Task Decomposition for Construction 3D printing

Chiang Hou Nam[1]

MSc Robotics and Computation

Supervisors:
PhD Julius Šustarevas,
Professor Simon Julier

Submission date: 27th September, 2021

Abstract

Additive manufacturing has been extensively researched throughout the years because of its inexpensiveness and efficiency in producing structures in a wide variety of shapes and sizes, which assisted the growth of different industries including manufacturing, medicine and construction. In the construction industry, multiple mobile robots are equipped with nozzles to extrude construction materials on-site to print structural components. However, the site environment can be very busy and many other tasks are being carried out simultaneously which can hinder the construction of the printing task. Therefore, with the increase in scale and complexity of these constructions, effectively decompose and allocate different parts of the printing task to individual robots with the awareness of obstacles in the environment will be extremely useful. The objectives in this study include investigating and performing navigation of single mobile manipulator with task consistent path planning, proposing methods for decomposing construction 3D printing tasks and integrating them with task consistent path planning algorithms for multiple robots to work in parallel. The feasibility and performance of the integrated systems are evaluated and compared. Results show successful decompositions in five sets of experiments with directions for future work which aims to improve the robustness of the final selected approach.

Word Count: 19787 words

Acknowledgements

I would like to pay my special regards to my supervisors Professor Simon Julier and his PhD student Julius Šustarevas for their patience to give invaluable guidance and insightful feedback whenever requested, which pushes me to sharpen my thinking and brought my work to a higher level. Without their encouragement and support, I would not be able to overcome difficult hurdles encountered and lead to the completion of this thesis.

I would also like to give thanks to the people that I met in the Robotics and Computation and Machine Learning cohort for being a strong and supportive network so that we can overcome the adversities we came across over the year.

Lastly, I would like to express my deepest gratitude to my friends and people who have faith in me for their strong support and encouragement throughout the academic journey.

Contents

List of Algorithms

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivation

Additive manufacturing (AM) or 3D printing is a technology that sequentially layers materials to build 3D structures based on digital 3D models by using computer control [1]. It has been significantly researched in recent years to meet the demand of building a variety of models and prototypes with different shapes and sizes efficiently [2] for different fields due to its low cost and low energy consumption [3]. Some of the fields are medicine [4][5], aerospace [6], construction [7], manufacturing [8], art and jewelry [9]. In general, recent 3D printed objects are built by using a technique named Fused Deposition Modeling (FDM). A basic structure of a 3D printer using FDM is shown in Figure 1.1. It uses heat to deposit semi-melted thermal sensitive plastic on a machine bed according to the instructions generated by a computer system [10].

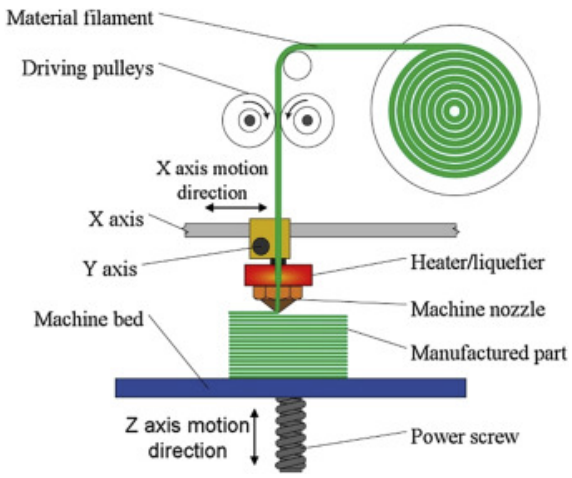


Figure 1.1: A Fused Deposition Modeling printer extruding material filaments through a heated machine nozzle on a machine bed to construct a manufacturing part [11]

FDM has a number of advantages such as high speed, high accuracy, low cost and is fairly easy to use. However, there are a few limitations. The first one is the size of the model being printed [12]. This is because the printing has to be printed using an overhead x, y axis frame

and the power screw has a limited motion in the z direction as illustrated in Figure 1.1. The second limitation is the material usage. The reason for this is because the material has to be extrudable. Although FDM is not ideal, since the advantages are more attractive, they outweigh the disadvantages overall. So the idea of depositing semi-melted materials in FDM has been extended to work with other compatible materials. For example, in the aerospace industry, people use FDM with thermoplastics to manufacture and prototype fixtures, jigs and end-use parts on aircrafts [13].

In construction, 3D printing robotic systems such as gantry systems [14] and mobile manipulators [15] were built to deposit concrete or other construction materials on sites. Since constructions 3D printing is usually large, people were interested in controlling multiple robots to work on the same printing in parallel to increase efficiency [15]. However, there are different considerations when assigning a section of the printing task to a robot. In this regard, a task decomposition method that helps facilitate the distribution of work to the robots will be beneficial for overall efficiency. In this study, methods of task decomposition for construction 3D printing will be proposed and implemented.

1.2 Aims and Objectives

This project intends to facilitate task distribution for multiple robots by proposing methods of task decomposition so that the decomposed tasks can be carried out in parallel, which in turns boost the efficiency of large scale construction 3D printing. The objectives of the study are:

1. To investigate and perform navigation of single mobile manipulator with task consistent path planning.
2. To investigate and propose methods for decomposing construction 3D printing tasks.
3. To integrate the proposed task decomposition methods with task consistent path planning for multiple robots to work in parallel.
4. To evaluate the feasibility and performance of the integrated systems.

1.3 Outline of the Thesis

The structure of the thesis is organized as follows:

- **Chapter 2**: A detailed literature review about different robotics systems used in construction 3D printing to understand the current latest technology being used. Then common task decomposition methods, path planning algorithms and task consistent path planning approaches will be investigated with the selection of appropriate task consistency approaches and path planning algorithms.
- **Chapter 3**: The problem definition, the approaches to extend path planning algorithms with task consistency and the proposed methods for task decomposition are presented in detail. In addition, a method to optimize the operations for faster computation will be discussed.

- **Chapter 4**: The five experimental setups in MATLAB simulations used to evaluate the feasibility of the proposed methods are described. Task decomposition and path planning results using the proposed methods are presented. Moreover, the limitations and difficulties of the study are discussed in this chapter.
- **Chapter 5**: Concludes the significance of this study. The contributions of this study are also included in this chapter.
- **Chapter 6**: Introduces future work which could solve the inadequacies and limitations of this study.

Chapter 2

Literature Review

In this chapter, a literature review on the current applications of Additive Manufacturing (AM) in the construction industry is first given. Then, an introduction of the current task decomposition approach and application will be given. After that, an overview of path planning algorithms and task consistent path planning will be given.

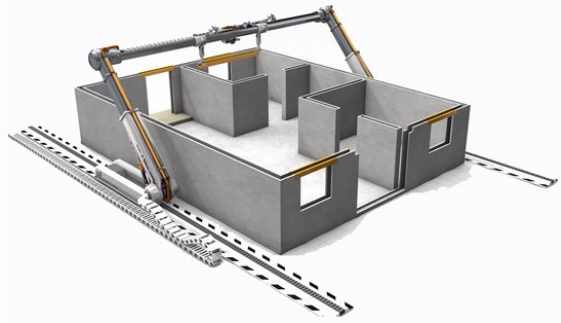
2.1 Additive Manufacturing in construction industry

This section aims to provide an overview of the development of Additive Manufacturing (AM) to understand the limitations that barred the technology from being more efficient and scalable in the realm of the construction industry. Some popular static and mobile AM systems including those being developed in the construction field will be presented, along with an analysis of their advantages and disadvantages.

2.1.1 Stationary Robotic Construction Systems

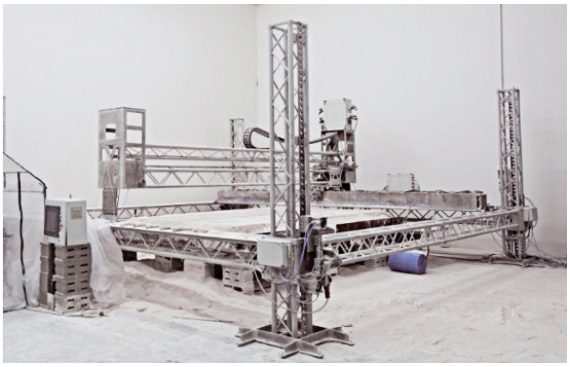
Gantry Systems

Inspired by Pegna who proposed solid freeform construction in 1997 [16], there has been increasing interest in exploring AM processes in the realm of civil engineering. One product of such exploration is gantry system [14][7]. There are different designs of gantry systems, but in general, they all consists of a frame that holds an overhead bridge which allows a mounted manipulator to move across a horizontal plane. In the case of 3D printing, this manipulator is a printing nozzle. Figure 2.1 shows some of these gantry systems including Contour Crafting [14], Concrete Printing [17], D-shape [18][19].



(a) Contour Crafting with gantry system [14]

(b) Concrete Printing with gantry system [17]



(c) D-shape with gantry system [18][19]

Figure 2.1: Gantry Systems

As we can observe in Figure 2.1, Gantry systems are frames that controls the motion of a concrete extruder in the 3D axis. Given the elegance of simplicity to install, it is has been a popular choice in the commercial world [20] over the years [21].

Although gantry systems have been recognized commercially, we can also observe a major drawback of gantry systems. That is, the structure being printed must be enclosed by the gantry framework. This means when it comes to large scale concrete printing, we need to build a giant gantry system [22]. This creates a large financial burden especially when the construction project is just one-off because after the project the giant frame will be idle most of the time and it also occupies a large volume of space. Creating very large gantry frames also raise structural, safety and printing quality concerns. To elaborate, since very long beams suffers a larger deflection under loading [23], and vibrations are unavoidable as the nozzle move around. Therefore, there will be a high chance that the printing quality worse than expected. Moreover, the instability of the printing framework may also raise safety concerns for the on-site workers since unstable structures have a higher chance of failure. The solution for improvement is to either use stronger materials to build the gantry frame or print the structure at a slower rate to minimize the oscillation induced by the nozzle movement [24]. However, they are highly undesirable because of the following two reasons. Firstly, using a stronger material for a gantry frame that will be rarely utilized is fundamentally counter-intuitive because it creates a larger financial burden. Secondly, printing the structure at a slower pace means more expense on manpower. In an extreme case, the construction project

may be more financially feasible if it is built in the traditional way. Observing Figure 2.1, we can tell that gantry systems suffer from printing an object from directions other than from the top because the nozzles are always pointing down. An example of such nozzle is shown in Figure 2.2[1]. Moreover, gantry systems also suffer from building structures that are larger than themselves. This limits the design of the printing structure and scalability. To liberate the restriction on printing size, people have applied nozzle equipped mobile robots to construction, which will be discussed in Section 2.1.2.

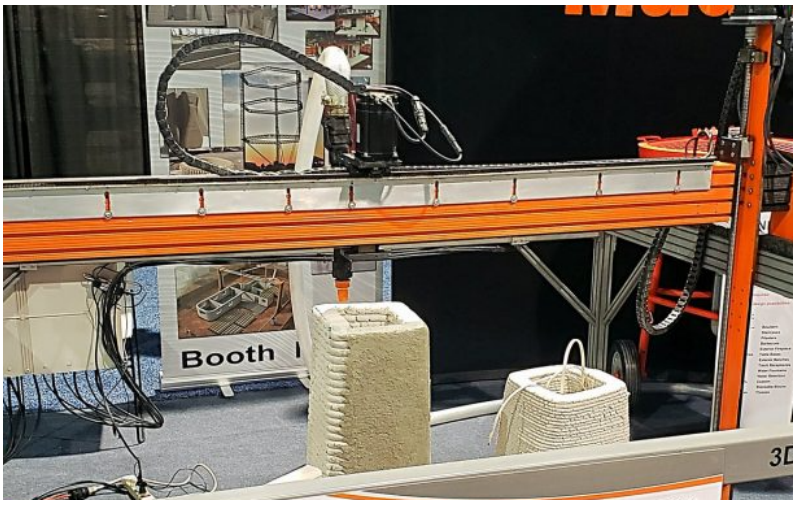


Figure 2.2: A gantry system with a concrete printing nozzle mounted, printing a structure

2.1.2 Mobile Robotic Construction Systems

Mobile Manipulators

Mobile Manipulator usually refers to robotic systems that have a high Degree of Freedom (DoF) robotic arm mounted on a moving platform using different methods of locomotion such as omnidirectional wheels [25] and legs like the Spot Arm developed by Boston Dynamics [26]. Combining a mobile platform and a robot arm with high DoF enables high-quality printing and an infinite workspace since a robot arm with many DoF offers a higher degree of flexibility [27] and putting such a robot arm on a moving platform liberates the arm from bounded reachability by carrying the arm to the region where it can approach the task.

One such mobile manipulator system is the Digital Construction Platform (DCP) developed by MIT [28]. The DCP consists of a five DoF hydraulic boom arm and a six DoF KUKA robotic arm sitting on a tracked vehicle. With these two arms, the boom arm was used to quickly position the end effector to the vicinity of a target pose, and the KUKA manipulator was used for fine positioning. Since precision is important, a collection of controllers is used for accurate positioning and oscillation minimization for the KUKA manipulator. Figure 2.3 shows the DCP printing a dome that had a diameter of 14.6m using foam as the building material. Besides printing structures, We can also see that mobile manipulators have been utilized in a variety of construction activities such as welding [29][30][31], drilling [32] and building formworks [33][34].

[1]Source: https://www.3dprintingbusiness.directory/company/mudbots-3d-concrete-printing/

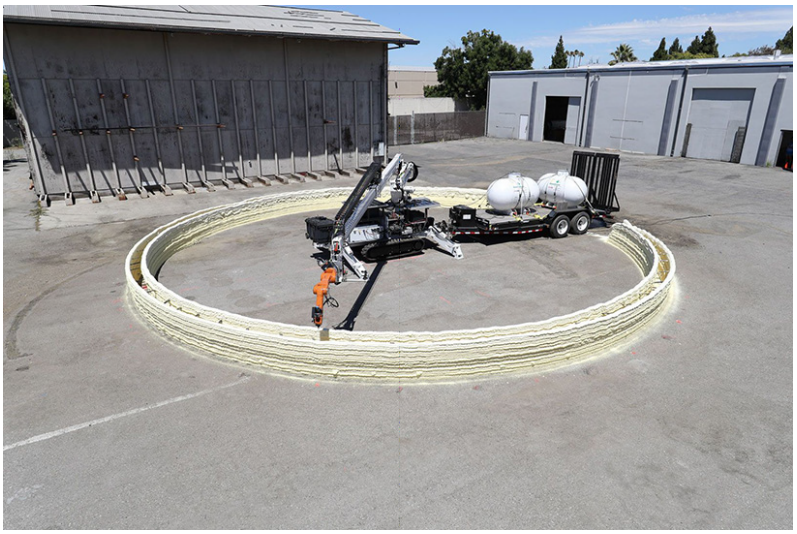


Figure 2.3: Digital Construction Platform (DCP) printing a 14.6m diameter dome [28]

Comparing mobile robotic construction systems with the stationary ones, we can observe that mobile manipulator systems resolved the restriction of working space that is rooted in stationary systems. The robot arms on mobile manipulators have a high DoF also means mobile manipulator systems can have a higher finishing quality. However, recognizing the indefinite 2D workspace of mobile manipulator systems, the vertical reach is restricted by the size of the robot arm. Albeit this limited the ability of mobile manipulator systems from building tall structures, we can always build temporary structures or parts like stairs that help to transport the robot to a higher level. This can also be observed in traditional construction methods as workers finish high rise buildings from the ground. In terms of build-ability, since similar constraints can be observed in traditional constructions, the bounded of vertical reach is essentially non-existent if we reasonably prioritize the parts that help robots to climb to higher levels into the organization of the construction plan. In this regard, using mobile manipulator systems is a more superior choice than the stationary ones which has the size constraint barred by the gantry frame. This in turns means that we can build large scale structures with mobile manipulator systems. However, using a single mobile manipulator for large scale structures can be extremely time-consuming. In this regard, an introduction of multi-agent systems will be given in Section 2.2.

2.2 Multi-agent Robotics and Swarm Systems

This section gives an introduction to swarm robots and mobile manipulators. Examples in multi-agent cases in construction will be given. This helps us to understand why using multiple mobile manipulators can liberate the operation in construction 3D printing from limited scalability and efficiency compared to the stationary 3D printers when it comes to large scale constructions.

2.2.1 Swarm Robots

As mentioned in Section 2.1.1, construction tasks are usually large in scale. Therefore, completing a construction 3D printing using a single robot will be difficult and time-consuming.

