RG-TREES: TRAJECTORY-FREE FEEDBACK MOTION PLANNING USING SPARSE RANDOM REFERENCE GOVERNOR TREES

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

FERHAT GÖLBOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

SEPTEMBER 2018

Approval of the thesis:

RG-TREES: TRAJECTORY-FREE FEEDBACK MOTION PLANNING USING SPARSE RANDOM REFERENCE GOVERNOR TREES

submitted by **FERHAT GÖLBOL** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

ABSTRACT

RG-TREES: TRAJECTORY-FREE FEEDBACK MOTION PLANNING USING SPARSE RANDOM REFERENCE GOVERNOR TREES

Gölbol, Ferhat
M.S., Department of Electrical and Electronics Engineering
Supervisor : Assoc. Prof. Dr. Afşar Saranlı
Co-Supervisor : Assist. Prof. Dr. Mustafa Mert Ankaralı

September 2018, 66 pages

Sampling based methods resulted in feasible and effective motion planning algorithms for high dimensional configuration spaces and complex environments. A vast majority of such algorithms as well as their application rely on generating a set of open-loop trajectories first, which are then tracked by feedback control policies. However, controlling a dynamic robot to follow the planned path, while respecting the spatial constraints originating from the obstacles is still a challenging problem. There are some studies which combine statistical sampling techniques and feedback control methods which address this challenge using different approaches. From the feedback control theory perspective, Reference Governors proved to be a useful framework for constraint enforcement. Very recently, Arslan and Koditschek (2017) introduced a feedback motion planner that utilizes Reference Governors that provably solves the motion planning problem in simplified spherical worlds. In this context, here we propose a "trajectory-free" novel feedback motion planning algorithm which combines the two ideas: random trees and reference governors. Random tree part of the algorithm generates a collision-free region as a set of connected simple polygonal regions. Then, reference governor part navigates the dynamic robot from one region to the adjacent region in the tree structure, ensuring it stays inside the current region and asymptotically reaches to the connected region. Eventually, our algorithm robustly routes the robot from the start location to the goal location without collision. We demonstrate the validity and feasibility of the algorithm on simulation studies.

Keywords: Reference Governor, Rapidly-exploring Random Trees (RRT), Sequential Composition, Obstacle Avoidance, Motion Planning

ÖZ

RG-TREES: SEYREK RASSAL REFERANS YÖNETİCİ AĞAÇLARI İLE YÖRÜNGESİZ GERİ BESLEMELİ HAREKET PLANLAMA

Gölbol, Ferhat
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi : Doç. Dr. Afşar Saranlı
Ortak Tez Yöneticisi : Dr. Öğretim Üyesi Mustafa Mert Ankaralı

Eylül 2018 , 66 sayfa

Örnekleme tabanlı yöntemler, yüksek boyutlu konfigürasyon uzayları ve karmaşık ortamlar için uygulanabilir ve etkili hareket planlama algoritmalarına imkan sağlamıştır. Bu algoritmaların büyük çoğunluğu, öncelikle bir açık döngü yörünge elde edip, sonradan bu yörüngeyi geri beslemeli kontrol ilkeleriyle takip etmeye dayalıdır. Ancak dinamik bir robotu planlanmış bir yolu izleyecek şekilde kontrol etmek, aynı zamanda engellerden kaynaklanan uzaysal kısıtlamalara uymasını sağlamak hala zorlu bir problemdir. Literatürde örnekleme tabanlı yöntemlerle geri beslemeli kontrol metotlarını birleştirerek bu problemi ele alan uygulamalar bulunmaktadır. Geri beslemeli kontrol teorisi literatüründe Referans Yöneticileri, kısıtlamaları uygulatmak için kullanışlı bir yöntemdir. Yakın zamanda Arslan ve Koditschek (2017), Referans Yöneticisini kullanarak hareket planlama problemini basit küre dünyalarda çözen bir yöntem ortaya koymuştur. Bu bağlamda bu çalışmamızda biz, rasgele ağaçlar ve referans yöneticilerini birleştiren yeni bir "yörüngesiz" hareket planlama algoritması sunuyoruz. Algoritmamızın rasgele ağaç kısmı, birbirine bağlı basit çokgensel bölgelerin bileşimi olan, içinde engel bulunmayan güvenli bir alan yaratır. Referans yöneticisi kısmının görevi, dinamik robotu bulunduğu bölgeden, ağaç üzerinde bu bölgenin bağlı olduğu diğer bölgeye götürmek ve bunu yaparken robotun bölgenin içinde kalmasını sağlamaktır. Bunun sonucunda algoritmamız robotu gürbüz bir şekilde, engellere çarpmadan başlangıç konumundan hedef konumuna taşır. Algoritmamızın geçerliliğini ve uygulanabilirliğini simülasyonlar üzerinde gösterdik.

Anahtar Kelimeler: Referans Yöneticisi, Hızlı Keşfeden Rasgele Ağaçlar, Sıralı Ardışık Bileşim, Engelden Kaçınma, Hareket Planlama

To my family

ACKNOWLEDGMENTS

I would like to start by thanking my supervisors Afşar Saranlı and Mustafa Mert Ankaralı for their support, guidance and help throughout my academic studies. They have always responded to my needs, they have listened to me patiently and given me advices whenever I needed. We had invaluable discussions in our research meetings, which helped me gain a deep understanding on how to approach scientific problems.

It was Ömür Arslan who introduced to me the concept of Reference Governors, which plays a major part in this study. I would like to thank him, for his presenting their 2017 ICRA paper in our laboratory, ATLAS, and for the helpful conversations we had afterwards. I am also grateful to Uluç Saranlı for his guidance in his course, Robot Motion Planning and Control, and in ATLAS.

I had the opportunity to work with an amazing group of people in Robotics and Autonomous Systems Laboratory, RoLab, and in ATLAS. Among them, special thanks go to Başer Kandehir, with whom I worked together throughout my M.S. studies. I thank Jean Piere Demir, Osman Kaan Karagöz, Halil İbrahim Uğurlu, Abdullah Cem Önem, Nurullah Gülmüş, Lütfullah Tomak, Ayşe Deniz Duyul, Merve Özen and Seyit Yiğit Sızlayan. They made the hours I spent in the laboratory and my M.S. journey enjoyable.

I am thankful to TÜBİTAK, the National Scientific and Technological Research Council of Turkey, for granting me their M.S. studies scholarship.

And above all, I am thankful to my mother, Feride Gölbol, my father, Heysem Gölbol, my sister Esra Gölbol, and my brother, Ali Gölbol, for their endless love and support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

RG	Reference Governor
RRT	Rapidly-exploring Random Trees
RRG	Rapidly-exploring Random Graph
O_∞, MOAS	Maximal Output Admissible Set

CHAPTER 1

INTRODUCTION

Obstacle avoidance motion planning is one of the most fundamental tasks of mobile robotics. The problem can be stated as follows: given the robot dynamics and the description of the environment, the motion planner finds a sequence of commands such that the robot reaches the goal configuration(s), while respecting the constraints such as avoiding collisions with the obstacles, remaining within velocity, acceleration and motor voltage limits. Traditionally, an off-line planner determines a collision-free trajectory that reaches a goal configuration for the kinematic model of the robot, and an on-line feedback controller follows that trajectory as closely as possible [1].

In this thesis, we propose a new trajectory-free, sampling based, computationally efficient motion planning algorithm that can handle arbitrary obstacle configurations provided that the system dynamics are linear or feedback linearizable [2]. The algorithm is composed of two stages: a random tree generation stage and a feedback motion control stage. The block diagram topology of our method is illustrated in. Fig. 1.1. The motion planning stage generates a "tree" that connects the start location to the goal location, where the nodes are collision-free, overlapping, square-shaped areas/volumes. The goal location is located inside the root of the tree, and start location is located at a leaf node. Then, the reference-governor-based motion control stage navigates the robot from the robot's current node to the parent node guaranteeing that no constraint violation occurs, eventually to the root node where the goal position resides.

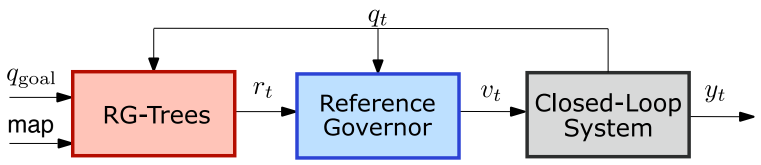


Figure 1.1: Block diagram of the algorithm. RG-Trees algorithm determines medium term set points, r_t, which are reachable from the current configuration q_t and steer to the goal location q_{goal}. Reference governor modifies r_t such that the closed-loop robot does not collide with the obstacles.

1.1 Literature Review

Several recent studies utilize the reference governor framework for motion planning applications under some constraints. Arslan and Koditschek [3] propose a provably correct, computationally efficient motion planner for a world of spherical obstacles. Petersen et al. [4] use rotating virtual hyperplane concept, introduced by Park et al. [5], which puts an additional, time varying constraint to avoid collision. Finally, without emphasizing motion planning, Gilbert and Kolmanovsky [6] suggest hybrid reference governors: the idea of dividing the configuration space into convex, collision-free subsets, which are managed using reference governors. Technically our method in this paper, random reference governor trees for feedback motion planning, shares some principles with this study.

There exist numerous modifications and extensions of RRT, each of which concentrating on a different aspect of the algoritm. For example, some researchers focused on the optimality aspects of the algorithm, [7–10], whereas some others extended the basic algorithm to solve dynamic motion planning problems [11–15]. On the other hand, similar to our method, some researchers modified the RRT algorithm (or other sampling based techniques) to use regions (*funnels*) instead of points, which are then acted as the "basins of attraction" of some local feedback control policies [1,16,17]. These feedback policies are connected in the essence of the sequential composition idea introduced by Burridge et al. [18]. Each region is associated with a feedback control policy, which is responsible for navigating the robot inside the area/volume to a final location which is also located inside a different but sequentially connected region.

1.2 Methodology and Contributions

Our algorithm consists of two stages: a random tree generation stage and a motion control stage. The random tree generation stage constructs, in an RRT-like manner, a directed tree of connected rectangular regions which are located in the obstacle-free region. The resultant directed tree structure can also be considered as a sequentially composed "funnel" set, similar to [1, 17]. In this context, the goal state is located inside the region (basin of attraction) of the root node/funnel. The motion control part is responsible for steering the robot from its current node to the parent node, ensuring that the robot always remains inside the current node and does not violate other constraints. When the robot enters the intersection of those two nodes, the parent node now becomes the active node, and the procedure repeats inside this new region. This process guarantees that the robot reaches the goal location, as it is in the root of the tree. In order to guarantee that the robot strictly stays inside the region defined by the active node and no other constraint violation occurs, we use the concept of reference governors. Overall, RG-Trees algorithm guarantees collision free dynamic navigation from any point that is located inside one of the convex regions to the goal location.

The main contribution of this thesis is the synthesis of reference governors, sampling-based planning, and sequential composition. We propose to split the constraints which ensure that the robot remains in the square region defined by the active node from all other constraints, such as speed and acceleration limits, and to use two RGs in a cascade configuration. We show the rotation, translation and scale invariance of the motion of a double integrator. With the remark that every square in the 2D plane is the unit square, which is rotated, translated and scaled suitably, we show that for all square-shaped nodes in the tree, we can reuse the RG which is calculated for the unit square. Therefore, we show that in the presence of other constraints, our algorithm calculates only 2

RGs, regardless of the number of nodes on the tree and their orientations. For each node, the algorithm uses the first RG by rotating, translating and scaling, and the second RG directly.

We also propose a couple of enhancements to the basic RG-trees algorithm. We first introduce a node/region expansion phase at each iteration of the tree generation step to enhance sparsity of the planning algorithm. We then propose a method to optimize the RG-tree structure which is similar to how RRG (Rapidly Exploring Random Graphs) optimizes the RRT [19]. However, although optimization step optimizes the path length, we show that it has some negative effect on the navigation speed of the robot due to small intersections between some nodes. In order to handle, this problem, we propose the usage of "gateway nodes" which are generated in real-time. We implemented and tested RG-Trees and its applicability and feasibility using Matlab simulations on two different 2D environments. In addition to some illustrative results, we also performed 1000 Monte-Carlo simulations to better assess the performance of different cases. We defined some performance parameters and presented their histogram plots and comparative combined box plots.

1.3 Organization of the Thesis

This thesis is organized as follows. In Chapter 2, we give the relevant background on reference governors, RRT and sequential composition. Subsequently, in Chapter 3, we introduce our fundamental algorithm, RG-Trees, which combines these ideas to formulate a guaranteed obstacle avoidance dynamic motion planner. We also explain our proposed improvements on the fundamental algorithm, which are node expansion, tree optimization, and gateway nodes. Then, to test the feasibility and effectiveness of the algorithm, we make Monte-Carlo simulations with different parameter configurations. We share the implementation details and results in Chapter 4. Finally, we draw some conclusions in Chapter 5.

CHAPTER 2

BACKGROUND

RG-Trees algorithm combines ideas and concepts from reference governors [20], rapidly-exploring random trees [21] and sequential composition of dynamic behaviors [18]. For the sake of completeness, we briefly covered the important principles of these methods in this Chapter.

2.1 Reference Governors

The reference governor is an add-on scheme/controller that modifies the reference command of a well-designed closed loop system [20] in order to enforce some pointwise-in-time state and control input constraints. In a reference governor based controller topology, one first designs a closed-loop controller ignoring the constrains emphasizing other performance metrics and criteria such as controller simplicity, speed of response, robustness, and disturbance rejection. Then, he/she adds a reference governor in a cascade configuration around the closed loop system to ensure constraint enforcement [6]. Fig. 2.1 illustrates the combined controller topology [22].

Consider a closed loop system given with discrete time state space model

$$\begin{aligned} q_{t+1} &= Aq_t + Bv_t \\ y_t &= Cq_t + Dv_t \end{aligned} \tag{2.1}$$

and constraint set, $y_t \in Y \ \forall t$. The task of RG is to find the optimal modified reference signal, v_t, given the initial state q_t and unconstrained reference r_t, such that

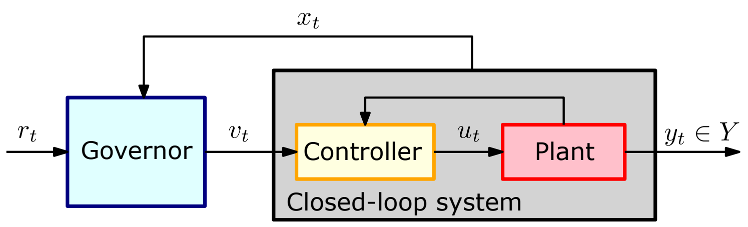


Figure 2.1: Block diagram of a reference governor applied to a closed loop system. In this type of control schemes, an engineer first designs the Controller (yellow) block ignoring the constrains, then he/she designs the reference governor such that whole closed-loop system satisfies the enforced constrains.

- v_t is as close to r_t as possible
- if v_t is kept constant, no constraint violation occurs in the subsequent motion

Definition 1 *Given a dynamical system $\mathcal{S}$ and a constraint set $y_t \in Y$, the pair of an initial state and a reference signal, (q_0, v), is output admissible if, starting from the initial state q_0 and keeping the reference signal constant at v, no constraint violation occurs in the subsequent motion.*

Definition 2 *Given a dynamical system $\mathcal{S}$ and a constraint set $y_t \in Y$, the set of all output admissible pairs is called the Maximal Output Admissible Set (MOAS), O_∞ [23].*

Algorithmic determination of O_∞ is explained in detail in [23] and summarized in Subsection 2.1.1. From [20], we have that if

- the matrix pair (C, A) is observable
- the matrix A is asymptotically stable
- the constraint set Y is a polytope containing the equilibrium point in its interior, i.e., there exist a matrix S and a vector s for which

$$Y = \{y \mid Sy \leq s\}^1 \tag{2.2}$$

[1] Note that the vector comparison operator $\leq$ here indicates that all elements of the LHS are smaller than or equal to the corresponding element in the RHS.

MOAS turns out to be a polytope:

$$O_\infty = \{(q_0, v) \mid H_q q_0 + H_v v \leq h\}, \tag{2.3}$$

for which H_q, H_v and h can be calculated offline, if the system is time invariant and the constraint set is static.

For MOAS calculated in (2.3), the Scalar Reference Governor is formulated as follows: the modified reference v is initialized to an initially output admissible value given the initial state q_0. Then, it is updated in a loop according to the following equation:

$$v_t = v_{t-1} + \kappa_t(r_t - v_{t-1}), \tag{2.4}$$

where κ_t is maximized subject to

- $0 \leq \kappa_t \leq 1$
- (q_t, v_t) is output admissible, i.e., $H_q q_t + H_v v_t \leq h$

2.1.1 Calculation of O_∞

Rewrite the state space equations given in (2.1) as

$$\begin{aligned} \bar{q}_{t+1} &= \bar{A}\bar{q}_t \\ y_t &= \bar{C}\bar{q}_t \end{aligned} \tag{2.5}$$

where

$$\bar{q}_t = [q_t \ v_t]^T, \quad \bar{A} = \begin{bmatrix} A & B \\ 0 & I \end{bmatrix}, \quad \bar{C} = \begin{bmatrix} C & D \end{bmatrix} \tag{2.6}$$

so that

$$y_t = \bar{C}\bar{A}^t\bar{q}_0 \tag{2.7}$$

and for the constraint set $y_t \in Y$, define the k-time-step maximal output admissible set

$$O_k = \left\{\bar{q}_0 \mid \bar{C}\bar{A}^t\bar{q}_0 \in Y \quad t = 0, 1, ..., k\right\} \tag{2.8}$$

Note that O_∞ is the infinite horizon maximal output admissible set.

Clearly,

$$O_\infty \subset O_{k_1} \subset O_{k_2} \quad \forall k_1, k_2 \in \mathbb{Z}^+, \ k_1 > k_2 \tag{2.9}$$

It can be shown that if $O_{k^*+1} = O_{k^*}$ for some $k^* \in \mathbb{Z}^+$, then $O_k = O_{k^*}\ \forall k \geq k^*$, thus $O_\infty = O_{k^*}$ [23]. Therefore, to calculate O_∞, it is sufficient to show that two consecutive sets in the sequence $O_1, O_2, ...$ are equal to each other.

Consider the constraint set given in (2.2). In order to show that two sets are equal, rewrite the constraint set as

$$O_k = \{y \mid Sy - s \leq 0\} = \left\{y \mid S\bar{C}\bar{A}^t\bar{q}_0 - s \leq 0 \quad t = 0, 1, .., k\right\} \tag{2.10}$$

Now, to check whether the newly introduced constraint in O_{k+1} reduces the set, solve the following optimization problem:

$$\text{maximize } J_{k+1}(\bar{q}_0) = S\bar{C}\bar{A}^{k+1}\bar{q}_0 - s \tag{2.11}$$

subject to

$$J_t(\bar{q}_0) = S\bar{C}\bar{A}^t\bar{q}_0 - s \leq 0 \quad t = 0, 1, .., k \tag{2.12}$$

This optimization procedure is a linear programming problem and there exist tools to solve it [24]. If all rows of the maximum value are less than 0, set $O_\infty = O_k$. Otherwise, try the same procedure for larger k.

2.1.2 1D Reference Governor Example

To illustrate the effect of RG on the reference signal and state variables, we implemented the one-dimensional double integrator system given in [20]. Continuous time system model and controller are given as

$$\begin{aligned} \dot{x_1} &= x_2 \\ \dot{x_2} &= u \\ u &= -0.917(x_1 - r) - 1.636x_2 \end{aligned} \tag{2.13}$$

and the constraint set is given as

$$|x_1| \leq 1, \quad |x_2| \leq 0.1, \quad |u| \leq 0.1 \tag{2.14}$$

The set point is $r = 0.5$. The system is discretized for the RG at a sampling rate of $T_s = 0.1s$. Given the output definition, $y_t = [x_1\ x_2\ u]^T$, state space equations take the form

$$x_{t+1} = \begin{bmatrix} 1 & 0.1 \\ -0.0917 & 0.8364 \end{bmatrix} x_t + \begin{bmatrix} 0 \\ 0.0917 \end{bmatrix} r_t,$$

$$y_t = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ -0.917 & -1.636 \end{bmatrix} x_t + \begin{bmatrix} 0 \\ 0 \\ 0.917 \end{bmatrix} r_t \tag{2.15}$$

and the constraint with respect to the definition in (2.2) can be written as

$$\begin{bmatrix} 1 & 0 & 0 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & -1 \end{bmatrix} y_t \leq \begin{bmatrix} 1 \\ 1 \\ 0.1 \\ 0.1 \\ 0.1 \\ 0.1 \end{bmatrix} \tag{2.16}$$

Fig. 2.2 illustrates a simulation result without and with RG for the given example system. It is clear that when there is no RG, i.e. Fig. 2.2(a), the system trajectories violate the constraints on x_2 and u. When we activate the RG, i.e. Fig. 2.2(b), the modified reference signal evolves nonlinearly and reaches in finite time to the unmodified reference such that no constraint violation occurs in the simulation.

