3.5: Comparison of the results of the first solution with all the sampling strategies on the top with Simple RRT, instead on the bottom with Augmented RRT.

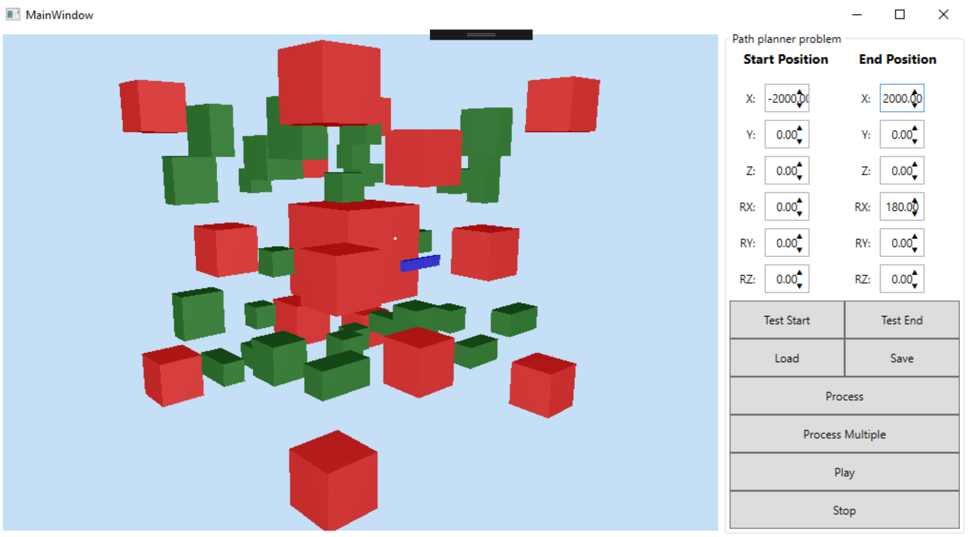
	Mean Path 1 $[mm]$	Stand. Dev.	Mean Path 2 $[mm]$	Stand. Dev.
Very Mixed	9692.30	1193.82	9634.52	1325.60
Mixed	9491.21	1292.30	9680.67	1326.63
Separated	9771.64	1181.63	9590.67	1413.905
Informed	9373.51	1500.31	9288.23	1340.18

Tab. 3.12: Final length path after the smoothing with the Simple RRT algorithm.

	Mean Path 1 $[mm]$	Stand. Dev.	Mean Path 2 $[mm]$	Stand. Dev.
Very Mixed	7851.73	1005.09	7937.62	1014.97
Mixed	8408.76	1187.39	8251.78	1234.44
Separated	8067.36	1073.64	8418.11	1147.23
Informed	8054.10	1159.17	7841.00	1294.95

Tab. 3.13: Final length path after the smoothing with the Augmented RRT algorithm.

3.2 Environment 2: Navigation Problem



Path planner problem

	Start Position		End Position
X:	-2000.00	X:	2000.00
Y:	0.00	Y:	0.00
Z:	0.00	Z:	0.00
RX:	0.00	RX:	180.00
RY:	0.00	RY:	0.00
RZ:	0.00	RZ:	0.00

Fig. 3.6: Environment 2: Navigation problem set up.

This is the classic navigation path planning problem, which consists in finding a collision-free route that passes through red and green obstacles shown in Figure 3.6. In this case there are many obstacles scattered throughout the entire workspace. They were randomly placed inside the space, without following a criterion. The moving object (depicted in blue), which dimensions are always the same, has to move from the initial pose $p_i = [-2m, 0, 0]$ and $(\theta_i, \phi_i, \eta_i) = (0, 0, 0)$ to the final pose $p_g = [2m, 0, 0]$ and $(\theta_g, \phi_g, \eta_g) = (\pi, 0, 0)$.

3.2.1 Uniform Rotational Sampling

In this section all results provided by a uniform rotational sampling are discussed and analyzed.

First of all let's consider the size of the trees. In the two tables 3.14 and 3.15 it is possible to notice that the mean total number of nodes is about 11 for the Simple RRT and about 110 for Augmented one. The augmented RRT is one hundred of nodes greater that the simple one. The second solution on average is found after 5 nodes with a simple RRT, instead after 30 vertices with augmented. The standard deviation of the samples is half the mean value in both algorithms. So the variability is quite large.

	First mean	Stand. Dev.	Second mean	Stand. Dev.
Very Mixed	11.82	5.99	15.47	6.17
Mixed	11.29	5.83	16.21	6.70
Separated	12.76	7.27	17.87	6.90
Informed	10.97	5.93	15.20	6.51

Tab. 3.14: Total number of the nodes in the two trees with the Simple RRT algorithm.

Regarding the computational time in Figure 3.7 it is shown a histogram for each

	First mean	Stand. Dev.	Second mean	Stand. Dev.
Very Mixed	116.46	64.84	143.04	71.01
Mixed	102.26	55.04	132.31	69.16
Separated	99.03	55.83	131.67	72.59
Informed	120.88	64.45	149.19	69.91

Tab. 3.15: Total number of the nodes in the two trees with the Augmented RRT algorithm.

test. On the left there are all graphs that use the simple RRT, instead on the right the augmented one. For each sampling strategy are considered both the solutions ($N = 1, 2$).

In order to provide a clear image, the horizontal axis is not composed by single values but by time intervals of length of 10 seconds. Only the last interval is open and solutions that take more than 2 minutes to find a route fall into it.

For all the two algorithms it is true that the Very Mixed technique is the best. With the simple RRT it takes 12 seconds and 4 more seconds to find respectively the first and second solution. Instead with the augmented it takes 40 seconds for the first and 14 for the second. It is preferable to the others also because it has few values in the highest intervals.

The mean and standard deviation of the computational time of the all the experiments are reported in Table 3.16. The time is reported in seconds. The two vertical lines in the middle of the Table divide the measurements done with the Simple RRT from the ones done with the Augmented. The term “Mean1” and “Mean2” indicate the mean between all the times used to calculate the first and the second solution respectively. While “S.Dev.” is the standard deviation of the previous mean.

	Mean1	S.Dev.	Mean2	S.Dev.	Mean1	S.Dev.	Mean2	S.Dev.
Very Mixed	12s	5s	4s	3s	40s	30s	14s	12s
Mixed	19s	13s	9s	8s	44s	39s	20s	30s
Separated	20s	18s	11s	14s	37s	32s	18s	20s
Informed	11s	5s	5s	3s	44s	37s	15s	17s

Tab. 3.16: Computational time mean and standard deviation of all the samples depicted in Figure 3.7. On the left the Simple algorithm, on the right the Augmented.

From the table it is possible to notice that with the Simple RRT also the Informed strategy has good performances, therefore in its graph of Figure 3.7 it has similar shape of the Very Mixed one.

Similarly for the Augmented RRT the Separated strategy could be also adopted with similar results of the Very Mixed.

What has already been said is clearly shown in Figure 3.8, where all the first solutions are compared directly. The blue and yellow histogram are the best for the simple RRT, instead the blue and the grey for the augmented.

The percentage reduction of the smoothing is summarized in the following tables Tab.3.17 and Tab.3.18. As for the previous experiment even in this case the presence of more nodes in the path provides a better smoothing. The reduction is about a 6% for the Simple algorithm, instead about 14% for the Augmented.

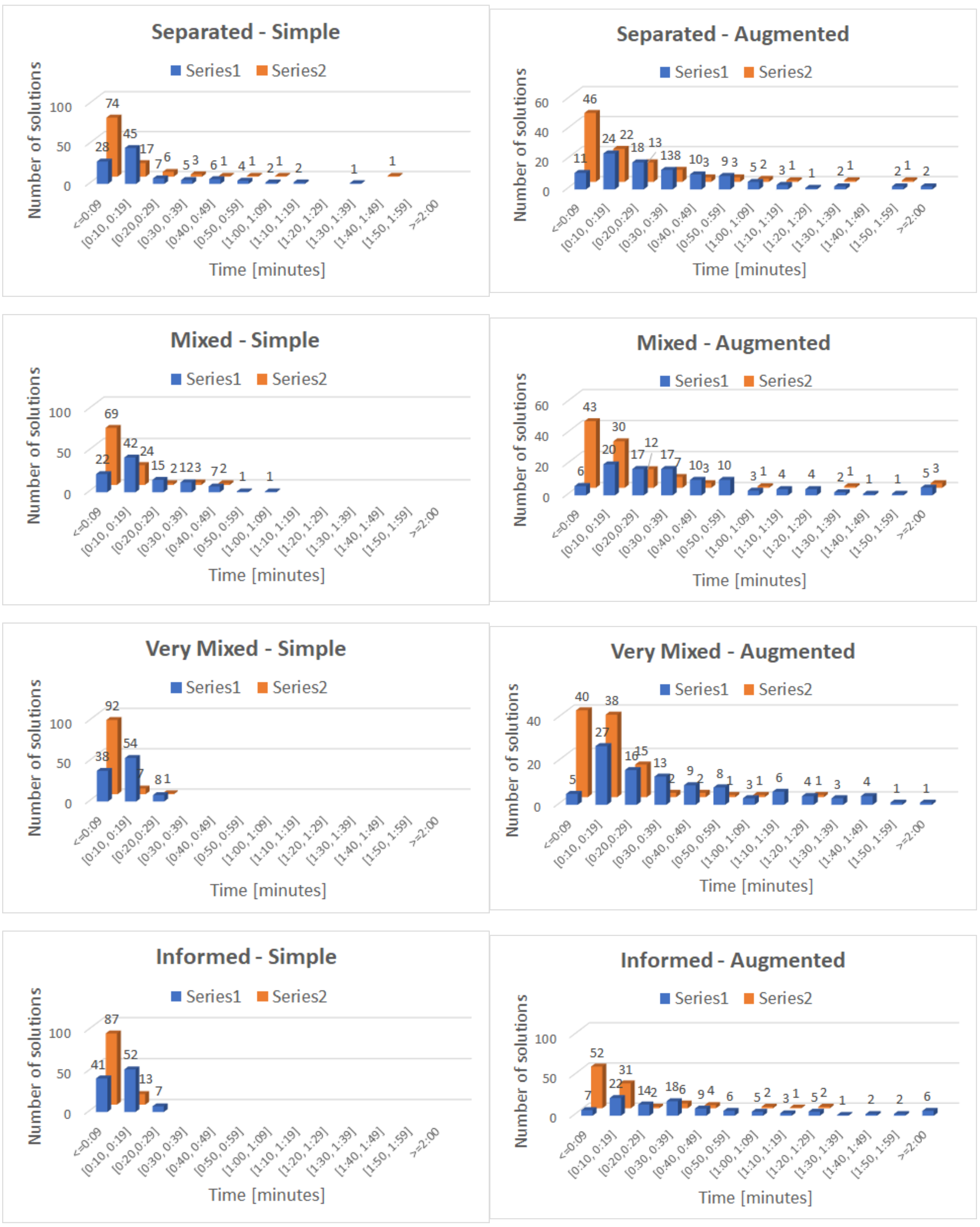
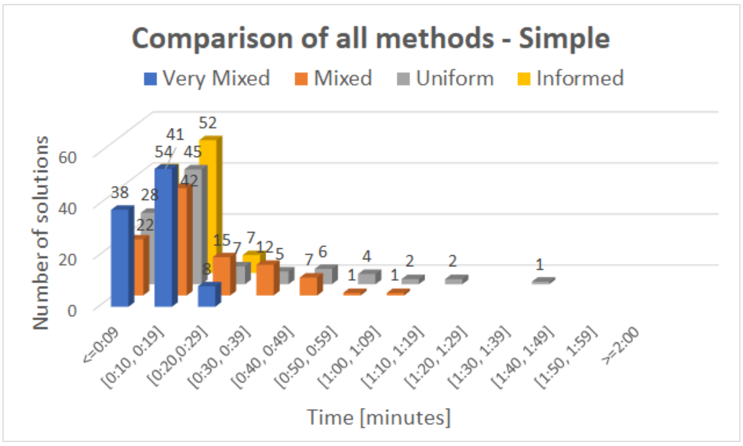


Fig. 3.7: Time required for the computation of the first (light blue) and second solution (orange) with all sampling techniques on both the algorithms (Simple RRT on the left, Augmented RRT on the right).