To solve this, using multiple robots [35] in a cooperative way will increase the efficiency of the task. Therefore, the way to control multiple robots cooperatively becomes important. One such control scheme is swarm robotics. It has been significantly researched to meet the demand of autonomous control of multiple agents, including Unmanned Ground Vehicles (UGVs) [36][37], Unmanned Aerial Vehicles (UAVs) [38], Satellites [39] and Autonomous Underwater Vehicles (AUVs) [40]. Figure 2.4 shows an illustration of the leader-follower coordination of AUVs to form a pattern.

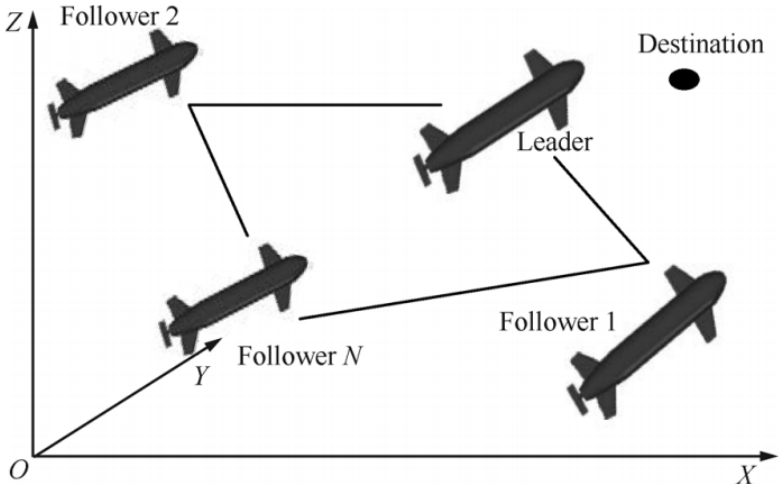


Figure 2.4: Coordination of AUVs [40]

In the past, most swarm systems were managed using centralized control protocols [41], leading to the requirement of communication between all agents and large computation loads. There are also studies about distributed control which requires local communication links between neighbours instead of all agents and distributes computation loads to agents [42], providing larger flexibility and scalability of the swarm system. Up to date, swarm systems have been used on a wide range of applications, including surveillance [43], rescuing after disasters [44], space interferometry [45] and also construction 3D printing [15]. Since we are interested in controlling multiple robots in 3D printing, the case of controlling multiple mobile robot manipulators will be discussed in Section 2.2.2.

2.2.2 Multi-agent Mobile Manipulators

As people try to use mobile manipulators for larger tasks or tasks that require several procedures, deploying more mobile manipulators to these tasks are desirable. This motivates research on controlling multiple mobile manipulators. One of such studies is [46], which used multiple robots to finish a series of tasks including transporting, alignment and fastening to assemble a wing panel on a wing box, which shows that applying multiple robots for distinctive individual or cooperative tasks is feasible. In terms of constructions, since construction tasks are usually large, we can divide the large task into small ones and allocate them to individual robots. With this, there are also research on multi-agent construction [47][48][49][50][51]. One of such a work which is similar to [46] is [47]. They proposed a method to plan paths for multiple mobile robotic cranes to transport a large construction part by fusing environmental data from sensors in a multi-agent system to detect any possible collisions, which has an impact on improving the safety of the construction site and productivity. Another work is [15] which proposed a system using multiple mobile manipulators to print one large structure as shown in Figure 2.5.

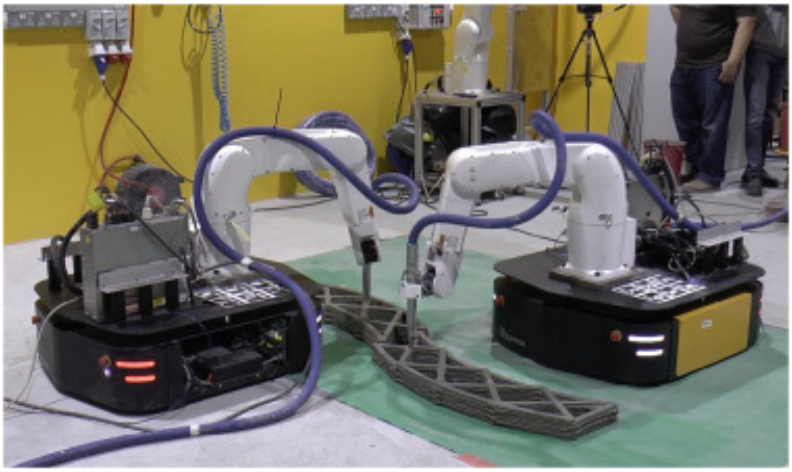


Figure 2.5: Two mobile manipulators printing a structure [15]

On the other hand, comparing with using a single manipulator, using a multi-agent approach also enjoys greater efficiency and scalability. This is also demonstrated in [15] that printing the same structure using two printers is two times faster than using just one printer.

Nonetheless, the boundless 2D workspace also comes with a cost. In addition to obstacles in the dynamic and busy environment on-site, collisions between the manipulators and the evolving printing task are highly not desirable because this will ruin the work done. Therefore, a strategy that can navigate the manipulators to build structures with the awareness of the growing structure and other obstacles is needed. A method that helps to solve this navigation problem is task consist path planning, which will be discussed in Section 2.5.

Having discussed the advantages and drawbacks of multi-agent mobile manipulators, we can see that using mobile manipulators with swarm techniques is a feasible solution for construction 3D printing. Since construction 3D printings are large scale printing tasks, a strategy that helps to break down the large task into small ones and distribute them to multiple robots is desired. One such method is task decomposition, which will be discussed in Section 2.3

2.3 Task decomposition

Task decomposition has been researched and explored because of several problems that arise in 3D printing. One such problem 3D printing intrinsically has is that supporting structures are needed for printing regions that overhang as shown on the left side of Figure 2.6, which can be difficult to remove if the printing is complex. It can also ruin the surface of the printing. Moreover, as more supports are needed, more waste of material is expected [52]. In this context, task decomposition helps to break down the printing in a way so that each part can be printed with minimal supporting structures. This is done by allowing printing materials to accumulate along different directions for different parts. A printing using such a method is shown on the right side of Figure 2.6.

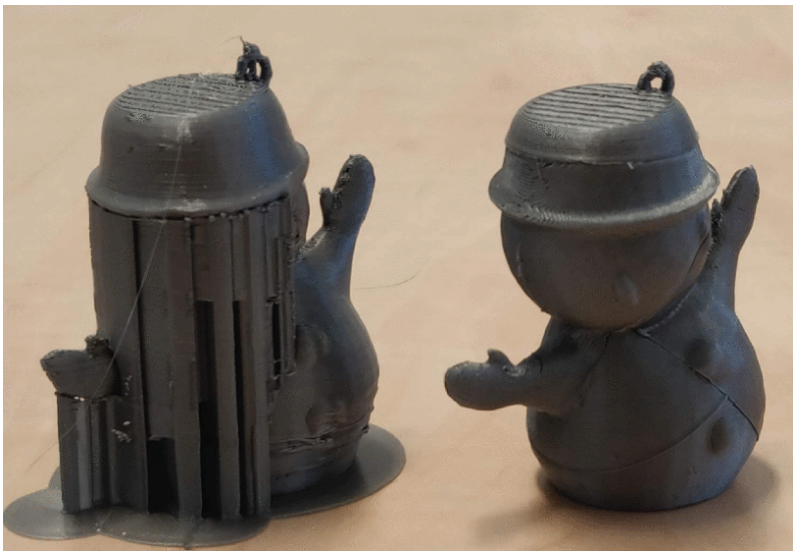
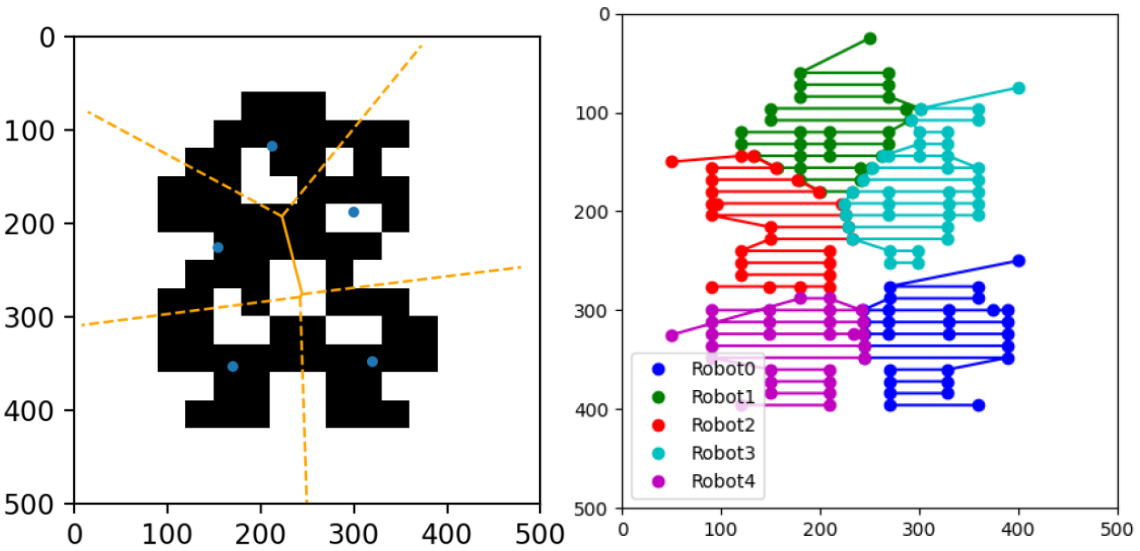


Figure 2.6: Left: A 3D printing structure with support for overhanging regions. Right: The same 3D printing with decomposition. [52]

Another issue is when there is a large task so that many robots collaborating on the task is desirable. Some of these tasks are agricultural operations [53] such as harvesting, logistic operations [54] and lawnmower [55]. In [54], a hierarchical coordination strategy that aims at partitioning the working space of an industrial environment into sectors for robots is provided by using a clustering algorithm. This effectively avoids unnecessary interactions between robots when operating because individual robots will mainly work in their assigned sections. However, in the context of construction 3D printing, it is impossible to have a predefined path like the case in [54]. Instead, we need to generate paths using path planning algorithms that can couple with task consistent constraints, because the task that the robot works on can be different every time.

Karpe et. al [56] use geodesic cellularization to divide an image of 3D printing tasks into geodesic cells by a modified k-means clustering algorithm with an objective function that aims to distribute the task equally to every robot as shown in Figure 2.7. Since the duration of printing is defined by the robot that used maximum time, using geodesic cellularization with k-means is a good way to help divide the workload equally.



(a) Geodesic cells

(b) Robot trajectories of distributed printing as a result of geodesic cellularization

Figure 2.7: Geodesic cellularization utilizing k-means algorithm [56]

In terms of construction 3D printing, Šustarevas et. al [57] has recently presented a solution to decompose construction 3D printing tasks. This solution consists of 4 steps. Firstly, the printing task is converted into a black and white binary image by a human. An example of such a conversion is illustrated in Figure 2.8. Secondly, it is transformed into an undirected graph which has its vertices and edges representing pixel occupancy and adjacency respectively as shown in Figure 2.9.

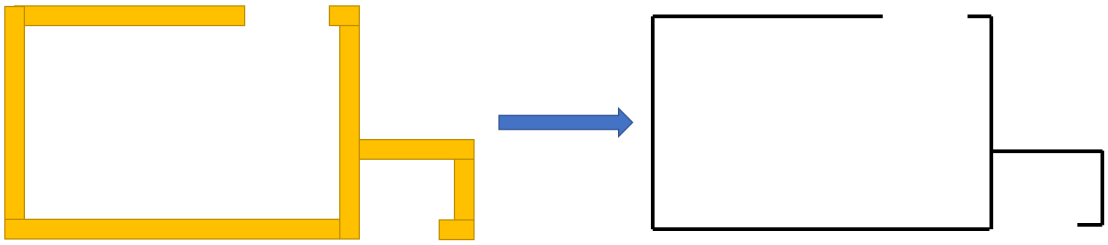


Figure 2.8: An example of a converting a printing task to a binary image. An illustration of the original printing task (Left). The converted binary image (Right)

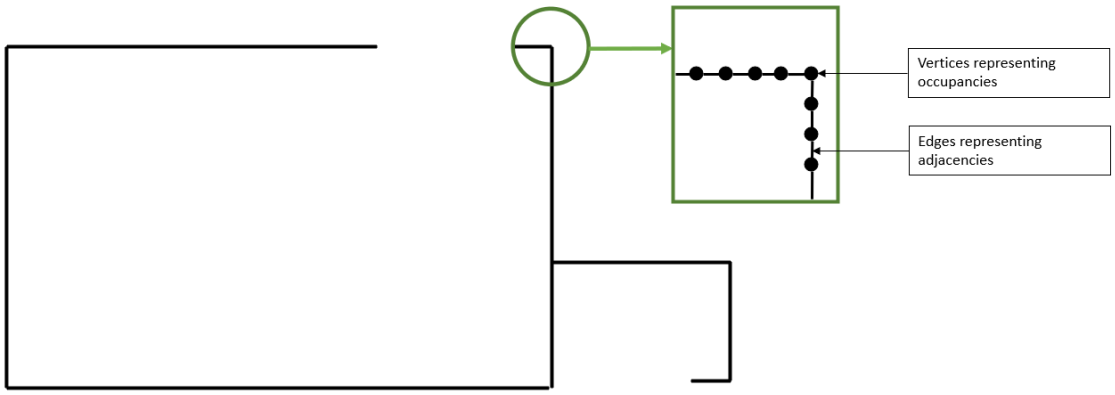


Figure 2.9: Binary image turned into a graph of vertices and edges

Then, the graph is scaled to include Cartesian coordinates of each printing point and the physical distance calculated from the coordinates as the edge weight. After that, a decomposition algorithm utilizing Depth-First-Search (DFS) is then used to travel through each vertex of the graph starting with a randomly chosen one. This algorithm decomposes the graph into segments based on two criteria:

1. If the segment is longer than a threshold sampled from a Normal distribution with a specified mean, the segment will terminate at that length and start a new segment.

2. If the DFS algorithm detects a graph branch before the threshold is reached, the segment will also be terminated at that point.

These resultant segments were then sequenced in a directed graph tree which has its nodes representing the segments and edges representing the adjacencies in space. The reason why adjacency is used to sequence the tasks is that the robot can be distributed in a way that congestion can be minimized. The result of decomposition and sequencing procedure for the case shown in Figure 2.8 is displayed in Figure 2.10 and the figure's upper right corner respectively. With this, we can then allocate the tasks to one or more robots. In their work, an auction approach with the consideration of the printing capacity of each robot and the distance between the end-effectors and the task points was used to allocate the segments to multiple robots which proved the capacity of mobile multi-robot systems for construction.

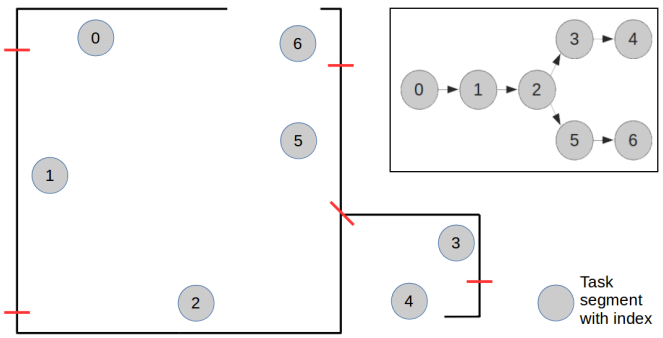


Figure 2.10: Example of construction 3D Printing task decomposition and sequencing [57]

The whole decomposition-sequencing-allocation process (task allocation process) described above can be summarized as in Figure 2.11.