2.2 RRT-Based Motion Planning

Rapidly-Exploring Random Trees (RRT) algorithm constructs a tree of feasible paths by incrementally adding a random, collision-free edge to the tree [21]. Its algorithmic parameters such as time complexity, space complexity, completeness and optimality are discussed in [19].

The algorithm works as follows: The tree is initialized to the start location as its root. Then, a random point q_{rand} is sampled from the obstacle-free space. The closest vertex to q_{rand} on the tree, $q_{nearest}$, is determined. Then, the robot configuration at $q_{nearest}$ is steered towards q_{rand}, giving a new point q_{new}. Usually the step length is limited, thus q_{new} is a closer point to $q_{nearest}$. Finally, if the link connecting q_{new} to $q_{nearest}$ is collision-free, the vertex q_{new} and the edge $(q_{new}, q_{nearest})$ are added to the tree. This procedure iterates until the goal

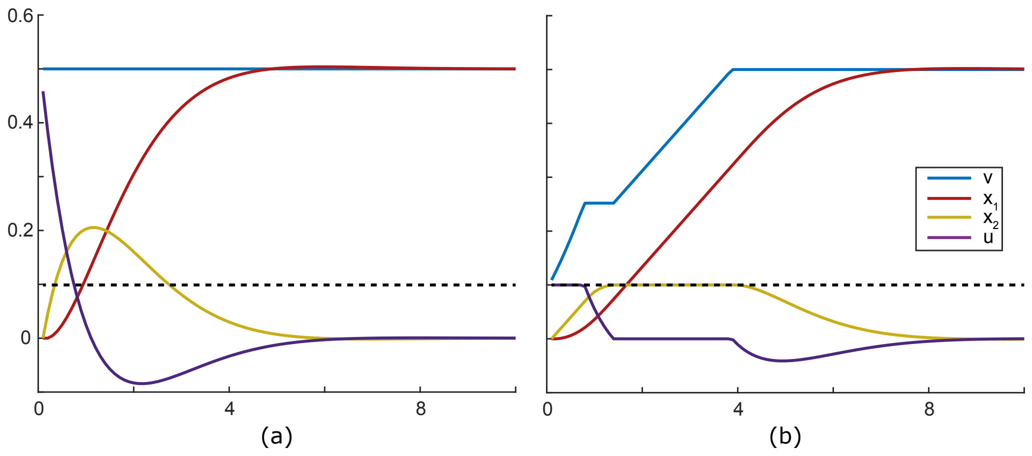


Figure 2.2: Simulation results of the example system (2.15) without RG (no constraints) and with RG (constraint enforcement). In these Simulations solid red, yellow, and dark blue lines belong to state trajectories and input respectively, wehereas solid light blue and dashed black lines illustrate the "reference" signal and the constraint limit respectively. (a) Unconstrained simulation (no RG) of the system in (2.15). In this case, reference signal is unmodified and constant at all times. Thus, both x_2 and u violate the constraints. (b) Simulations with RG. RG successfully enforces the constraints by modifying the reference signal.

location is reached by a vertex. The procedure is given in Algorithm 1 [19], and illustrated in Fig. 2.3.

If the samples are drawn uniformly, the probability that any particular vertex in the tree becomes $q_{nearest}$ is proportional to the area of its Voronoi region. In addition, larger Voronoi regions occur on the frontier of the tree. As a result, RRT tends to grow towards the unexplored areas.

Algorithm 1 RRT

```
1:  $T \leftarrow$ InitializeTree($V \leftarrow \{q_{init}\}$, $E \leftarrow \emptyset$)
2:  for $i = 1$ to $N$ do
3:      $q_{rand} \leftarrow$ SampleFreeSpace($i$)
4:      $q_{nearest} \leftarrow$ Nearest($G = (V, E)$, $q_{rand}$)
5:      $q_{new} \leftarrow$ Steer($q_{nearest}$, $q_{rand}$)
6:      if ObstacleFree($q_{nearest}$, $q_{new}$) then
7:          $V \leftarrow V \cup \{q_{new}\}$
8:          $E \leftarrow E \cup \{(q_{nearest}, q_{new})\}$
9:      end if
10: end for
11: return $G = (V, E)$
```

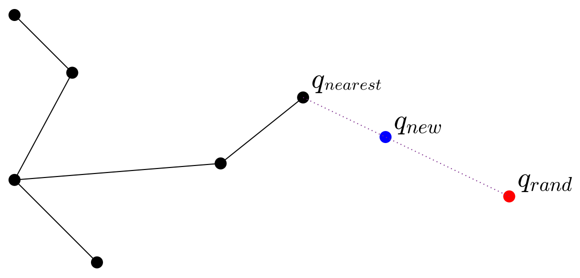


Figure 2.3: RRT tree expansion process. A random state, q_{rand} is sampled from the free space. The nearest node on the tree, $q_{nearest}$, is steered towards q_{rand} to obtain a new state, q_{new}. If the connection between $q_{nearest}$ and q_{new} is collision-free, q_{new} is added to the tree.

2.3 Sequential Composition

In the previous literature, the metaphor of a *funnel* is used to describe feedback motion planners. The funnel's task is to collect a large set of initial configurations and steer them to a smaller set of goal configurations. In this metaphor, the region of attraction of the feedback controller is the inlet of the funnel, and goal configuration set is its outlet. An example funnel is illustrated in Fig.2.4.a.

It is usually difficult, sometimes even impossible, to find a single global funnel, whose inlet is the entire free space [25, 26], as depicted in Fig.2.4.b. In such cases, the global funnel can be constructed as a sequential composition of local funnels [1, 17, 18]. Each local funnel's task now is to steer the configurations in its inlet set to the next funnel, eventually to the *master funnel*, whose outlet is the goal configuration. This sequential composition is illustrated in Fig. 2.4.c.

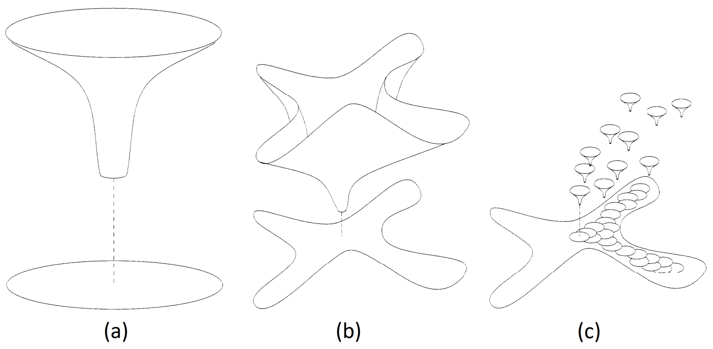


Figure 2.4: Sequential composition. (a) The funnel steers a large set of initial conditions to a smaller set. (b) Finding a global control policy for a non-convex obstacle-free set is difficult. (c) Constructing the control policy as a composition of local funnels is simpler.

CHAPTER 3

RG-TREES: A NOVEL FEEDBACK MOTION PLANNING APPROACH

RG-Trees algorithm consists of two stages. In the first stage, a tree structure that is composed of connected square shaped regions is generated using the map of the environment, ignoring the robot motion dynamics. First region is created around the goal location. Then, tree extension is repeated until one of the newly created regions covers the start location. In this way we ensure that there is a “safe connection” between the start location and the goal location.

In the second stage, a reference governor based control strategy that navigates the dynamic robot to goal location is used. It is important to note that this algorithm does not generate an open-loop trajectory. Instead, it calculates a set of connected regions where the robot can be controlled safely. The trajectory of motion is then determined by the equations of motion of the robot.

The basic RG-Trees algorithm and node expansion process have been reported in a conference paper [27].

3.1 Tree Generation

The aim of the tree generation stage is to cover the configuration space, fully or partially, by connected nodes (regions) for which the robot’s motion can be controlled independently, to construct a global motion planner [1, 17]. In [1], configuration space is covered by balls probabilistically, and a graph structure is constructed. Then, global navigation problem is reduced to a search problem on the graph. In [17], instead of a graph, a tree structure is used and tree generation is stopped as soon as initial and goal configurations are connected, emphasizing computational simplicity over path optimality. In these methods the nodes are chosen to be circular, because of the applied control methods (Navigation functions in [1] and Lyapunov function based controller in [17]), although there are studies to use rectangular regions of attraction with navigation functions [28].

In this work, we use a reference governor based controller. Since finite determination of reference governors is guaranteed under convex polygonal constraint set [20], our algorithm's nodes are not circular. For the sake of simplicity and feasibility, we adopted square-shaped regions. Note that one can use the algorithm using any convex polygon or a set of predetermined convex polygons, with only slight modifications.

We illustrate the tree generation algorithm in Fig.3.1. The algorithm starts from the goal location. A node is generated around that point (see Sec.3.1.1), and this node is expanded (see Sec.3.1.2). Then, until a stop condition is satisfied, the following procedure is repeated:

- a random point, q_{rand}, is sampled from the space which is not covered by a node
- from the covered area, the closest point to this sample, $q_{nearest}$, is calculated
- a new node is generated and expanded around that point

Termination condition can be that the configuration space is probabilistically covered [1]. However, in our example, tree generation is terminated as soon as the start location is covered by a node, as in [17]. The rationale behind this choice is that tree generation algorithm is incremental, if the robot leaves the covered space, new nodes can be added on-line, and the algorithm is sufficiently fast for a variety of systems to be performed in real time.

After all nodes are generated, the constructed tree is optimized with respect to approximate path length (see Sec.3.1.3). Pseudocode for RG-Tree generation is given in Algorithm 2.

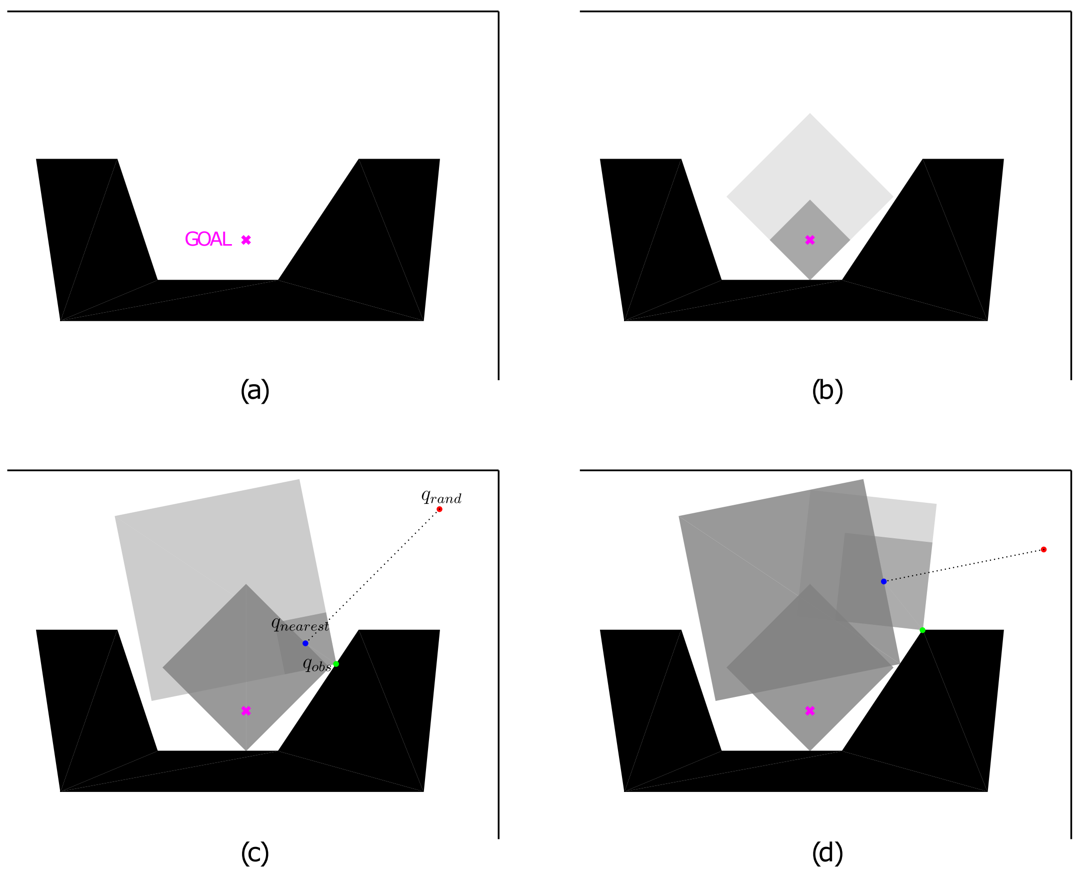


Figure 3.1: RG-Tree generation algorithm: (a) Initial map of the arena, (b) A node generated(dark grey) and expanded(light grey) around goal position, (c) A random point(red) sampled from the free space, closest point to the previous tree(blue) calculated, a node generated(dark grey) and expanded(light grey) around that point, (d) Another node is generated and expanded as in (c).

3.1.1 Node Generation

When a new node is to be generated around a point $q_{nearest}$, the closest point on an obstacle to that point, q_{obs}, is calculated. Then, a square region is constructed with the former point being its center and the latter being its first corner.

The corner point is stored as the offset of the node's reference frame, (x_0, y_0), and the angle between the first edge and horizontal is stored as the node's rotation, θ, see Fig. 3.2. Moreover, edge length of that square is stored as the node's scale.

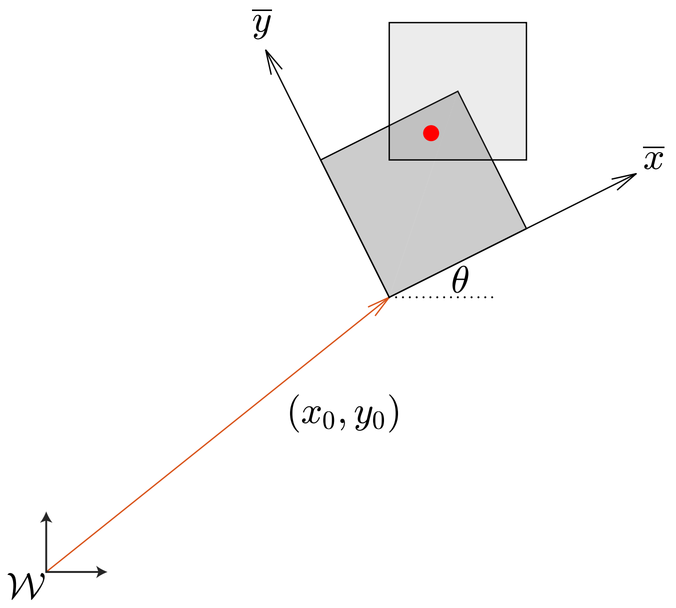


Figure 3.2: Two consecutive nodes, CurrentNode(dark grey) and NextNode(light grey), generated by the algorithm. The centroid of the intersection(red point) is the set-point in CurrentNode. $(\bar{x}, \bar{y})$ is CurrentNode's coordinate frame, offset by (x_0, y_0) and rotated by θ from the global frame.

This procedure can be formalized as follows: Let WO_i be the i$^{\text{th}}$ obstacle in the workspace and WO be the obstacle set,

$$WO = \bigcup_i WO_i. \tag{3.1}$$

Let also $\mathcal{B}$ be the set of all points which are covered by a square node,

$$\mathcal{B} = \bigcup_i Node_i. \tag{3.2}$$

Then,

$$q_{nearest} = \underset{q \in \mathcal{B}}{\arg\min} \|q - q_{rand}\| \tag{3.3}$$

$$q_{obs} = \underset{q \in WO}{\arg\min} \|q - q_{nearest}\| \tag{3.4}$$

This definition ensures that a ball with center $q_{nearest}$ and radius $r = \|q_{nearest} - qobs\|$ resides inside the obstacle-free region, and therefore that any polygon which is inscribed by this ball also resides totally inside the obstacle-free region. We choose the polygon to be a square with center being $q_{nearest}$ and one of its corners being q_{obs}.

$$\begin{aligned} Node_i.\text{offset} &= q_{obs} \\ Node_i.\theta &= \angle\left(q_{nearest} - q_{obs}\right) - \frac{\pi}{4} \\ Node_i.\text{scale} &= \|q_{nearest} - q_{obs}\| \cdot \sqrt{2} \end{aligned} \tag{3.5}$$

3.1.2 Node Expansion

Usage of larger nodes yields two major advantages. Firstly, since larger nodes cover larger areas of the free space, they result in a more sparse tree. Secondly, they allow the robot to move faster. The reference governor ensures that the robot's motion is confined to the current node, which implicitly incurs a velocity constraint on the robot. Larger nodes result in looser constraints.

For the sake of simplicity, the nodes are expanded in discrete steps. At each step, the node's scale is multiplied by a constant factor γ, keeping the offset and the rotation unaltered. If the expanded node collides with an obstacle, the last expansion is reverted. If no collision occurs, the procedure repeats.

Larger values for multiplication constant γ yield lower calculation time with lower spatial resolution, and smaller values yield better spatial resolution at the expense of slower calculations. A constant value of $\gamma = 1.2$ resulted in satisfactory results in our experiments.

3.1.3 RG*-Trees: Optimized RG-Trees

A randomly generated tree is not likely to be optimal. However, in some applications, the path is expected to be optimal with respect to a given cost function, such as path length or execution time. To ensure optimality, some extra computational steps must be taken.

Several attempts have been made to find the "optimal" versions of sampling based techniques im the literature. Among those, two are based on RRT: RRG and RRT* [19]. In RRG, instead of a tree structure that is used in RRT, new edges are generated (keeping the nodes fixed) such that tree structure is transformed into an undirected *graph.* The algorithm starts like RRT, it attempts to connect the sample point q_{new} to the closest node on the graph, as given in Algorithm 1. If this connection is successful, RRG algorithm goes on to attempt to connect q_{new} to all nodes within a neighborhood of it. After the graph generation, optimal path is found using a graph search algorithm, usually A*. RRG is best suited for multiple query missions with distinct start and goal configurations.

RRT* constructs the optimal tree incrementally. The tree is obtained by removing from RRG the edges which are not on a shortest path from any leaf to the root. This algorithm also starts like RRT. If the sample point q_{new} can be connected to the tree, RRT* attempts to connect q_{new} to all nodes within a neighborhood of it. The algorithm constructs a *local* graph of connected nodes this way. It runs a search algorithm on this local graph, and rewires the optimal tree accordingly, if necessary. Since it stores the same nodes as and only a subset of the edges of RRG, RRT* is more memory-efficient compared to RRG. Having a directed tree structure, this algorithm is used for single query missions or multiple query missions with a single goal location.

Similar to RRT*, RG-Trees algorithm returns the optimal path in a tree data structure. However, unlike RRT*, in our algorithm there is no need for an optimisation step at every sample locally. Instead, RG-Trees algorithm performs the optimization as a batch after the full RG-tree is constructed, similar to RRG, putting emphasis on sparsity of the tree. Yet, in contrast to RRG, the graph structure is not constructed by finding all connections at each sample either. Tree-to-graph conversion is also carried out after the full tree is generated. In this context, compared to both optimization steps in RRT* and RNG, the additional computational burden of RG*-Trees is quite limited.

RG-Trees first generates a tree of connected nodes, as RRT does. The nodes are square-shaped regions in the *obstacle-free* workspace, see Section 3.1.1. Since all nodes reside totally in the obstacle-free space, a direct feasible path exists from a point P_1 in a node $node1$ to another point P_2 in another node $node2$ if and only if the regions defined by these nodes have overlapping interiors [1,6]:

$$\mathrm{int}\,(node1 \cap node2) \neq \emptyset. \tag{3.6}$$

Therefore, the graph can be constructed by finding and connecting all pairs of intersecting nodes. All intersecting pairs of N polygons can be found with Bentley-Ottmann algorithm [29] with $O(N \log N)$ time complexity. For the sake of simplicity, the Euclidean distance between the centers of the overlapping nodes are used as the edge weights, i.e. transition costs, although better heuristics can be formulated. The resulting graph is similar to the Random Neighborhood Graph [1].