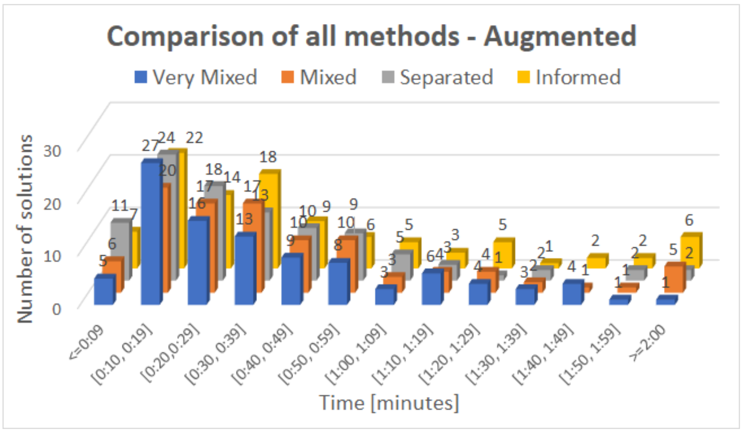


Fig. 3.8: Comparison of the results of the first solution with all the sampling strategies on the top with Simple RRT, instead on the bottom with Augmented RRT.

There is no surprise to find out that lengths of path calculates with Augmented RRT are on average smaller than the one find out with Simple RRT, as reported in Tab. 3.19 and 3.20.

	First % Reduction	Stand. Dev.	Second % Reduction	Stand. Dev.
Very Mixed	6.64%	9.02	7.94%	9.45
Mixed	5.30%	7.91	5.35%	6.94
Separated	7.14%	8.86	6.52%	8.09
Informed	5.24%	7.54	7.21%	9.32

Tab. 3.17: Percentage reduction from first to second solution with the Simple RRT algorithm and all the sampling techniques.

	First % Reduction	Stand. Dev.	Second % Reduction	Stand. Dev.
Very Mixed	13.86%	7.40	15.51%	7.87
Mixed	13.09%	8.51	15.45%	8.67
Separated	14.04%	6.96	16.96%	8.01
Informed	13.91%	7.74	16.26%	7.83

Tab. 3.18: Percentage reduction from first to second solution with the Augmented RRT algorithm and all the sampling techniques.

	Mean Path 1 [mm]	Stand. Dev.	Mean Path 2 [mm]	Stand. Dev.
Very Mixed	8872.38	1605.90	9226.36	2049.25
Mixed	8697.87	1610.34	9416.56	1911.47
Separated	9036.25	1832.02	9547.96	2107.41
Informed	9172.00	2250.60	9265.10	2267.41

Tab. 3.19: Final path length after the smoothing with the Simple RRT algorithm.

	Mean Path 1 [mm]	Stand. Dev.	Mean Path 2 [mm]	Stand. Dev.
Very Mixed	7612.31	1515.46	7640.45	1441.83
Mixed	7478.00	1120.92	7628.75	1482.64
Separated	7527.18	1416.01	7868.81	1634.86
Informed	7413.14	1421.87	7273.92	1240.54

Tab. 3.20: Final path length after the smoothing with the Augmented RRT algorithm.

3.2.2 Polar Rotational Sampling

In this paragraph all the results obtained with a polar rotational sampling are explained.

The topics to be discussed are always the same:
at first the number of nodes presented in the two trees is exposed, then the computational time and finally the final path length and smoothing action are analyzed.

Let's start from the beginning, in Table 3.21 and 3.22 the mean number of nodes and the standard deviation are reported. The Simple RRT finds the first path after the generation of twelve nodes and the second after sixteen nodes. Instead the Augmented needs for the first one hundred of nodes and for the second one hundred and forty.

This results are comparable and similar to the ones already found with the uniform rotational sampling.

	First mean	Stand. Dev.	Second mean	Stand. Dev.
Very Mixed	12.42	5.93	16.40	6.53
Mixed	11.04	6.57	16.27	8.57
Separated	13.29	7.34	18.38	8.67
Informed	11.29	6.52	14.96	6.87

Tab. 3.21: Number of the nodes' tree with the Simple RRT algorithm.

	First mean	Stand. Dev.	Second mean	Stand. Dev.
Very Mixed	117.26	62.08	148.93	60.07
Mixed	104.45	53.26	136.61	70.54
Separated	96.52	44.66	140.44	71.21
Informed	113.37	58.26	135.41	68.47

Tab. 3.22: Number of the nodes' tree with the Augmented RRT algorithm.

The computational time respects what already said for the number of nodes, in fact Table 3.23 has similar times as the ones reported in the previous paragraph. On average twelve seconds are needed to find the first path for the Simple RRT, instead 36 for the Augmented. This are good results.

All graphs of Figure 3.9 show the computational times. In this specific case the Very

	Mean1	S.Dev.	Mean2	S.Dev.	Mean1	S.Dev.	Mean2	S.Dev.
Very Mixed	14s	9s	6s	8s	48s	41s	22s	24s
Mixed	11s	5s	6s	4s	35s	29s	18s	23s
Separated	12s	4s	7s	5s	36s	28s	23s	28s
Informed	11s	5s	5s	3s	35s	27s	12s	14s

Tab. 3.23: Computational time mean and standard deviation of all the samples depicted in Figure 3.9. On the left the Simple algorithm, on the right the Augmented.

Mixed technique, even if it has not bad performances, is the worst strategy in both Simple and Augmented. Informed and Mixed are the best for Simple and Augmented too.

Simple algorithm is better than Augmented one in this case, not only for the average time performance but also because it has got few solutions that take lot of time to find a valid path. In Figure 3.9 simple and augmented graphs have the same horizontal axis. This choice was made to immediately identify the big difference in performance.

The graphical comparison of the performance in finding the first solution is shown in Figure 3.10.

The last element to discuss is the length of the final path and how the smoothing acts in these experiments. In Table 3.24 and 3.25 the percentage reduction is taken into account. The values inside the tables are very similar to the ones of the previous paragraph, where there are uniform sampling of rotations. When the Simple RRT builds the path, the smoothing is not so large, the reduction is about the 7%. However in case of Augmented algorithm the reduction is the double, about 14%. As expected the final length of the Augmented is smaller (it is about 7500) than the Simple (it is more the 8500). All measurements are shown in Table 3.26 and 3.27.

	First % Reduction	Stand. Dev.	Second % Reduction	Stand. Dev.
Very Mixed	7.01%	10.23	8.30%	9.29
Mixed	5.13%	7.73	6.02%	8.63
Separated	6.87%	8.91	7.07%	8.75
Informed	6.66%	10.23	7.88%	9.51

Tab. 3.24: Percentage reduction from first to second solution with the Simple RRT algorithm and all the sampling techniques.

	First % Reduction	Stand. Dev.	Second % Reduction	Stand. Dev.
Very Mixed	13.23%	8.02	15.00%	7.27
Mixed	13.76%	8.07	15.90%	7.92
Separated	13.55%	8.41	15.04%	8.95
Informed	13.19%	7.44	14.23%	7.14

Tab. 3.25: Percentage reduction from first to second solution with the Augmented RRT algorithm and all the sampling techniques.

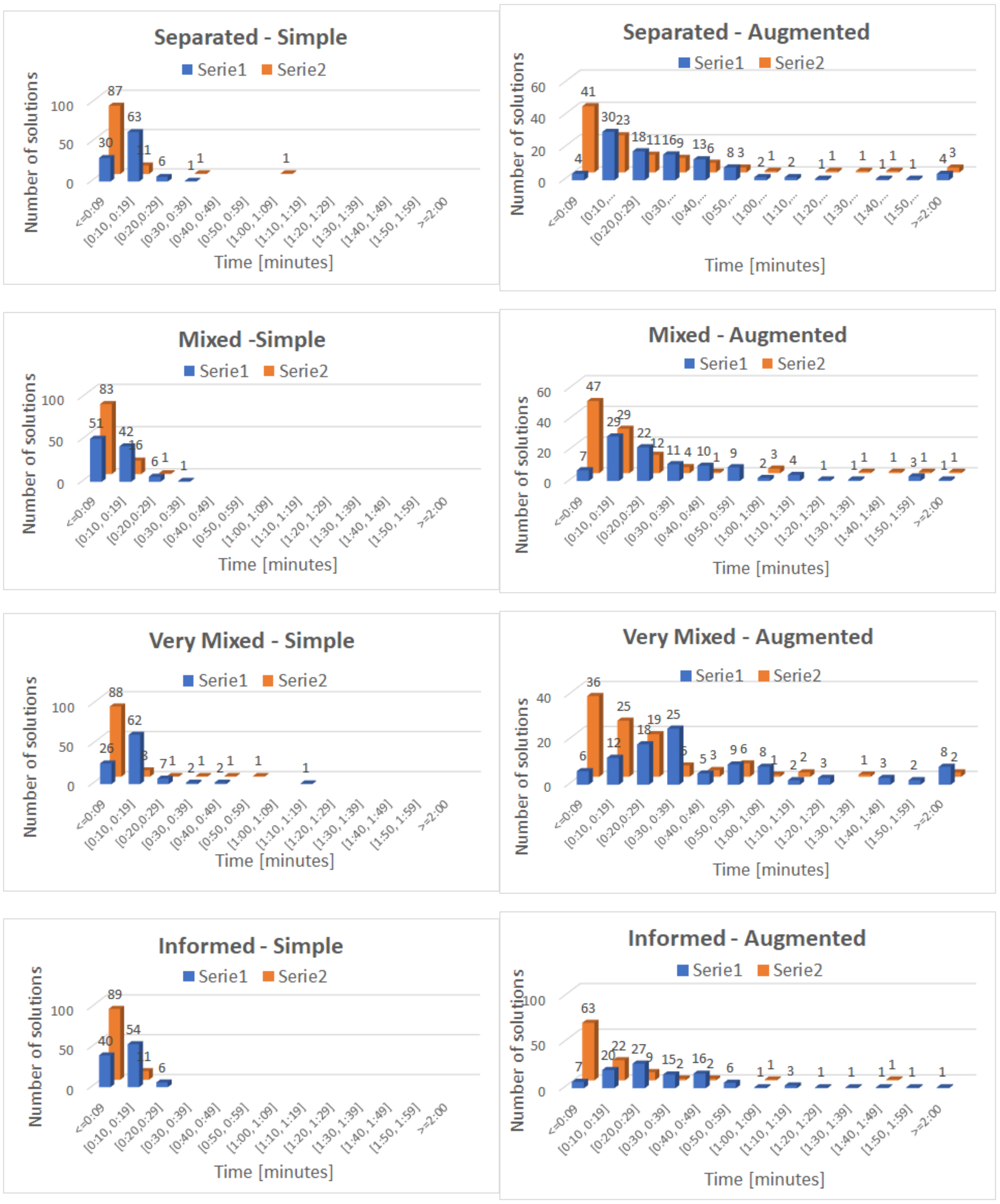
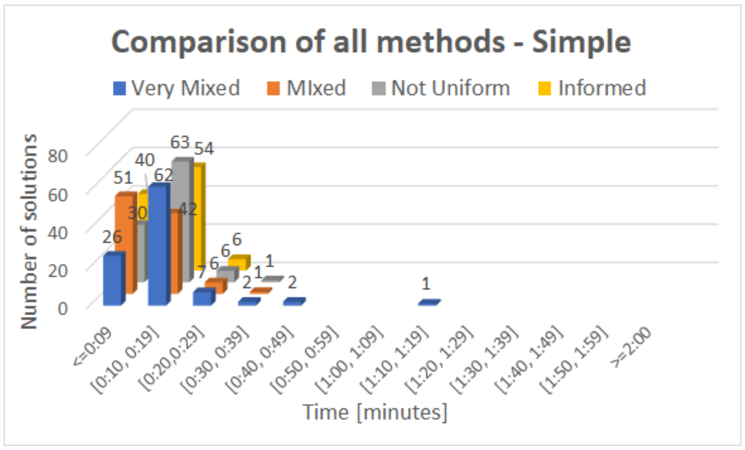


Fig. 3.9: Time required for the computation of the first (light blue) and second solution (orange) with all sampling techniques on both the algorithms (Simple RRT on the left, Augmented RRT on the right).



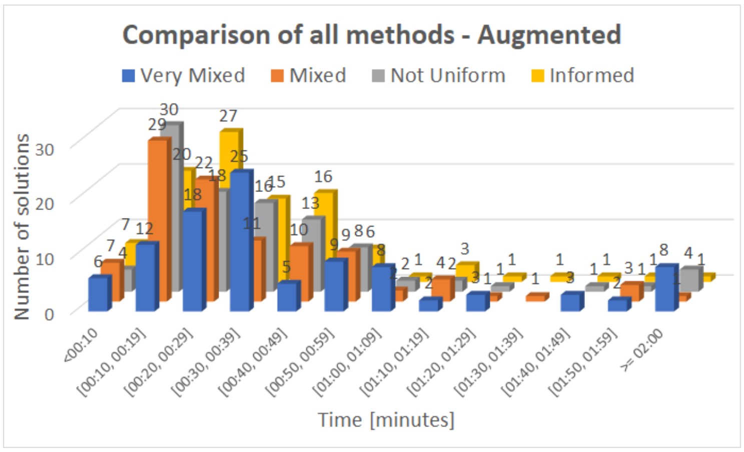


Fig. 3.10: Comparison of the results of the first solution with all the sampling strategies on the top with the Simple RRT, instead on the bottom with the Augmented RRT.