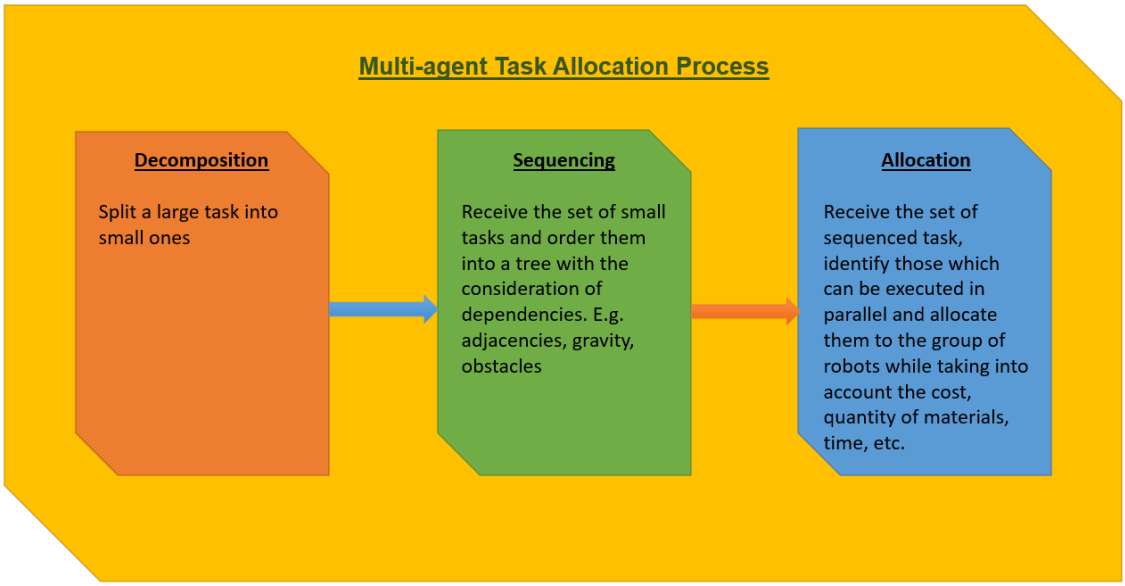


Figure 2.11: Task allocation process flow chart

Since the tasks are decomposed at length based on a number sampled from a Normal distribution, there may still be chances that the allocated tasks need to be redistributed. For example, a task can be assigned to a robot with insufficient battery life and the robot may not be able to finish the task. The task left over will then need to reallocate to another robot. To optimize the whole process, it would be nice to have the allocation information pass to the decomposition algorithm so that it can output segments in a more optimal way such that the number of reallocation procedures can be minimized. However, without a decomposed set of tasks, the allocation procedure will not be able to distribute tasks to individual robots. In this regard, Zlot's work [58] has attempted to solve this puzzle with a market-based approach distributed algorithm which decomposes and allocate tasks to robots simultaneously and continuously.

While we can see that much research has been done to propose ideas that help to decompose a large task into small ones, they are very specific to a particular problem. This makes sense because different classes of problems should be decomposed in different ways. Concerning construction mobile 3D printing, although a task distributing system was developed by Šustarevas et. al, the decomposition algorithm will fail when there exist obstacles that can block the robot from printing the task. This is because, as mentioned earlier, the tasks are decomposed at lengths sampled from a Normal distribution. By doing this, there may exist obstacles in the middle of the decomposed tasks. Since a construction site is a busy and dynamic environment, detailed scheduling and project management is required to complete constructions on time. So, there can be many different tasks happening simultaneously on a floor. Therefore, a task decomposition strategy that can be aware of obstacles will be extremely useful. For example, along with the construction of structural components like walls and columns, water pipe planning can be done at the same time. In this case, the water pipes will be obstacles for the mobile robots to print the structures. In this thesis,

we address this problem by proposing a strategy that can decompose tasks with the consideration of static obstacles so that the construction printing task can be allocated to multiple robots in a way that minimizes interactions with obstacles. This will be described in Chapter 3.

With the decomposed tasks, each robot has to know how they should navigate themselves so that the tasks can be printed using their end-effectors. In the next section, path planning will be introduced.

2.4 Path Planning

To facilitate multiple robots to work on a shared task in 3D printing, every robot has to know where they should start and how they should navigate themselves along with their tasks. In this regard, path planning is needed for every robot. Path planning is an extensively researched area that provides solutions to move an object from the starting point to the destination. In general, the problem defined in the domain of path planning is to find a trajectory connecting the start and goal points. This trajectory consists of a sequence of valid configurations describing the states of the object which are sampled from the free configuration space $\mathcal{X}_{free}$. In the context of a mobile robot, the configuration space in Cartesian or polar form has been widely used.

One category of the path planning algorithms is optimal algorithms [59][60][61]. Optimal algorithms aim to minimize a cost function subject to dynamic and static constraints [62], which is similar to the problem that we are trying to solve - finding an optimal path under the static constraint of obstacles and dynamic constraints from the busy environment. However, a huge computational time is needed if we want to have a high-quality path. Therefore, approximation techniques [63] are needed and this leads to the research of sampling-based algorithms.

In this section, we will first present the problem definition for path planning to give an idea of what path planning is trying to solve, then a literature review on three sampling-based path planners, Rapid Exploring Random Tree (RRT*) Bi-directional RRT* (B-RRT*) and Fast March Tree (FMT*) will be given. The advantages and disadvantages, along with their appropriateness for mobile 3D printing will also be discussed.

2.4.1 Problem Definition for Path Planning

Let $\mathcal{X} \subseteq \mathbb{R}^d$ be a d dimension configuration space such that each element of $\mathcal{X}$ is a configuration of the object to navigate, which is a robot in our case. In $\mathcal{X}$, we have the fixed obstacle space $\mathcal{X}_{obs} \subset \mathcal{X}$ which represents the configurations that the robot collides with the obstacles and the obstacle-free space $\mathcal{X}_{free} := \mathcal{X} \setminus \mathcal{X}_{obs}$ which represents the configurations that the robot can move freely.

With this setting, the path planning problem is to search a collision-free path $\sigma(t) \in \mathcal{X}_{free} \forall t \in [0, 1]$ linking an initial configuration $\sigma(0) = q_{start} \in \mathcal{X}_{free}$ and a goal configuration $\sigma(1) = q_{goal} \in \mathcal{X}_{free}$. This collision-free path $\sigma(t)$ should contains a sequence of configurations that navigates the robot from q_{start} to q_{goal} with all the edges lies entirely in $\mathcal{X}_{free}$ [64][65].

2.4.2 RRT Algorithms

In terms of mobile robots, Rapid Exploring Random Tree (RRT) [64] methods for path planning are extremely well researched [66][67][68][69][70][71][72][73][74] and are classified as sampling-based planners. First of all, the script of the original RRT algorithm pseudo code that links the starting configuration q_{start} and the goal configuration q_{goal} is presented in Algorithm 1:

Algorithm 1 RRT

```
1:  procedure RRT(q_start, q_goal)
2:      isGoal ← false
3:      reached ← false
4:      TREE.INSERT(q_start)                    ▷ Start poses are assigned NULL as parent.
5:      while true do                                                    ▷ Main loop.
6:          [q_rand, isGoal] ← RANDCONFIG(𝒳, q_goal)
7:          q_nearest ← NEAREST(TREE, q_rand)
8:          [q_new, reached] ← EXTEND(q_nearest, q_rand, ϵ_inc, ϵ_reach)
9:          if ISVALID(q_new) then
10:             TREE.APPEND(q_nearest, q_new)
11:             if isGoal ∧ reached then
12:                 Break
13:             end if
14:         end if
15:     end while
16:     Return σ ← PATH(q_new, TREE)
17: end procedure
```

In Algorithm 1, we can see that RRT plans paths by iteratively performing the following procedures after initializing the RRT tree with q_{start}:

- **Line 6**: Randomly sample a robot configuration q_{rand} from the configuration space $\mathcal{X}$. If the configuration sampled happens to be the same as q_{goal}, isGoal is assigned as `true`.
- **Line 7**: With the sampled configuration q_{rand}, find the nearest neighbour $q_{nearest}$ from the tree using a distance metric which is typically the square norm.
- **Line 8**: Try to extend the tree from $q_{nearest}$ to q_{rand} with a length increment of ϵ_{inc}, this process continues until the length extended from $q_{nearest}$ exceeds the maximum edge length ϵ_{reach}. the resultant configuration is then assigned as q_{new}. If q_{rand} can be reached within the length ϵ_{reach}, the variable reached is assigned as `true`. Since obstacles can exist between the two points q_{rand} and $q_{nearest}$, every extended configuration will be checked if it is in the space $\mathcal{X}_{free}$ using the function IsValid(), if the extended configuration is not valid, the previous configuration is restored and assigned as q_{new}.
- **Line 9**: Check if q_{new} is in $\mathcal{X}_{free}$.
- **Line 10**: Assign $q_{nearest}$ as the parent of q_{new} and append q_{new} to the tree

- **Line 11 - 13**: Since isGoal means that the configuration q_{rand} generated in Line 6 is q_{goal} and reached means that q_{rand} is reached within the length ϵ_{reach}, getting both of these variable `true` simutaneously means that a path linking q_{start} and q_{goal} is found and the configuration representing q_{goal} is q_{new} at the moment. So the iterative procedure can be terminated.

- **Line 16**: Trace back from q_{new} to q_{start} and return the path σ.

With the procedure of RRT presented, Figure 2.12 shows an illustration of a resultant RRT tree in a space with obstacles and the path found.

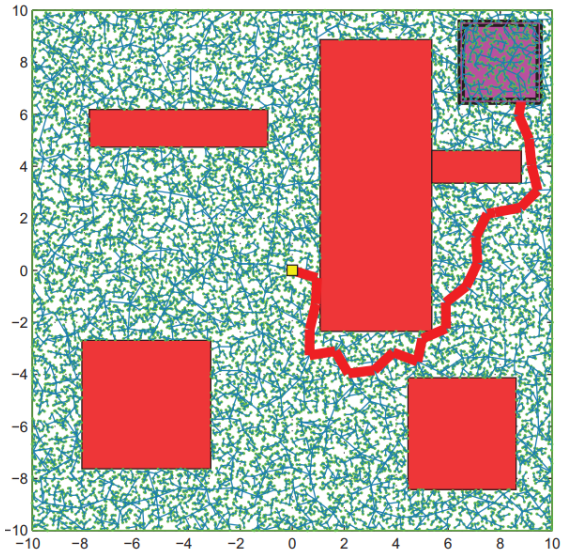


Figure 2.12: RRT searching a path linking the starting point (yellow dot) to the goal point (purple square) with the exploring tree indicated in green edges [75]

We can see from Figure 2.12 that the tree generated by the RRT algorithm explores the space in great detail, a path with cost = 21.02 indicated in red is found linking the starting point to the goal point. However, the path jiggles a lot and there are rooms to make it shorter.

In this context, an optimal version of RRT named RRT* [75] has a better capacity in finding a more optimal path. The pseudo-code of the RRT* algorithm is shown in Algorithm 2.

Algorithm 2 RRT*

1: **procedure** RRT*(q_{start}, q_{goal})
2: isGoal ← `false`
3: reached ← `false`
4: TREE.INSERT(q_{start}) ▷ Start poses are assigned $NULL$ as parent.
5: **while** `true` **do** ▷ Main loop.
6: $[q_{rand}, isGoal]$ ← RANDCONFIG($\mathcal{X}$, q_{goal})
7: $q_{nearest}$ ← NEAREST(TREE, q_{rand})
8: $[q_{new}, reached]$ ← EXTEND($q_{nearest}, q_{rand}$, ϵ_{inc}, ϵ_{reach})
9: **if** ISVALID(q_{new}) **then**
10: q_{neigh} ← NEAR(TREE, q_{new}, r)
11: q_{min} ← CHOSEPARENT($q_{new}, q_{nearest}, q_{neigh}$)
12: TREE.APPEND(q_{min}, q_{new})
13: REWIRE(TREE, q_{new})
14: **if** isGoal ∧ reached **then**
15: **Break**
16: **end if**
17: **end if**
18: **end while**
19: **Return** σ ← PATH(q_{new}, TREE)
20: **end procedure**

Comparing the RRT* algorithm shown in Algorithm 2 and the RRT algorithm shown in Algorithm 1, we can see that rather than directly appending a valid q_{new} to the tree after Line 8, the following procedures are introduced:

- **Line 10**: After verifying that q_{new} is in $\mathcal{X}_{free}$, a set of configurations q_{neigh} in the RRT* tree which are located within a radius r is identified.
- **Line 11**: Find a point $q_{min} = q \in q_{neigh}$ such that the cost of the path connecting from $q_{start} \longrightarrow q_{min} \longrightarrow q_{new}$ is minimized. I.e.: $q_{min} = \arg\min_{q \in q_{nearest} \cup q_{neigh}} cost(q, q_{new})$
- **Line 12**: Assign q_{min} as the parent of q_{new}.
- **Line 13**: Configurations in q_{neigh} except q_{min} are examined to see if the path $q_{start} \longrightarrow q_{new} \longrightarrow q_{neigh}$ has a less cost. Their parents will change to q_{new} if that is the case.

Again, with the above algorithm, a result of RRT* path planning is shown in Figure 2.13.

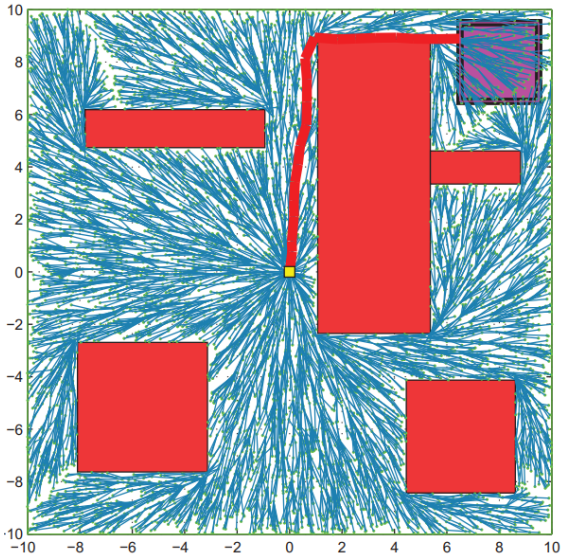


Figure 2.13: RRT* searching a path linking the starting point (yellow dot) to the goal point (purple square) with the exploring tree indicated in blue edges [75]

Figure 2.13 shows that the RRT* tree has also explored the space very well. Compared to the original RRT, the paths extended from the starting point are straighter than that of RRT. This is due to the rewiring process which keeps finding the path with minimum cost for a neighbourhood in every iteration. As we can observe, the resultant path highlighted in red is also straighter than that in RRT, which is more optimal and has a lower cost = 14.51.

The RRT class of planners usually gives suboptimal but fast solutions [76][75], meaning that this class of algorithms are probabilistic complete such that finding a feasible path is very likely if sufficient time is given [66]. This is the case because any new state sampled in the environment will be connected to the existing states in the RRT tree. Moreover, RRT* will provide the optimal path if infinity time is given or infinity number of nodes are allowed to explore, which is also known as asymptotically optimal [75]. Although this is practically impossible, this allows us to control the optimality of the resultant path at the expense of runtime.

In terms of obstacles, since we are dealing with obstacles that can be located anywhere within the space $\mathcal{X}$, they can be organized into configurations like narrow corridors and 'bug traps'. These kinds of configurations are proved challenging for mono-directional sampling-based algorithms. In the case of 3D printing, since there can have many layers of work happening together within the construction site, these obstacle configuration is possible to happen. Moreover, since we would like to decompose a large task into sections, we can treat the start point and the endpoint of each section as the origins of the path exploring trees. In this regard, bi-directional RRT* is also explored.

Bi-directional RRT (B-RRT*) [77][78], as the name suggested, means two trees exploring in both directions. Comparing B-RRT* with RRT* there are some extra steps required to enable the growth of both trees and connections between them. Algorithm 3 [78] shows the pseudo-code of B-RRT*, note that the TREE in Algorithm 2 is now TREE_FORWARD for clarity.

Algorithm 3 B-RRT*

1: **procedure** B-RRT*(q_{start}, q_{goal})
2: $cost_{best} \leftarrow \infty$ ▷ Cost of the best path found.
3: $\sigma_{best} \leftarrow$ `NULL` ▷ Array storing the sequence of configurations of the best path found.
4: TREE_FORWARD.INSERT(q_{start}) ▷ Start poses are assigned $NULL$ as parent.
5: TREE_BACKWARD.INSERT(q_{goal})
6: **for** $i = 1$ to N **do** ▷ Main loop.
7: $q_{rand} \leftarrow$ RANDCONFIG($\mathcal{X}$, q_{goal})
8: $q_{nearest} \leftarrow$ NEAREST(TREE_FORWARD, q_{rand})
9: $q_{new} \leftarrow$ EXTEND($q_{nearest}, q_{rand}$, ϵ_{inc}, ϵ_{reach})
10: **if** ISVALID(q_{new}) **then**
11: $q_{neigh} \leftarrow$ NEAR(TREE_FORWARD, q_{new}, r)
12: $q_{min} \leftarrow$ CHOSEPARENT($q_{new}, q_{nearest}, q_{neigh}$)
13: TREE_FORWARD.APPEND(q_{min}, q_{new})
14: REWIRE(TREE_FORWARD, q_{new})
15: $q_{connect} \leftarrow$ NEAREST(TREE_BACKWARD, q_{new})
16: $[\sigma_{sol}, cost_{sol}] \leftarrow$ CONNECT(TREE_BACKWARD, $q_{new}, q_{connect}$)
17: **if** $cost_{sol} < cost_{best}$ and $\sigma_{sol} \neq$ `NULL` **then**
18: $cost_{best} \leftarrow cost_{sol}$
19: $\sigma_{best} \leftarrow \sigma_{sol}$
20: **end if**
21: SWAPTREES(TREE_FORWARD, TREE_BACKWARD)
22: **end if**
23: **end for**
24: **Return** $\sigma \leftarrow$ PATH(σ_{best}, TREE_FORWARD, TREE_BACKWARD)
25: **end procedure**

From Algorithm 3, the first thing we can notice is that we now have two trees, the forward growing tree TREE_FORWARD and the backward growing tree TREE_BACKWARD initialized with q_{start} and q_{goal} respectively. The second thing is instead of checking the two flags isGoal and reached for termination as in RRT and RRT*, we now have a $cost_{best}$ and σ_{best}. We utilize these two variables to get the lowest cost path that we can have within N iterations. This is because we are now growing trees from both the start point and the goal point, so the goal point has already been "reached" from the beginning. Along with this modification, the following procedures are added compared to RRT*:

- **Line 15**: After rewiring, find the nearest neighbour $q_{connect}$ of q_{new} in the backward tree.
- **Line 16**: Try to connect q_{new} to $q_{connect}$, this involves several sub-steps:
  - Try to extend an edge from q_{new} to $q_{connect}$ with collision checking, the extended point is assigned as x_{new}.
  - Find a set of neighbours x_{neigh} around x_{new} in the backward growing tree.