Finally, we apply Dijkstra's algorithm to find the shortest paths from all leaf nodes to the root. Like RRT*, we remove all edges which are not on a shortest path and convert the graph into a directed optimal tree. Fig. 3.3 and Fig. 3.4 illustrate the tree optimization process.

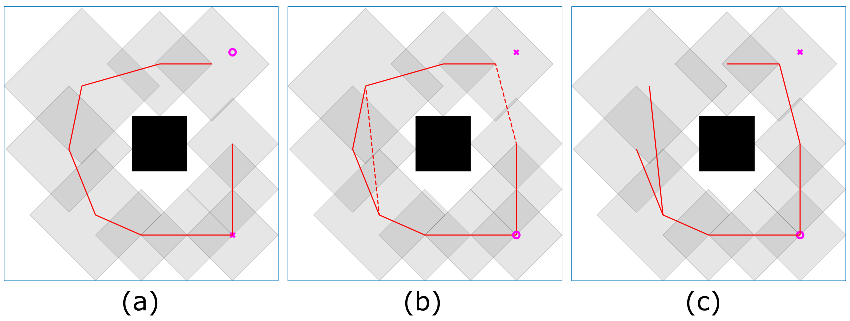


Figure 3.3: RG-Tree optimization. (a) A random tree generated by the algorithm. The robot needs to turn CCW around the obstacle. (b) Neighborhood graph that connects all intersecting node pairs. Dashed lines illustrates the newly generated edges. (c) The optimal tree obtained by running Dijkstra's algorithm on the graph. A shorter, CW path is found.

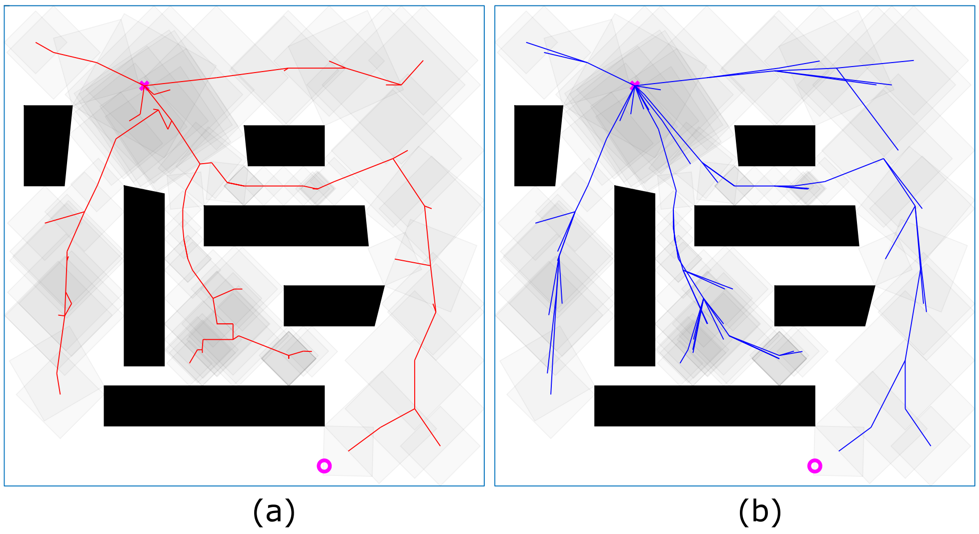


Figure 3.4: (a) The tree generated before optimization, (b) Optimal tree with the same nodes

Algorithm 2 RG Tree Generation

```
1:  $GoalNode \leftarrow \text{GenerateSquareRegion}(q_{goal})$
2:  $GoalNode \leftarrow \text{Expand}(GoalNode)$
3:  $T \leftarrow \text{InitializeTree}(GoalNode)$
4:  for $k = 1$ to $K$ do
5:      $q_{rand} \leftarrow \text{SampleFree}()$
6:      $q_{nearest} \leftarrow \text{Nearest}(T, q_{rand})$
7:      $Node_k \leftarrow \text{GenerateSquareRegion}(q_{nearest})$
8:      $Node_k \leftarrow \text{Expand}(Node_k)$
9:      $T.\text{InsertNode}(Node_k)$
10:     if $q_{init} \in T$ then
11:         $T \leftarrow \text{OptimizeTree}(T)$
12:         return T
13:     end if
14: end for
15: Return $NoSolution$
```

3.2 Motion Control

Gilbert and Kolmanovsky [6] proposed a hybrid reference governor method, which divides the configuration space into overlapping, convex sets, and designs a *different* reference governor for each set. However, calculation of MOAS for a given dynamical system and constraint set is computationally expensive. In our study, for computational efficiency and considering limitations in real-time motion planning applications, we propose a method to minimize the number of MOAS calculations. Observe the following:

- MOAS calculation is linear in the constraint set,
- If the equations of motion of the system are symmetric under rotation, MOAS can be rotated (and translated),
- A reference governor with a large number of constraints can be divided into two or more reference governors which address different constraints of the older reference governor.

The first statement can be rewritten as follows: If

$$MOAS\left(\{y \mid Sy \leq s\}\right) = \{(q_0, v) \mid H_q q_0 + H_v v \leq h\}, \tag{3.7}$$

for a particular system, then

$$MOAS\left(\{y \mid Sy \leq \alpha s\}\right) = \{(q_0, v) \mid H_q q_0 + H_v v \leq \alpha h\} \ \forall \alpha. \tag{3.8}$$

That is, for all square-shaped positional constraints of the form

$$\{(x, y) \mid 0 \leq x \leq \alpha,\ 0 \leq y \leq \alpha\}, \tag{3.9}$$

it would suffice to calculate MOAS for the constraint set

$$\{(x, y) \mid 0 \leq x \leq 1,\ 0 \leq y \leq 1\}. \tag{3.10}$$

The second statement indicates that, since the double integrator system is symmetric under rotation, any square-shaped positional constraint set can be converted into the form given in (3.9). For example, for a 2D dynamic robot with the canonical state variable $q = [x\ y\ \dot{x}\ \dot{y}]^T$, for the square node given in Fig. 3.2, conversion is given by

$$\begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & \cos\theta & -\sin\theta \\ 0 & 0 & \sin\theta & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} \bar{x} \\ \bar{y} \\ \dot{\bar{x}} \\ \dot{\bar{y}} \end{bmatrix} + \begin{bmatrix} x_0 \\ y_0 \\ 0 \\ 0 \end{bmatrix} \tag{3.11}$$

The last statement allows that the square-shaped positional constraints, which are required by the algorithm, constitute one reference governor, $RG1$, while all other constraints such as velocity and control input limitations constitute another reference governor, $RG2$, if necessary. Then, for any node of the generated tree, $RG1$ is rotated, translated and scaled accordingly while $RG2$ is used directly. Design of $RG2$ for speed and acceleration constraints is given in Section 3.2.1.

RG-Trees motion control algorithm is as follows: Initially the node where the robot is located is stored as $CurrentNode$ and its parent as $NextNode$. Similarly, the reference governors obtained by rotating, translating and scaling $RG1$ according to these nodes are stored as $CurrentRG$ and $NextRG$, respectively. Recall that the task of the motion control part is to navigate the robot from its current node to the parent node, and eventually to the root node of the tree where the goal location is situated. Therefore, the medium-term non-modified reference signal, r_t, should be inside the intersection of $CurrentNode$ and $NextNode$. We use the centroid of this intersection for r_t:

$$r_t = \text{centroid}(CurrentNode \cap NextNode) \tag{3.12}$$

We also initialize the modified reference signal to the robot's initial location, which is guaranteed to be output admissible for $CurrentRG$ if the robot is initially stationary:

$$v_0 = q_{init} \tag{3.13}$$

Then, in the control loop, v_t is updated according to (2.4), using $CurrentRG$ and $RG2$ in series. When the pair of robot's state and the modified reference, (q_t, v_t), is output admissible for $NextRG$, the robot is said to *enter* $NextNode$. Then, $CurrentNode$ is updated to $NextNode$, and $NextNode$ to its parent node. $CurrentRG$, $NextRG$ and r_t are also updated accordingly. If $CurrentNode$ is the root node, the medium-term reference signal r_t is set to q_{goal}.

3.2.1 Adding Speed and Acceleration Limits

In 2D, speed limit can be written as

$$v_x^2 + v_y^2 \leq v_{max}^2. \tag{3.14}$$

Definition 3 *A set Y is finitely determined if it can be described by a finite set of linear inequalities, that is, if it can be written as in (2.2) with a finite-size matrix S and a finite-size vector s.*

The constraint (3.14) cannot be written as in (2.2) with finite-size S and s, therefore MOAS is not guaranteed to be finitely determined, as in (2.3). Thus, we use a finitely determined inner approximation of this constraint set by constraint tightening [30]. We approximate the circle described in (3.14) with a 32-sided regular polygon, as illustrated in Fig.3.5. Since the polygon's outer radius is v_{max}, the robot is guaranteed to not exceed the speed limit. Moreover, the inner radius of the polygon is $v_{max} \cdot \cos(\pi/32) = 0.9952 \cdot v_{max}$, therefore approximation error is less than 0.5%.

We formulate the acceleration limit similarly as a 32-sided polygon.

3.2.2 Gateways

The reference governor part ensures that the dynamical robot stays inside the square region defined by $CurrentNode$ for all time. Therefore, it is expected

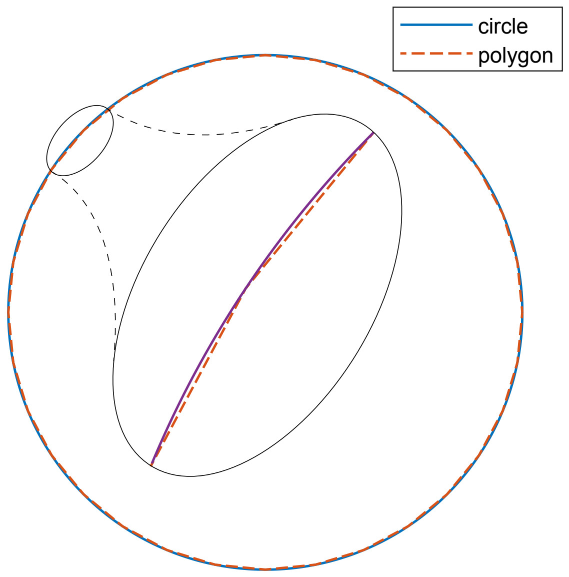


Figure 3.5: Approximating speed limit constraint with a 32-sided polygon. Circle: the actual constraint set, Polygon: its finitely determined inner approximation.

to slow down the robot as it approaches the borders of the region. In the desired operation, before approaching the border, the robot enters $NextNode$. Since $NextNode$'s border is farther away than that of $CurrentNode$, the robot continues its motion without slowing down. However, because the tree is generated randomly and then optimized, it is possible that the intersection between $CurrentNode$ and $NextNode$ is small. In such cases, we observed in our simulations that the robot slows down before entering $NextNode$. In order to prevent this effect, we propose using temporary nodes between $CurrentNode$ and $NextNode$, that we call *gateways*.

When the robot's speed falls below a threshold, the algorithm estimates that a small intersection case occurred. To increase the intersection between consecutive nodes, RG-Tree execution algorithm generates and expands a temporary node, $Gateway$, around the centroid of the intersection of $CurrentNode$ and $NextNode$. Then, it updates the connection $CurrentNode \rightarrow NextNode$ as $CurrentNode \rightarrow Gateway \rightarrow NextNode$ on the tree. This procedure is illustrated in Fig. 3.6.

Algorithm 3 RG Tree Execution

```
1: $CurrentNode \leftarrow T.\mathrm{LastNode}()$
2: $NextNode \leftarrow CurrentNode.\mathrm{Parent}()$
3: $r \leftarrow \mathrm{Centroid}(CurrentNode \cap NextNode)$
4: $v \leftarrow q_{init}$
5: $RG1 \leftarrow \mathrm{ReferenceGovernor}(UnitSquareConstraints)$
6: $CurrentRG \leftarrow \mathrm{RTS}(RG1, CurrentNode)$    ▷ Rotate, Translate and Scale
   RG1 according to CurrentNode
7: $NextRG \leftarrow \mathrm{RTS}(RG1, NextNode)$
8: while $q_{goal}$ not reached do
9:     if in $GoalNode$ then
10:        $r \leftarrow q_{goal}$
11:    else if $\mathrm{OutputAdmissible}(q, NextRG)$ then
12:        $CurrentNode \leftarrow NextNode$
13:        $NextNode \leftarrow NextNode.\mathrm{Parent}()$
14:        $r \leftarrow \mathrm{Centroid}(CurrentNode \cap NextNode)$
15:        $CurrentRG \leftarrow NextRG$
16:        $NextRG \leftarrow \mathrm{RTS}(RG1, NextNode)$
17:    end if
18:    if speed $<$ THRESHOLD then
19:        $Gateway \leftarrow \mathrm{GenerateSquareRegion}(r)$
20:        $Gateway \leftarrow \mathrm{Expand}(Gateway)$
21:        $Gateway.Parent \leftarrow NextNode$
22:        $NextNode \leftarrow Gateway$
23:        $NextRG \leftarrow \mathrm{RTS}(RG1, Gateway)$
24:        $r \leftarrow \mathrm{Centroid}(CurrentNode \cap Gateway)$
25:    end if
26:    $v \leftarrow \mathrm{ModifiedReference}(q, r, v, CurrentRG)$
27:    $v \leftarrow \mathrm{ModifiedReference}(q, r, v, RG2)$
28: end while
```

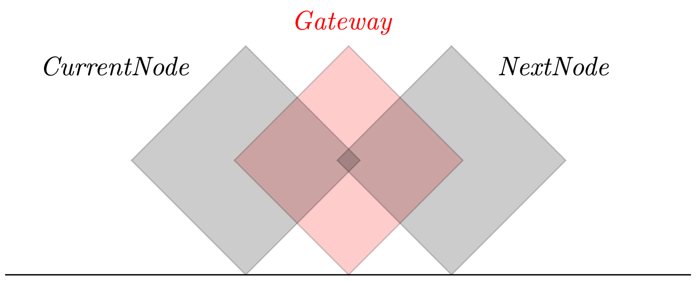


Figure 3.6: Originally, the connection is $CurrentNode \rightarrow NextNode$, for which the intersection is small. We generate a temporary node $Gateway$ and update the connection as $CurrentNode \rightarrow Gateway \rightarrow NextNode$. In the new tree, intersections between consecutive nodes are larger.

CHAPTER 4

IMPLEMENTATION AND RESULTS

We illustrate the flow of the RG-Trees algorithm and its feasibility on computer simulations. We investigate the effects of node expansion, tree optimization and gateway usage by repeating the simulations with different parameter combinations. For each combination, we make N=1000 Monte-Carlo simulations and give their histogram plots. We also test the effects of velocity and acceleration limits, discussed as $RG2$ in Chapter 3. Finally, we test multiple-query performance of the algorithm. In this test, after the robot reaches goal location, we relocate it to another point on the map and measure the replanning time. We made Monte-Carlo simulations.

This chapter gives implementation details and simulation results.

4.1 Robot Motion Model

In the simulations, a fully-actuated, double-integrator, planar, point robot model is used. State vector of the robot is $q = [x\ y\ \dot{x}\ \dot{y}]^T$, and the input vector is $u = [\ddot{x}\ \ddot{y}]$. Full state of the robot is measured. The system is discretized using forward Euler method. Discrete time state space model of the open-loop robot is then,

$$A = \begin{bmatrix} 1 & 0 & T_s & 0 \\ 0 & 1 & 0 & T_s \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},\ B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ T_s & 0 \\ 0 & T_s \end{bmatrix},$$

$$C = I_4,\ D = 0, \tag{4.1}$$

The reference governor works on the closed-loop system. In order to emphasize the effectiveness of the algorithm, as the inner loop controller, an underdamped PD controller is chosen:

$$u = \begin{bmatrix} K_p(r_x - x) - K_d\dot{x} \\ K_p(r_y - y) - K_d\dot{y} \end{bmatrix} = \begin{bmatrix} -K_p & 0 & -K_d & 0 \\ 0 & -K_p & 0 & -K_d \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix} + \begin{bmatrix} K_p & 0 \\ 0 & K_p \end{bmatrix} \cdot \begin{bmatrix} r_x \\ r_y \end{bmatrix} \tag{4.2}$$

The controller's parameters are $T_s = 0.05s$, $K_p = 4.1$ and $K_d = 2.2$. If these parameters and (4.2) are substituted in (4.1) the overall discrete time state space model of the closed loop system is obtained as follows:

$$A = \begin{bmatrix} 1 & 0 & 0.05 & 0 \\ 0 & 1 & 0 & 0.05 \\ -0.205 & 0 & 0.89 & 0 \\ 0 & -0.205 & 0 & 0.89 \end{bmatrix},\ B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0.205 & 0 \\ 0 & 0.205 \end{bmatrix} \tag{4.3}$$

4.2 Effects of the Reference Governor

In RG-Trees algorithm, the square regions, which are defined by the nodes, are placed totally in the obstacle-free space. Therefore, collision avoidance is guaranteed only if the robot stays inside such square regions, and it is RG's task to keep the robot inside the region. In this section, we demonstrate the necessity for the RG on an example scenario.

In Fig. 4.1, the robot starts from the point $(0.1, 0.1)$ at a 1m/s velocity in the x direction. The goal location is $(0.95, 0.95)$, and the admissible region is $0 \le x, y \le 1$, which is shown in grey. Without RG (left), the robot crosses the node boundary, which might or might not result in a collision. When RG is used, the robot stays within the node all the time, therefore collision avoidance is ensured.

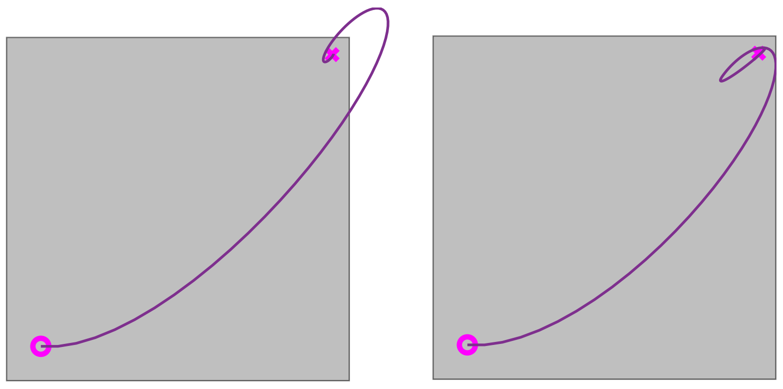


Figure 4.1: Effect of the RG. The robot starts from ‘o’ at 1m/s initial velocity in the horizontal direction. The goal location is ‘x’. *Left.* In the unconstrained simulation, the robot crosses the node boundary. *Right.* The RG ensures that the robot stays inside the node.

4.3 Performance Metrics

In this thesis we introduced the fundamental RG algorithm and also proposed some potential improvements. In order to have a comparative analysis between different methods, we utilize some performance metrics as described below

Success Rate: The number of successful missions out of N=1000 simulations

CPU Time: The time it takes for the algorithm to find a path

Number of Nodes: The total number of nodes on the tree

Path Depth: The number of nodes on the path

Average Speed: The average speed of the robot during its motion

Arrival Time: The time it takes for the robot to reach goal location

Path Length: The total length of the robot’s trajectory

4.4 Results on Sample Map 1

Firstly, we tested RG-Trees algorithm on the sample map given in Fig. 4.2. The boundary of the arena is square shaped, with edge length 12m. There exist six different polygonal obstacles in the arena.

Fig. 4.3 illustrates three different alternative solutions with the same initial and goal configuration. Note that there exist different homotopy classes of solutions; therefore, the probability distribution of performance metrics are expected to be multi-modal. Respective subsections provide the histogram results of the performance metrics.