	Mean Path 1 $[mm]$	Stand. Dev.	Mean Path 2 $[mm]$	Stand. Dev.
Very Mixed	8826.97	1942.31	9687.81	2460.35
Mixed	9154.23	1924.26	9462.59	2017.83
Separated	9367.79	1982.98	9810.54	2378.09
Informed	8783.29	2078.97	8964.67	1961.53

Tab. 3.26: Final path length after the smoothing with the Simple RRT algorithm.

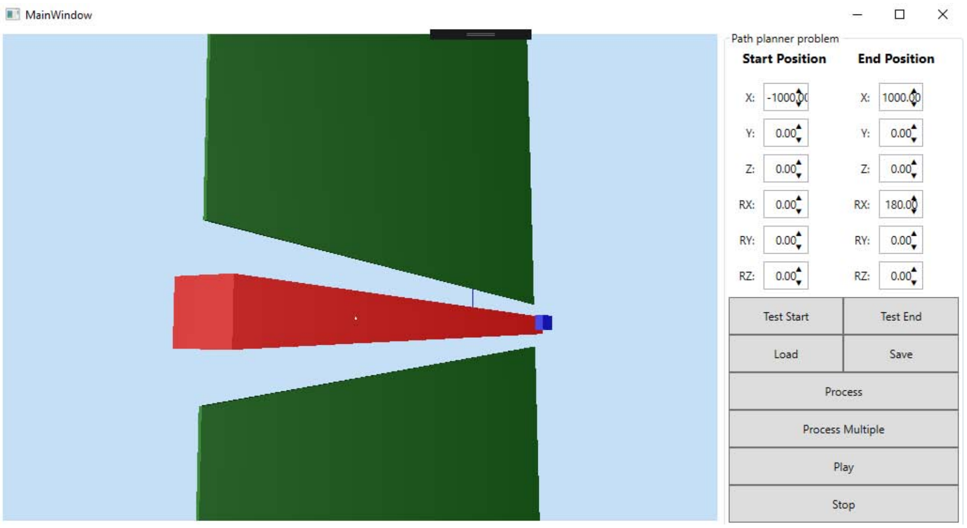
	Mean Path 1 $[mm]$	Stand. Dev.	Mean Path 2 $[mm]$	Stand. Dev.
Very Mixed	7494.79	1179.34	7400.09	1279.84
Mixed	7748.97	1582.33	7926.48	1603.22
Separated	7649.52	1607.68	8416.48	1930.94
Informed	7458.32	1200.94	7337.55	1165.97

Tab. 3.27: Final path length after the smoothing with the Augmented RRT algorithm.

3.3 Environment 3: Narrow Passages Problem

The moving object (blue), with the same dimensions as before, has to move from the initial pose $p_i = [-1m, 0, 0]$ and $(\theta_i, \phi_i, \eta_i) = (0, 0, 0)$ to the final pose $p_g = [1m, 0, 0]$ and $(\theta_g, \phi_g, \eta_g) = (\pi, 0, 0)$.

This is the most difficult experiment, because the rod must necessarily go through one of the two thin corridors to reach the final pose.



Path planner problem

Start Position		End Position	
X:	-1000.00	X:	1000.00
Y:	0.00	Y:	0.00
Z:	0.00	Z:	0.00
RX:	0.00	RX:	180.00
RY:	0.00	RY:	0.00
RZ:	0.00	RZ:	0.00

Fig. 3.11: Environment 3: Narrow passages problem set up.

In this environment the Separated techniques is not used, only the first solution with respectively uniform or polar distribution is found. The second solution, which was found only with the octree-based distributions is not done, because it takes to much time.

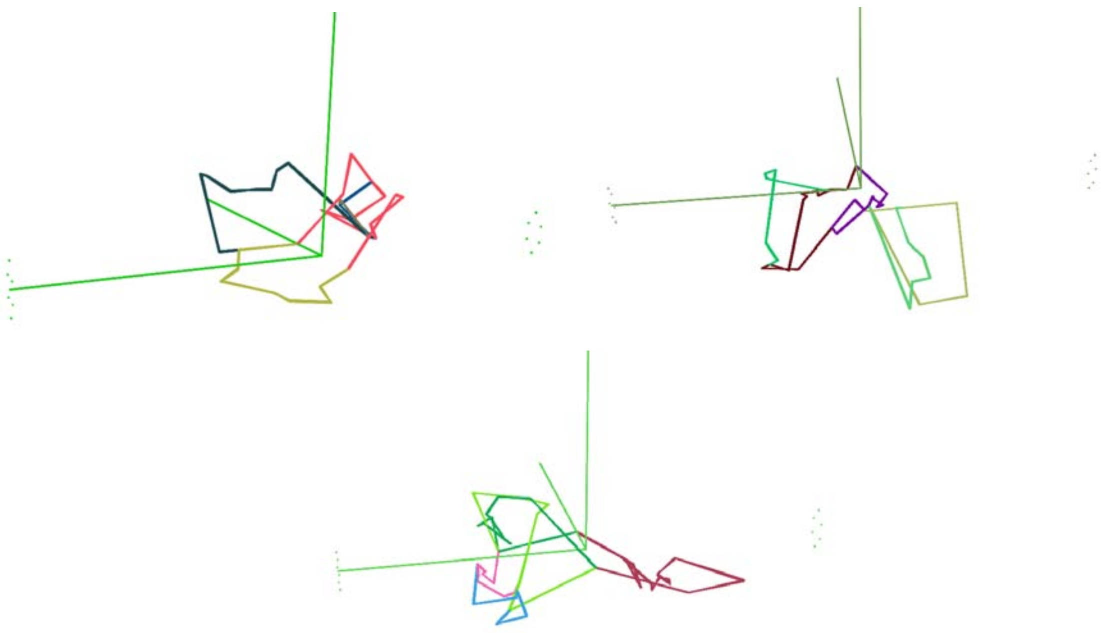


Fig. 3.12: Five different paths in the same environment with the same RRT.

Another important aspect concerns the differences between paths calculated on same trees. In this type of environment is more evident the effectiveness of the `Delete` method describes in Chapter 2. Every new path is different from all the others already found.

In Figure 3.12 are reported 3 tests where algorithms search for five paths ($N = 5$). In light green are depicted the three Cartesian axes, instead all colored broken lines represent the different paths. No path is perfectly equivalent to another, sometimes they differ by one or two points, but other times they are completely different. It could happen that some solutions go through the upper free corridor, instead others go through the lower one.

The ability to change the path structure a lot can offer benefits in possible future works (see Section 3.6).

3.3.1 Uniform Rotational Sampling

Passing through holes and narrow passages is the most complicate issue for sampling based path planners. Sampling randomly in position and rotations in all the space is not a good strategy to find out small apertures, that need a precise pose.

As expected in this case the number of nodes required for both trees increases a lot with respect to the previous tests. Thousands nodes are necessary in both algorithms, in particular four thousands of nodes are required when the Uniform sampling is used alone.

Mean and standard deviation of all measurements are reported in the following tables 3.28 and 3.29.

	First mean	Stand. Dev.	Second mean	Stand. Dev.
Very Mixed	1460.40	1295.03	2059.19	1468.74
Mixed	2963.96	2795.58	4568.35	3231.21
Uniform	4171.35	6602.75	-	-
Informed	1655.67	1524.30	2269.11	1860.33

Tab. 3.28: Number of the nodes' tree with the Simple RRT algorithm.

	First mean	Stand. Dev.	Second mean	Stand. Dev.
Very Mixed	2437.98	2074.35	3430.49	3264.40
Mixed	4439.01	4258.46	6141.25	5560.91
Uniform	4749.10	5341.08	-	-
Informed	2505.99	2648.65	3477.94	3340.14

Tab. 3.29: Number of the nodes' tree with the Augmented RRT algorithm.

Computational time reflects what already shown with the number of nodes, but also reveals more information on the two algorithms.

Working with the Augmented algorithm is not recommended, on average the computational time is about 20 minutes for all sampling strategies except for the Very Mixed that takes 5 minutes.

The very Mixed strategy provides the best results also for the Simple algorithm, the mean is about 2:33 minutes. From Table 3.30 is possible to notice how faster the Simple algorithm is with respect to the Augmented one in this environment.

However the standard deviation in all tests is very high, so there is a lot of variability on the computational time.

	Mean1	S.Dev.	Mean2	S.Dev.	Mean1	S.Dev.	Mean2	S.Dev.
Very Mixed	02:33	02:59	01:00	01:24	04:49	07:20	06:16	18:44
Mixed	04:36	04:56	02:40	03:27	21:12	35:13	12:25	26:51
Uniform	05:28	09:07	-	-	24:15	50:45	-	-
Informed	02:59	03:36	01:12	01:54	17:01	50:07	18:21	41:31

Tab. 3.30: Computational time mean and standard deviation of all the samples depicted in Figure 3.13. On the left the Simple algorithm, on the right the Augmented.

In Figure 3.13 all results are shown, first and second solution respectively in blue and orange are reported below. However as already said, the first couple of graphs has no second histogram, because the time required to conclude the second set of measurements with only the octree strategies takes too much time.

It is worth to notice that the horizontal time scaling are different between left and right graphics. The horizontal axes are composed by time intervals of length of one minutes until 8 minutes, then Simple and Augmented differ and group the time intervals in different ways. Simple bidirectional algorithm is faster than the Augmented one, so the last interval is composed by all solutions that take more than ten minutes to find a solution. Instead for the Augmented the three last intervals of the horizontal axis group more solution, from eight to fifteen minutes, from fifteen to thirty minutes and finally more than thirty minutes to find a solution.

In the two comparison graphics of Figure 3.14 is possible to compare the sampling strategies. The best sampling strategies is the Very Mixed one. It has the lowest computational time and also the smallest number of “outliers”, where with this term are identified all the solutions that take long time.

For the Augmented algorithm only the Very Mixed sampling is competitive with the Simple algorithm’s solutions.

Considering now the length of the final path and the percentage reduction of the smoothing, new observations can be done in this environment.

First of all, looking at Table 3.31 and 3.32 it is possible to see how smaller is the difference of the smoothing applied on the two RRTs. In previous sections (Environment 1 and 2) one common consideration was that the smoothing performs better where there are more nodes in the path, i.e. in the Augmented algorithm. In this environment however the smoothing is able to save and reduce about a 40% of the initial path length in both algorithms. This means that the cutting off procedure works similarly in both RRTs.

Maybe the big portion of free space, on the left and on the right of the green thin foils, helps the smoothing.

The lengths of the final paths are all reported in Table 3.33 and 3.34. On average, the length of the final path is more than double the linear distance between the start and the target pose, which is two meters.