  - Try to extend an edge from q_{new} to every point $x \in x_{neigh}$ and return the lowest cost path σ_{sol} linking $q_{start} \longrightarrow q_{new} \longrightarrow x \longrightarrow q_{goal}$ and its corresponding cost $cost_{sol}$. This function will return $\sigma_{sol} =$ `NULL` and $cost_{sol} = \infty$ if no path is found.

- **Line 17 - 20**: If a path with a lower cost $cost_{sol}$ is found, the cost and its corresponding path will be stored as $cost_{best}$ and σ_{best} respectively.
- **Line 21**: Swap the forward growing tree and backward growing tree so that the algorithm will grow the backward growing tree in the next iteration.
- **Line 24**: With σ_{best}, trace back from q_{goal} to q_{start} and return the path found as σ.

With B-RRT* illustrated, Figure 2.14[2]shows an example of B-RRT* tree exploring in an environment along with an RRT* version for comparison.

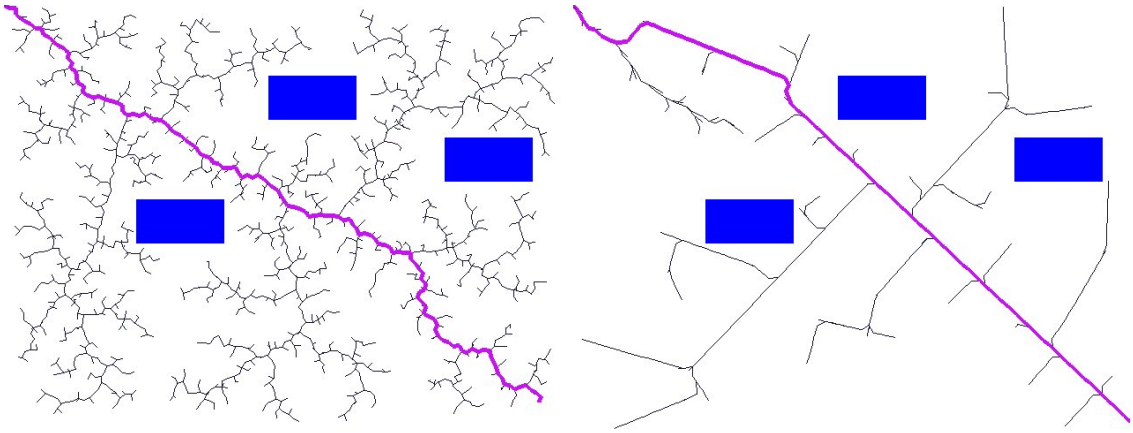


Figure 2.14: RRT* (left) and B-RRT* (right) searching a path highlighted in purple linking two point, one at the upper left corner, another one at the lower right corner in the same environment, the exploring tree is indicated in black edges

Figure 2.14 shows RRT* and B-RRT* trees growing at the upper left corner and the lower right corner. The author's program terminates whenever a path is found. Comparing with RRT*, we can notice that B-RRT* can find a path linking the start point and goal point with significantly less exploration. The reason for this is due to the fact that two trees are growing from both the start point and the endpoint in an alternating manner, so the biggest issue for B-RRT* is to find a point in each tree that is feasible for connecting the two trees together. This is a relatively easier problem comparing to that RRT* need to solve, which is to explore and find the exact goal point in the space $\mathcal{X}_{free}$. In terms of B-RRT*, both trees have more and more endpoints as it grows. Therefore, it is much easier for B-RRT* to find a path because there can be many endpoints on both trees for B-RRT* to connect. However, B-RRT* can return a path that has a significantly larger cost since the connection can happen on points that are very far away from the optimal path. Figure 2.15[3] shows a B-RRT* result that gives a path that has a cost far from optimal. Although RRT* can also result in suboptimal paths, the overall costs are usually less than that in B-RRT* on average. The reason for this is because RRT* needs to explore the space more to find

[2]Source: https://github.com/vvrs/RRT_Algorithms

the goal point, so before it locates the goal point the space is usually better explored compared to B-RRT*. Therefore, it has a higher chance to give a lower cost path. In this regard, it seems that allowing the B-RRT* algorithm to run for more iterations or defining termination criteria is a way to encourage B-RRT* to find a more optimal path.

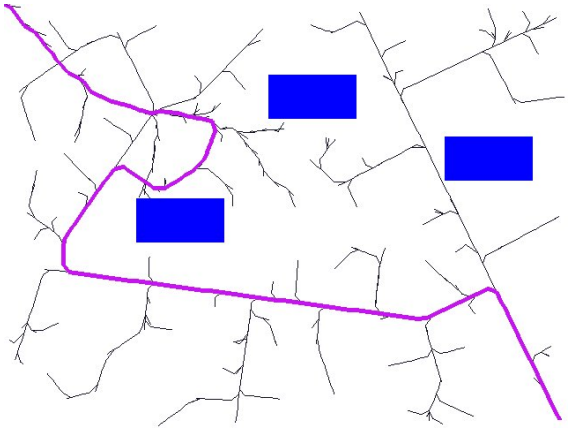


Figure 2.15: Result of B-RRT* exploring in the same environment as Figure 2.14 with a path found (highlighted in purple) with significantly larger cost / longer distance

Concerning optimality, Qureshi et. al [78] has provided a comprehensive comparison between RRT* and B-RRT* for finding the optimal paths. Their results show that for all the five environments they have tested on, B-RRT* can always find the optimal path in a fewer number of iterations and in less time with less memory usage. To further give an idea of the efficiency of finding the optimal path, they have modelled an optimal path convergence rate (CR) defined as:

$$CR = \frac{c(\sigma_{init}) - c(\sigma^*)}{t_{init} - t^*} \tag{2.1}$$

where $c(\cdot)$ is the cost function in terms of Euclidean distance, σ_{init} and t_{init} is the initial path found and the corresponding time spent, σ^* and t^* is the optimal path and the time used to find the optimal path respectively. Using (2.1), RRT* has a mean CR around 0.8 unit/s while B-RRT* has a mean CR around 1.9 unit/s [78]. This indicates that although B-RRT* may not be able to find the optimal path in the first place, it is still significantly faster than RRT* to arrive an optimal solution.

In general, we can see that RRTs are very well researched and explored algorithms for mobile robots. Together with its low computation cost and ease of implementation, using RRTs for mobile robot 3D printing is a suitable approach. However, RRT search paths in a naturally greedy manner because any new node will only be linked to its nearest node that existed in the RRT tree. Therefore, paths generated by RRT are sub-optimal. Albeit RRT* and B-RRT* has a step that relinks nodes to guarantee that each node arrived with a minimum cost path [75], this step consumes extra computational power which can be avoidable if we utilize dynamic programming to find the best node to connect. Concerning this, Fast March Tree is explored in Section 2.4.3.

[3]Source: https://github.com/vvrs/RRT_Algorithms

2.4.3 FMT* Algorithm

Fast March Tree (FMT*) was introduced by Janson et. al [65] and is a fairly new method for path planning. The pseudo-code of FMT* is first introduced in Algorithm 4. Then, the features and advantages will be discussed.

Algorithm 4 FMT*

1: **procedure** FMT*(q_{start}, q_{goal})
2: $V \leftarrow q_{start} \cup$ SAMPLE($\mathcal{X}$, n)
3: $V_{unvisited} \leftarrow V \setminus q_{start}$
4: $V_{open} \leftarrow q_{start}$
5: $V_{closed} \leftarrow$ `NULL`
6: TREE $\leftarrow$ `NULL`
7: $z \leftarrow q_{start}$
8: **while** `true` **do**
9: $V_{open,new} \leftarrow$ `NULL`
10: $N_z \leftarrow$ NEAR($V \setminus z$, z, r_n)
11: $X_{near} \leftarrow N_z \cap V_{unvisited}$
12: **for** $x \in X_{near}$ **do**
13: $N_x \leftarrow$ NEAR($V \setminus x$, x, r_n)
14: $Y_{near} \leftarrow N_x \cap V_{open}$
15: $y_{min} \leftarrow \arg\min_{y \in Y_{near}} [cost(y) + cost(y, x)]$
16: **if** ISVALID(y_{min}, x) **then**
17: TREE.INSERTEDGE(y_{min}, x)
18: $V_{open,new} \leftarrow V_{open,new} \cup x$
19: $V_{unvisited} \leftarrow V_{unvisited} \setminus x$
20: $cost(x) = cost(y_{min}) + cost(y_{min}, x)$
21: **end if**
22: **end for**
23: $V_{open} \leftarrow (V_{open} \cup V_{open,new}) \setminus z$
24: $V_{closed} \leftarrow V_{closed} \cup z$
25: $z \leftarrow \arg\min_{y \in V_{open}} cost(y)$
26: **if** $z \neq q_{goal}$ or $V_{open} \neq$ `NULL` **then**
27: **Break**
28: **end if**
29: **end while**
30: **Return** $\sigma \leftarrow$ PATH(z, TREE)
31: **end procedure**

From Algorithm 4, we observed that FMT* consists of the following steps:

- **Line 2**: The algorithm start by statically sampling n points over the space $\mathcal{X}$ and store them in a set V along with q_{start}.
- **Line 3 - 6**: The set V is then partitioned into 3 subsets:

  - $V_{unvisited}$ as the set of unvisited points that have not been added to the tree.
  - V_{open} as the set of visited points that have been added to the tree with their valid, collision-free cost-to-arrive calculated and being further considered for connections with the points in $V_{unvisited}$.
  - V_{closed} as the set of points that have been added to the tree and will not be further considered for new connections.

- **Line 9**: Create a temporary set $V_{open,new}$ to store points $x \in X_{near}$ that found connected successfully and will be added to V_{open} after all $x \in X_{near}$ are considered.
- **Line 10 - 11**: Find a set of neighbours X_{near} that belongs to the set $V_{unvisited}$ within a specified radius r_n.
- **Line 12 - 15**: For every point $x \in X_{near}$, find another set of neighbours belongs to V_{open} within the radius r_n and identify the point y_{min} which has the lowest cumulative cost from $q_{start} \longrightarrow y_{min} \longrightarrow x$.
- **Line 16 - 21**: If the edge y_{min}, x is collision free, insert the edge to the tree, add point x to $V_{open,new}$ and remove it from $V_{unvisited}$. Calculate the cumulative cost for the point x.
- **Line 23 - 24**: Add $V_{open,new}$ to the set V_{open}, z is excluded from V_{open} because it has been explored at this stage. For the same reason, z is added to V_{closed}.
- **Line 25**: Update z as the point with the minimum cost in the set V_{open}.
- **Line 26 - 28**: Terminate while loop if goal is reached or V_{open} is empty.
- **Line 30**: Return the final path σ by tracing back from q_{goal} to q_{start} using z.

From the above, we can see that FMT* consist of three major features [65]:

1. Static sampling of the space $\mathcal{X}$ can connect neighbours using disk-connected graphs, meaning that two vertices can be connected if the distance between them is below a specified value r_n.
2. Graph search and graph construction happens at the same time.
3. Dynamic programming recursion is used with "lazy" cost determination. It is "lazy" in the sense that the cost calculation will ignore the presence of obstacles. Whenever an edge connecting the vertices intersects obstacles, the algorithm will skip the sample being considered and leave it for later to search for other connections.

Having layout the pseudo code and features, Figure 2.16 presents the tree of RRT* and FMT* for comparison.

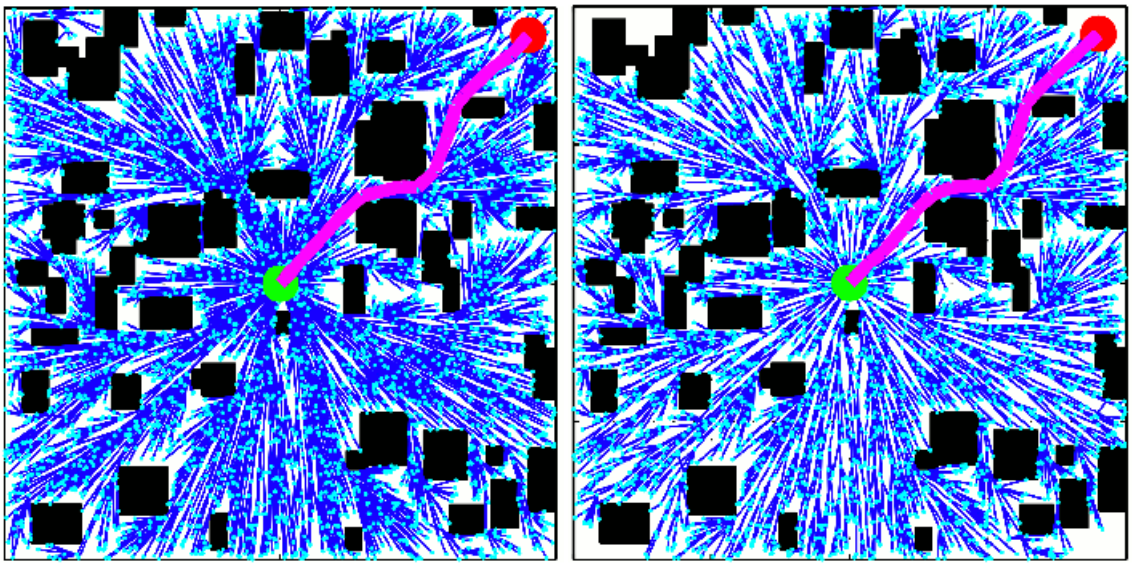


Figure 2.16: RRT* (left) and FMT* (right) searching a path highlighted in purple starting from the green region to the red region in the same environment. The cyan dots and blue edges are the vertices and edges of the exploring trees respectively. [79]

From Figure 2.16 it can be observed that FMT* is similar to RRT* in the sense that it will grow a tree of trajectories very much like that in RRT*. This is because FMT* searches and constructs the path at the same time to form a dynamic tree structure. However, they are yet very different procedures. While FMT* statically sampled point in the space and connect vertices in the order of drawing of the samples to expand outwards, RRT* greedily extend to a new sample from points of the tree reached at that time. When RRT* comes to cases where the solution path is required to be long and winding, a significant amount of time is wasted by attempting to extend the tree in unreachable regions before a valid point is found. In this case, FMT* is more efficient since all the points in the space have already been sampled. So the number of trial and error procedures is hugely reduced by just considering the neighbours around a certain point within a specified radius r_n. This shows an advantage of FMT* over RRT*. In [65], simulations have also been carried out and showed that FMT* can always reach a feasible solution faster than RRT*, regardless of having obstacles or not.

In terms of optimality, Janson et. al [65] proved that FMT* has a probabilistic convergence rate upper bound $\mathcal{O}\left(n^{-\frac{1}{d+\rho}}\right)$ in obstacle-free scenarios, where n is the number of points sampled, d is the dimension of the configuration space $\mathcal{X}$ and ρ is a constant which is arbitrarily small. This implies that as $n \longrightarrow \infty$, the probability of suboptimal connections will approach zero. For comparison, LaValle et. al [80] has proved that RRT has an exponential convergence rate bound, so FMT* actually converges to optimal solutions at a slower pace since it is not exponential. Moreover, the bound stated above is with the condition that the radius for searching neighbours

r_n is defined by:

$$r_n = 2(1+\eta)\left(\frac{1}{d}\right)^{\frac{1}{d}}\left(\frac{\mu(\mathcal{X}_{free})}{\zeta_d}\right)^{\frac{1}{d}}\left(\frac{\log(n)}{n}\right)^{\frac{1}{d}} \tag{2.2}$$

where d is the dimension of the space $\mathcal{X}_{free}$, $\mu(\mathcal{X}_{free})$ is the volume of the space $\mathcal{X}_{free}$, ζ_d is the volume of the d-dimension unit sphere, n is the number of samples and η is a positive value [65]. Since $\eta > 0$, it implies that we may not able to cherry-pick a value for the radius. In this study, the number of samples and the radius will be chosen based on experimental results and will be further discussed in Chapter 4.