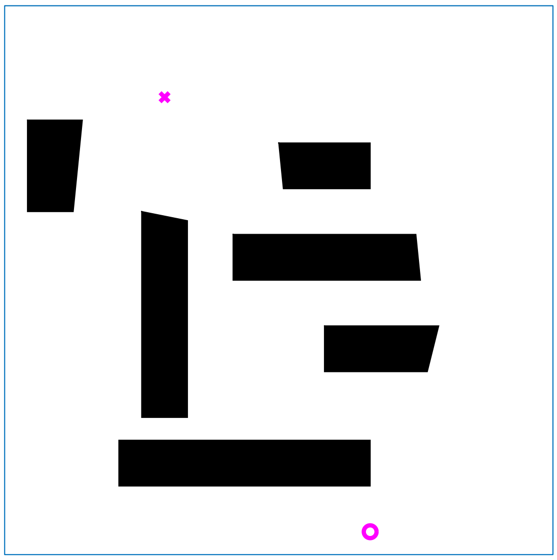


Figure 4.2: Sample Map 1. 'o' is the start location and 'x' is the goal location.

4.4.1 Case I: Basic RG-Trees Algorithm

In this case, node expansion, tree optimization, and gateways are all disabled. In 997/1000 simulations, the robot reached the goal location within the 60s time limit. We observe that in the failure cases, intersection between some nodes become too small which slows down the robot motion accordingly due to the enforced constraints.

On average, RG-Trees algorithm found a path in 0.14s, with 106 nodes, of which

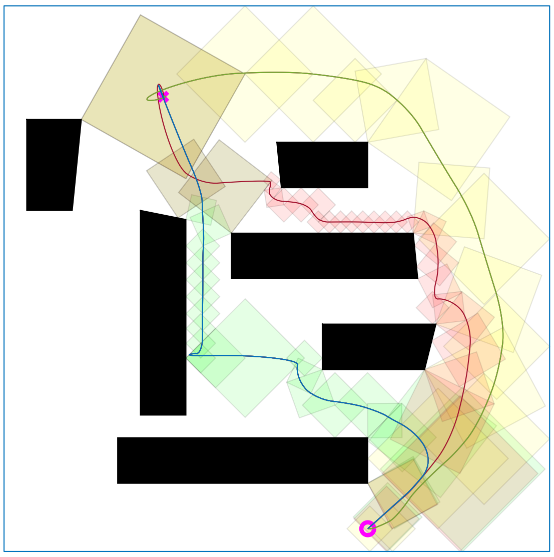


Figure 4.3: Three different solution paths for Map 1, belonging to three different homotopy classes.

21 are on the path between the start and goal configurations. Again on average, the robot reaches the goal location in 17.48s, with an average speed of 0.99m/s. The resulting average trajectory length is 16.87m. Histogram plots of CPU time, path length, arrival time and average speed are given in Fig. 4.4. A sample tree and path generated with this parameter configuration are given in Fig. 4.5. As expected, some histogram plots are multi-modal.

4.4.2 Case II: RG-Trees with Node Expansion

In this case, node expansion is enabled, but tree optimization and gateways are disabled. In these MC simulations, the robot reached in all 1000 simulations within 60s time limit. On average, RG-Trees algorithm found a path in 0.20s, with 70 nodes, of which 18 are on the path. The robot reached the goal location in 14.80s, at an average speed of 1.09m/s. The resulting average trajectory length is 15.81m long.

The node expansion process also increases the area of the intersections between different nodes, thus we have never observed a failure case that is reported in Case I. In addition to this improvement, thanks to larger nodes, the robot moves

10% faster on average, and since the robot is allowed to move farther away from the obstacle boundaries, average path length is 6% shorter. As a result, the robot reaches the goal location in 15% less time. The additional CPU time is only 0.06s.

Histogram plots of CPU time, path length, arrival time and average speed are given in Fig. 4.6. A sample tree and path generated with this parameter configuration are given in Fig. 4.7. These results show that node expansion adds an almost negligible amount of computational complexity (on this map), yet provides substantial performance improvements in other aspects.

4.4.3 Case III: RG-Trees with Tree Optimization

In this case, tree optimization is enabled, but node expansion and gateways are disabled. We observed that in 989 of 1000 simulations, the robot reached the goal location within 60s time limit. It can be seen that the number of failed cases is even higher than the Case I. We suspect that the reason for this is the occasional problematically small intersection regions between consecutive nodes on the new optimized path. Such examples can be observed in Fig. 4.9.

On the other hand, in this case on average the algorithm found a path in 0.34s, with 107 nodes, of which 16 are on the path. The robot reached the goal location in 18.03s, at an average speed of 0.93m/s. The resulting trajectory is 16.22m long. If the intersection is too small the robot unavoidably halts, otherwise it slows down, which is the reason of the decrease in average speed. The resulting path is slightly shorter.

Histogram plots of CPU time, path length, arrival time and average speed are given in Fig. 4.8. A sample tree and path generated with this parameter configuration are given in Fig. 4.9. These results suggest that performing optimization with other improvements disabled is not a preferable option.

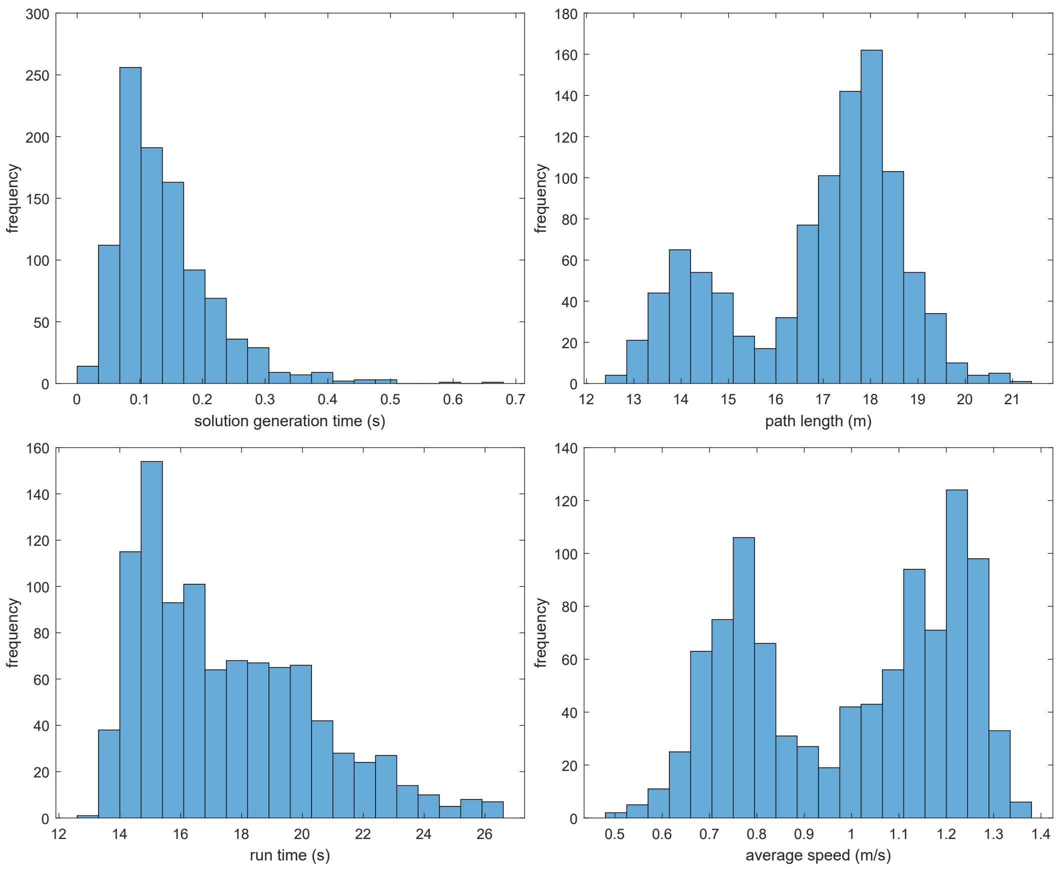


Figure 4.4: Histogram plots for Case I, where node expansion and tree optimization are disabled.

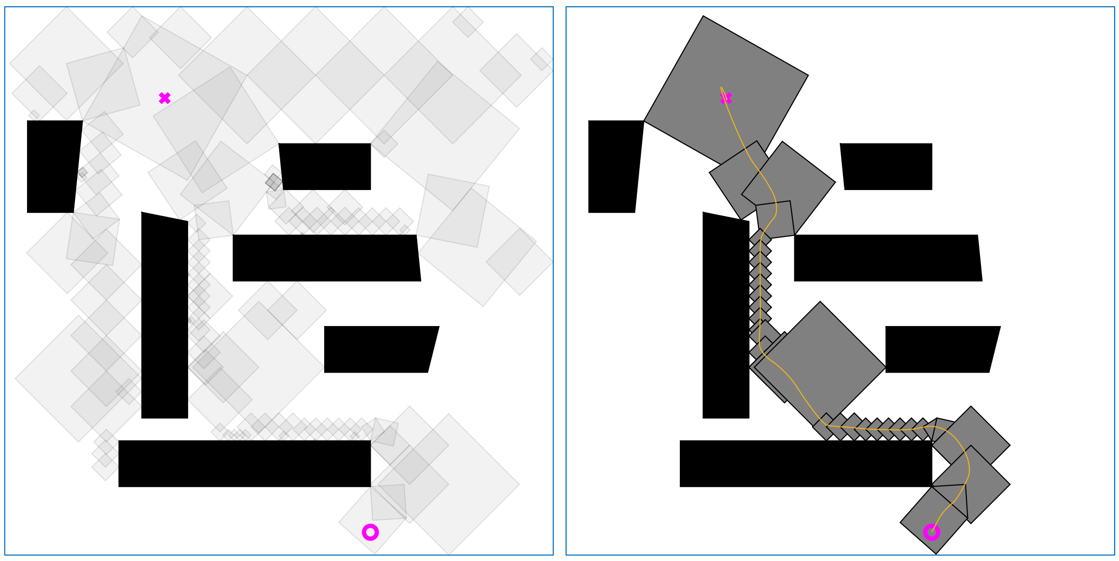


Figure 4.5: An example tree and solution path for Case I.

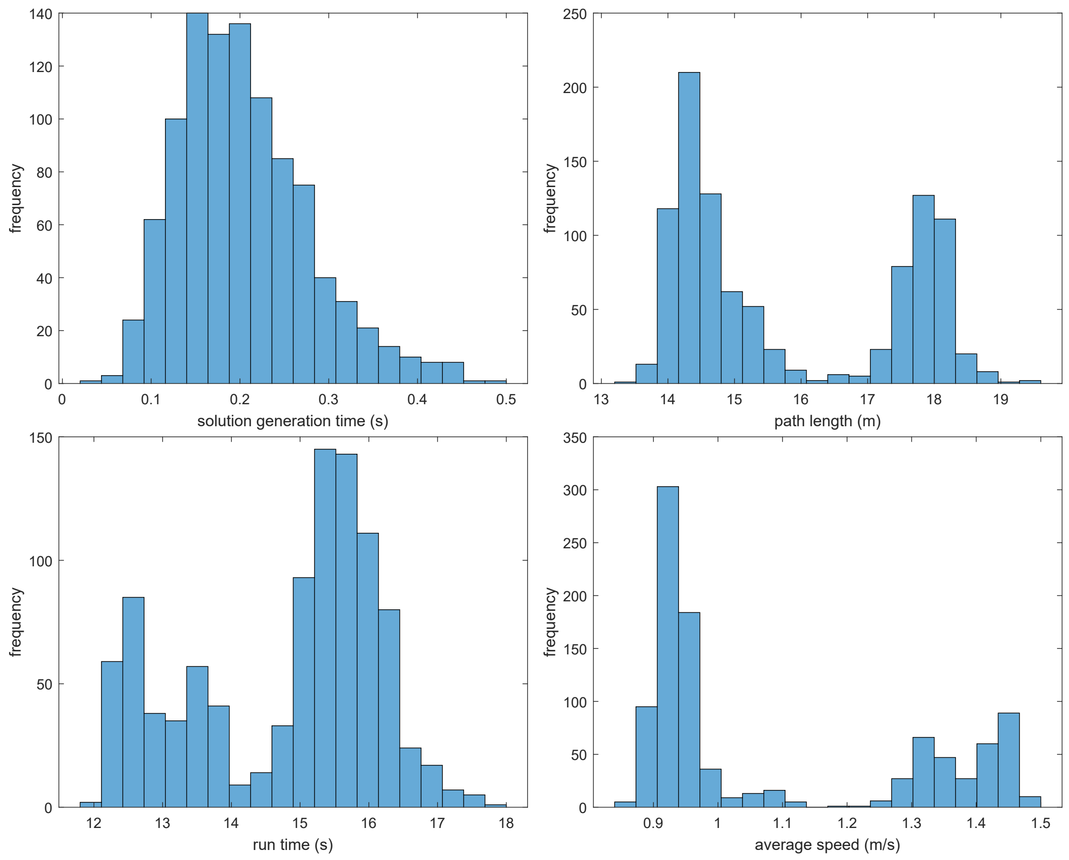


Figure 4.6: Histogram plots for Case II, where node expansion is enabled and tree optimization is disabled.

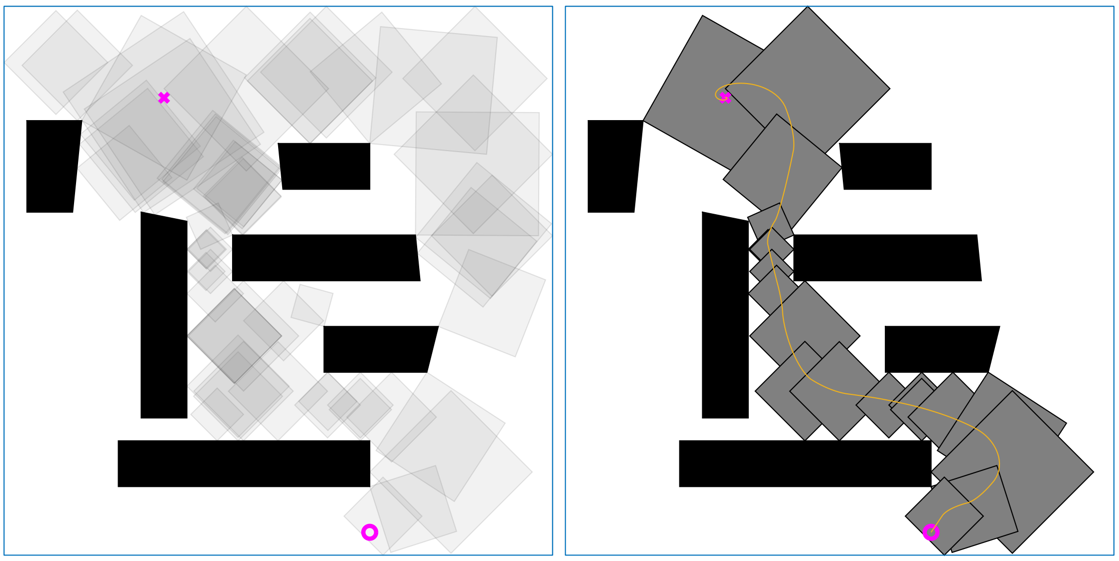


Figure 4.7: An example tree and solution path for Case II.

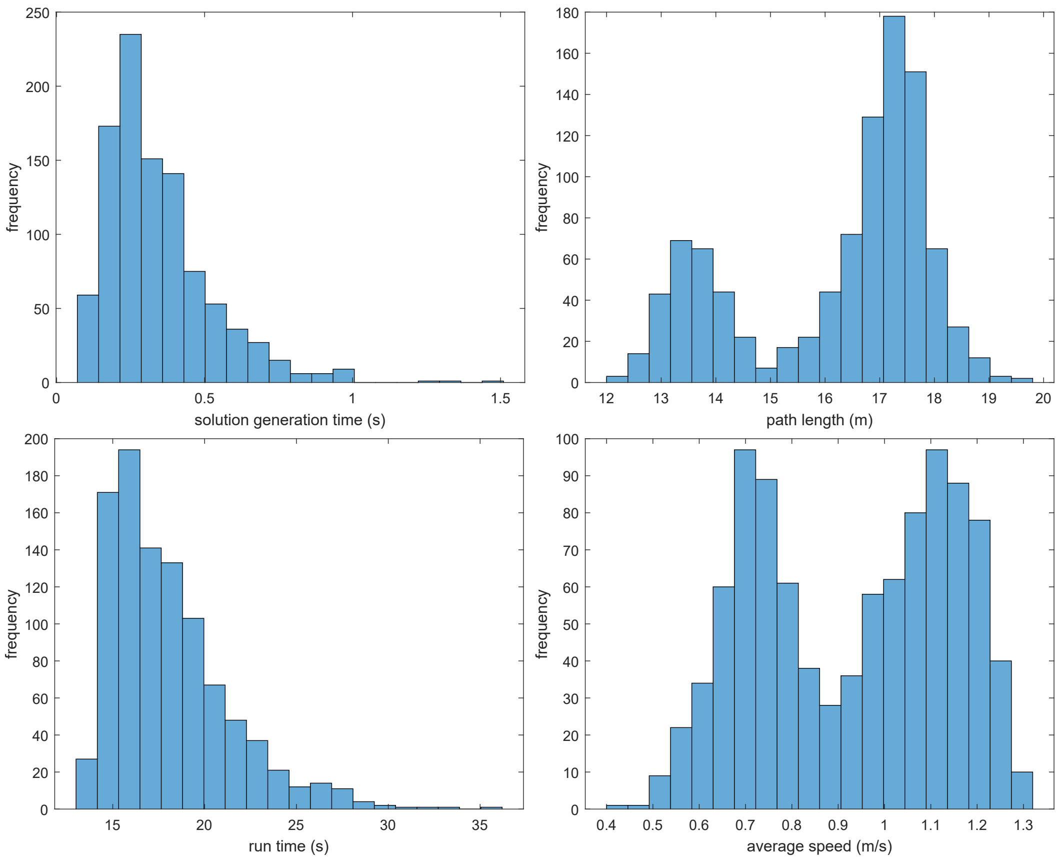


Figure 4.8: Histogram plots for Case III, where node expansion is disabled and tree optimization is enabled.

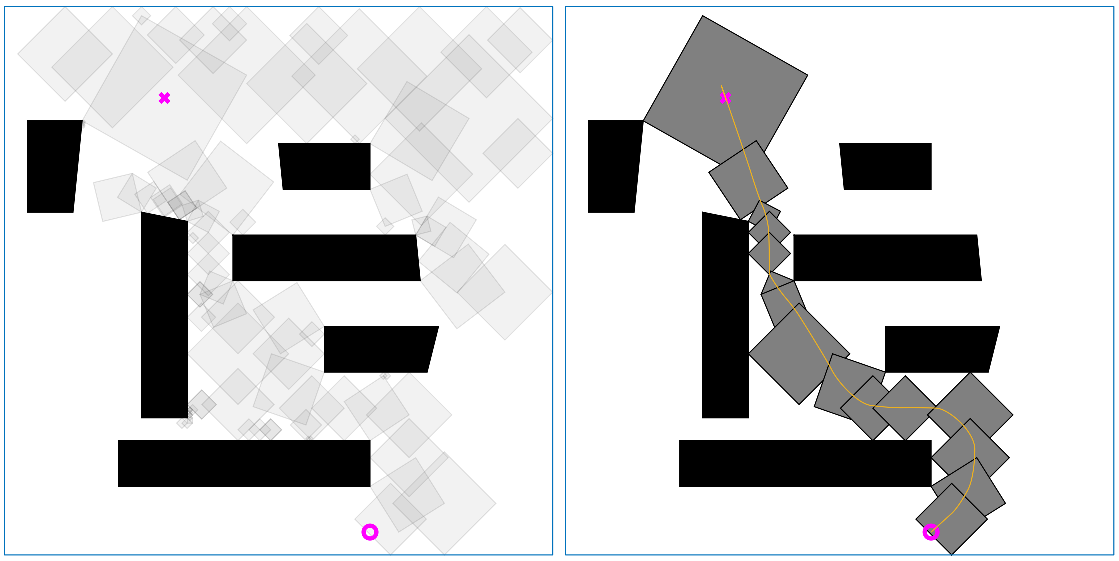


Figure 4.9: An example tree and solution path for Case III.