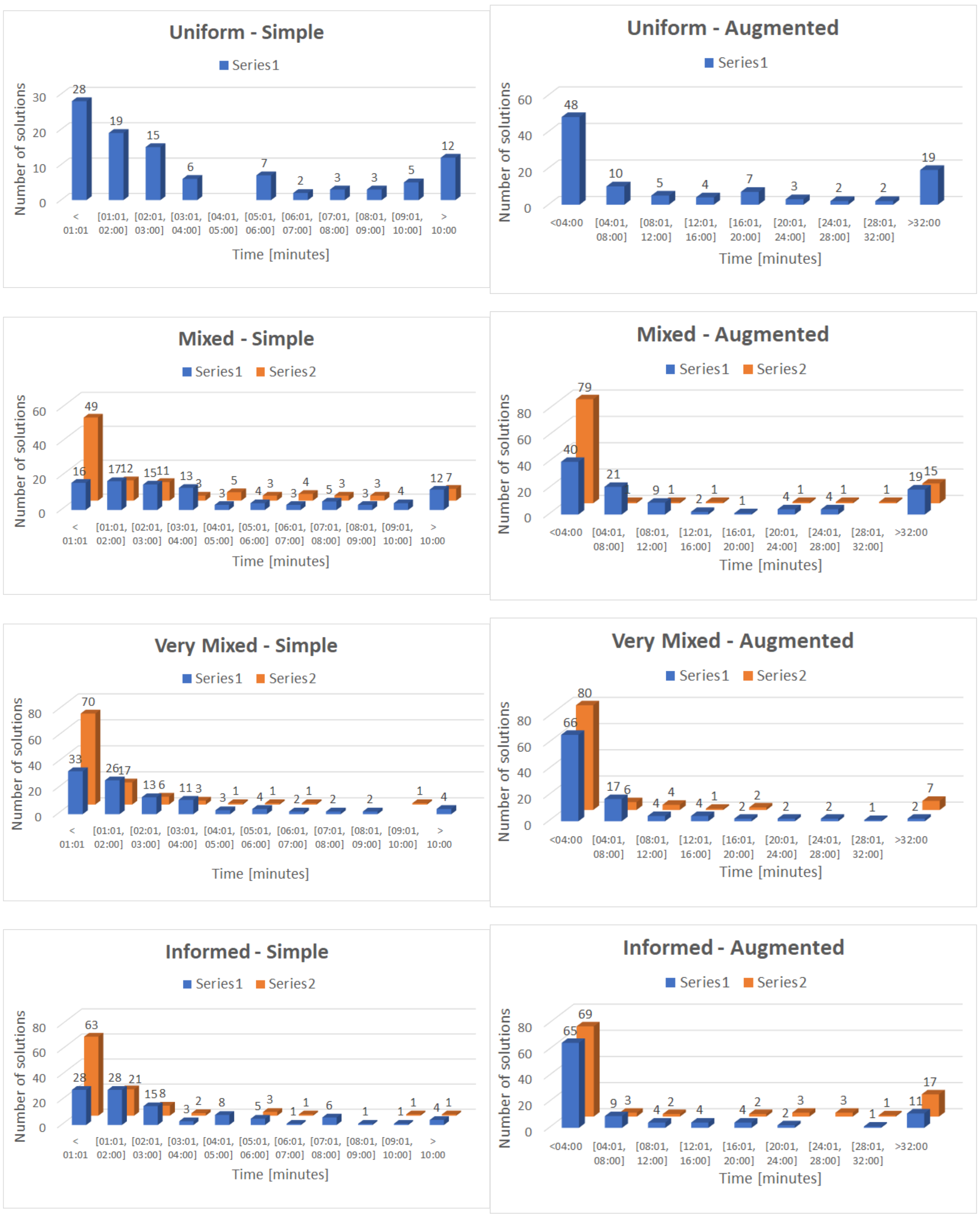


Fig. 3.13: Time required for the computation of the first (light blue) and second solution (orange) with all sampling techniques on both the algorithms (Simple RRT on the left, Augmented RRT on the right).

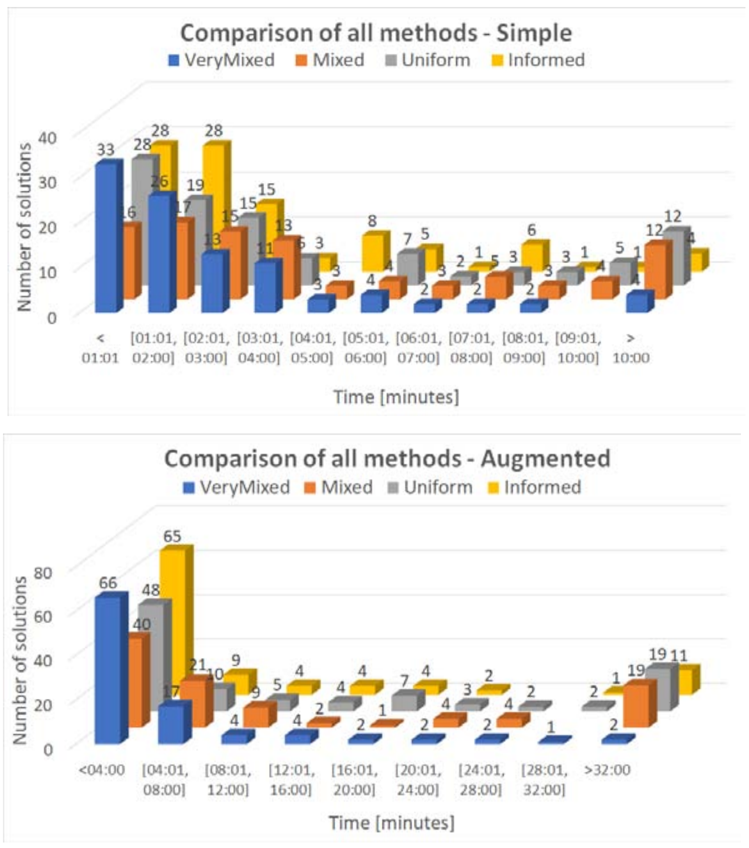


Fig. 3.14: Comparison of the results of the first solution with all the sampling strategies on the top with Simple RRT, instead on the bottom with Augmented RRT.

	First % Reduction	Stand. Dev.	Second % Reduction	Stand. Dev.
Very Mixed	44.59%	17.20	46.13%	15.12
Mixed	47.10%	17.04	50.24%	15.62
Uniform	49.88%	15.94	-	-
Informed	41.53%	16.77	42.25%	15.50

Tab. 3.31: Percentage reduction from first to second solution with the Simple RRT algorithm and all the sampling techniques.

	First % Reduction	Stand. Dev.	Second % Reduction	Stand. Dev.
Very Mixed	36.91%	18.75	42.35%	19.13
Mixed	39.39%	15.55	41.71%	16.75
Uniform	39.46%	16.15	-	-
Informed	29.47%	15.24	29.46%	16.50

Tab. 3.32: Percentage reduction from first to second solution with the Augmented RRT algorithm and all the sampling techniques.

	Mean Path 1 $[mm]$	Stand. Dev.	Mean Path 2 $[mm]$	Stand. Dev.
Very Mixed	5545.74	1914.80	5391.16	1779.08
Mixed	6283.33	1892.44	6434.42	1911.30
Uniform	6342.36	1938.14	-	-
Informed	4813.70	1648.36	4469.53	1588.75

Tab. 3.33: Final path length after the smoothing with the Simple RRT algorithm.

	Mean Path 1 $[mm]$	Stand. Dev.	Mean Path 2 $[mm]$	Stand. Dev.
Very Mixed	4671.99	1789.35	4699.05	1759.41
Mixed	6425.08	1938.29	6442.94	1944.62
Uniform	6064.91	1789.66	-	-
Informed	4778.13	1864.75	4735.48	1693.04

Tab. 3.34: Final path length after the smoothing with the Augmented RRT algorithm.

3.3.2 Polar Rotational Sampling

In this paragraph all results derived from a polar sampling of rotations are discussed. It is worth to notice that having a polar distribution means having an heavy biased rotation though the polar regions. In this environment, this sampling could bring great advantages, if the polar regions are oriented like the reduced aperture, or great disadvantages if the polar regions are not parallel to the aperture.

Some preliminary tests were done in order to understand how the polar regions are oriented. This tests were made only with one sampling strategies (the Very Mixed) and only with one algorithm (the Simple), that were chosen without a specific criterion. In all tests the algorithm and sampling strategy selected were used,instead the environment changes. One hundred tests were done with the aperture horizontal, as depicted in Figure 3.11, the second hundred of measurements were done with traversal aperture (slanted by 45 degrees) and the last with vertical slits.

In Table 3.35 the mean and standard deviation for each test of the computational time are reported. Only the time required to find out the first path is considered.

	First mean	Stand. Dev.
Horizontal	00:43	00:35
45 degrees	01:31	01:30
Vertical	02:51	02:47

Tab. 3.35: Preliminary test to verify the orientation of the polar regions.

The values in Table 3.35 are reported in minutes and it is evident that the sampling is not uniform because changing the apertures orientations modifies the time required to find a path. In particular the polar regions are oriented parallel to the y-axis and this sampling influences a lot the performances. The algorithm results very fast with horizontal apertures, however if the slits are vertical the performances are worst.

In the following considerations it is important to keep in mind that we are working in a favorable situation, so in other cases results could be worst.

In Table 3.36 and 3.37 the number of nodes required to build the two trees respectively

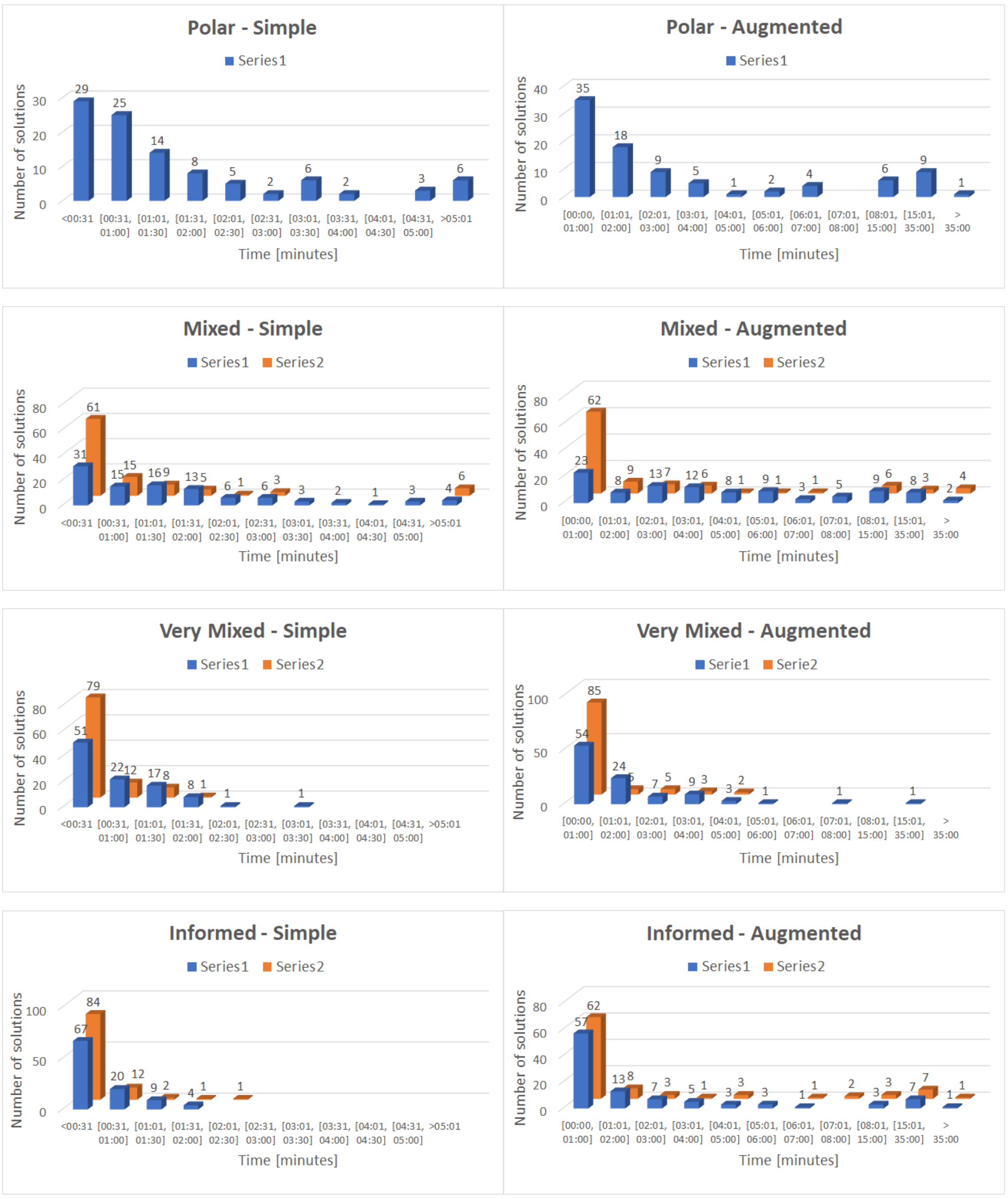


Fig. 3.15: Time required for the computation of the first (light blue) and second solution (orange) with all sampling techniques on both the algorithms (Simple RRT on the left, Augmented RRT on the right).

	First mean	Stand. Dev.	Second mean	Stand. Dev.
Very Mixed	445.32	382.88	637.70	464.06
Mixed	918.02	824.75	1445.66	1293.36
Polar	1054.40	1208.46	-	-
Informed	332.17	314.30	530.29	450.99

Tab. 3.36: Number of the nodes' tree with the Simple RRT algorithm.

	First mean	Stand. Dev.	Second mean	Stand. Dev.
Very Mixed	1335.37	1028.30	1610.00	1194.75
Mixed	2315.49	1950.22	3216.24	3007.55
Polar	2240.30	2258.09	-	-
Informed	1625.76	1594.80	2103.53	1880.69

Tab. 3.37: Number of the nodes' tree with the Augmented RRT algorithm.

	Mean1	S.Dev.	Mean2	S.Dev.	Mean1	S.Dev.	Mean2	S.Dev.
Very Mixed	00:43	00:35	00:18	00:21	01:37	02:45	00:31	00:57
Mixed	01:31	01:28	00:56	01:47	06:15	09:26	05:42	20:01
Polar	01:42	02:18	-	-	04:58	08:00	-	-
Informed	00:29	00:25	00:16	00:21	04:10	11:58	03:07	07:31

Tab. 3.38: Computational time mean and standard deviation of all the samples depicted in Figure 3.15. On the left the Simple algorithm, on the right the Augmented.

when the simple and augmented algorithms are used are shown.