Having discussed RRTs and FMT*, we now have the tool to navigate robots from a start point to a goal point. To sum up, the path planning algorithms discussed have their edges:

- RRT*: An optimal version of the original RRT.
- B-RRT*: Better handling difficult obstacle configurations like narrow gaps and "bug trap", faster to arrive at a feasible solution than RRT* but can return paths that are far from optimal.
- FMT*: Faster in the sense that it will not waste time attempting to extend the tree in unreachable regions before a valid point is found.

Therefore, it is worthwhile to explore and compare RRT* and FMT* along with task decomposition. This will be further discussed in Chapter 3.

Nonetheless, we can observe that all path planning algorithm tries to explore the whole space given. This can be time-consuming and not efficient since we may need to explore a large space to get a path. Concerning practicality, since infinite time is not given, there will be chances that the solutions returned from the path planning algorithms are far from optimal. Furthermore, in addition, to navigate the robots from the start to the goal, we would also like to have the paths close enough to the printing tasks such that the robot can reach the printing locations in the context of 3D printing. To facilitate the path planning algorithms so that they can be oriented toward printing tasks, task consistent path planning is introduced in Section 2.5.

2.5 Task Consistent Path Planning

We can see that path planning helps us to find paths that link the starting points and the goals, but whether the robots can use the resultant paths to work on their tasks is not guaranteed. To make sure the path planner can provide paths such that the robot can operate on the task, task consistent planning is required. In this section, we will first introduce the idea of controlling the shape of the resultant path in path planning algorithms. Then we will introduce the constraints raised from mobile manipulators and their relations with a printing task. After that, a set of strategies that encourages the constrained path planning algorithms to sample towards to goal point will be given.

2.5.1 Adding Constraints to Path Planning Algorithms

From the perspective of path planning, we would like to minimize the cost of the path navigating a robot from the start point to the endpoint. Gammell et. al [79] has achieved this minimization by enclosing the two points into an ellipse sampling domain which restrict the path planning algorithm to sample points in the region only as shown in Figure 2.17. When there are no obstacles, the ellipse degenerates to a line and the path planned becomes a straight line running from the start point to the endpoint, which is optimal in this case.

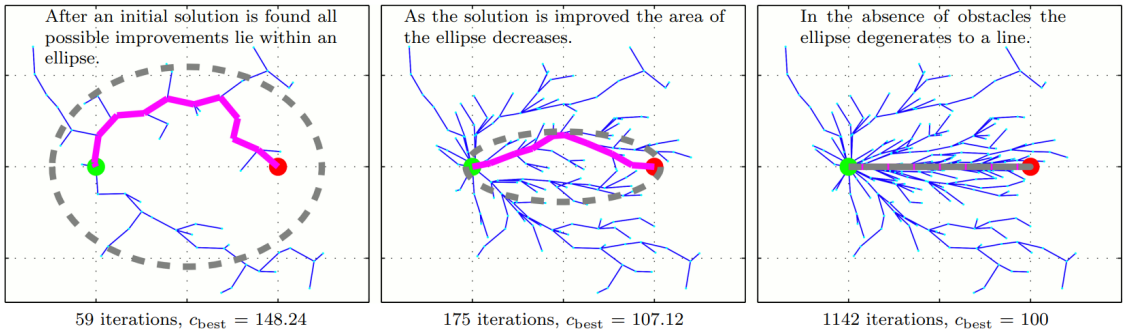


Figure 2.17: RRT* informed with a contracting ellipse sampling domain, planning a path from the start point in green to the goal point in red at its 59^{th}, 175^{th} and 1142^{th} iterations [79]

From [79], we can see that the shape of the path can be controlled by limiting the sampling space of the path planning algorithm. In the context of mobile 3D printing, a path should be generated around the printing task. With the mentioned idea, to limit the sampling space so that a path can generate in such a way, defining a region that encloses the printing task is a sensible thing to do. An example of this is presented in Figure 2.18. A robot is drawn to illustrate that the base pose sampled should only be in the sampling region.

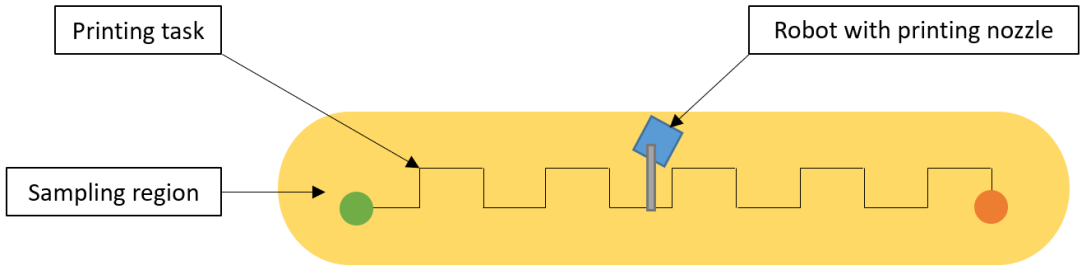


Figure 2.18: Example of a printing task starting from the green region and ends at the orange region with a sampling region (yellow) that path planning algorithm should sample base pose from

To define this region, since the mobile robot needs to extend its manipulator to work on the task, we would like to consider how far away the mobile robot can be from the printing task. With this regard, research that defines workable regions of mobile manipulators is investigated.

2.5.2 Constraints of Mobile Manipulators and Task Consistency

Literature shows there were research adopting this idea to achieve specific tasks, some of them are [81][82][83]. In particular, Oriolo et. al[83] introduced the compatible platform region which is a region that contains robot base poses $q \in \mathbb{R}^3$ that are considered to be compatible with the constraint of the end-effector pose $p \in \mathbb{R}^3$ such that an inverse kinematic solution for q is very likely to exist. It is 'very likely' because whether the solution exists also depends on the orientation of q and the vertical component of p. If q has an orientation that keeps the end-effector from reaching p_i, there is no inverse kinematics solution for this base pose q.

Since for an end-effector pose p_i, where $i = 0, ..., s$ represents the progress of the task, the furthest point that the base pose can be located is when the arm of the manipulator is fully extended (in singularity), the compatible platform region is constructed by drawing a circle centred at an end-effector pose p_i with the arm of the robot fully extended and passing through the robot base centre as shown in Figure 2.19.

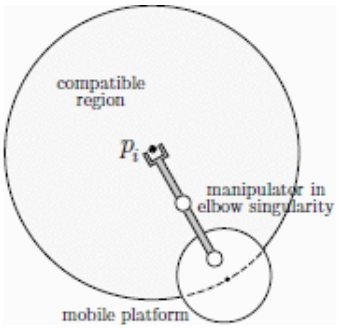


Figure 2.19: Compatible platform region [83]

Oriolo et. al use this region to sample base pose in their RRT-like path planner to get collision-free paths $p(i) = [p_0, ..., p_i, ..., p_s]$ for the end-effector. For every end-effector pose p_i, the existence of an inverse kinematic solution is subsequently validated after sampling the base pose q_i. From this, we can see that since the sampling of poses in the path planner is restricted by the compatible platform region for every progress i, the end-effector can arrive $p_i \forall i$ as illustrated in Figure 2.20. If we overlap all the compatible platform regions from progress $i = 0, ..., s$, we will resemble a region similar to that in Figure 2.18, and the path generated from the planning algorithm grows around the task. Therefore, the path generated is task consistent. This is also shown in Figure 2.20 by black dots linked with black edges.

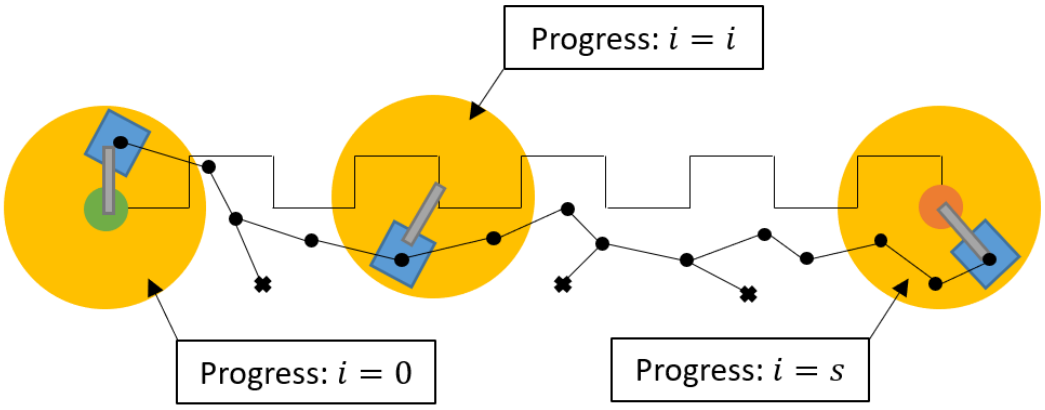


Figure 2.20: Path planned for the task displayed in Figure 2.18 with compatible platform region (in yellow) at progress $i = 0$, $i = i$ and $i = s$

With the path planning algorithm and the compatible platform region, we can plan paths that navigate a mobile manipulator from a start point to an endpoint. However, since the path planning algorithms we have mentioned are sampling-based, there may still be cases that the path planned grows backwards along with the task. This can ruin existing printings because the robot can bump into printed materials. In this regard, a set of strategies that helps the path planning algorithm to plan a path that grows along with the progress of the task will be useful. Šustarevas et. al [84] has come up with such a set of strategies to encourage the path planning algorithm to always explore towards to goal. This set of strategies is as follows:

1. Coupling the points in the robot configuration space $\mathcal{X}$ with the progress variable $s \in [0, s_f]$. To illustrate, if the base pose is $q_i = [x_{qi}, y_{qi}, \theta_{qi}] \in SE(2)$ for an end-effector pose $p_i = [x_{pi}, y_{pi}, \theta_{pi}] \in SE(2)$, then the coupled version will become $\breve{q} = [x_{qi}, y_{qi}, \theta_{qi}, s]$. With this, the path planning algorithms can keep track of the progress of the task.

2. Modifying the distance metric used to find neighbours $d(\cdot, \cdot)$ to become progress aware to avoid the path from growing backwards along with the task. This is important since printing backwards can ruin existing printings. Let q_1, q_2 be the points in the configuration space, the progress aware metric is as follows:

$$\hat{d}(\breve{q}_1, \breve{q}_2) = \begin{cases} d(q_1, q_2) & \text{if } s_2 > s_1 \\ \infty & \text{otherwise} \end{cases} \tag{2.3}$$

   Since finding the nearest neighbour for a pose q_i means finding the minimum distance of $d(\cdot, \cdot)$ between q_i and all other poses, forcing the distance of the neighbours with smaller progress s to ∞ will filter them out for the consideration of being the nearest neighbour. So, the nearest neighbour found using (2.3) will return a neighbour that has a progress value larger than that of q_i. This pushes the path planning algorithms to always look for poses with increasing processes towards the goal.

3. Sampling poses from a map similar to the compatible platform region.

4. With the awareness that some path planning algorithm will extend the path planning tree up to a specified threshold for each connection (like ϵ_{reach} in Algorithm 2), the sampled base poses and the extended poses are checked for collisions and their presence in the compatible platform region like map. This ensures the poses added to the tree are all collision-free, with valid inverse kinematic solutions and with orientations at most θ_{thres} pointing away from the task.

With the above, task consistency in mobile 3D printing can be demonstrated. In their paper, RRT* was used to demonstrate the results. Concerning Algorithm 2 (RRT*), the above strategies were fused into the algorithm by redefining the functions below:

Algorithm 5 Redefined functions for task consistent RRT* (TCRRT*)

procedure isValid(q)
    valid $\leftarrow$ isNotInCollisionWithEnvironment(q)
    valid $\leftarrow$ valid $\wedge$ isNotInCollisionWithTask(q, task($[0, q_s]$))
    valid $\leftarrow$ valid $\wedge$ inMap(q)
    Return valid
end procedure

procedure RandConfig(s_{max})
    if rand( ) $< \beta_{goal}$ **then**
        $s_{rand} \leftarrow s_f$
        isGoal $\leftarrow$ `true`
    else
        $s_{var} \leftarrow 0.1 s_f$ ▷ Standard deviation
        $s_{rand} \leftarrow \max(0, \min(s_f, s \sim \mathcal{N}(s_{max}, s_{var})))$
        isGoal $\leftarrow$ `false`
    end if
    $[x, y, \theta] \leftarrow$ sampleFromMap($p(s_{rand})$)
    Return $q_{rand} = [x, y, \theta, s_{rand}]$
end procedure

From the above functions, we can see that the isValid() function has been extended to examine if the base poses q is in their map or not, in addition, to check collisions with obstacles in the environment and the evolving printing task (from progress $s = 0$ up till the progress of the base pose $s = q_s$ being checked). For the function RandConfig(), it has been modified in a way that the current furthest progress s_{max}, which is stored and updated in each path planning iteration, is used to sample a value from a Normal distribution. By defining the mean of the distribution as s_{max} and the standard deviation s_{var} as $0.1 s_f$, a progress s_{rand} is sampled from the Normal distribution and clipped between the domain $[0, s_f]$. This progress is then used to identify the corresponding end-effector position and sample a base pose configuration q_{rand} from the corresponding map. In this way, the sampling of base poses will mainly focus around the current furthest progress s_{max}, and will also occasionally sample from a progress that is far behind. This can increase the chances of finding a shorter path and escape from difficult regions since building paths from earlier progress

can result in a sequence of base pose configurations that is easier to get around the aforementioned regions or even leads to a shorter solution. An example of this is illustrated in Figure 2.21.

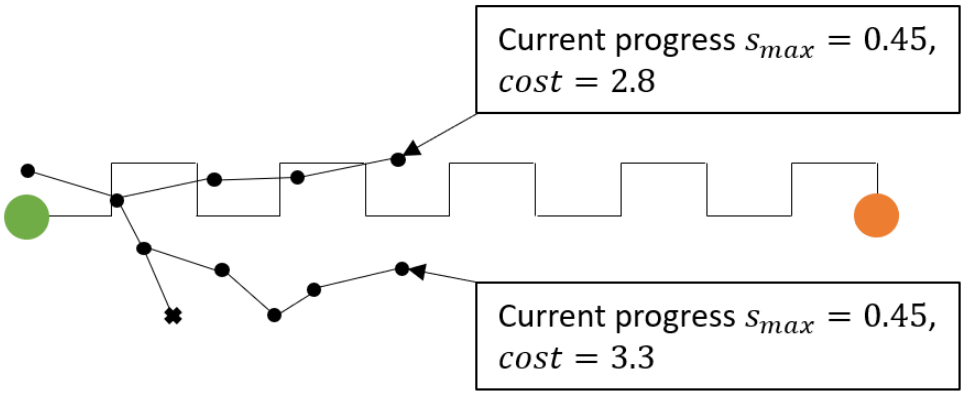


Figure 2.21: An example showing a path planning algorithm exploring a path for the task described in Figure 2.18 at current furthest progress $s_{max} = 0.45$, a path is generated by occasionally sampling from early progress and has a lower cost

From Figure 2.21, we can observe that a shorter path above the task is constructed by branching out from a pose sampled earlier (with a progress $s < s_{max}$). The resultant cost is less than that of the path below the task. With this, the path planning algorithm is informed with the constraint of the 3D printing task and the mobile manipulator. Therefore, a task consistent path planner is produced.

2.6 Summary

In this chapter, stationary and multi-agent mobile robotic construction systems are reviewed, relevant knowledge of path planning and task consistency is introduced, the importance of task decomposition is discussed and the need for an obstacles aware task decomposition is identified. In the next chapter, the methodology and implementation of obstacles aware task decomposition will be discussed.

Chapter 3

Obstacle-Based Decomposed Task Consistent Path Planning

This chapter presents the strategy for obstacle-based multi-agent task consistent path planning. The first part of this chapter will introduce the problem definition of what we are trying to solve. Then the task consistent path planning schemes using RRT* and FMT*, along with the strategy for decomposing construction 3D printing task based on obstacles will be given. Finally, the way of optimizing the path planner operations will be discussed.

3.1 Problem Definition

Let $\mathcal{X} \subseteq \mathbb{R}^3$ be the configuration space that each element of $\mathcal{X}$ is a configuration of the base frame of the mobile robot manipulator. In this space $\mathcal{X}$ we have the fixed obstacle space $\mathcal{X}_{obs} \subset \mathcal{X}$ representing the configurations that the robot collides with the obstacles, the evolving printing task $\mathcal{T}(s) \in SE(3), \subset \mathcal{X}$ which is parametrized by the progress variable $s \in [0, 1]$ and the free space $\mathcal{X}_{free} := \mathcal{X} \setminus \{\mathcal{X}_{obs} \cup \mathcal{T}(s)\}$ representing the configurations that the robot can move freely.