4.4.4 Case IV: RG-Trees with Node Expansion and Tree Optimization

In this case, node expansion and tree optimization are both enabled, but gateways are disabled. In this case, in 985/1000 simulations, the robot reached the goal location within 60s time limit. On average, RG-Trees algorithm found a path in 0.33s, with 70 nodes, of which 12 are on the path. The robot reached the goal location in 15.46s, at an average speed of 0.98m/s. The resulting path is 14.88m long.

The optimization is made with respect to path length, therefore the path is slightly shorter than Case II (expansion on, optimization). However, due to small intersection cases average speed of the robot is lower, thus it takes longer for the robot to reach the goal location. In addition to these, there is a significant number of fail cases compared to Case II.

Histogram plots of CPU time, path length, arrival time and average speed are given in Fig. 4.10. A sample tree and path generated with this parameter configuration are given in Fig. 4.11.

4.4.5 Comparison of Parameter Combinations

Boxplots for the four parameter combinations are given in Fig. 4.12. It can be seen that when the node expansion is enabled; the resulting path is shorter, tree is more sparse, average speed of the robot is higher and the arrival time is smaller, as compared to when node expansion is disabled (Fig. 4.12, (b), (d) vs (a), (c)). Solution generation time is slightly larger when tree optimization is disabled, however with optimization enabled, it is smaller thanks to sparsity of the tree. It can be concluded that node expansion is advantageous in almost every respect.

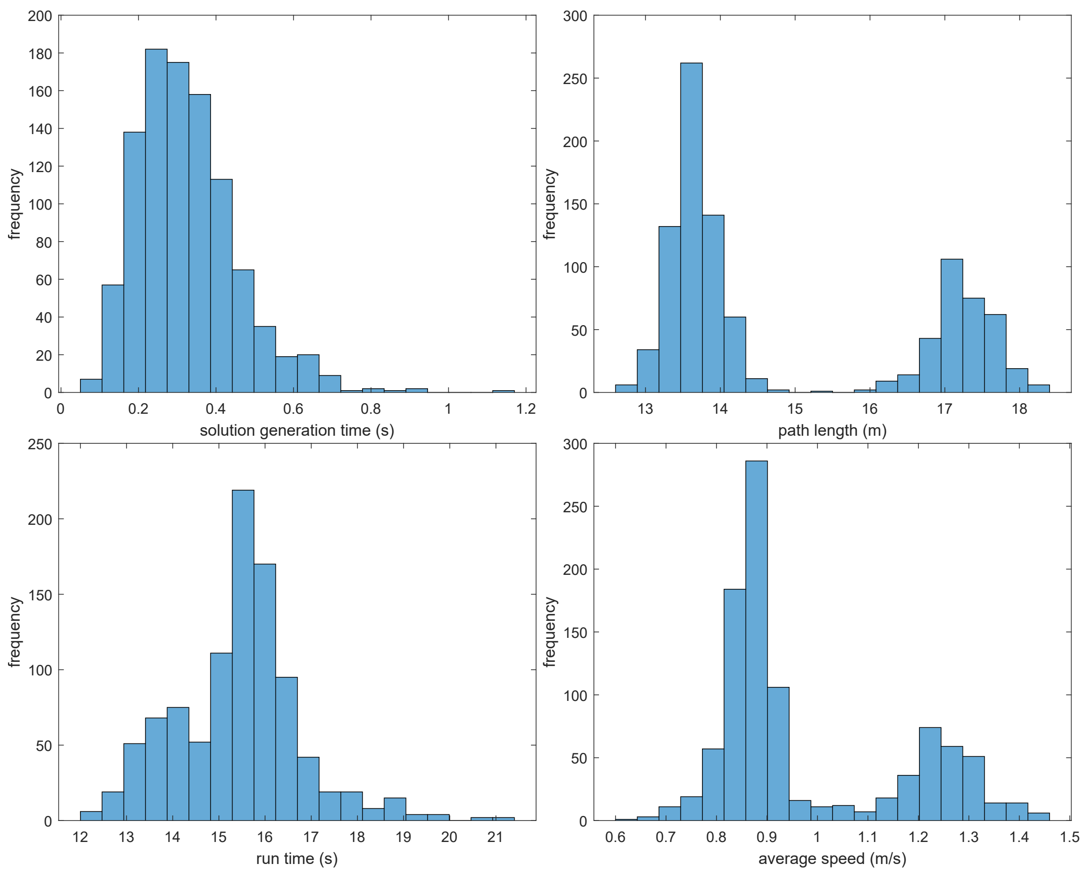


Figure 4.10: Histogram plots for Case IV, where node expansion and tree optimization are enabled.

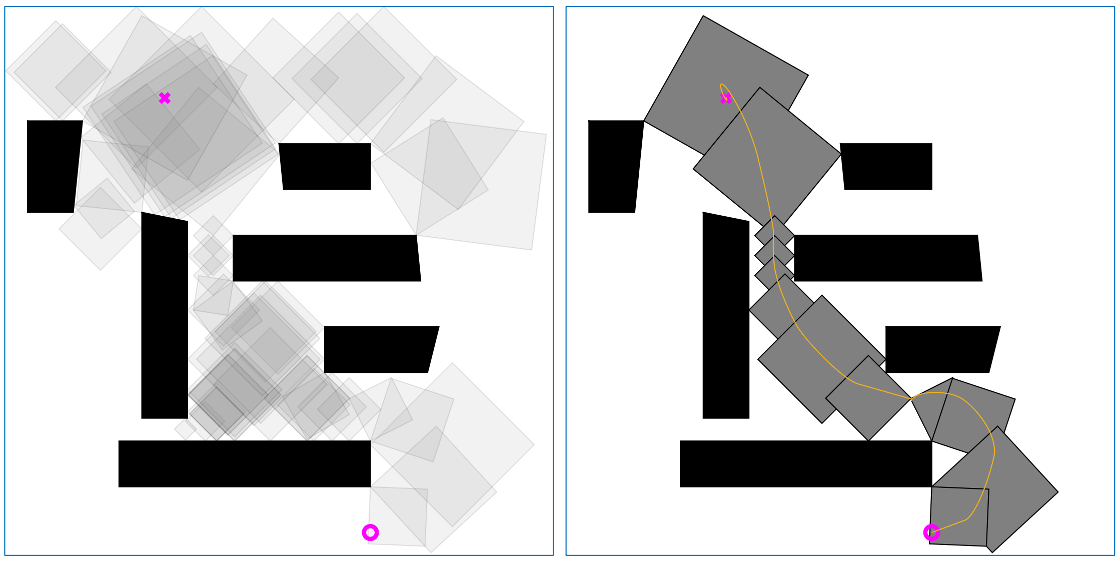


Figure 4.11: An example tree and solution path for Case IV.

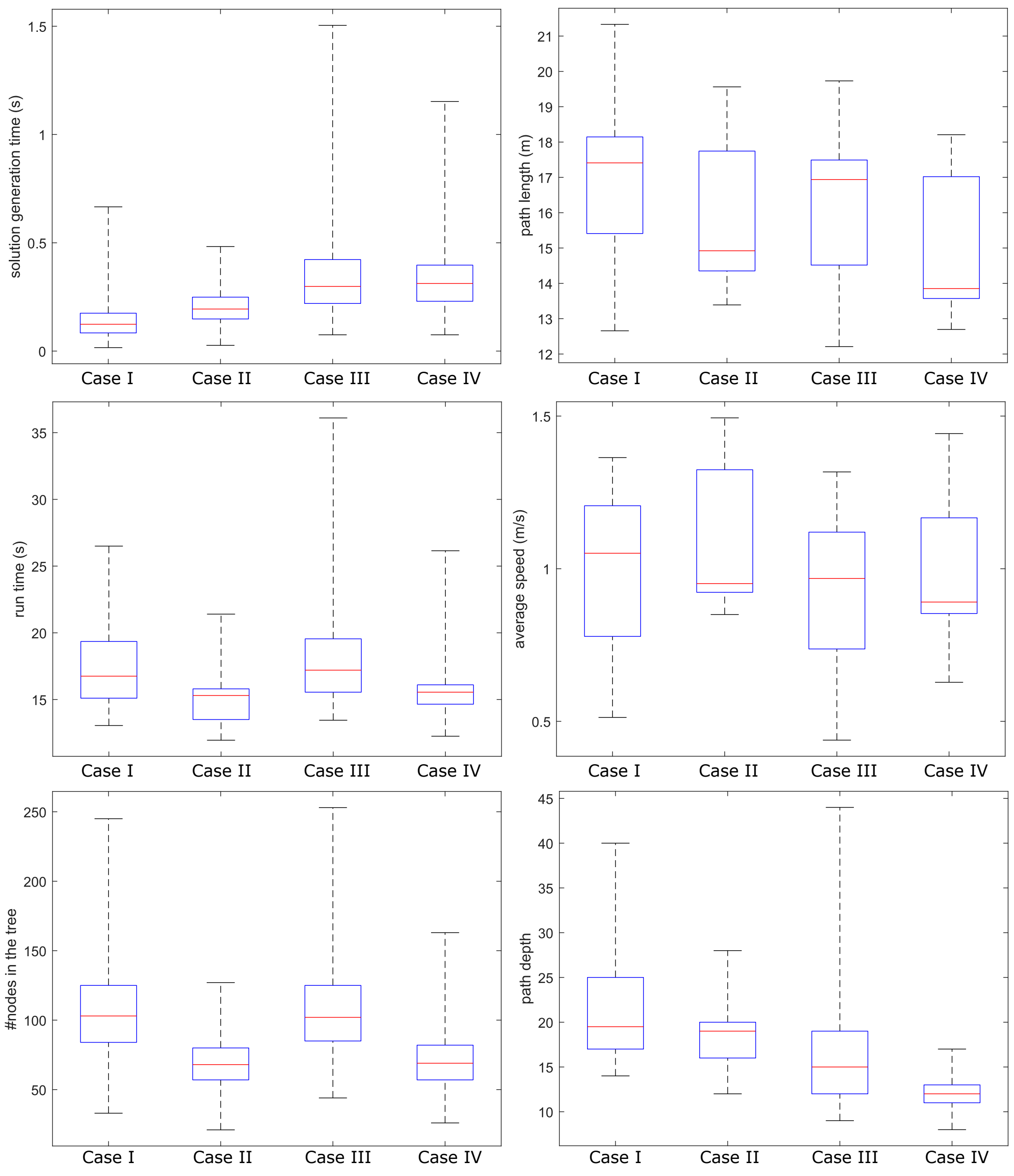


Figure 4.12: Box plot of parameter combinations. Red mark indicates the median. Bottom and top of the box are 25% and 75% points, respectively. Case I: node expansion and tree optimization disabled, Case II: node expansion enabled, tree optimization disabled, Case III: node expansion disabled, tree optimization enabled, Case IV: node expansion and tree optimization enabled

On the other hand, the benefits of tree optimization are disputable for these results. Resulting path length is shorter and solution path depth is smaller, since

Dijkstra's algorithm is used. However, due to small intersection cases, average speed is lower, and in some simulations the robot halts before arriving at the goal location. Since the optimization is made after the tree is fully constructed, average number of nodes in the tree is not affected. For the same reason, solution generation time is larger.

4.4.6 Case V: Effects of Gateway Nodes

In this case, both node expansion, tree optimization, and gateways are enabled. In the MC runs of this case, the robot reached the goal location within 60s time limit in all of the 1000 MC simulations . On average, RG-Trees algorithm found a path in 0.32s, with 78 nodes, of which 20 are on the path. The robot reached the goal location in 14.38s, at an average speed of 1.06m/s. The resulting path is 15.11m long.

Histogram plots of CPU time, path length, arrival time and average speed are given in Fig. 4.13. A sample tree and path generated with this parameter configuration are given in Fig. 4.14. Boxplots comparing this configuration with Case II and Case IV are given in Fig. 4.15.

It can be seen from these results that gateway nodes resolve the problems induced by tree optimization, while still enjoying the benefits the tree optimization process. Similar to Case II, every simulation succeeded with this configuration. Moreover, the average speed is as high and convergence time is as low as Case II (i.e. no optimization), while path length is obviously reduced.

Overall, for this map we can conclude that if additional computational cost is not significant Case V provides best overal result, however if computational cost is critical Case II seems to be the best configuration.

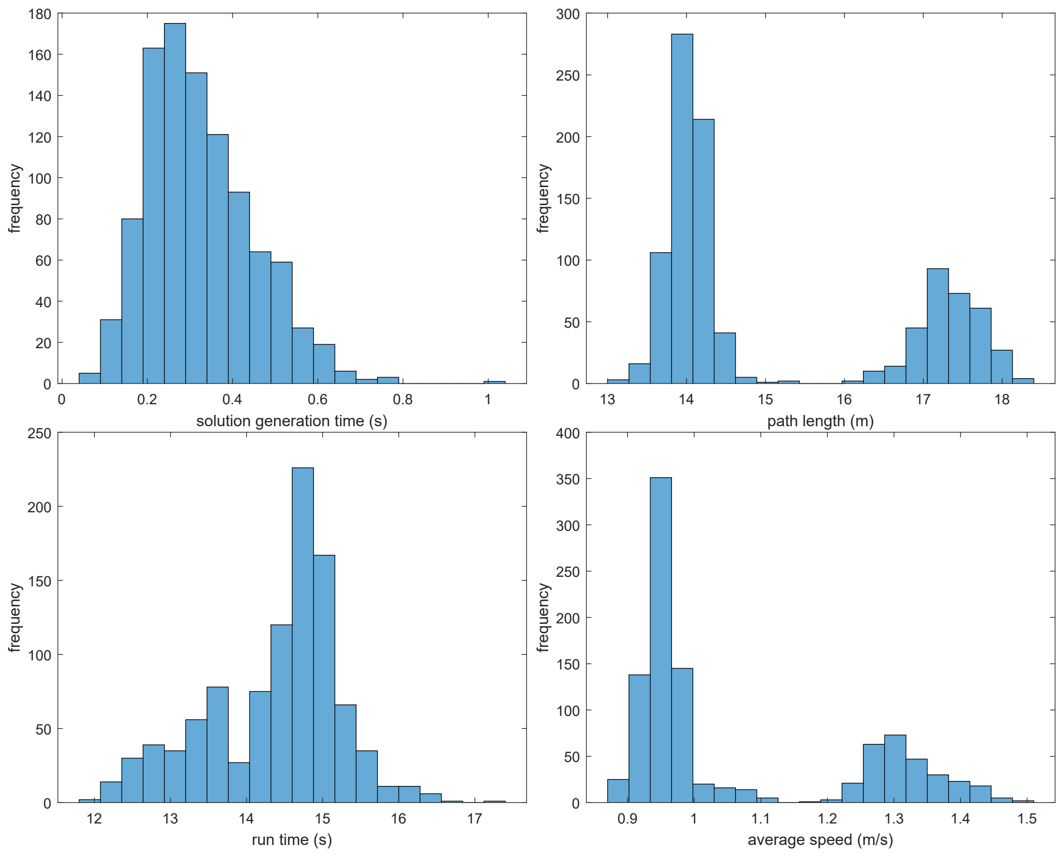


Figure 4.13: Histogram plots for Case V, where node expansion and tree optimization are enabled and gateways are used

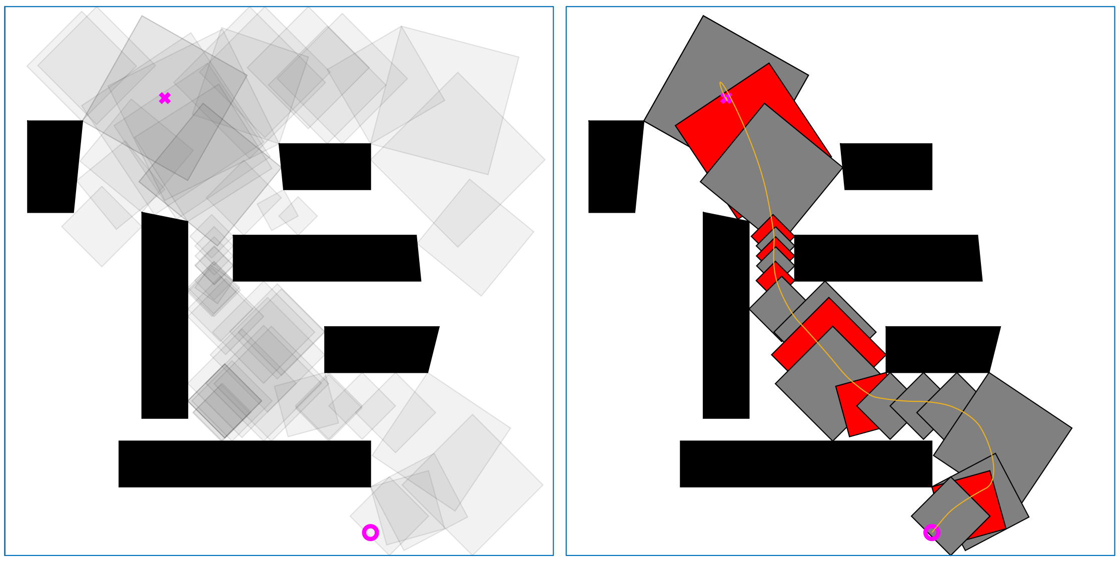


Figure 4.14: An example tree and solution path for Case V. Gateway nodes are shown in red.

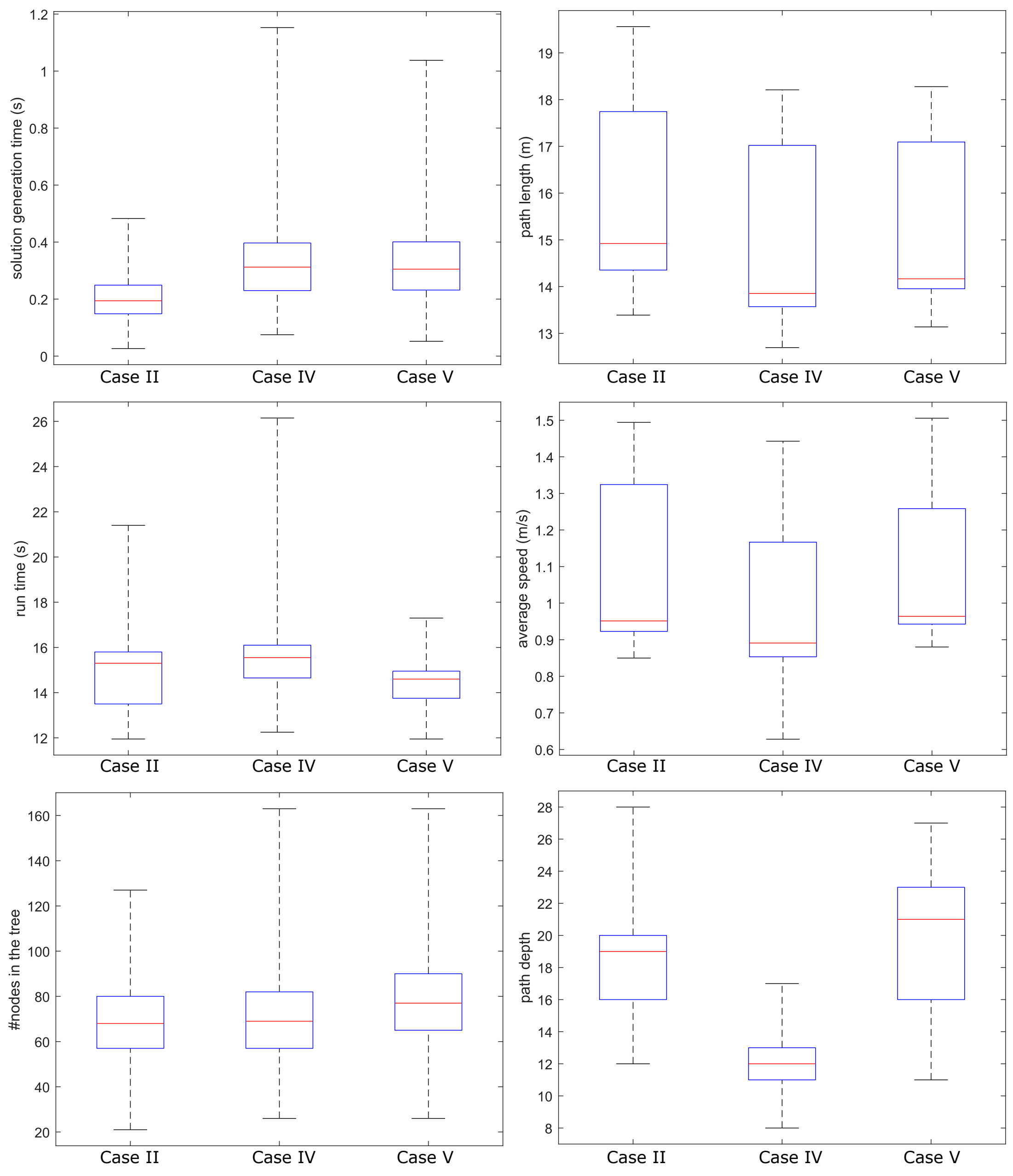


Figure 4.15: Box plot of parameter combinations. Red mark indicates the median. Bottom and top of the box are 25% and 75% points, respectively. Case II: node expansion enabled, tree optimization disabled, Case IV: node expansion and tree optimization enabled, Case V: node expansion and tree optimization enabled and gateways used

4.4.7 Effects of Speed and Acceleration Limits

Traditionally, the RG is used for enforcing state and input constraints, such as velocity and acceleration. RG-Trees algorithm allows using a second RG for this purpose, as explained in Subsection 3.2.1.