The number of nodes is very high, especially when the Augmented RRT works. The Simple builds trees of size about 500-1000 nodes, instead the Augmented builds trees of size about 1500-2000 vertices.

Table 3.38 reports all computational times in mean and standard deviation and Figure 3.15 shows how each test behaves.

The Very Mixed sampling is the most efficient in both algorithms. Especially for Augmented RRT it is the unique sampling strategy that provides a result in less than 4 minutes.

The horizontal time scale is different between Simple and Augmented graphs, because the choice of the algorithms in this case influences a lot the computational time.

In Figure 3.16 all performances of the sampling strategies that look for the first solution are compared. The Simple algorithm has good results with the Very Mixed and the Informed sampling, instead the Augmented, as already said, works well with the Very Mixed one.

For what concerning the final path length and the action of the smoothing Table 3.39 3.40 and 3.41 and 3.42 summarized all the principal information.

For both Simple and Augmented the smoothing reduces about the 40% of the path. The sampling strategy that provides less smoothing is the Informed one. A possible explanation of that fact could be that this strategy samples near the goal pose and does not wander around the entire space.

On average the length found by Simple and Augmented RRT are comparable, it is

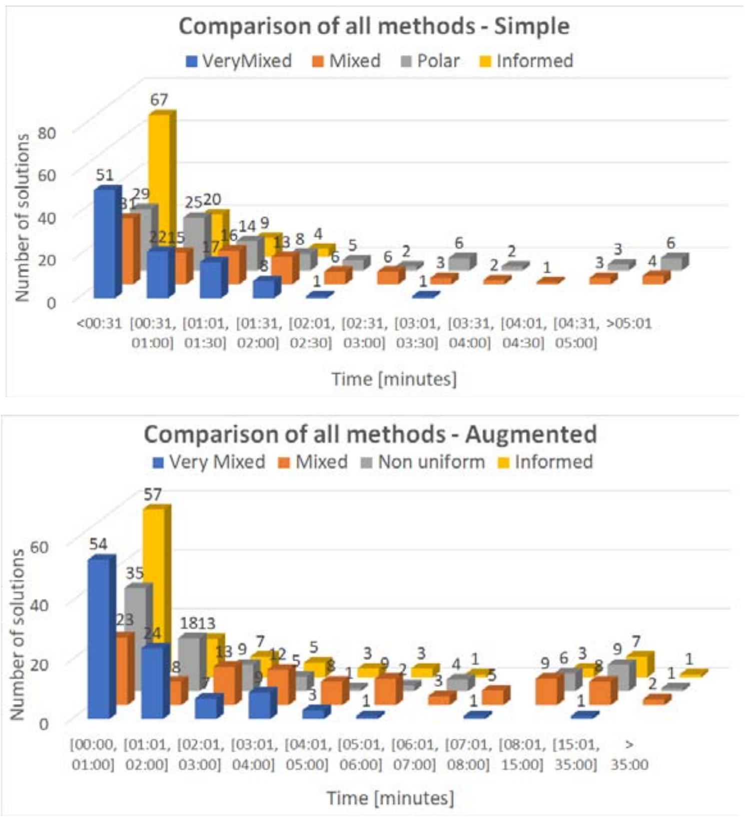


Fig. 3.16: Comparison of the results of the first solution with all the sampling strategies on the top with Simple RRT, instead on the bottom with Augmented RRT.

	First % Reduction	Stand. Dev.	Second % Reduction	Stand. Dev.
Very Mixed	37.92%	16.78	42.70%	18.87
Mixed	43.77%	18.09	47.98%	14.91
Polar	45.98%	17.06	-	-
Informed	33.07%	17.86	38.56%	18.07

Tab. 3.39: Percentage reduction from first to second solution with the Simple RRT algorithm and all the sampling techniques.

	First % Reduction	Stand. Dev.	Second % Reduction	Stand. Dev.
Very Mixed	33.94%	19.80	38.13%	18.58
Mixed	38.14%	18.10	39.67%	17.57
Polar	37.34%	16.94	-	-
Informed	29.31%	17.64	34.39%	18.07

Tab. 3.40: Percentage reduction from first to second solution with the Augmented RRT algorithm and all the sampling techniques.

about 5000/6000.

	Mean Path 1 $[mm]$	Stand. Dev.	Mean Path 2 $[mm]$	Stand. Dev.
Very Mixed	5336.70	1901.18	5252.44	1866.59
Mixed	6305.28	1820.23	6223.57	1912.33
Polar	6303.39	1710.98	-	-
Informed	4896.73	1641.60	4808.24	1850.66

Tab. 3.41: Final path length after the smoothing with the Simple RRT algorithm.

	Mean Path 1 $[mm]$	Stand. Dev.	Mean Path 2 $[mm]$	Stand. Dev.
Very Mixed	4897.53	1963.96	4998.18	1941.62
Mixed	5998.03	1750.43	6232.94	1906.54
Polar	6208.44	1604.99	-	-
Informed	4734.86	1639.47	4479.96	1497.25

Tab. 3.42: Final path length after the smoothing with the Augmented RRT algorithm.

3.4 Discussion and Observations

It is possible to extract some general considerations from all tests previously discussed.

In general it is true that the second solution requires less time to be found out. Usually having trees, already built, helps the algorithm to connect rapidly the two trees. These RRTs are not multiple query, because they cannot solve different problems, but for one specific task they can find out different solutions. The brief discussion at the beginning of section 3.3 on path diversity still holds for the other two environments.

As for the computational time, it is possible to deduce that having a simple RRT structure, with not a large number of nodes, helps the algorithm to speed up the calculations. A large number of nodes slows down the algorithm, however the presence of more vertices in the final path usually provides better smoothing. It could be a good choice to select the Simple RRT to resolve a specific task and then to add intermediate nodes in the final path.

Especially where it is employed the uniform sampling of rotations, mixing sampling strategies allows to speed up the search for a path. The Very Mixed strategy offers greater performance in the case of uniform sampling of rotations because the polar one in lucky cases (such as environment 3 with horizontal openings) offers some advantages on its own. So sometimes with polar rotational sampling is sufficient to provide an Informed distribution or a Cartesian sampling which is focused on the target.

However, it is worth noting that working with polar rotations can also provide worse performance, so it is not advisable to rely on such sampling if you are not familiar with the working environment.

Conclusions

3.5 Final Considerations

In Chapter 3 all measurements were discussed and analyzed looking at three dominant components: the number of nodes belonging to the two trees, the computational time needed to find out the first and second paths and finally the action of the smoothing associated to the final length of the path.

The analysis of the previous chapter shows some results that will be resumed in the following:

- The three environments are reported in order of complexity. The first is the simplest one and it witnesses that both the algorithms and all sampling strategies work properly. The second environment is a classic navigation problem that allows you to consider the first differences on the various approaches used. Instead the third one is a difficult problem that can be solved only with some techniques.
- The search for the second solution with an already built tree improves temporal performance.
- Having a simple construction of the trees (i.e. Simple Bidirectional RRT) provides better results than a more complex built tree (i.e. Augmented RRT). In the second and third environments, the advantages of a simple construction are visible in the best temporal performances. This fact underlines that what really affects the temporal performances of the RRT algorithms is the sampling strategy adopted.
- Between all the sampling strategies proposed, especially when there is uniform sampling of rotations, the best strategy is the Very Mixed one. This implies that mixing different sampling strategies provides better solutions than using them alone. The Very Mixed manages to perform well in all the environments and independently from the algorithm used.
- The sampling of rotations is the most delicate issue to deal with. The uniform distribution ensures uniform coverage of the rotation space, but sometimes it is not necessary to rotate in all directions and a good approach would be to limit movement. However you can limit rotations only when some information on the environment are known a priori. It could be interesting to mix together also different sampling of rotations (i.e uniform and polar) and see what happens on the temporal performances. In this way no a priori information is needed and the sampling could guaranteed the complete coverage of rotations space too.
- The action of the smoothing is more incisive when there are more points (vertices) inside the path, it could be reasonable to add some points within the path in order to have smoother paths.

3.6 Possible Improvements

As already said in Chapter 1, usually path planning algorithms are designed and projected on the robot's configuration space. So a sequence of joint's variables is the output of the planning algorithms and the end-effector pose is just a consequence of this sequence, thanks to the direct kinematics.

This choice was done because working with the direct kinematic of the robot is faster then using the inverse kinematics, that from the end-effector pose recovers all the chain of joint's variables. The inverse kinematics is more delicate and not trivial, however when a robot is manipulating a big payload also the direct kinematics is not so efficient. The inefficiency is due to the collision detection procedure, when the payload is greater than the robot it could happen that the most number of collisions are provided by the payload.

Taking these considerations into account, the reason behind this study is well explained. In this work, a long, thin rod, which is an example of a common manipulator payload, has the ability to independently plan its movements. In the future, it may be interesting to connect a robotic arm to the rod and to study whether performing an inverse kinematic procedure could bring some advantages in applications where there are high payloads.

Another important consideration, done in section 3.3, turns out to be very useful for such purpose: the possibility to have different paths to be tested. The inverse kinematics is more complicate than the direct one, this is why one of the goals of this work is to find big or small variances between paths in the same environment. In this way you can have more paths to be recovered in joint space.

Bibliography

[1] B. Siciliano, L. Sciavicco, L. Villani, G. Oriolo, *Robotics, modeling, planning and control*, New York: Springer-Verlag, 2008.

[2] T. Kunz dissertation, *Time-optimal sampling-based motion planning for manipulators with acceleration limits*, School of Interactive Computing, Georgia Institute of Technology, May 2015.

[3] T. Lozano-Pérez and M. A. Wesley, *An algorithm for planning collision-free paths among polyhedral obstacles*, Communications of the ACM, Vol.22, No.10, 1979, pp. 560-570.

[4] S. Lahouar, S. Zeghloul, L. Romdhane, *Path-planning for manipulator robots in cluttered environments*, in: International Design Engineering TechnicalConferences Computers and Information In Engineering Conference, DETC2005-84993.LongBeach, CA, September2005.

[5] J.-C. Latombe, *Robot Motion Planning* Kluwer, 1991.

[6] Steven M. La Valle, *Planning algorithms*, University of Illinois, 2006, Chapter 5 pp. 185-248.

[7] Colm Ã'Dúnlaing, Chee Keng Yap, Micha Sharir, *Retraction: A new approach to motion-planning*, January 1983.

[8] J.R. Andrews, and N. Hogan N, 1983, *Impedance Control as a Framework for Implementing Obstacle Avoidance in a Manipulator*, Control of Manufacturing Processes and Robotic Systems, Eds. Hardt, D.E. And Book, W., ASME, Boston, 243-251.

[9] Y. Koren, Senior Member, IEEE and J. Borenstein, Member, IEEEThe University of Michigan, Ann Arbor, *Potential Field Methods and Their Inherent Limitationsfor Mobile Robot Navigation*, Proceedings of the IEEE Conference on Robotics and Automation, Sacramento, California, April 7-12, 1991, pp. 1398-1404.

[10] Steven M. La Valle, *Rapidly-Exploring Random Trees: A New Tool For Path Planning*, Department of computer science, Iowa State University, Ames IA 50011 USA, 1998.

[11] Wikipedia, *https://en.wikipedia.org/wiki.*

[12] Man, K. F., Tang, K. S. and Kwong, S. (1999), *Genetic Algorithms*, Ed. Springer, 1st Edition, London, UK.

[13] O. Castillo and L. Trujillo, *Multiple objective optimization genetic algorithms for path planning in autonomous mobile robots*, Dept. of Computer Science, Tijuana Institute of Technology Tijuana, Mexico; International Journal of Computers, Systems and Signals, Vol. 6, No. 1, 2005.

[14] Kuffner, J.J.: *Effective sampling and distance metrics for 3D rigidbody path planning.* In: Proc. International Conference on Robotics and Automation (2004).

[15] K. Shoemake, *Uniform Random Rotations*, ser. Graphics Gems III. Academic Press, 1992, pp. 124-132.

[16] James J. Kuffner, Jr. Steven M. LaValle, *RRT-Connect: An Efficient Approach to Single-Query Path Planning*, In Proc. 2000 IEEE Intâl Conf. on Robotics and Automation (ICRA 2000).

[17] D. E. Knuth. *L'arte della programmazione del computer, volume 2: algoritmi Seminumerical.* Addison-Wesley, Reading, MA, terza edizione, 1997.