A compatible base region $CR(p(s)) \subset \mathcal{X}$ is a region that contains the robot base poses $q(s)$ at task progress s that are considered to be compatible with the constraint of the end-effector pose $p(s)$ such that an inverse kinematic solution for $q(s)$ is very likely to exist.

With this setup, the problem is divided into two parts:

1. **Obstacle-based Task Decomposition**: With the condition that the end-effector poses $p(s)$ closely following the task $\mathcal{T}(s)$, decompose the task $\mathcal{T}(s)$ into n sections $\mathcal{T}_1(s_1), ..., \mathcal{T}_i(s_i), ..., \mathcal{T}_n(s_n)$ with $s_i \in [s_{si}, s_{gi}]$ where s_{si} and s_{gi} denotes the start and goal progress at section i respectively. For a task $\mathcal{T}_i(s_i)$, the decomposition of task occurs whenever all base poses at the k^{th} progress $q(s^k)$ sampled from $CR(s^k)$ and all base poses at $k + 1^{th}$ progress $q(s^{k+1})$ sampled from $CR(s^{k+1})$ always require an edge E to connect such that $E \cap \mathcal{X}_{obs}$. When this happens, we can define $s^k := s_{gi}$ and $s^{k+1} := s_{s(i+1)}$. This puts $\mathcal{T}_i(s_i)$ to an end and starts a new section $\mathcal{T}_{i+1}(s_{i+1})$.

2. **Task Consistent Path Planning**: For each decomposed task $\mathcal{T}_i(s_i)$, search a collision-free path $\sigma_i(s_i) \in \mathcal{X}_{free} \forall s_i \in [s_{si}, s_{gi}]$ linking an initial configuration $\sigma_i(s_{si}) = q_{start,i} \in \mathcal{X}_{free}$

and a goal configuration $\sigma_i(s_{gi}) = q_{goal,i} \in \mathcal{X}_{free}$. This collision-free path $\sigma_i(s_i)$ should contains a sequence of configurations that navigates the base frame of the robot from $q_{start,i}$ to $q_{goal,i}$ with all the edges lies entirely in $\mathcal{X}_{free}$. This path should progress along the section of printing task $\mathcal{T}_i(s_i)$ so that the robot can deposit materials on it.

3.2 Methodology

3.2.1 Task Consistent Path Planning Schemes

Compatible Base Region

Mentioned in Section 2.5.2, Oriolo et. al [83] use a disk shape region to define the region that the base poses q of a robot for an end-effector pose p can be found very likely. In this study, we will use a doughnut shape for this region as shown in Figure 3.1. A robot in the region is displayed to illustrate that a base pose q can be sampled for the corresponding end-effector pose p.

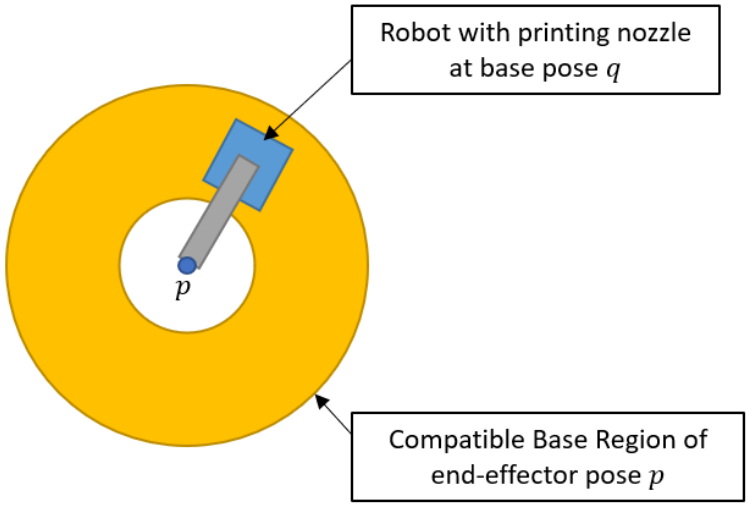


Figure 3.1: Compatible Base Region in donut shape

A doughnut shape is used because the end-effector of the robot has a high chance to collide with the robot itself in the innermost region of the disk shape compatible platform region suggested by Oriolo et. al. Excluding the innermost region can increase the chance of sampling base poses that have valid inverse kinematic solutions. In this study, a compatible base region with inner radius = 0.2m and outer radius = 0.8m is used.

Task Consistent RRT*

Task consistent RRT* (TCRRT*) has been explored by Šustarevas et. al [84] and proved to be functional. In this study, the TCRRT* developed by Šustarevas et. al is implemented and used to compare results with task consistent FMT*. The pseudo-code of the TCRRT* was presented in Algorithm 2 and 5.

Task Consistent FMT*

To extend FMT* described in Section 2.4.3 to become task consistent FMT* (TCFMT*), the following modifications were made:

- First of all, as mentioned in Section 2.5 we need to restrict the region where the algorithm can sample configurations so that task consistency can be achieved. In FMT* we introduced the compatible base region $CR(p(s))$ to the sampling function. In this way, the FMT* algorithm only samples configurations from $CR(p(s))$ for all end-effector poses p from progress $s \in [0, 1]$ with collision checks done in the isValid() function described in Algorithm 5. In this way, the poses sampled are all collision-free and orientated towards the task.
- Throughout the running of the algorithm, the current furthest progress s_{max} is tracked and the algorithm terminating condition is changed from $z = q_{goal}$ to $s_{max} = 1$. I.e. the base pose that helps the robot to print the last bit of the task is found.
- When we are searching for neighbours, we apply modified distance metrics accordingly so that the exploring tree can always grow towards the goal with an increasing progress s. In line 10, 11 of Algorithm 4, we search for poses $x = [x_2, y_2, \theta_2, s_2]$ in $V_{unvisited}$ around $z = [x_1, y_1, \theta_1, s_1]$, these poses should have a larger value of s such that $s_2 > s_1$, so the distance metric (2.3) is used for finding neighbours in $V_{unvisited}$. In line 13 and 14, the algorithm tries to link the poses x found in $V_{unvisited}$ back to a pose $y = [x_3, y_3, \theta_3, s_3]$ in V_{open}. Since y is already connected with the tree it should have its progress $s_3 < s_2$. These two scenarios are illustrated in Figure 3.2. Please note that in the right side of the figure, the node z is renamed to y because it is in the neighbour set Y_{near} as the algorithm is searching neighbours for q.

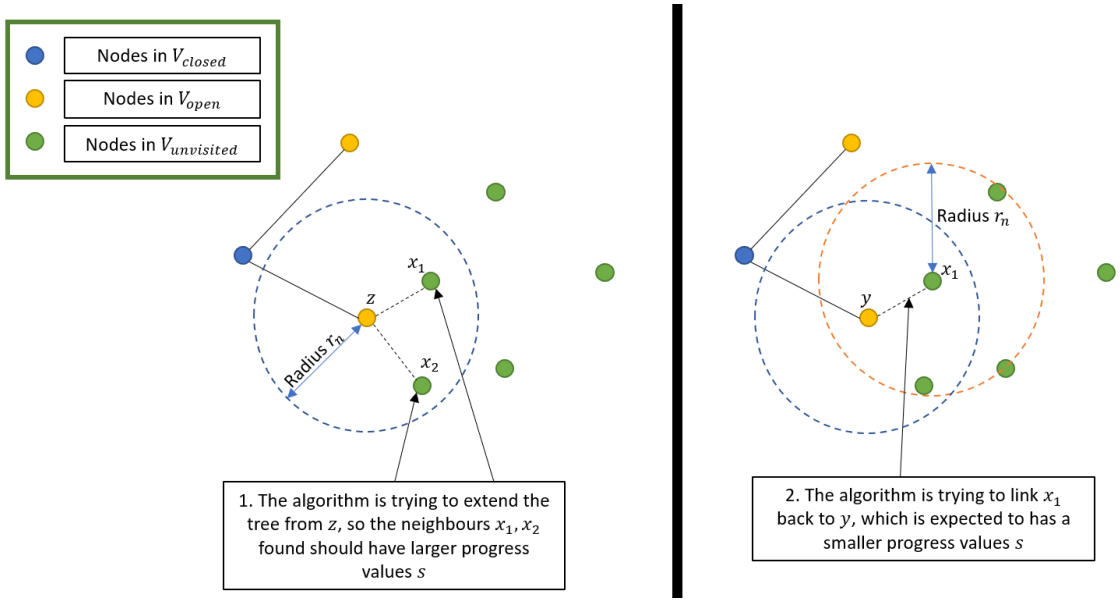


Figure 3.2: Example showing FMT* algorithm finding neighbours in $V_{unvisited}$ (left) and when finding neighbours in V_{open} (right)

In this regard, a new metric (3.1) is used for this search of neighbours:

$$\breve{d}(\breve{q}_1, \breve{q}_2) = \begin{cases} d(q_2, q_3) & \text{if } s_3 < s_2 \\ \infty & \text{otherwise} \end{cases} \tag{3.1}$$

With (3.1), the neighbour searching function will always return neighbours that have a smaller progress value.

3.2.2 Task Decomposition

Task Decomposition with TCFMT*

Recalling that FMT* will skip any poses in $V_{unvisited}$ that those in V_{open} cannot reach (line 16 in Algorithm 4), the set V_{open} will be used up whenever the obstacles arranged in a way that the robot cannot pass through because the algorithm cannot find a way to connect poses on the other side of the obstacles. A scenario of this is illustrated in Figure 3.3.

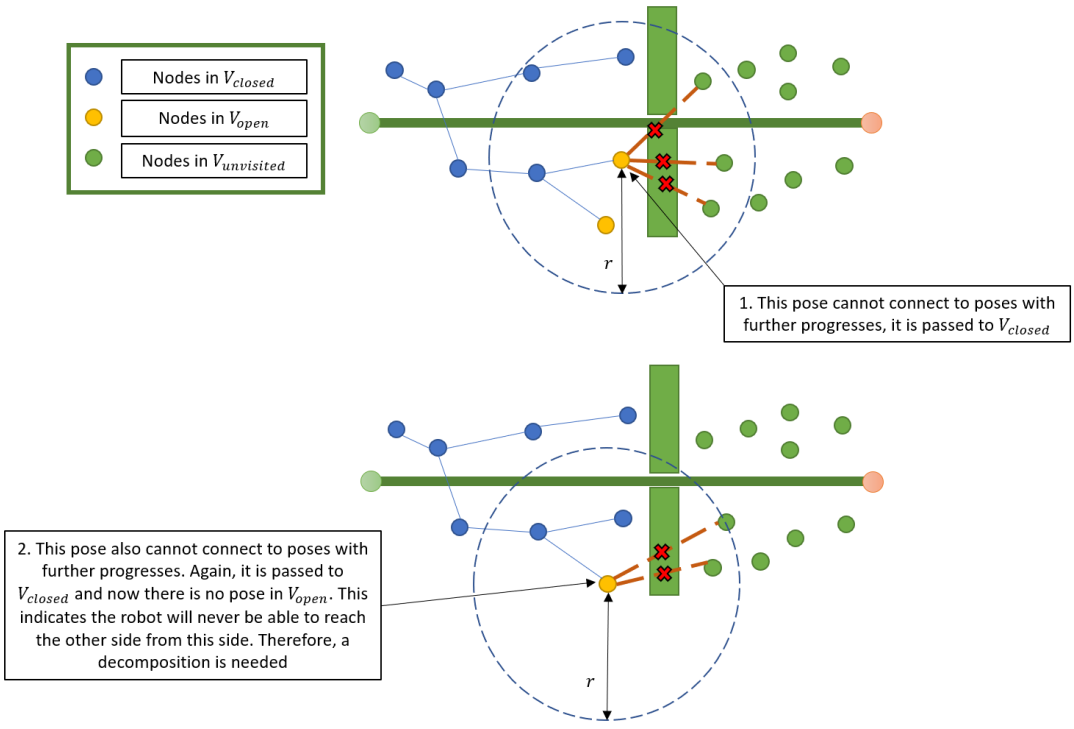


Figure 3.3: Scenario showing no poses in V_{open} can connect to poses with larger progress value s in $V_{unvisited}$, and V_{open} eventually becomes empty. Invalid connections attempting to make are indicated with red crosses and brown dotted lines.

This is exactly the case described in part 1 of the problem definition, the algorithm used all the poses in the connected tree (V_{open}) to try to connect unvisited poses that have further progress but failed because they all require edge that intersects with the obstacles. This phenomenon is leveraged so that when V_{open} is found empty, we go to enquire $V_{unvisited}$ for a configuration that

has a progress further than the current furthest progress. This pose is subsequently stored in V_{open} and assigned to z. This pushes the algorithm to start a new tree $\mathcal{T}$ and continue the planning. With this, the algorithm should be able to decompose the whole printing task $\mathcal{T}(s)$ to n sections $\mathcal{T}_1(s_1), ..., \mathcal{T}_i(s_i), ..., \mathcal{T}_n(s_n)$ when necessary.

Besides, we also observed that even though we have a huge sample of points, a point in it may still fail to find a valid neighbour because of the complex configuration space that we are using, the constraints that we have for task consistency and collision checks. In this regard, the radius r_n is implemented in a way such that it allows an increment $\epsilon_{r,inc}$ until at least one neighbour can be found. The radius is subsequently reset to the original value of r_n after that FMT* iteration. We would like to argue that this has a trivial effect on the quality of the resultant path because of two reasons.

- Firstly, even though a node is connected far away, the algorithm will always explore from the points that have the lowest cost in V_{open} to extend since FMT* operates in a lazy manner. Provided that the radius is large enough, the resultant tree will be well explored and it has a high chance that the path involving such a point will be filtered because that long edge also has a high cost.
- Secondly, this modification also does not affect task decomposition. This is further breakdown into two arguments:
  - Notice that when the tree hit an obstacle, the radius of the search will expand with the setting above until it finds a point on the other side of the obstacle. However, there will still be no connection through the obstacles because of the collision checking.
  - When it comes to places where the tree really cannot get through, the points in V_{open} will still use up because no points can find a valid neighbour, no matter how large is the radius.

With this, we would be interested in minimizing the fluctuation of the changing radius subject to the combination of the specified radius and the number of samples mentioned in Section 2.4.3. This will be further discussed in Section 4.1.1.

We are also aware that path planners may need to plan paths in relatively confined regions. In these cases, TCFMT* may struggle to find a path for the robot to work because the configurations sampled may have very different orientations such that even they appear to be near the searching subject, they are very 'far' away from the perspective of our task consistent distance metric (2.3) and (3.1). To mitigate this issue, we propose the following procedures for TCFMT* pre-sampling for every progress value s:

1. Sample a point and check if it does not collide with the task and is in the compatible base region.
2. If the above is true, check if the point collides with any obstacles.
3. If the point collides with an obstacle, sample again.
4. If a valid point can be found within a self-defined number of trials n_{sample_trials} with the above procedures, add the point to the set V.

5. If a valid point cannot be found in n_{sample_trials} times, still add the latest sampled point to the set V and adaptively sample k more points with procedure 1 - 3 for that progress s.

Please note that adding points that collides with obstacles will not destroy any TCFMT* procedures because of the nature that FMT* will simply ignore any point that has a collision when it is trying to make connections.

With the above procedures, there will be more points sampled around the obstacles. This has two benefits:

1. This provides an abundance of points for TCFMT* to try to connect points with further progress value s. So, if the algorithm is still not able to find a valid connection with these many points, a need for decomposition is more persuasive to declare.

2. If the obstacle congesting region is a narrow pathway, then these additional points will assist TCFMT* to find a valid configuration to connect. This is because the valid points sampled in a narrow pathway are more constrained. To further elaborate, the orientation of the poses may need to be aligned more with the printing task. Connections are more difficult to make in such a scenario because there may not exist a point that has a configuration with a compatible orientation that TCFMT* can connect the existing tree to.

Task Decomposition with TCRRT*

As mentioned in Section 2.3, it is demonstrated in [57] that a 3D printing task can be decomposed into sections of task by a cut-off threshold sampled from a Normal distribution. However, obstacles have not been taken into account. Since another major source of concern in the case with obstacles is that the obstacles block the robot(s) from printing, our algorithm will find possible points for decomposition by targeting obstacle congesting areas that have printing tasks nearby. A high-level pseudocode description for the whole decomposition process is first presented in Algorithm 6.

Algorithm 6 Task Decomposition

```
1: procedure TaskDecomposition(task_ROI_opening_angle)
2:     edges ← GetInvolvedEdges( )
3:     checklines ← GetChecklines(edges)
4:     roughbreakpoints ← GetRoughBreakPoints(checklines)
5:     obsbreakpoints ← MergeWithIRM(roughbreakpoints)
6:     breakpoints ← AscendingSort(obsbreakpoints)
7:     Return breakpoints
8: end procedure
```

We will now look at the functions one by one. To start with, the function GetInvolvedEdges() is listed in Algorithm 7.