When the speed limit is set to $v_{max} = 0.8 m/s$, the robot reaches this maximum speed as quickly as possible, and moves at that speed for the rest of its motion. This also provides a practical benefit, since in general it is not very desirable if autonomous vehicle changes its speed frequently. In this RG indirectly solves this problem in a formal way. A sample simulation path and the robot's speed and acceleration graphs are given in Fig. 4.16. As a result of the 32-sided polygon approximation, the speed ripples around its maximum value.

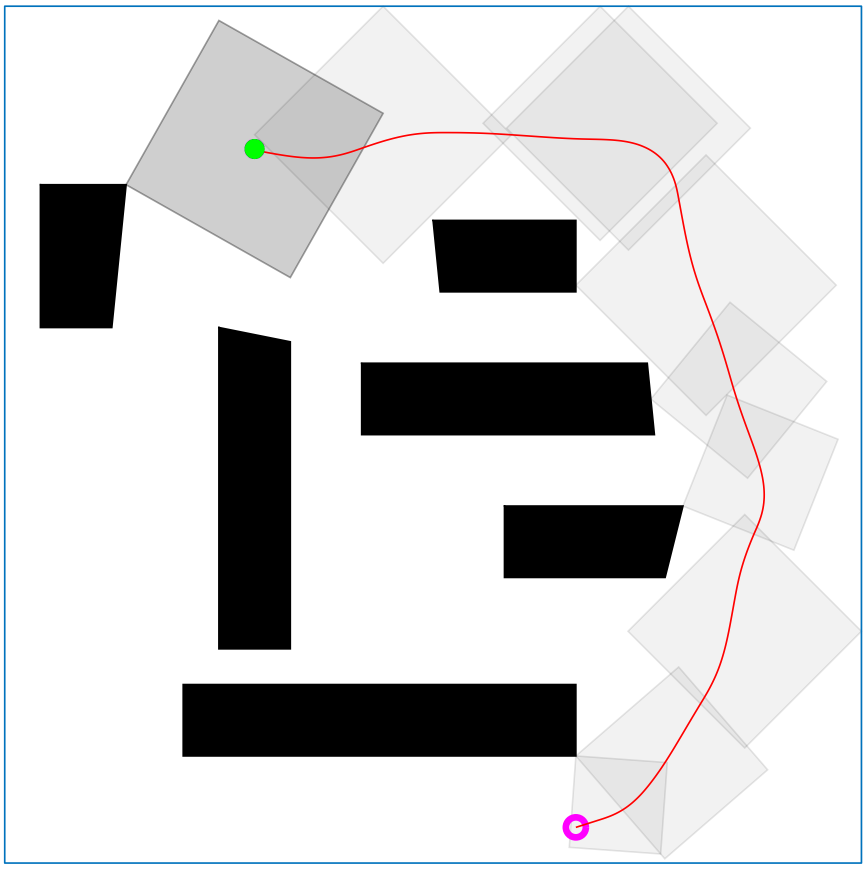
When the speed limit is increased to a value between the robot's minimum and maximum speeds, the robot can no longer move at this speed for all time. However, speed limitation is respected, the robot moves at most at the maximum speed. A sample simulation result for $v_{max} = 1.2 m/s$ is given in Fig. 4.17. Similarly, when the robot's acceleration is limited to $a_{max} = 0.8 m/s^2$, the robot respects this limitation, see Fig. 4.18.

4.4.8 Multiple-Query Tests

RG-Trees constructs a tree structure of square nodes, the root of which contains the goal location. Thus, it plans the robot's motion not only from start to the goal, but from every point which is covered by a node to the goal. Moreover, tree generation is incremental: even when the start location is switched to a point which is not covered by any node, instead of constructing the tree from scratch, the algorithm builds it on top of the previously constructed tree, which generally takes a shorter time.

In this subsection, we provide Monte-Carlo simulation results of the following multiple-query experiment. In this scenario, the robot starts from the point $(8, 0.5)$ as in the previous subsections and Case V configuration is adopted in these MC runs. After the robot reaches the goal, it is relocated/teleported to the point $(1, 1)$, which might or might not be already covered by a node, and the simulation with the same goal location is repeated. Finally, the robot is teleported to the point $(5, 4)$ and the same procedure is repeated. Sample trajectory of and MC run the robot is given in Fig. 4.19.

Tree generation and recalculation times are stored and their histogram plots are given in Fig. 4.20. If one observes the histogram plots of recalculation times for second and third solutions, he/she can see that there are peaks around 0, which corresponds to cases where the robot is already inside a covered area. If we isolate this part, average (re)calculation time is $0.35s$, $0.26s$ and $0.03s$ for the first, second and third calculations.



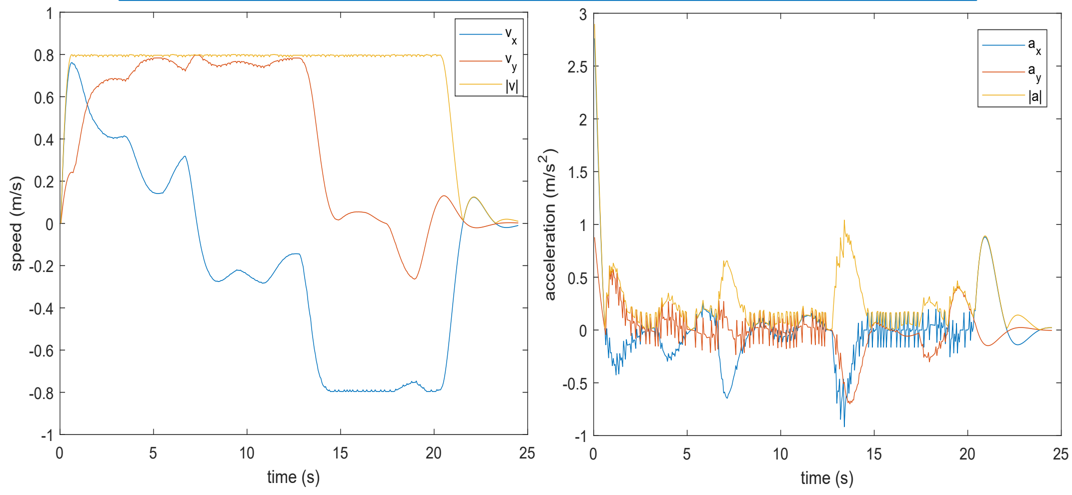


Figure 4.16: An example solution path and its v-t and a-t graphs when speed limit is set to 0.8 m/s.

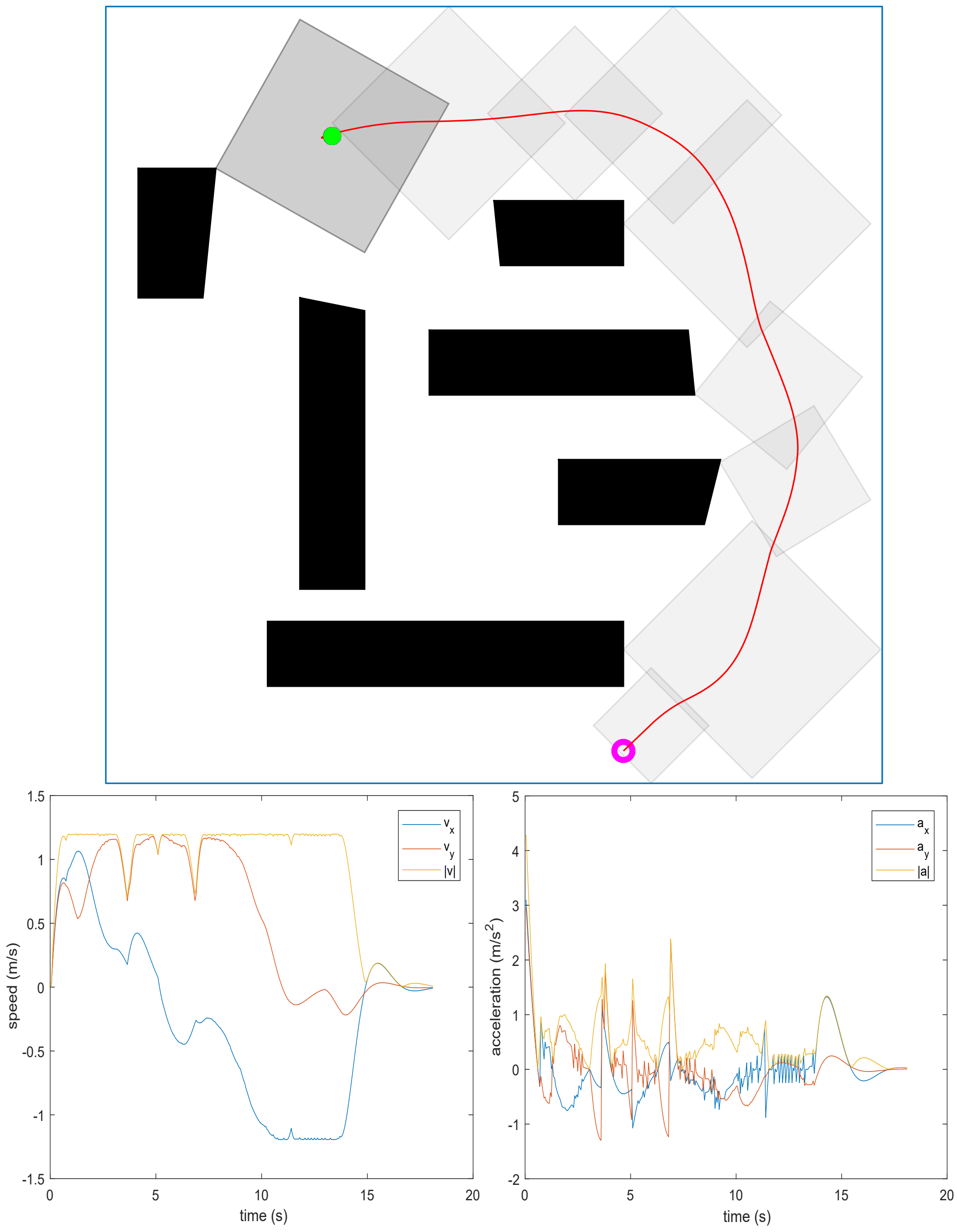


Figure 4.17: An example solution path and its v-t and a-t graphs when speed limit is set to 1.2 m/s.

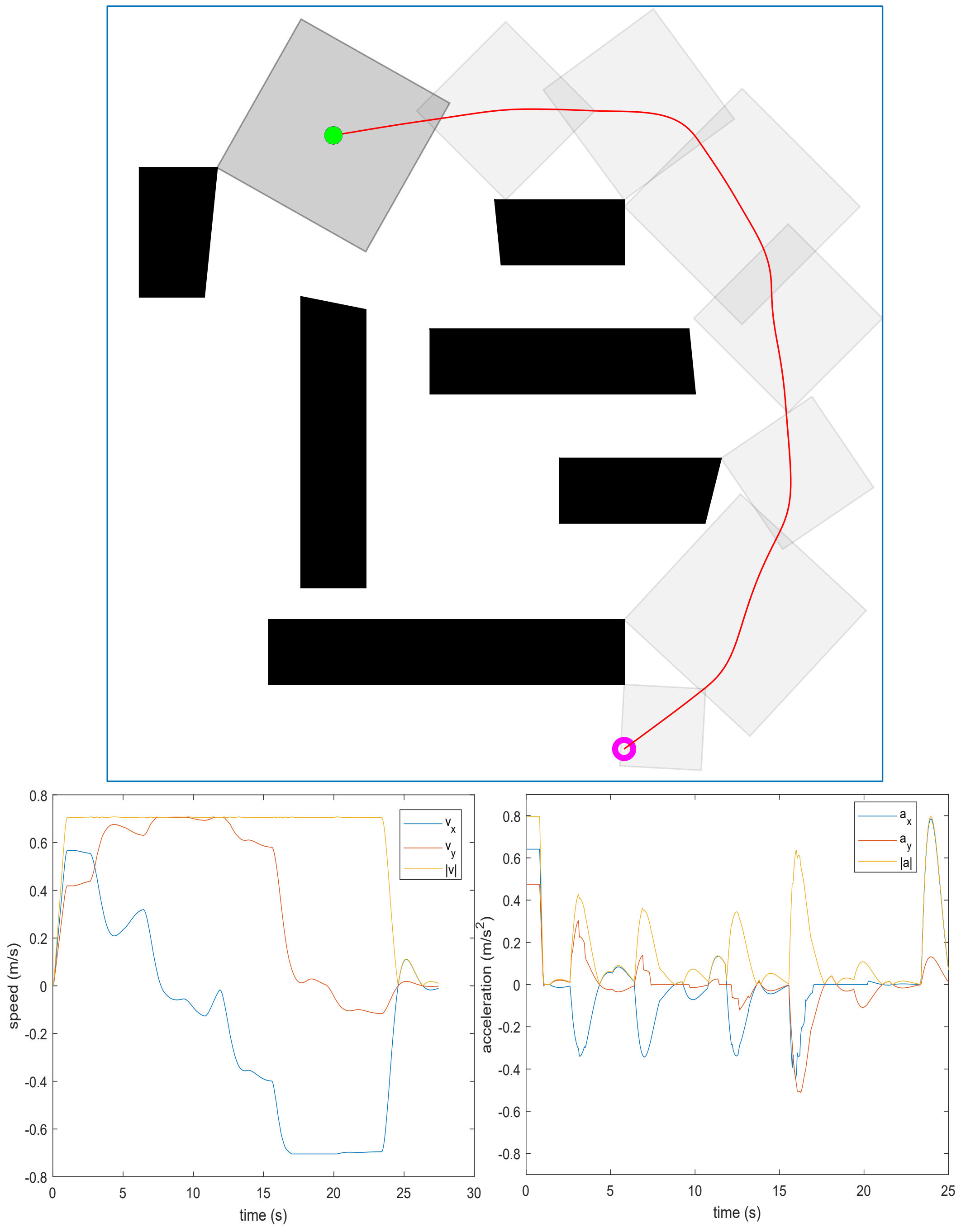


Figure 4.18: An example solution path and its v-t and a-t graphs when acceleration limit is set to 0.8 m/s^2.

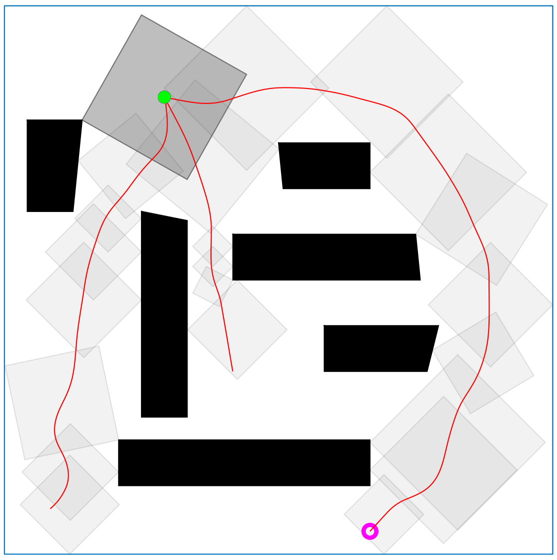


Figure 4.19: Multiple query example. The robot starts from (8, 0.5) initially. After it reaches the goal, it is relocated to (1, 1) and then to (5, 4), both of which might or might not be covered already. The robot reuses the previously generated tree.

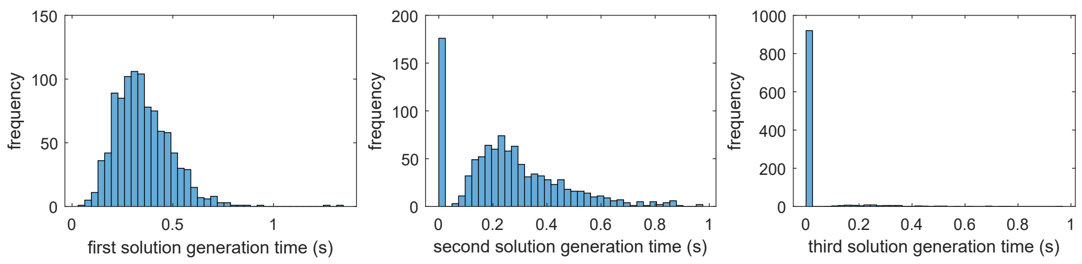


Figure 4.20: Histogram plots for tree generation and recalculation. Mean CPU time is 0.35s, 0.26s and 0.03s for the first, second and third calculations respectively. For the second calculation, in 176 simulations had already been covered. This number is 920 for the third calculation.

4.5 Results on Sample Map 2

The second map that we tried RG-Trees algorithm is given in Fig. 4.21. This map features local minima, from which potential field based planners suffer [10,31,32]. RG-Trees relies on random sampling, and the probability that it finds a solution approaches 1 as the number of nodes tends to infinity, provided that there exists a solution. That is, the algorithm is "probabilistically complete". We verified this claim in our experiments. If we choose the node count limit (K in Algorithm 2) small, the algorithm returns no solution. However, when we allow larger number of nodes, the algorithm finds a solution in all simulations.

The second arena is rectangular, with dimensions $8m$ x $12m$, and there exist 4 obstacles in the map. The arena is almost symmetric, therefore the histogram plots are unimodal.

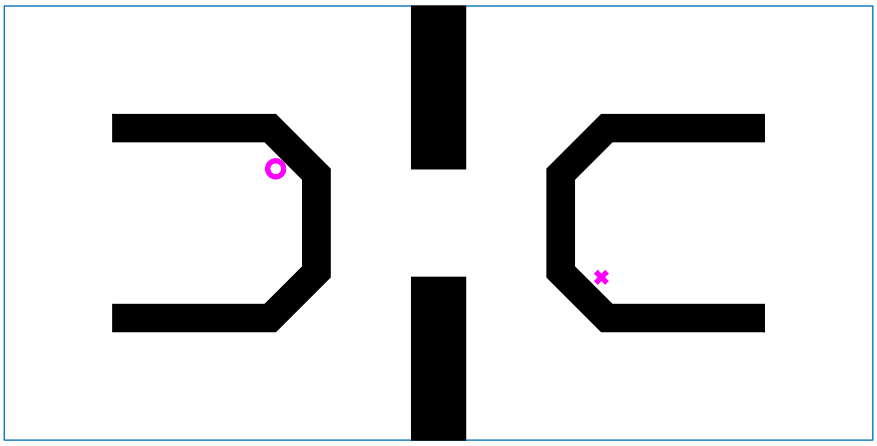


Figure 4.21: Sample Map 2, which features local minima. 'o' is the start location and 'x' is the goal location.

4.5.1 Case I: Basic RG-Trees Algorithm

In this case, node expansion, tree optimization and gateways are disabled.In 983/1000 simulations, the robot reached the goal location within 60s time limit.

On average, RG-Trees algorithm found a path in 0.22s, with 242 nodes, of which 52 are on the path. The robot reached the goal location in 33.59s on average, with an average speed of 0.92m/s. The resulting average trajectory length is 30.79m. Histogram plots of CPU time, path length, arrival time and average speed are given in Fig. 4.22. A sample tree and path generated with this parameter configuration are given in Fig. 4.23.