Appendix A

Algebraic Groups and Orientation Representation

A.1 Algebraic Group

An algebraic group is an algebraic structure equipped with an inner product $(\cdot)$ and characterized by the following four properties:

- *Closeness* Group $\mathcal{G}$ is close:

$$\forall g_1, g_2 \in \mathcal{G} : g_1 \cdot g_2 \in \mathcal{G} \tag{A.1.1}$$

- *Associativity* Group $\mathcal{G}$ is associative:

$$\forall g_1, g_2, g_3 \in \mathcal{G} : g_1 \cdot (g_2 \cdot g_3) = (g_1 \cdot g_2) \cdot g_3 \tag{A.1.2}$$

- *Existence of the neutral element*

$$1 \in \mathcal{G} \textit{ such that } g \cdot 1 = 1 \cdot g = g \tag{A.1.3}$$

- *Existence of the inverse element*

$$g^{-1} \in \mathcal{G} \textit{ such that } g \cdot g^{-1} = g^{-1} \cdot g = 1 \tag{A.1.4}$$

Now it is presented a list of different algebraic groups of interest, with some notes regarding their meaning.

The **General Linear** group $\mathbb{GL}(n)$ is the set of all the linear transformations/matrices $A \in \mathbb{R}^{n \times n}$ that are non-singular and equipped with the standard matricial product.

$$\begin{aligned} A : \mathbb{R}^n &\to \mathbb{R}^n \\ x &\to Ax. \end{aligned} \tag{A.1.5}$$

The **Affine Linear** group $\mathbb{A}(n)$ is define by a matrix $A \in \mathbb{R}^{n \times n}$ and a vector $b \in \mathbb{R}^n$ with the map:

$$\begin{aligned} A : \mathbb{R}^n &\to \mathbb{R}^n \\ (A, b) : x &\to Ax + b. \end{aligned} \tag{A.1.6}$$

The **Orthogonal** group $\mathbb{O}(n)$ is define by a matrix $A \in \mathbb{R}^{n \times n}(A \in \mathbb{GL}(n))$ invertible such that the inner product between vectors in $\mathbb{R}^n$ is preserved:

$$\forall x, y \in \mathbb{R}^n : \langle Ax, Ay \rangle = \langle x, y \rangle. \tag{A.1.7}$$

It is the group of orthogonal matrices:

$$\langle Ax, Ay \rangle = x^T A^T Ay = x^T y = \langle x, y \rangle \to A^T A = I. \tag{A.1.8}$$

We have distance preserving, angle preserving and area preserving transformations, in other words they are isometries.

We are interested in a subgroup of $\mathbb{O}(n)$, which is the **Special Orthogonal** group $\mathbb{SO}(n)$:

$$\mathbb{SO}(n) = \{A \in \mathbb{O}(n) \textit{ such that } \det A = 1\}. \tag{A.1.9}$$

Note that $\mathbb{SO}(n)$ is a group of rotations (for example $\mathbb{SO}(2)$) describes planar rotations).

The orthogonal groups are a subgroup of the general linear group, while the next one is a subgroup of the affine group.

The **Euclidean** group $\mathbb{E}(n)$ is define by an affine transformation, where $R \in \mathbb{O}(n)$, $T \in \mathbb{R}^n$:

$$\begin{gathered} A : \mathbb{R}^n \to \mathbb{R}^n \\ x \to Rx + T. \end{gathered} \tag{A.1.10}$$

We are interested in a particular subgroup of $\mathbb{E}(n)$, where $R \in \mathbb{SO}(n)$, called **Special Euclidean** group $\mathbb{SE}(n)$.

A.2 Orientation Representation

A rigid body is completely described in space by its position and orientation (in brief pose) with the respect to a reference frame.

Usually for the position is taken the center of mass of the object and its position with respect to the orthonormal reference frame is:

$$O' = o'_x x + o'_y y + o'_z z \tag{A.2.1}$$

Instead the rigid body orientation is obtained looking at the orthonormal frame attached to the body from the reference frame.

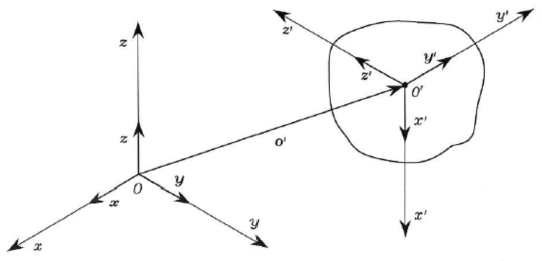


Fig. A.1: Bounding Box.

The orientations can be expressed in terms of rotation matrices. A rotation matrix $R \in \mathbb{R}^n$ belongs to the Special Orthogonal group $\mathbb{SO}(n)$. When $n = 2$ there are planar rotations, however with $n = 3$ there are spatial rotations.

A rotation matrix has three equivalent geometrical meanings:

- it represents the mutual orientation between two coordinate frames;

- it is the operator that allows the rotation of a vector in the same coordinate frame;

- it describes the coordinate transformation between the coordinates of a point expressed in two frames.

However there exist two other ways to describe the frame orientation in addition to the rotation matrix:

- *Three-angle representation:* this is the minimal representation, three independent parameters are sufficient to describe orientation of a rigid body in the space. In fact the minimal space representation of the $\mathbb{SO}(n)$ group requires $\frac{n(n-1)}{2}$ parameters.

  There is also a theorem that states that: *the Euler's rotation theorem* affirms that any two independent orthonormal coordinate frames can be represented by a sequence of rotations (not more than three) about coordinate axes, where no two successive rotations may be about the same axis.

  The most popular set of angles is the triplets of roll-pitch-yaw angles (θ, ϕ, η) respectively about the z-axis, the y-axis and the x-axis.

- *Quaternion representation:* this is a four-parameters representation, alternative to the axis-angle representation. The quaternion is an extension of the complex number, a hyper-complex number, and it is written as a scalar s plus a vector v:

$$\hat{q} = s + v = s + v_1 i + v_2 j + v_3 k. \tag{A.2.2}$$

  TO represent rotations are required quaternions with unit magnitude: $|\hat{q}| = 1$. It has the special property that it can be considered as a rotation of an angle θ about a unit vector r, which are related to the quaternion's component by:

$$s = \cos\frac{\theta}{2}, \qquad v = (\sin\frac{\theta}{2})r. \tag{A.2.3}$$

Appendix B

Description of Object's Size, Position and Shape

There exist many methods to represent a concise description of the size, position and shape of an object. In the following are reported some useful techniques.

The simplest representation of size and shape of is the *bounding box* for a point set (S) in N dimensions is the box with the smallest measure (area, volume, or hypervolume in higher dimensions) within which all the points lie.

It differs from the *convex hull* of a shape, which is in geometry the smallest convex set that contains it. For a bounded subset of the plane, the convex hull may be visualized as the shape enclosed by a rubber band stretched around the subset.

Formally, the convex hull may be defined either as the intersection of all convex sets containing a given subset of a Euclidean space, or equivalently as the set of all convex combinations of points in the subset. Convex hulls of open sets are open, and convex hulls of compact sets are compact. Every convex set is the convex hull of its extreme points. The convex hull operator is an example of a closure operator. In Figure B.1 is shown the difference between the bounding box and the convex hull of a two-dimensional object with a particular geometry.

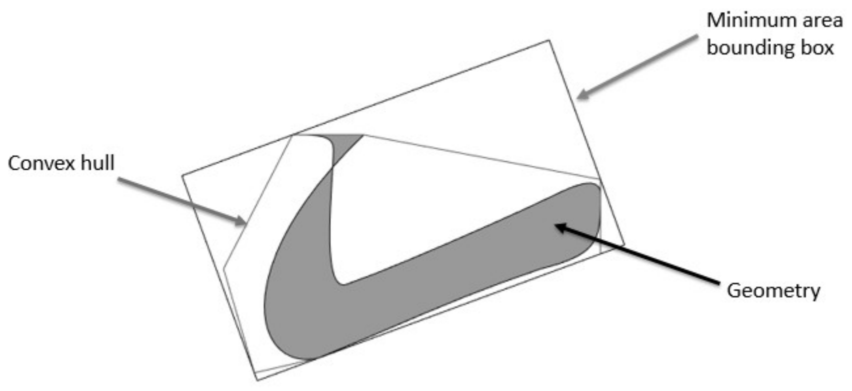


Fig. B.1: Example of bounding box and convex hull in two-dimensional space.

A *polygon mesh* is a collection of vertices, edges and faces that defines the shape of a polyhedral object in 3D computer graphics and solid modeling. The faces usually consist of triangles (triangle mesh), quadrilaterals (quads), or other simple convex polygons (n-gons), since this simplifies rendering, but may also be more generally composed of concave polygons, or even polygons with holes. As polygonal meshes are extensively used in computer graphics, algorithms also exist for ray tracing, collision detection, and rigid-body dynamics with polygon meshes.

Objects created with polygon meshes must store different types of elements. These include vertices, edges, faces, polygons and surfaces all reported in Figure B.2.

- **Vertex:** A position (usually in 3D space) along with other information such as color, normal vector and texture coordinates.

- **Edge:** A connection between two vertices.
- **Face:** A closed set of edges, in which a triangle face has three edges, and a quad face has four edges. A polygon is a coplanar set of faces. In systems that support multi-sided faces, polygons and faces are equivalent. Mathematically a polygonal mesh may be considered an unstructured grid, or undirected graph, with additional properties of geometry, shape and topology.
- **Surfaces:** More often called smoothing groups, are useful, but not required to group smooth regions.

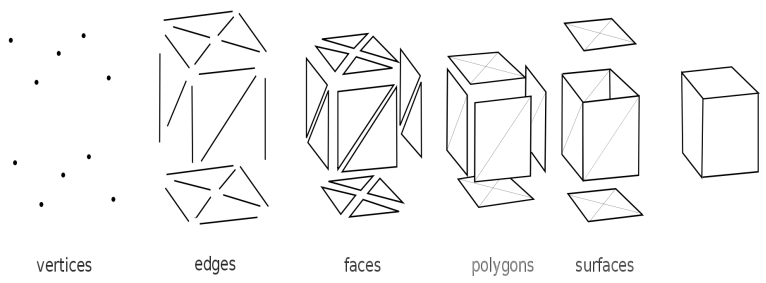


Fig. B.2: Elements that can be stored in a mesh regions.

Polygon meshes may be represented in a variety of ways, using different methods to store the vertex, edge and face data.

Appendix C

Vostok Interface

In this thesis a graphical interface was built in order to allow some user to work without the need of using a less intuitive C sharp code.

The graphical interface is depicted in Figure C.1. The workspace is loaded and shown in the big light blue space on the left, instead on the right there are a list of all possible commands. Let's consider all of them in details.

On the top of the right white space there are the Cartesian coordinates and the angles (expressed in degrees) of the initial and final pose of the target to move. They can be all modified putting the desired value on the small grid. In Figure C.1 is reported the initial and final pose of the moving object used in Environment 1 or 3, just for example.

Instead below there are some buttons. The two upper buttons "Test Start" and "Test End" verify if the starting and goal configurations set previously are in collision of not, if they are in collision an error message appears on the display.

Below these two buttons there are "Load" and "Save", these respectively are used to load a program/workspace already stored and to save the current project if the user wants to remember it.

The "Process" and "Process Multiple" buttons were adopted to test the algorithm in the current ambient, the first runs the algorithm one time, instead the second runs the algorithm one hundred of times.

The button "Play" can be work only when the process is concluded, if it is pressed before the "Process" button an error occurs. This button is called "Play" because it starts moving the target through the path already found in the process stage.

The last button is the "Stop" which interrupt the movement of the target to move.

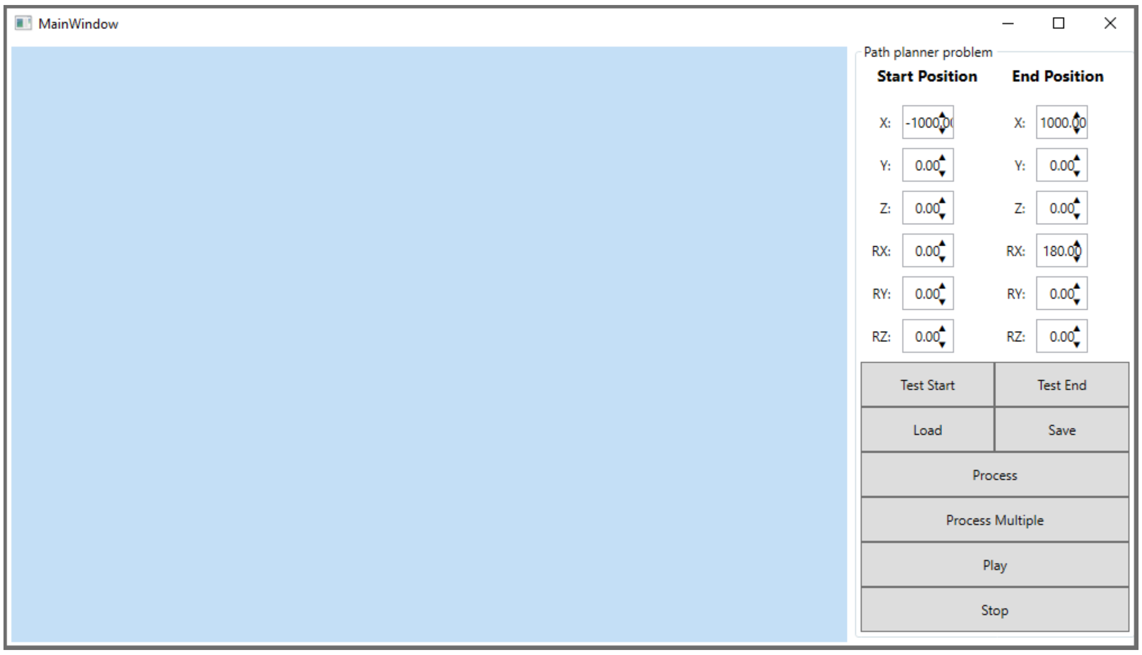


Fig. C.1: The interface of the Vostok plugin.