Algorithm 7 GetInvolvedEdges

```
1: procedure GetInvolvedEdges( )
2:     for each obstacle in obstacles do
3:         for each edge in obstacle do        ▷ Check if each edge is in compatible base region
4:             if PointToLineDistance(printing task, edge) < compatible base region
   outer radius then
5:                 edges.append(edge)
6:             end if
7:         end for
8:     end for
9:     Return edges
10: end procedure
```

This function makes use of the compatible base region introduced earlier in Section 3.2.1. Any obstacle intersecting the region means there is a chance for the robot to collide with the obstacle as illustrated in Figure 3.4.

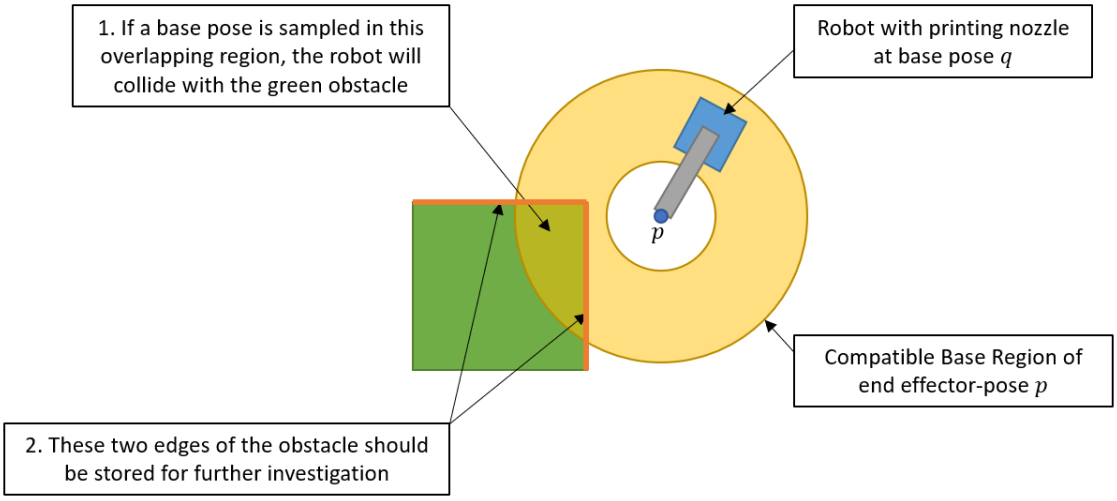


Figure 3.4: Scenario showing an obstacle intersecting with a compatible base region

In this regard, the edges of the obstacles involved in the overlapping region should be stored for future investigation. The reason why we cannot immediately say that we need a decomposition for the task here is that there can still be cases that do not satisfy what has been described in Part 1 of the problem definition, which means the situation may not be so congested that the robot cannot bypass the obstacles. One such case is shown in Figure 3.5.

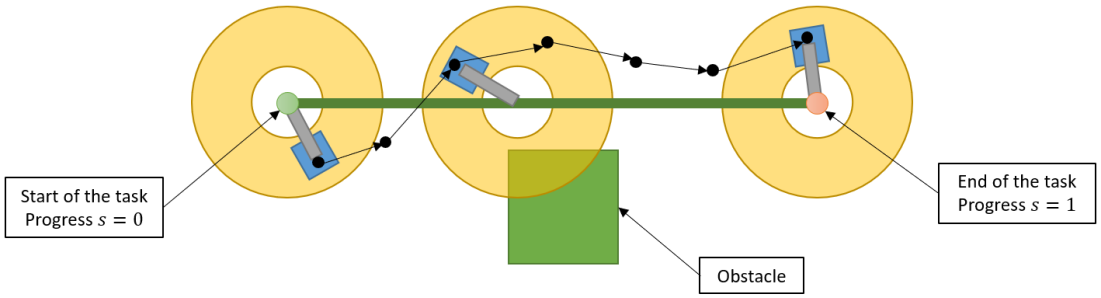


Figure 3.5: Scenario showing a path (in black) is built without decomposition

As observed in Figure 3.5, the path found can bypass the obstacle. Therefore, the robot instructed with this path can print the task (highlighted in dark green) from the start to the end without collision. To further illustrate, a case that describes the condition of problem definition 1 is in Figure 3.6.

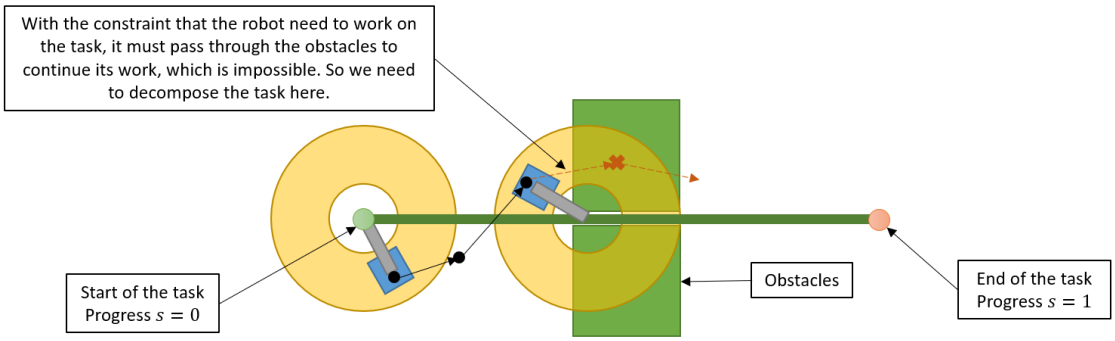


Figure 3.6: Scenario showing a path (in black) cannot further proceed, which is the condition requiring a edge $E \cap \mathcal{X}_{obs}$ to connect (dotted arrows and red markers)

To check if an edge is inside a compatible base region, we calculate the shortest distances between each coordinate of the task and the edges of the obstacles (line 4 in Algorithm 7) and check if there is any distance shorter than the outer radius of the compatible base region. The reason why it is checked in such a way is that the shortest distance between each task coordinate and the edges is the same as the distance between each task coordinate and the obstacles. A distance shorter than the compatible base region radius indicates the obstacle intersects with the compatible base region as illustrated in Figure 3.4. The calculation of the shortest distance is included in Appendix A.1 for the sake of completeness.

We have now narrowed down the region of interest to be within the compatible base regions of every end-effector poses needed for the task. With all the locations of the involved edges known, we can now proceed to investigate the distances between the edges. We are interested in identifying inter-edge distances that are shorter than the diagonal size of the robot. This is because such a condition indicates that the gap between the two obstacles is not wide enough for the robot to pass through. Figure 3.7 shows three cases that describe the condition. These gaps (in green) are recorded as check-lines to identify the task coordinates where we possibly need to decompose the task in the next step.

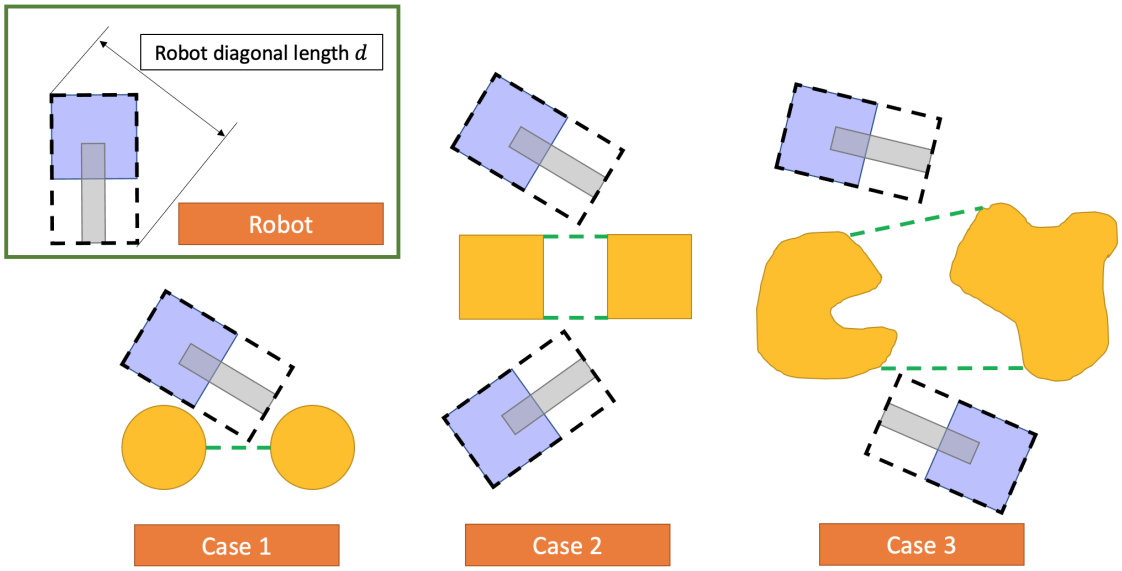


Figure 3.7: Three cases of obstacle configurations (in yellow) with regions that cannot fit the diagonal length of the robot indicated in green dotted lines. A robot indicated with the diagonal length is illustrated at the upper left corner.

The diagonal length of the robot is used to do the checking. This is to minimize the time needed for the path planning algorithm to break into the gap between the obstacles by attempting to find the correct orientation for the base pose. This is justified with Figure 3.8.

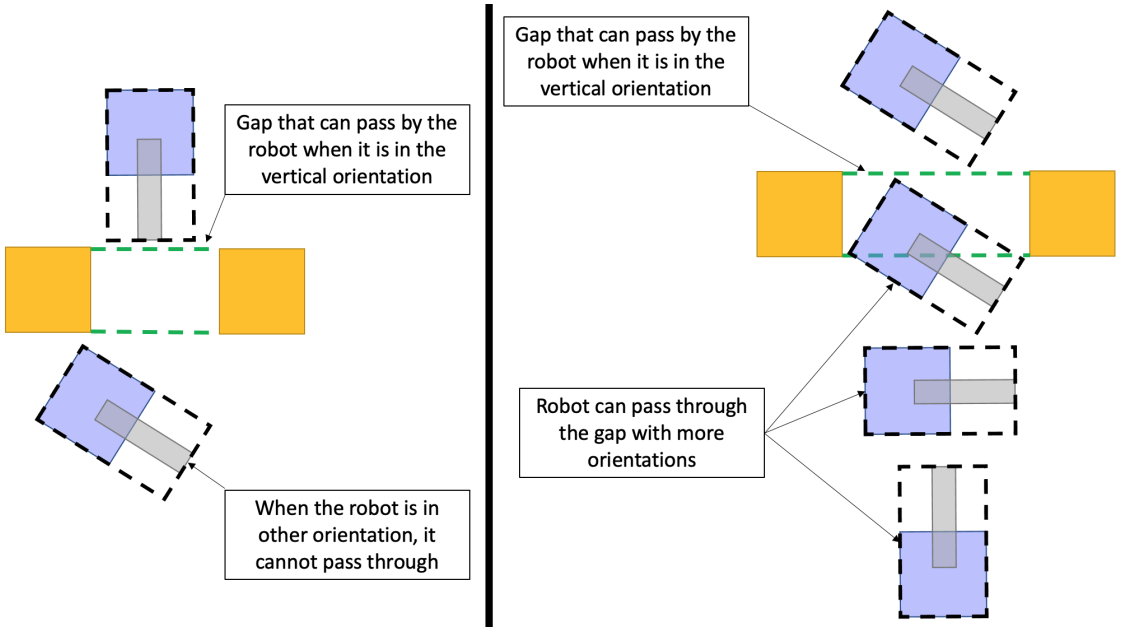


Figure 3.8: Example showing a gap checked using the width of the robot (Left) and that using the diagonal length (right)

From Figure 3.8, we can see that the path planning algorithm may need more sampling for cases on the left side because there is/are orientation(s) that the robot cannot pass through. In

contrast, the case on the right side requires less sampling because the robot can pass through with all the orientations.

With the above description, the pseudo-code that can identify such regions is in Algorithm 8.

Algorithm 8 GetChecklines

```
1:  procedure GetChecklines(edges)
2:      for each edge_i in edges do
3:          for each edge_j in edges do
4:              if edge_i ≠ edge_j ∧ (edge_i ∧ edge_j ∉ same obstacle) then
5:                  [d1, checkline_1] ← PointToLineDistance(edge_j.vertex_1, edge_i)
6:                  [d2, checkline_2] ← PointToLineDistance(edge_j.vertex_2, edge_i)
7:                  if d1 ≤ robot size then
8:                      checklines.append(checkline_1)
9:                  end if
10:                 if d2 ≤ robot size then
11:                     checklines.append(checkline_2)
12:                 end if
13:             end if
14:         end for
15:     end for
16:     Return checklines
17: end procedure
```

In Algorithm 8, we examine the distance between the two vertices in each edge with all other edges to identify the aforementioned gaps. The coordinates linking the gaps are stored and returned to identify relevant task coordinates and their progress in the next function GetRoughBreakPoints().

The function GetRoughBreakPoints() aims to identify the progress variable s by locating task coordinates closest to the check-lines. With these progress identified, the path planning algorithm can be informed and start trees on the corresponding locations. The assumption of this function the printing task should pass through the check-lines. We think this assumption makes sense because it is not necessary to navigate the robot through a narrow gap if it has nothing to print in there. In Algorithm 9, the minimum distances of every coordinate in the printing task to every check-line are calculated. Then we compare these minimum distances with the mean distance between the printing task points d_{task_diff}. If they are less than the mean of d_{task_diff}, then we treat the corresponding progress as a likely point that the path planning algorithm should start a new path. We use the mean of d_{task_diff} as the upper bound of the minimum distance calculated because of two reasons. Firstly, the printing task has been discretized into a sequence of coordinates with approximately equal distance, taking the mean of d_{task_diff} to describe a sense of how far the task points are away from each other. If a minimum distance is shorter than this distance, that means it is very likely that the check-line intersect the printing task, and a new tree should start from the nearest task point. Secondly, since the mean of d_{task_diff} describe the inter-task point distance, we can use this value to exclude check-lines that have no task passing

through. This is illustrated in Figure 3.9.

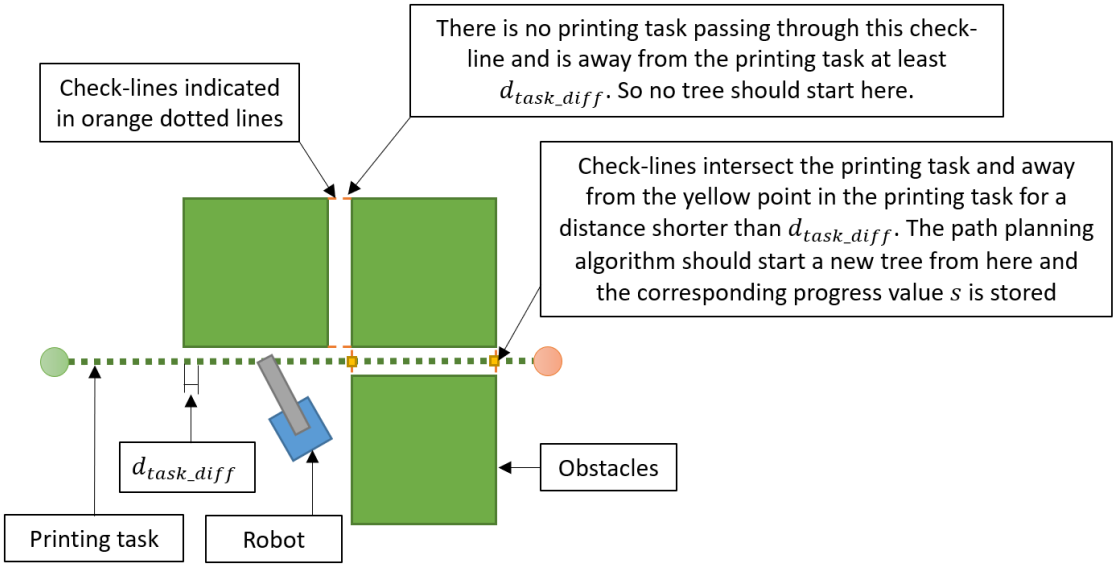


Figure 3.9: Scenario illustrating the determination of "rough" breakpoints for printing task points going through green obstacles

With this, the pseudo code that do this is presented in Algorithm 9.

Algorithm 9 GetRoughBreakPoints

```
procedure GetRoughBreakPoints(checklines)
    for each checkline in checklines do
        [min_dist, idx] ← min(PointToLineDistance(task.coord, checkline))
        if min_dist < mean(d_task_diff) then        ▷ Mean distance between the printing task points
            roughbreakpoints.append(task.s(idx))
        end if
    end for
    Return roughbreakpoints
end procedure
```

Combining Algorithm 7, 8 and 9, we have now located places where:

1. The obstacles are arranged in a way that the robot is hard to pass through. (Algorithm 7, 8)
2. The robot has to get through the obstacles if there is no task decomposition. (Algorithm 9)

Notice that the progress identified in Algorithm 9 are named "rough" breakpoints. This is the case because the results return from the above three algorithms can be very close together. Figure 3.10 shows two such cases.