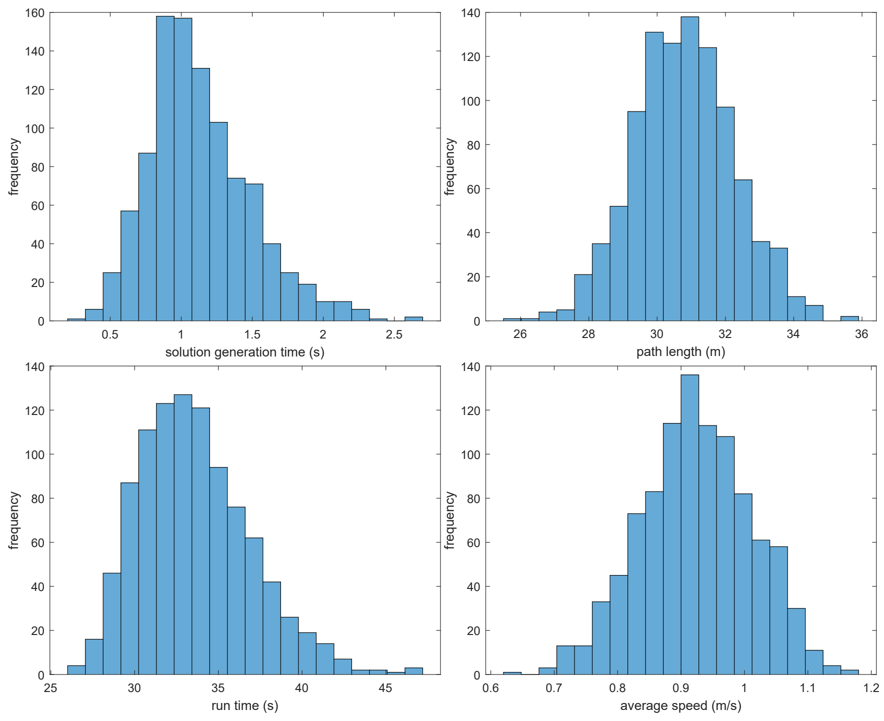


Figure 4.22: Histogram plots for Case I, where node expansion and tree optimization are disabled.

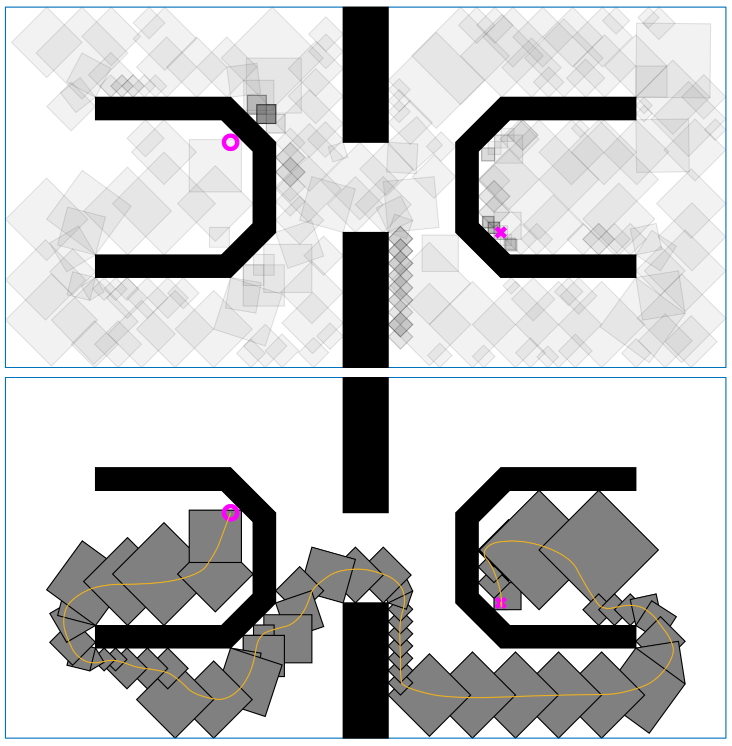


Figure 4.23: An example tree and solution path for Case I.

4.5.2 Case II: RG-Trees with Node Expansion

In this case, node expansion is enabled but tree optimization and gateways are disabled. In all 1000 simulations, the robot reached the goal location within 60s time limit. On average, RG-Trees algorithm found a path in 0.35s, with 122 nodes, of which 35 are on the path. The robot reached the goal location in 23.72s, at an average speed of 1.29m/s. The resulting average trajectory length is 30.67m.

Similar to the result in Map1, node expansion widens the nodes and their intersections; thus, the robot runs faster and all simulations succeed.

Histogram plots of CPU time, path length, arrival time and average speed are given in Fig. 4.24. A sample tree and path generated with this parameter configuration are given in Fig. 4.25.

4.5.3 Case III: RG-Trees with Tree Optimization

In this case, tree optimization is enabled but node expansion and gateways are disabled. In 963 of 1000 simulations, the robot reached the goal location within 60s time limit. On average, RG-Trees algorithm found a path in 0.75s, with 243 nodes, of which 36 are on the path. The robot reached the goal location in 33.71s, at an average speed of 0.84m/s. The resulting trajectory length is 28.12m.

Solution path depth and length are better than Case II. However, due to small intersection problem, more simulations get stuck, and in those who succeed, average speed is smaller than Case II.

Histogram plots of CPU time, path length, arrival time and average speed are given in Fig. 4.26. A sample tree and path generated with this parameter configuration are given in Fig. 4.27.

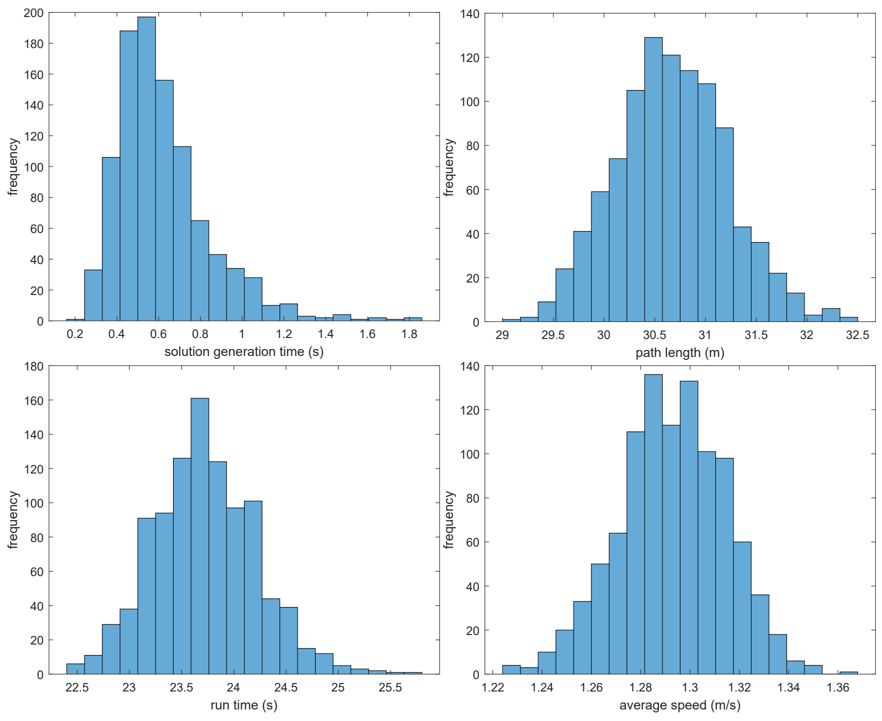


Figure 4.24: Histogram plots for Case II, where node expansion is enabled and tree optimization is disabled.

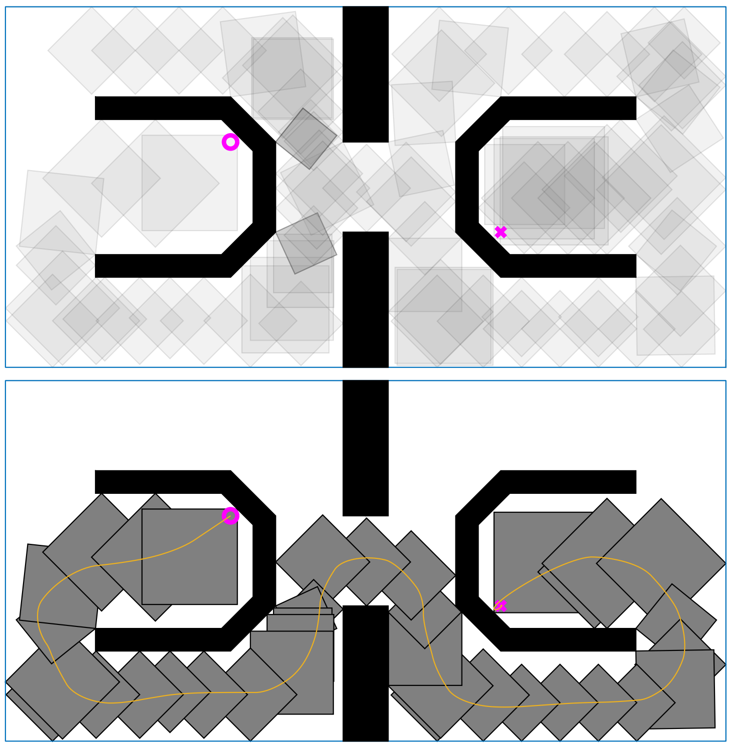


Figure 4.25: An example tree and solution path for Case II.

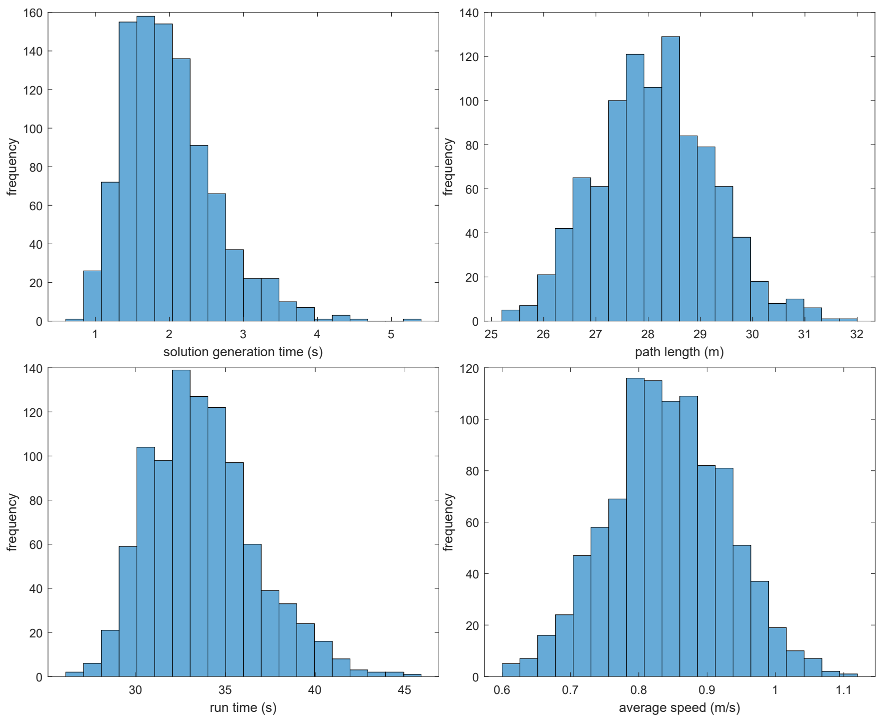


Figure 4.26: Histogram plots for Case III, where node expansion is disabled and tree optimization is enabled.

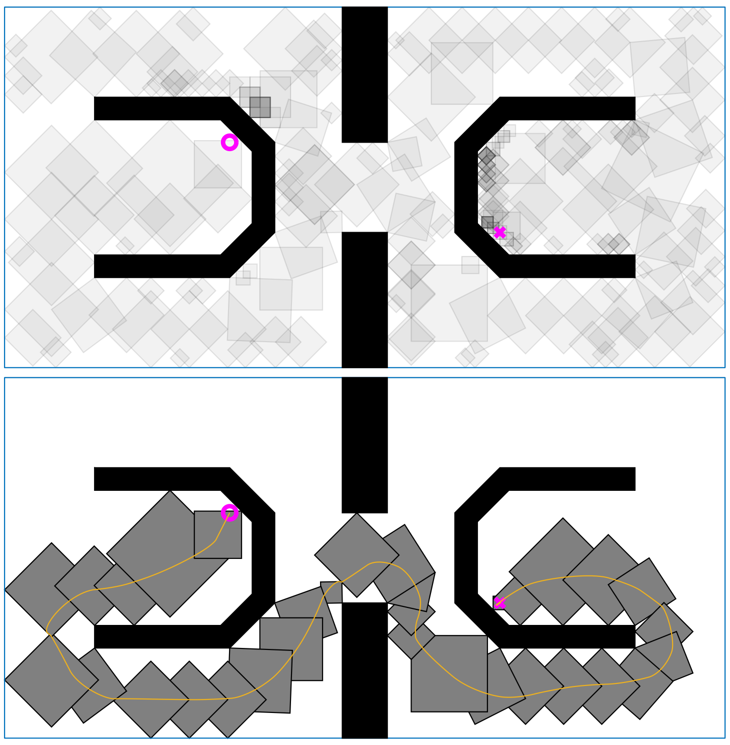


Figure 4.27: An example tree and solution path for Case III.

4.5.4 Case IV: RG-Trees with Node Expansion and Tree Optimization

In this case, node expansion and tree optimization are enabled, but gateways are disabled.In 950/1000 simulations, the robot reached the goal location within 60s time limit. On average, RG-Trees algorithm found a path in 0.93s, with 121 nodes, of which 21 are on the path. The robot reached the goal location in 25.51s, at an average speed of 1.13m/s. The resulting path is 28.50m long.

Solution depth is significantly reduced. However, this configuration also suffers from small intersections. Even more simulations get stuck. Average speed is smaller than Case III, where tree optimization is disabled.

Histogram plots of CPU time, path length, arrival time and average speed are given in Fig. 4.28. A sample tree and path generated with this parameter configuration are given in Fig. 4.29.

4.5.5 Comparison of Parameter Combinations

Boxplots of the parameter combinations are given in Fig. 4.30. It can be seen that when node expansion is enabled, the tree contains a significantly smaller number of nodes. As a result, solution generation time is significantly lower. Similar to the results on Map1, when node expansion is enabled, the robot moves at a higher average speed, and reaches the goal sooner.

When tree optimization is enabled, solution generation time increases, because even when tree optimization is enabled, the tree is generated exactly as when it is disabled, and optimization is performed as an extra. This extra time should not be a major problem if the planning is done offline. Moreover, in some applications, even this larger time is small enough to be performed in real-time [4]. Optimization is made on approximate path length, thus resulting path is shorter. However, due to small intersections, average speed of the robot is smaller, hence arrival time is larger.

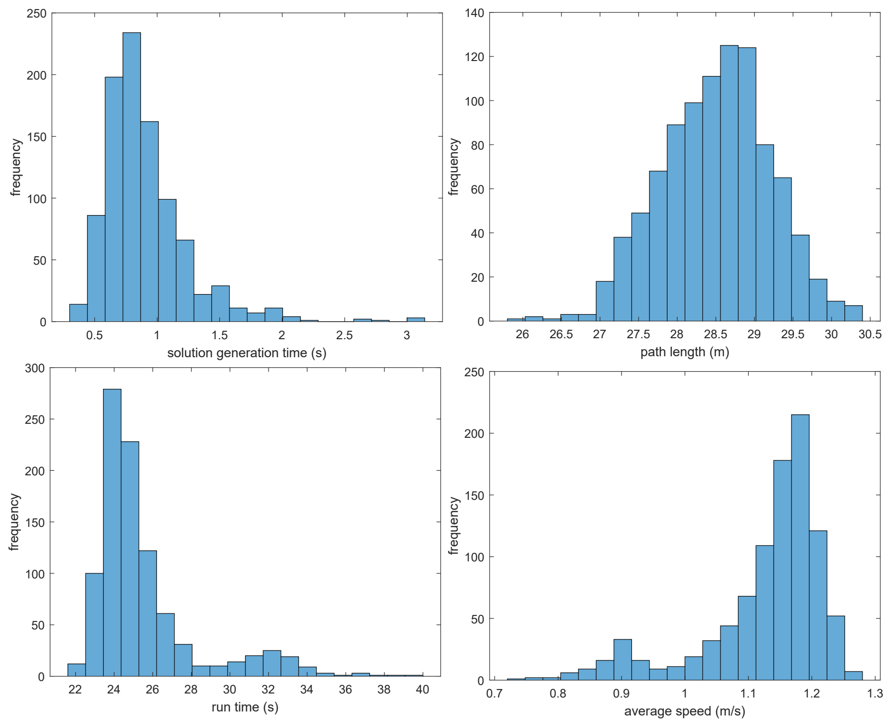


Figure 4.28: Histogram plots for Case IV, when node expansion and tree optimization are enabled.

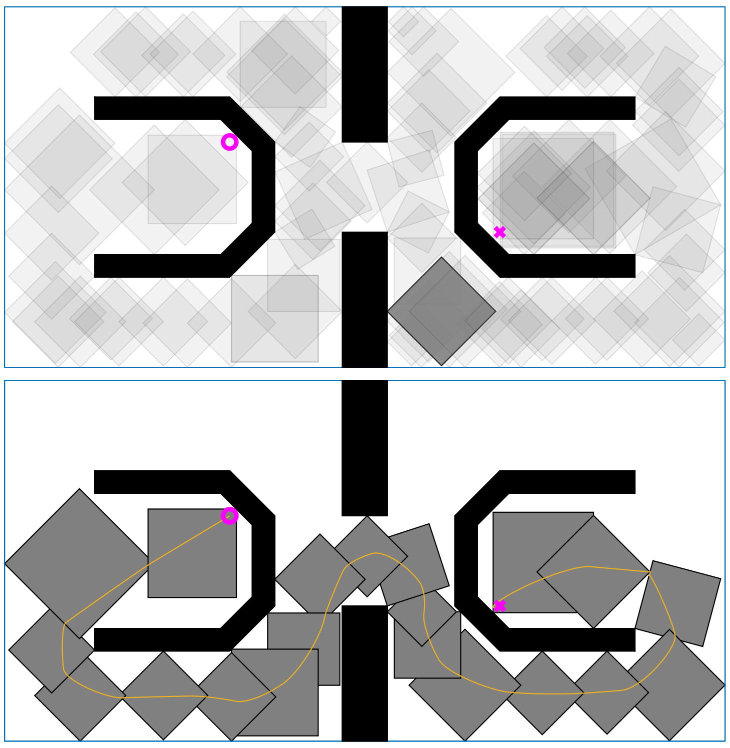


Figure 4.29: An example tree and solution path for Case IV.