Appendix D

Graphical Results

In this appendix are reported all the graphical results already discussed in Chapter 3. In particular they show the length of final paths.

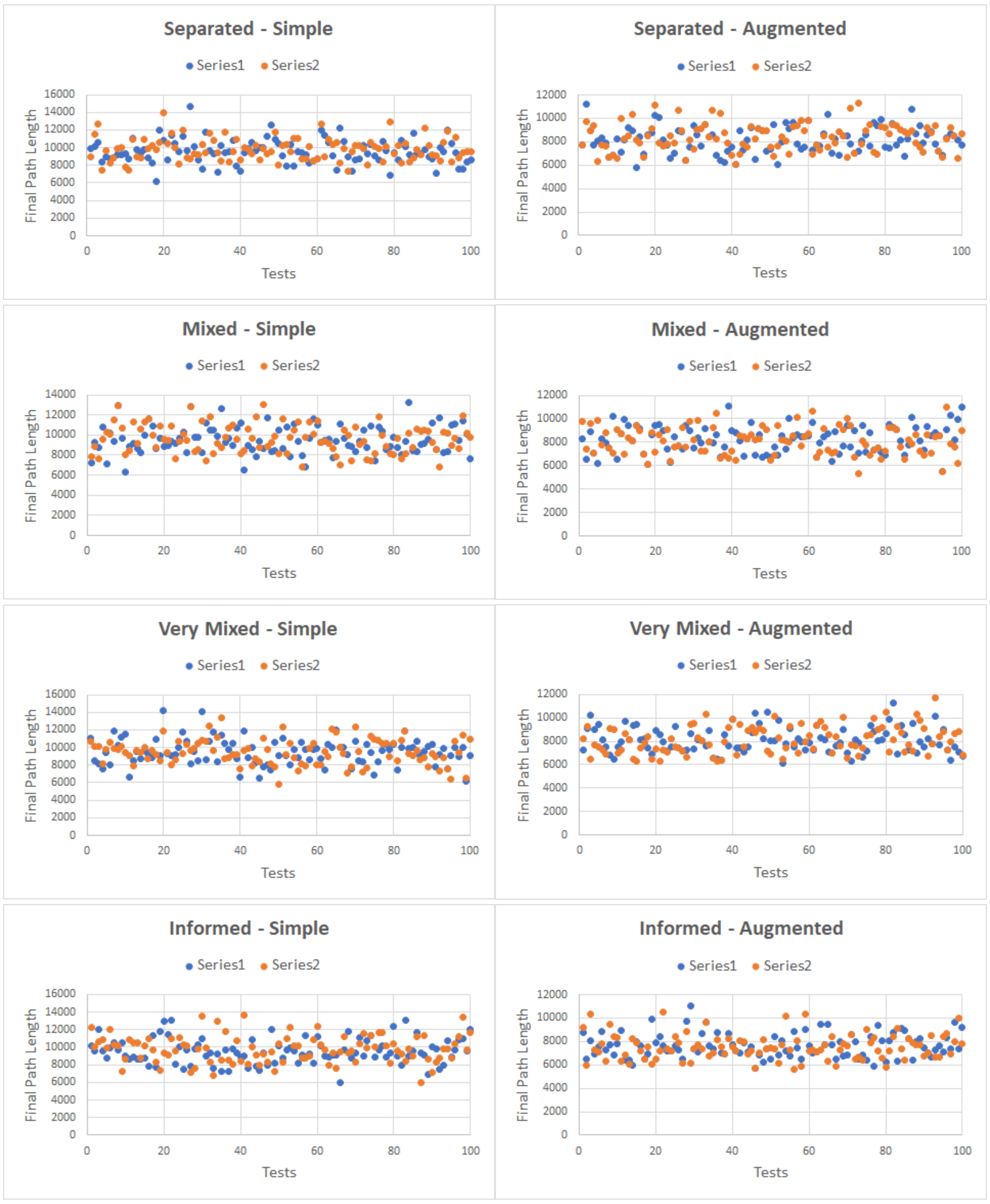


Fig. D.1: *Environment 1: Uniform Rotational Sampling.* The final path length of the path after the smoothing of the first (light blue) and second solution (orange), on the left with the Simple RRT, on the right the Augmented.

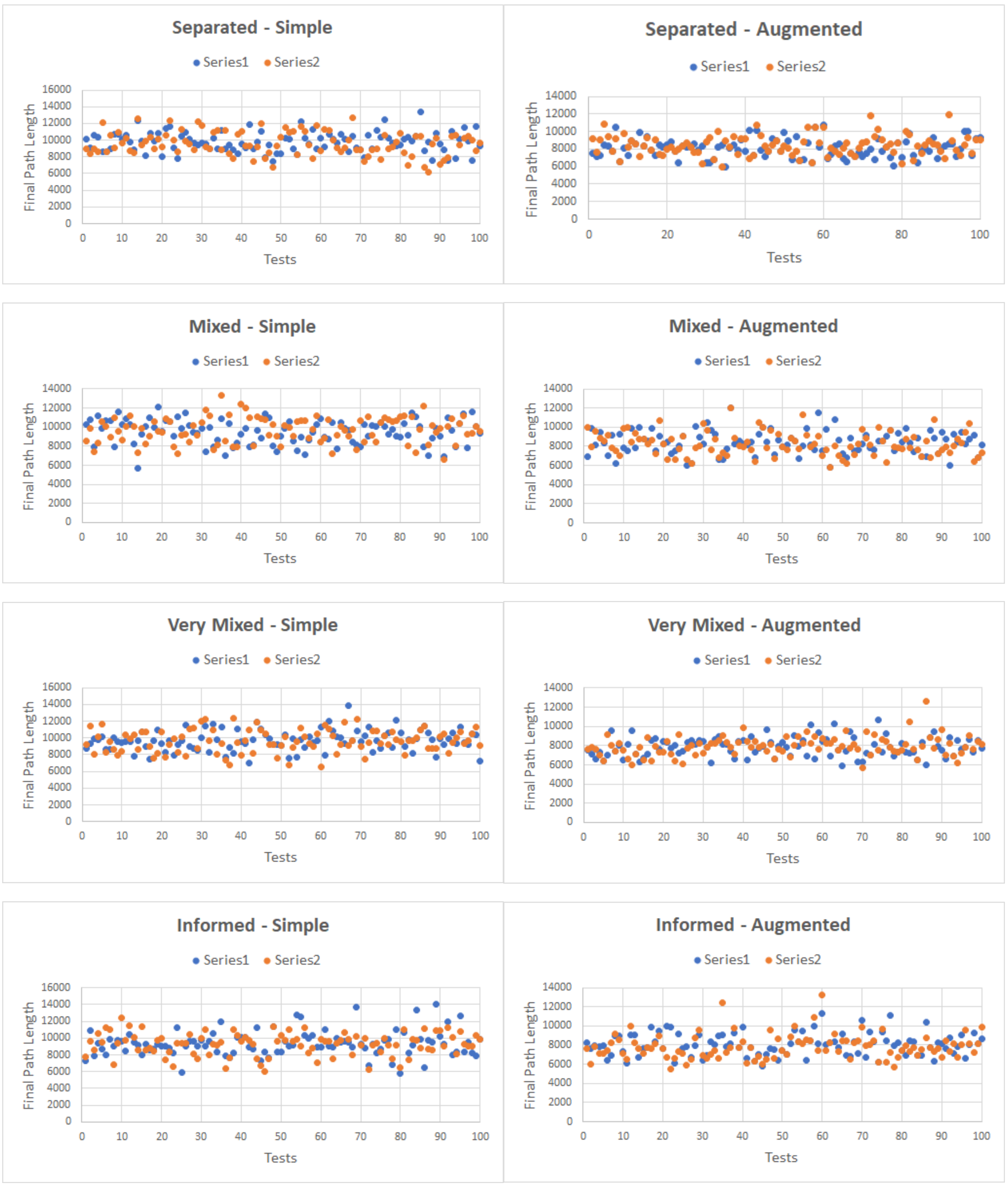


Fig. D.2: *Environment 1: Polar Rotational Sampling.* The final path length of the path after the smoothing of the first (light blue) and second solution (orange), on the left with the Simple RRT, on the right the Augmented.

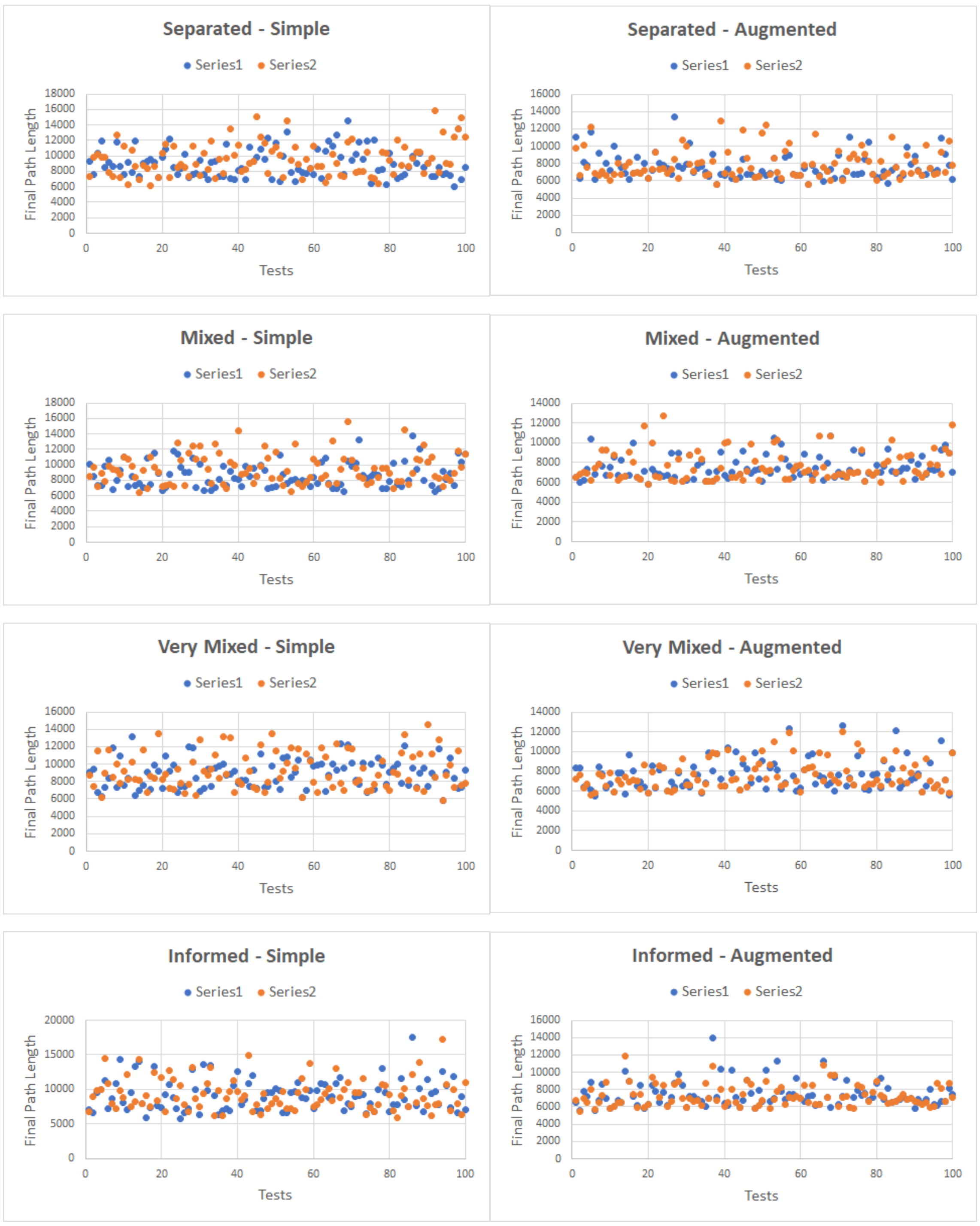


Fig. D.3: *Environment 2: Uniform Rotational Sampling.* The final path length of the path after the smoothing of the first (light blue) and second solution (orange), on the left with the Simple RRT, on the right the Augmented.