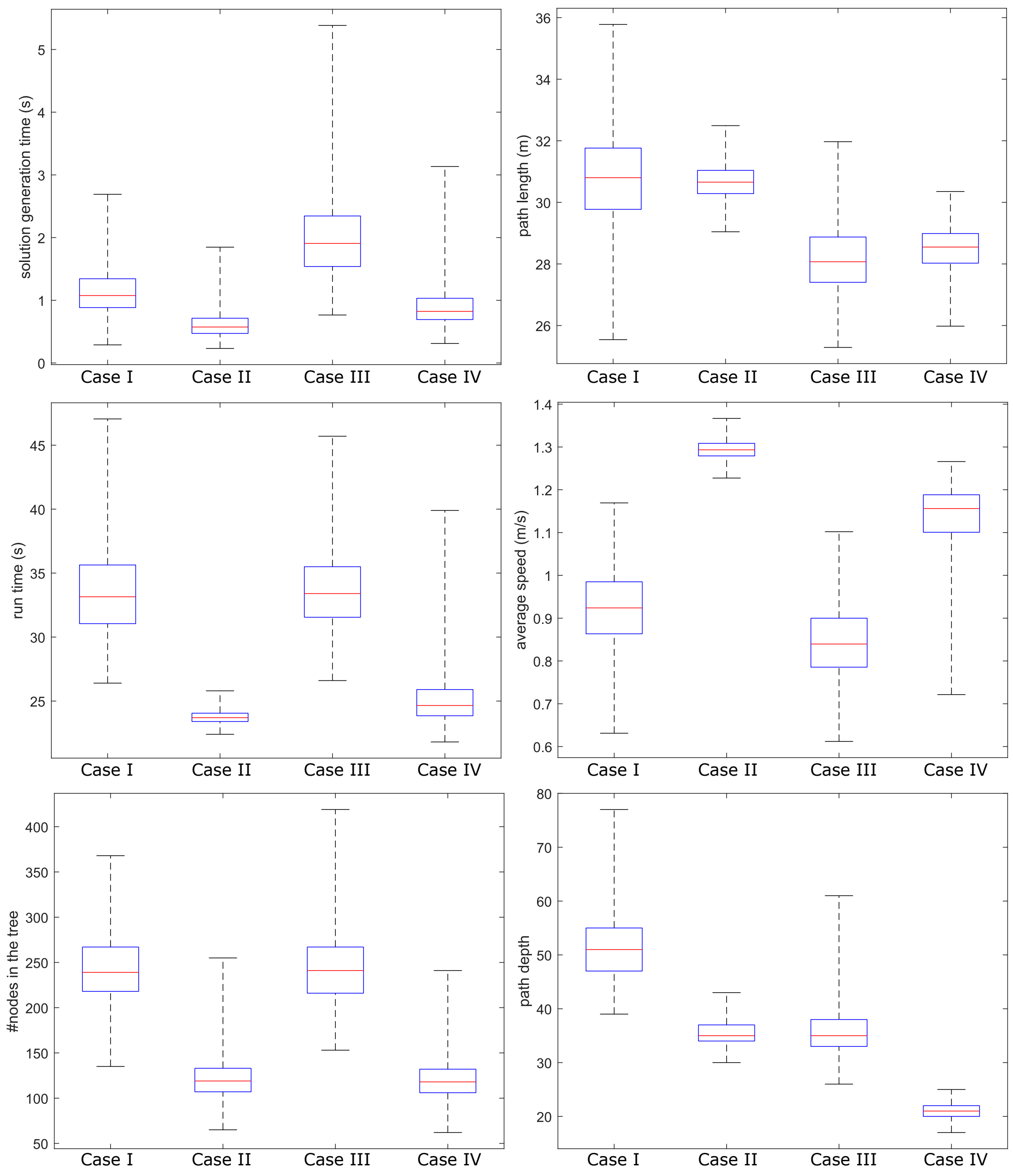


Figure 4.30: Box plot of parameter combinations. Red mark indicates the median. Bottom and top of the box are 25% and 75% points, respectively. Case I: node expansion and tree optimization disabled, Case II: node expansion enabled, tree optimization disabled, Case III: node expansion disabled, tree optimization enabled, Case IV: node expansion and tree optimization enabled

4.5.6 Case V: Effects of Gateway Nodes

In this case, all node expansion, tree optimization and gateway nodes are enabled. In all 1000 simulations, the robot reached the goal location within 60s time limit. On average, RG-Trees algorithm found a path in 1.10s, with 134 nodes, of which 34 are on the path. The robot reached the goal location in 22.90s, at an average speed of 1.26m/s. The resulting trajectory is 28.79m long.

Histogram plots of CPU time, path length, arrival time and average speed are given in Fig. 4.31. A sample tree and path generated with this parameter configuration are given in Fig. 4.32. Boxplots comparing this configuration with Cases II and IV are given in Fig. 4.33.

It can be seen from the boxplots that with gateway nodes, the resulting path is as short as in Case IV, and average speed of the robot is as high as in Case II.

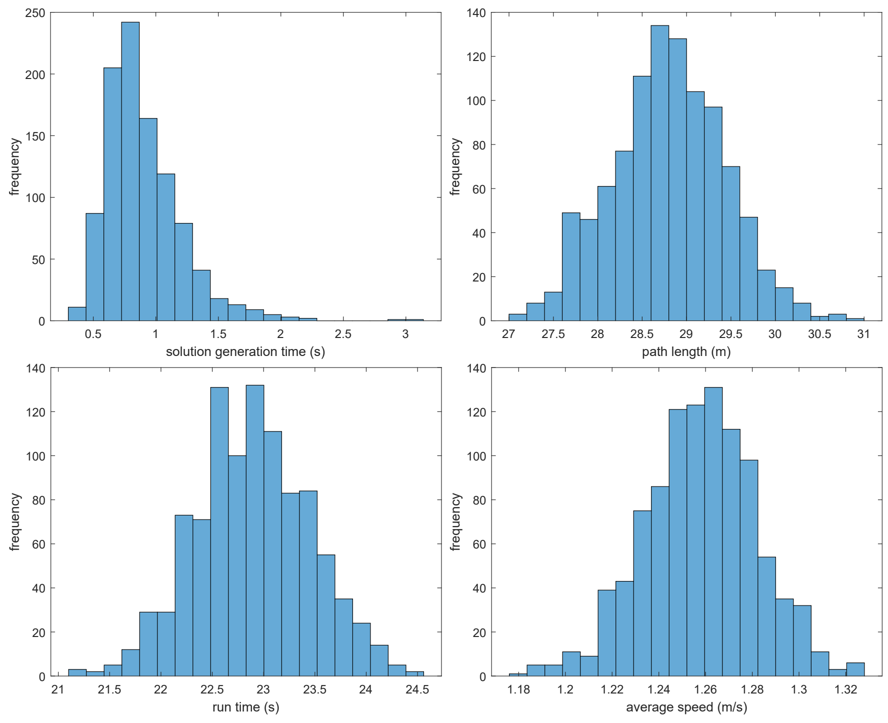


Figure 4.31: Histogram plots when node expansion and tree optimization are enabled and gateways are used

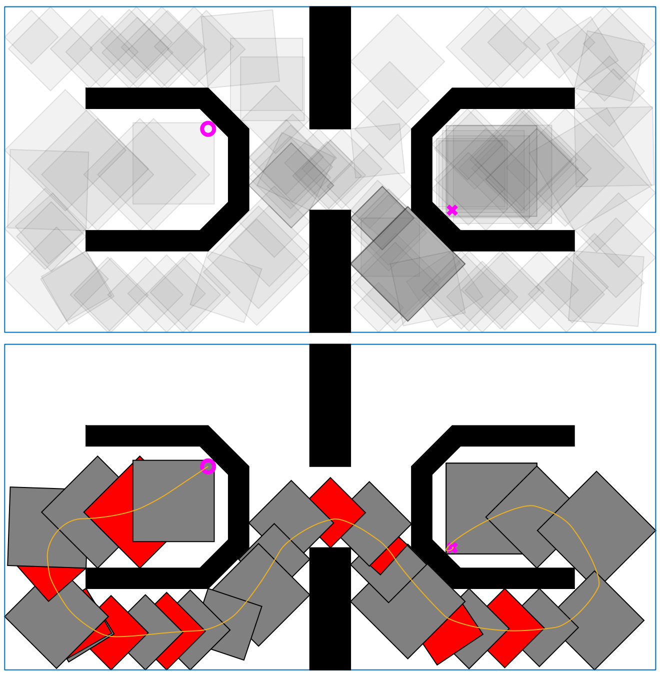


Figure 4.32: An example tree and solution path when node expansion and tree optimization are disabled and gateways are used. Gateway nodes are shown in red.

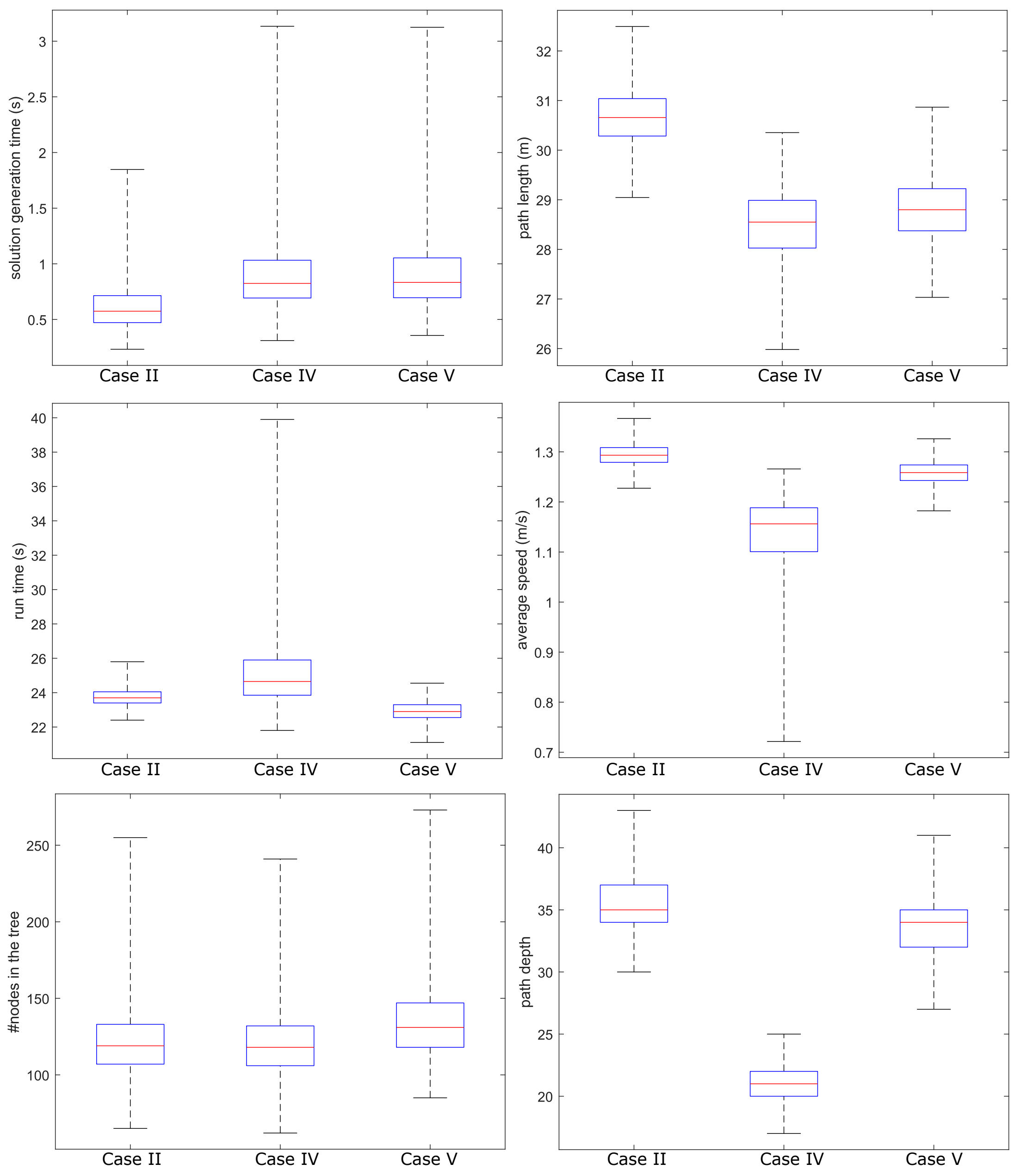


Figure 4.33: Box plot of parameter combinations. Red mark indicates the median. Bottom and top of the box are 25% and 75% points, respectively. Case II: node expansion enabled, tree optimization disabled, Case IV: node expansion and tree optimization enabled, Case V: node expansion and tree optimization enabled and gateways used

CHAPTER 5

CONCLUSIONS

5.1 Summary of Results

In this thesis, we presented RG-Trees, a new obstacle avoidance motion planning algorithm for a robotic system whose dynamics is linear and time-invariant and with a closed-loop controller that is asymptotically stable. The algorithm is a combination of a reference governor with Random Sequential Composition, a derivative of rapidly exploring random trees adapted for systems with dynamics. The algorithm, through random sampling of the obstacle-free space generates and grows square-shaped overlapping regions that are then connected to each other sequentially. This process starts from the goal location (a node) and proceeds until the start location (another node) is reached. When both start and goal locations are covered, the reference governor part of the algorithm steers the robot from the square region it is currently in to the next, sequentially connected node. This steering procedure repeats until the robot reaches the goal region and then the goal location. The RG ensures that the robot stays inside its current region, and since these regions are guaranteed to be obstacle-free, collisions are avoided. We have also showed that with some assumptions, a single RG design in a canonical reference frame can easily be transformed to be used in all connected regions through simple translation, rotation and scaling. This allows significant computational savings as compared to designing separate governors for these regions.

The algorithm requires the obstacles to be mathematically defined shapes and the planning is realized in the 2D work space. Thus the algorithm cannot be easily generalized to higher dimensional complex configuration spaces spaces. In particular, configuration space for higher dimensional robots have obstacles that cannot be easily defined mathematically. However, the RG component successfully handles the robot dynamics and constraints created by the physical obstacles. This makes it possible to handle planning in 2D space for a planar robot with dynamics, a higher dimensional state-space.

We defined three design parameters, namely node expansion, tree optimization and gateway nodes, and seven performance metrics, namely success rate, calculation time, arrival time, number of nodes in the tree and on the path, average speed of the robot and resulting dynamic path's length. We derived and implemented the algorithm in detail for an example case of a double integrator robot controlled with a PD controller. We have tested the effects of these design parameters in Monte-Carlo simulations in Matlab. We have presented the distributions of performance metrics as histogram plots.

We have observed a phenomenon where two consecutive nodes on the path becomes too small and the robot slows down to ensure that the robot stays within the region while converging to the overlap region. This may even result in a stopped robot. We have overcome this difficulty with a variant of our algorithm with "gateway nodes". Note that in all simulations, the tree is generated and a path is returned successfully. However, in some cases, such paths turned out to be infeasible. In our simulations, node expansion generally solves this problem.

5.2 Future Work

In this thesis, we have implemented the algorithm only for a double integrator robot in 2D environment. However, the algorithm can be directly generalized to higher dimensions by incorporating higher dimensional polygons (e.g. cubes for 3D environment). On the other hand, in its current form, RG-Trees is limited to systems where the dynamics are linear or feedback linearizable. The most challenging but exciting future work will be the generalization of the algorithm to nonlinear and nonholonomic systems. We would also like to address uncertainties and process and measurement noises.

REFERENCES

[1] L. Yang and S. M. LaValle, "A framework for planning feedback motion strategies based on a random neighborhood graph," in *Robotics and Automation (ICRA), 2000 IEEE International Conference on*, vol. 1, pp. 544–549, IEEE, 2000.

[2] S.-R. Oh and S. K. Agrawal, "A reference governor-based controller for a cable robot under input constraints," *IEEE Transactions on Control Systems Technology*, vol. 13, no. 4, pp. 639–645, 2005.

[3] O. Arslan and D. E. Koditschek, "Smooth extensions of feedback motion planners via reference governors," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pp. 4414–4421, IEEE, 2017.

[4] C. Petersen, A. Jaunzemis, M. Baldwin, M. Holzinger, and I. Kolmanovsky, "Model predictive control and extended command governor for improving robustness of relative motion guidance and control," in *Proc. AAS/AIAA space flight mechanics meeting*, 2014.

[5] H. Park, S. Di Cairano, and I. Kolmanovsky, "Model predictive control for spacecraft rendezvous and docking with a rotating/tumbling platform and for debris avoidance," in *American Control Conference (ACC), 2011*, pp. 1922–1927, IEEE, 2011.

[6] E. G. Gilbert and I. V. Kolmanovsky, "Set-point control of nonlinear systems with state and control constraints: A lyapunov-function, reference-governor approach," in *Decision and Control (CDC), 1999 IEEE 38th Annual Conference on*, vol. 3, pp. 2507–2512, IEEE, 1999.

[7] A. Perez, R. Platt, G. Konidaris, L. Kaelbling, and T. Lozano-Perez, "Lqr-rrt*: Optimal sampling-based motion planning with automatically derived extension heuristics," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pp. 2537–2542, IEEE, 2012.

[8] B. Akgun and M. Stilman, "Sampling heuristics for optimal motion planning in high dimensions," Georgia Institute of Technology, 2011.

[9] F. Islam, J. Nasir, U. Malik, Y. Ayaz, and O. Hasan, "Rrt*-smart: Rapid convergence implementation of rrt* towards optimal solution," in *Mechatronics and Automation (ICMA), 2012 International Conference on*, pp. 1651–1656, IEEE, 2012.

[10] O. Arslan, V. Pacelli, and D. E. Koditschek, "Sensory steering for sampling-based motion planning," in *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pp. 3708–3715, IEEE, 2017.

[11] J. Kim and J. P. Ostrowski, "Motion planning a aerial robot using rapidly-exploring random trees with dynamic constraints," in *Robotics and Automation (ICRA), 2003 IEEE International Conference on*, vol. 2, pp. 2200–2205, IEEE, 2003.

[12] S. M. LaValle and J. J. Kuffner Jr, "Randomized kinodynamic planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

[13] S. Karaman and E. Frazzoli, "Sampling-based optimal motion planning for non-holonomic dynamical systems," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pp. 5041–5047, IEEE, 2013.

[14] J. J. Park and B. Kuipers, "Feedback motion planning via non-holonomic rrt* for mobile robots," in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pp. 4035–4040, IEEE, 2015.

[15] S. Dalibard, A. El Khoury, F. Lamiraux, A. Nakhaei, M. Taïx, and J.-P. Laumond, "Dynamic walking and whole-body motion planning for humanoid robots: an integrated approach," *The International Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1089–1103, 2013.

[16] R. Tedrake, I. R. Manchester, M. Tobenkin, and J. W. Roberts, "Lqr-trees: Feedback motion planning via sums-of-squares verification," *The International Journal of Robotics Research*, vol. 29, no. 8, pp. 1038–1052, 2010.

[17] E. Ege and M. M. Ankarali, “Feedback motion planning of unmanned surface vehicles via random sequential composition,” in review.

[18] R. R. Burridge, A. A. Rizzi, and D. E. Koditschek, “Sequential composition of dynamically dexterous robot behaviors,” *The International Journal of Robotics Research*, vol. 18, no. 6, pp. 534–555, 1999.

[19] S. Karaman and E. Frazzoli, “Sampling-based algorithms for optimal motion planning,” *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[20] I. Kolmanovsky, E. Garone, and S. Di Cairano, “Reference and command governors: A tutorial on their theory and automotive applications,” in *American Control Conference (ACC), 2014*, pp. 226–241, IEEE, 2014.

[21] S. M. LaValle, “Rapidly-exploring random trees: A new tool for path planning,” 1998.

[22] E. Garone, S. Di Cairano, and I. Kolmanovsky, “Reference and command governors for systems with constraints: A survey on theory and applications,” *Automatica*, vol. 75, pp. 306–328, 2017.

[23] E. G. Gilbert and K. T. Tan, “Linear systems with state and control constraints: The theory and application of maximal output admissible sets,” *IEEE Transactions on Automatic Control*, vol. 36, no. 9, pp. 1008–1020, 1991.

[24] N. Karmarkar, “A new polynomial-time algorithm for linear programming,” in *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pp. 302–311, ACM, 1984.

[25] R. W. Brockett *et al.*, “Asymptotic stability and feedback stabilization,” *Differential Geometric Control Theory*, vol. 27, no. 1, pp. 181–191, 1983.

[26] D. E. Koditschek, “Task encoding: Toward a scientific paradigm for robot planning and control,” *Robotics and Autonomous Systems*, vol. 9, no. 1, p. 5, 1992.

[27] F. Golbol, M. M. Ankarali, and A. Saranli, "Rg-trees: Trajectory-free feedback motion planning using sparse random reference governor trees," in *Intelligent Robots and Systems (IROS), 2018 IEEE/RSJ International Conference on*, IEEE, 2018, to appear.

[28] N. J. Cowan, "Navigation functions on cross product spaces," *IEEE Transactions on Automatic Control*, vol. 52, no. 7, pp. 1297–1302, 2007.

[29] J. L. Bentley and T. A. Ottmann, "Algorithms for reporting and counting geometric intersections," *IEEE Transactions on Computers*, no. 9, pp. 643–647, 1979.

[30] H. R. Ossareh, "Reference governors and maximal output admissible sets for linear periodic systems," *arXiv preprint arXiv:1804.09262*, 2018.

[31] J. Ng and T. Bräunl, "Performance comparison of bug navigation algorithms," *Journal of Intelligent and Robotic Systems*, vol. 50, no. 1, pp. 73–84, 2007.

[32] N. Buniyamin, W. W. Ngah, N. Sariff, and Z. Mohamad, "A simple local path planning algorithm for autonomous mobile robots," *International journal of systems applications, Engineering & development*, vol. 5, no. 2, pp. 151–159, 2011.