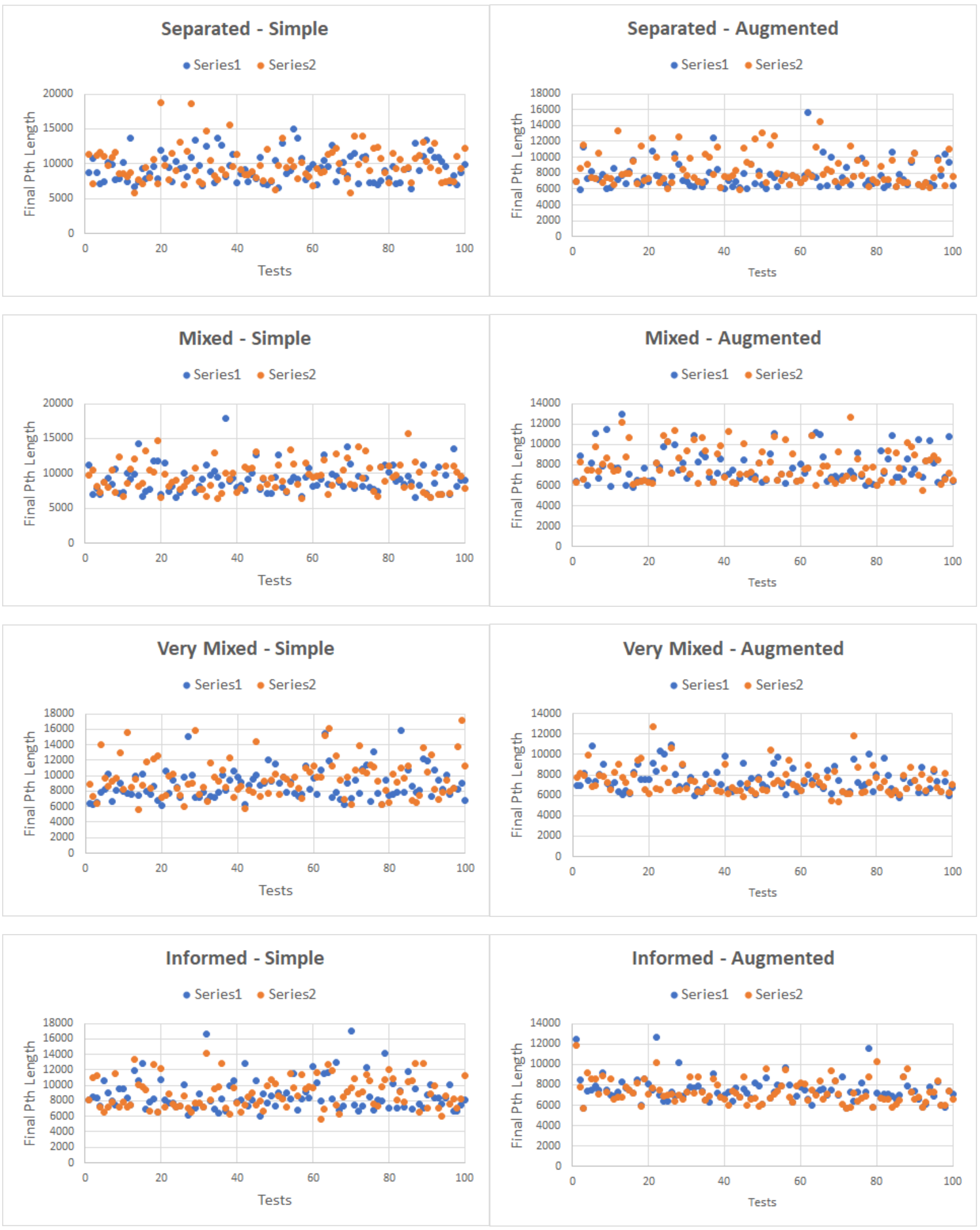


Fig. D.4: *Environment 2: Polar Rotational Sampling.* The final path length of the path after the smoothing of the first (light blue) and second solution (orange), on the left with the Simple RRT, on the right the Augmented.

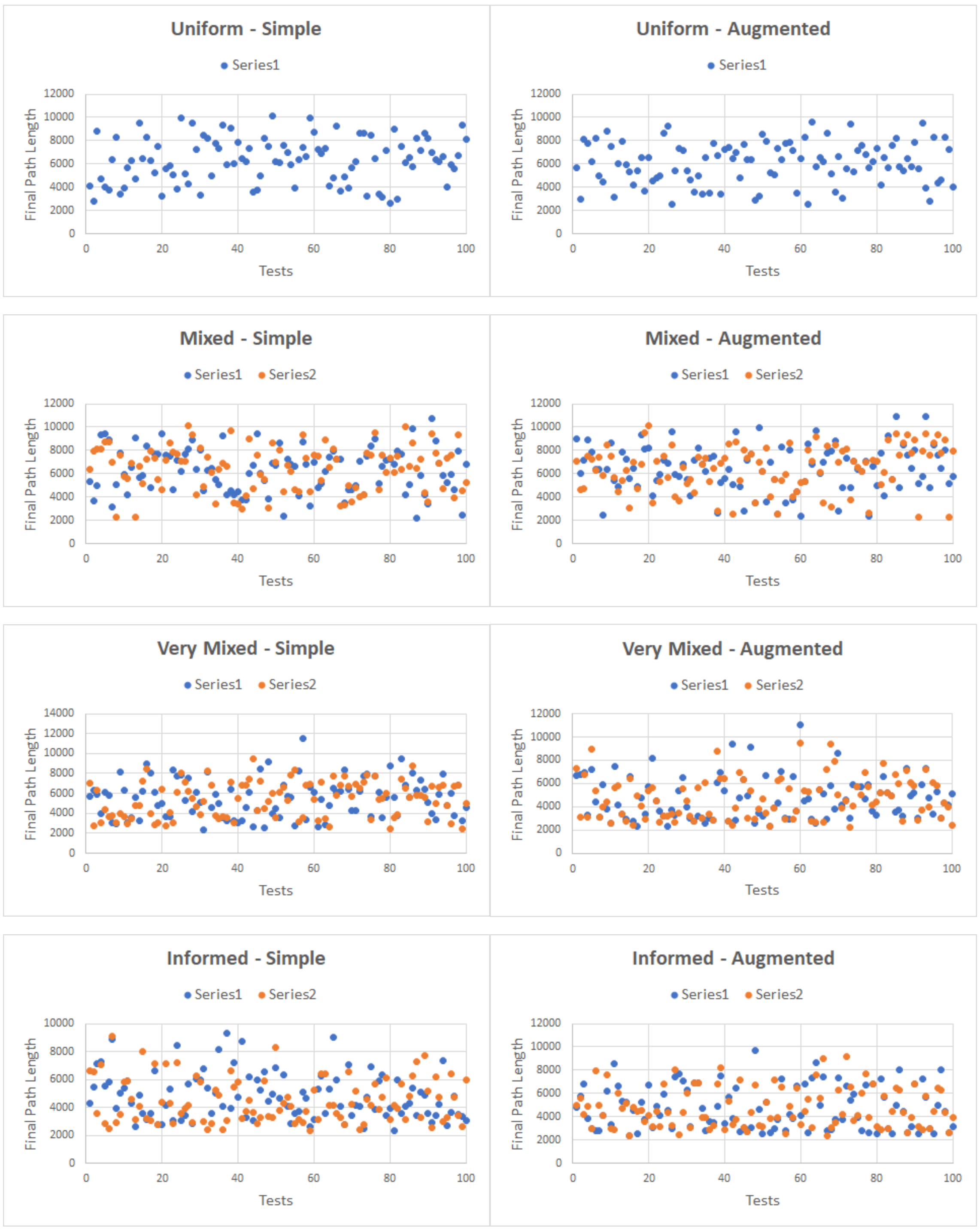


Fig. D.5: *Environment 3: Uniform Rotational Sampling.* The final path length of the path after the smoothing of the first (light blue) and second solution (orange), on the left with the Simple RRT, on the right the Augmented.

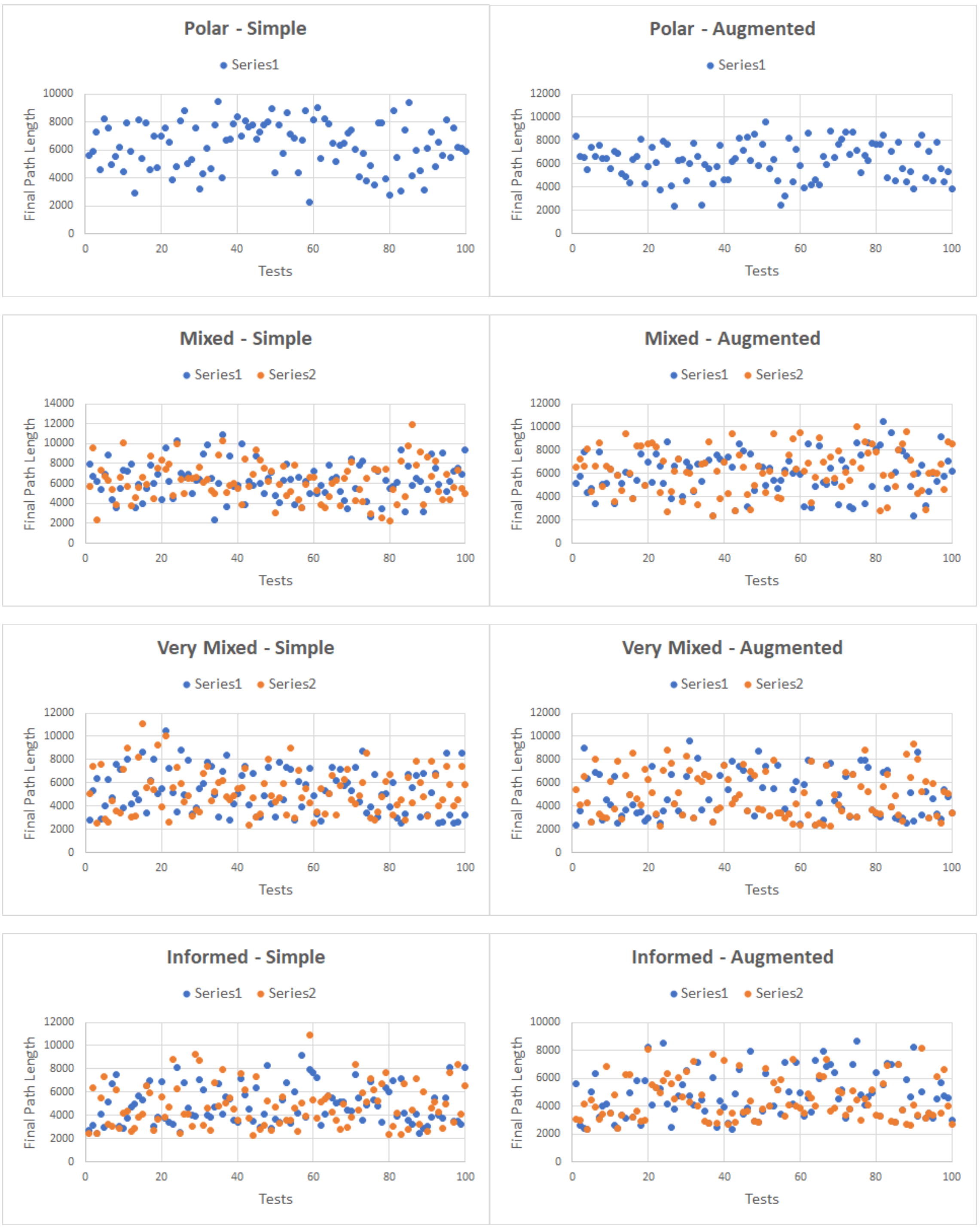


Fig. D.6: *Environment 3: Polar Rotational Sampling.* The final path length of the path after the smoothing of the first (light blue) and second solution (orange), on the left with the Simple RRT, on the right the Augmented.

Special thanks go to my parents and Filippo for the constant support and encouragement they gave me during my school career.

Special thank goes to Prof. Ruggero Carli for his collaboration in this thesis.

Special thank goes to Euclid Labs S.r.l. and in particular to Ing. Roberto Polesel for following me step by step during this project.