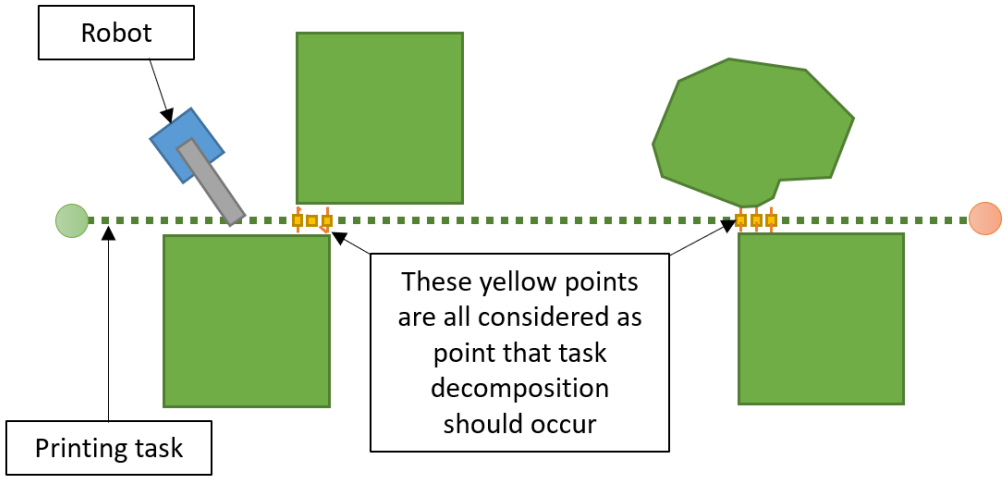


Figure 3.10: A scenario showing two cases that "rough" breakpoints (yellow) identified may be able to merge locally

With this issue, it would be good to have a function that can optimize the results by investigating whether these groups of points can be merged locally. Zooming into the left cases presented in Figure 3.10, one can tell the three "rough" breakpoints can be reduced to just one breakpoint in the middle. If we look at the compatible base region of these three points, we can see that they are highly overlapped as shown in Figure 3.11. The overlapping area also indicates that if we place the robot in the overlapping region it can reach the task points involved.

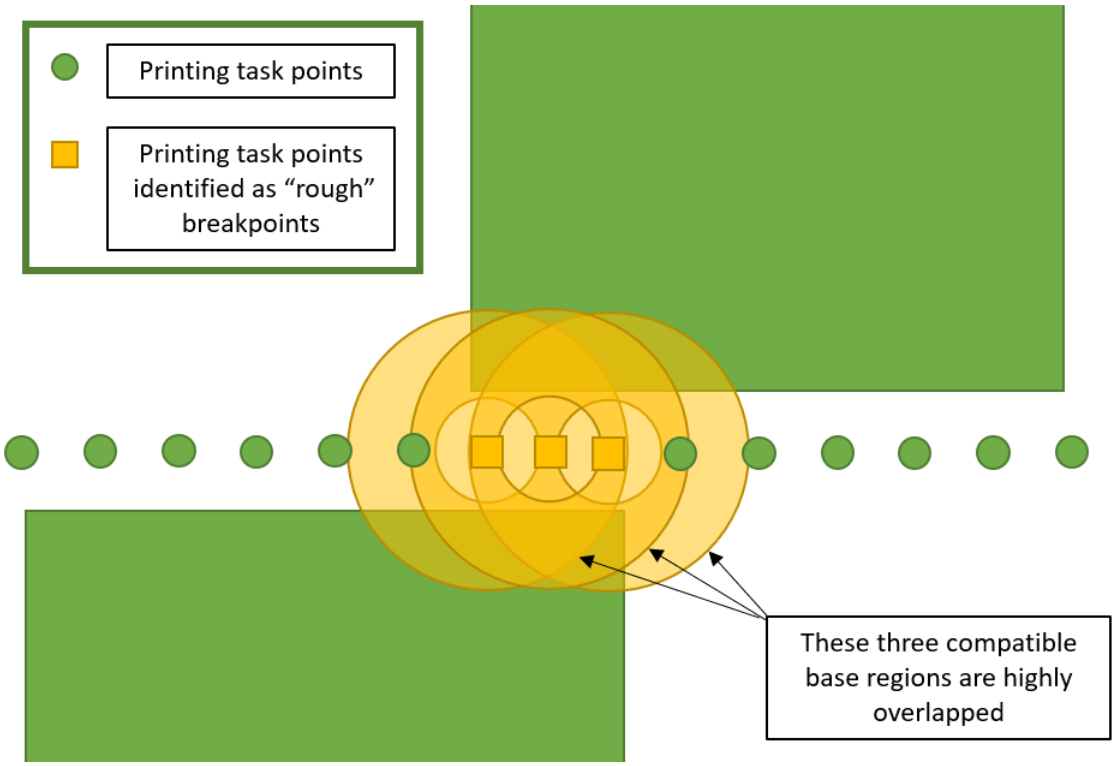


Figure 3.11: Compatible base regions of "rough" breakpoints are highly overlapped

Now zooming out back to the whole picture in Figure 3.10, we can also notice that none of these compatible base regions overlaps with the compatible base regions of the "rough" breakpoints determined on the right side, this is shown in Figure 3.12.

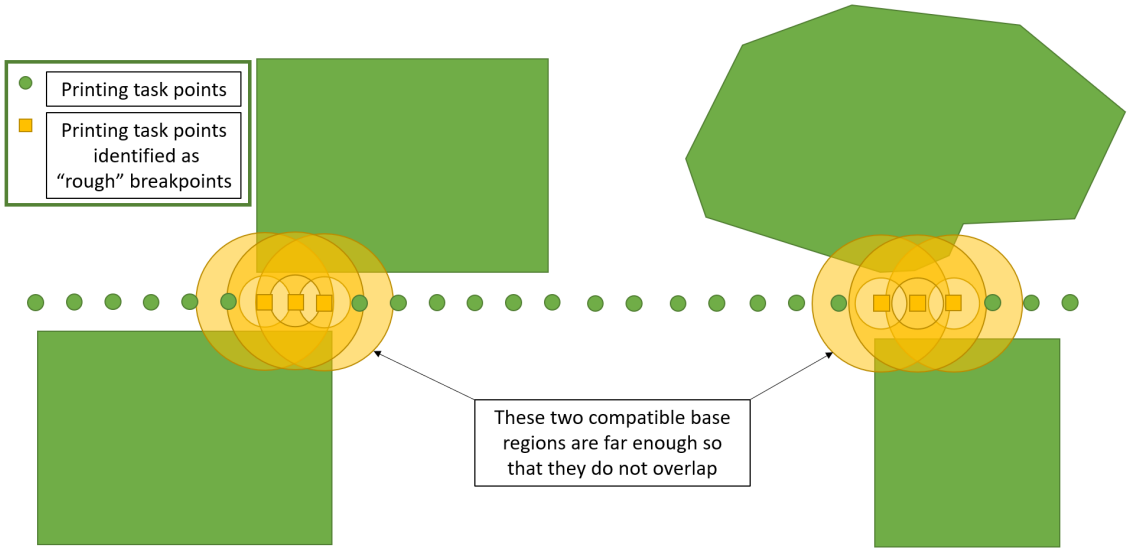


Figure 3.12: Compatible base regions do not overlap

From these observations, it seems that using the overlapping area to determine whether these "rough" breakpoints can be merged is a way to go. However, this is under the assumption that the second set of "rough" breakpoints should be sufficiently far away from the last "rough" breakpoint of the previous set. This in turn means a second set of obstacles should be sufficiently far away from the first set. This makes sense because if the obstacles are so packed together, the robot will have a high chance of colliding with obstacles. And in these cases, some sections of the task are also impossible to print. This is illustrated in Figure 3.13.

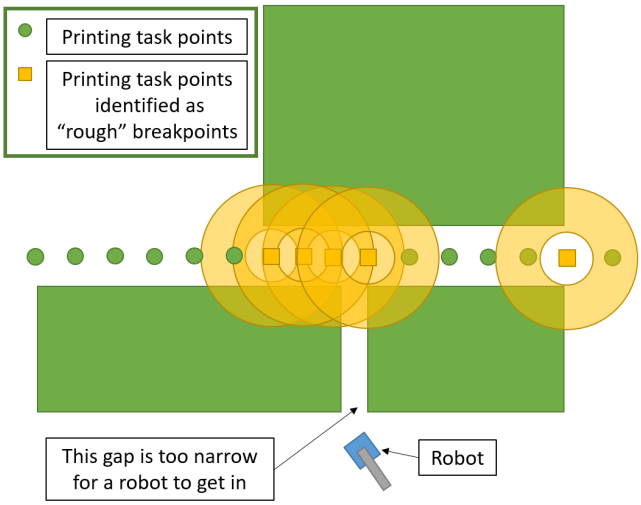


Figure 3.13: A scenario show a robot cannot get into a narrow passage. So the small section enclosed between the obstacles cannot be printed

With this, a function that merge "rough" breakpoints locally is proposed in Algorithm 10.

Algorithm 10 MergeWithCR

```
1: procedure MergeWithCBR(roughbreakpoints, α_overlap)
2:     while true do                          ▷ Recall that roughbreakpoints are actually progresses
3:         n ← size(roughbreakpoints)
4:         progress_overlap_mat ← NaN(n, n)
5:         for each s_1, i in roughbreakpoints do                ▷ Check all combinations of progress
6:             for each s_2, j in roughbreakpoints do
7:                                 ▷ CR(p(s)): Compatible base region of end-effector pose at progress s
8:                 if area(CR(p(s_1)) ∩ CR(p(s_2)))/area(CR) > α_overlap then
9:                     progress_overlap_mat(i, j) ← s_2
10:                end if
11:            end for
12:        end for
13:        progress_to_check ← OmitNAN_row_mean(progress_overlap_mat)
14:        diag_progress_overlap_mat ← DiagEntries(progress_overlap_mat)
15:        Overlapping_progress ← progress_overlap_mat − diag_progress_overlap_mat
16:        Mean_overlap_progress ← OmitNAN_row_mean(Overlapping_progress)
17:        if All(Mean_overlap_progress) = 0 then
18:            break
19:        end if
20:        decomposition_progress ← NULL
21:        for k = 1 to size(progress_to_check) do
22:            decomposition_progress.append(progress_to_check(k))
23:        end for
24:        roughbreakpoints ← unique(decomposition_progress)
25:    end while
26:    decomposition_progress ← NULL
27:    for k = 1 to size(progress_to_check) do
28:        decomposition_progress.append(progress_to_check(k))
29:    end for
30:    breakpoints ← unique(decomposition_progress)
31:    Return breakpoints
32: end procedure
```

Algorithm 10 operates in the following steps:

- **Line 4 - 12**: we first look at all pairs of compatible base regions in the "rough" breakpoints. Then we store these progress s in a matrix if they are found to have an overlapping percentage of area larger than a self-defined overlap threshold $\alpha_{overlap}$. A self-defined threshold is used because this allows the user to control how 'local' the merging process is. For an extreme instance, if we define $\alpha_{overlap} = 100\%$ there will be no merging at all because only the compatible base regions located at the same spot can have an overlapping of 100%. On

the other end, if we define $\alpha_{overlap} = 0\%$, all the progress will be merged. Therefore, this threshold can control how 'local' the merging processes are. In addition, the progress are stored as the double for loop indexed into the matrix. It is done this way so that we can know which pair of compatible base regions satisfied the condition in line 8. Here we assume the compatible base region is a doughnut shape polygon and so there will be cases that this assumption is not applicable.

- **Line 13 - 15**: Exclude the diagonal entries of the matrix, This is done because the diagonal entries represent the intersection of the same compatible base region, and we would like to investigate if the aforementioned condition is valid between different compatible base regions only. This also creates a terminating criterion for terminating the while loop.
- **Line 16**: The mean progress is calculated. We calculate the mean because a large overlapping region means that the "rough" breakpoints are very close together. In this case, we can merge these "rough" breakpoints and represent these "rough" breakpoints by taking the mean of them. By doing this iteratively, each rough breakpoint cluster will eventually converge to one breakpoint.
- **Line 17 - 19**: When every mean calculated above is zero, every "rough" breakpoints cluster has converged to one breakpoint and further operation is not necessary.
- **Line 20 - 24**: Update the variable roughbreakpoints with the unique set of the progress calculated and continue with the iterations.
- **Line 26 - 31**: Return the unique set of progress as the final result.

With Algorithm 10, the whole decomposition process for TCRRT* is concluded. Knowing that the printing path must start at progress $s = 0$ and ends at $s = 1$, we can inform TCRRT* about the location of start and end points with the breakpoints calculated above. For example, if we have breakpoints $= [0.23, 0.58, 0.78]$, combining the start and end points will become $[0, 0.23, 0.58, 0.78, 1]$. So, the start points in this case are $[0, 0.23, 0.58, 0.78]$ and the end points are $[0.23, 0.58, 0.78, 1]$. In this way, we can see that the combining effect of Algorithm 7 - 10 returned information that helps decomposing the printing task $\mathcal{T}(s)$ into n sections $\mathcal{T}_1(s_1), ..., \mathcal{T}_i(s_i), ..., \mathcal{T}_n(s_n)$.

In general, octomaps are used to represent obstacles in applications. However, one can convert the octomaps to polygons by extracting the vertices and edges to use the above-mentioned method.

3.3 Implementation

In this study, all the codes, tests and experiments were implemented and simulated using MATLAB R2021a in a computer with specification summerized in Table 3.1.

Table 3.1: Computer specification

Components	Specification
Processor	Intel®i7-6700K CPU @4.00GHz (8 CPUs)
Memory	16384MB RAM
Graphic card	NIVIDIA GeForce GTX1070
Operating System	Windows 10 Pro 64-bit (10.0 Build 19043)

In this section, the way of optimizing the path planning operations will be discussed.

3.3.1 Accelerating Path Planning with k-dimensional Tree

To enable the path planning algorithm for planning long paths, we must speed up the algorithm. One of the steps that take up much time in the path planner is finding the nearest neighbour. When we want to find the nearest neighbour for a node, we calculate the distances from the node to all existing nodes. It would be great if such calculations are minimized. In this regard, k-dimensional tree (KD Tree) data structure is used to organize the nodes generated in the path planner. KD Tree organizes k-dimensioned data in a binary tree structure. At each level of the tree, if the value of the corresponding dimension of a node is less than an existing node on this tree, it will be inserted on the left side of the existing node, and on the right side otherwise. To do this, when a new node is confirmed in every iteration of the path planner, we insert it to the KD Tree. So, whenever we want to find the nearest neighbour, we can search through the KD Tree. By searching through the tree, a lot of distance calculations can be skipped. Therefore, using a KD Tree significantly accelerates the path planner.

However, with the use of KD tree, there is an issue for the configuration space. Uptil now we define a configuration as $q = [x, y, \theta, s]$ and the domain of the angle θ is $[-\pi, \pi]$. Since there is a sudden jump from $-\pi$ to π, the search of nearest neighbour can be wrong because of the way KD tree organize the data. For example, in $-\frac{9}{10}\pi$, $\frac{9}{10}\pi$, $\frac{1}{10}\pi$ the nearest neighbour of $\frac{9}{10}\pi$ should be $-\frac{9}{10}\pi$ but with a KD tree it will return $\frac{1}{10}\pi$. A way of work around is to map θ to $\cos\theta$ and $\sin\theta$. By doing this, although the sudden jump will just occur in $\cos\theta$ and $\sin\theta$, the value of θ can be restored by using $\arctan(\frac{\sin\theta}{\cos\theta})$. With this, our poses throughout the program is implemented using $q = [x, y, \cos\theta, \sin\theta, s]$ instead.

Although there is a KD Tree provided in MATLAB, these two KD Tree functions are self-implemented. This is because the KD Tree in MATLAB requires initialization every time we want to find the nearest neighbours which occupy a lot of memory. By using the self-implemented KD Tree functions, we can insert nodes into the KD Tree on the fly as the path planner operates, which theoretically only takes $\mathcal{O}(nlog(n))$ time on average [85].

To ensure the usage of the KD Tree is beneficial for the path planner, a test is conducted. In this test, a node is fixed and a list of nodes are randomly generated from one node to 3000 nodes. Then, the nearest neighbours are found by using two methods:

1. Calculating all distances using square norm. Then find the minimum distance. (Method 1)
2. Use KD Tree structure with square norm. (Method 2)

10 runs are performed and the run time is recorded for each size of the node list. The result is visualized in Figure 3.14.

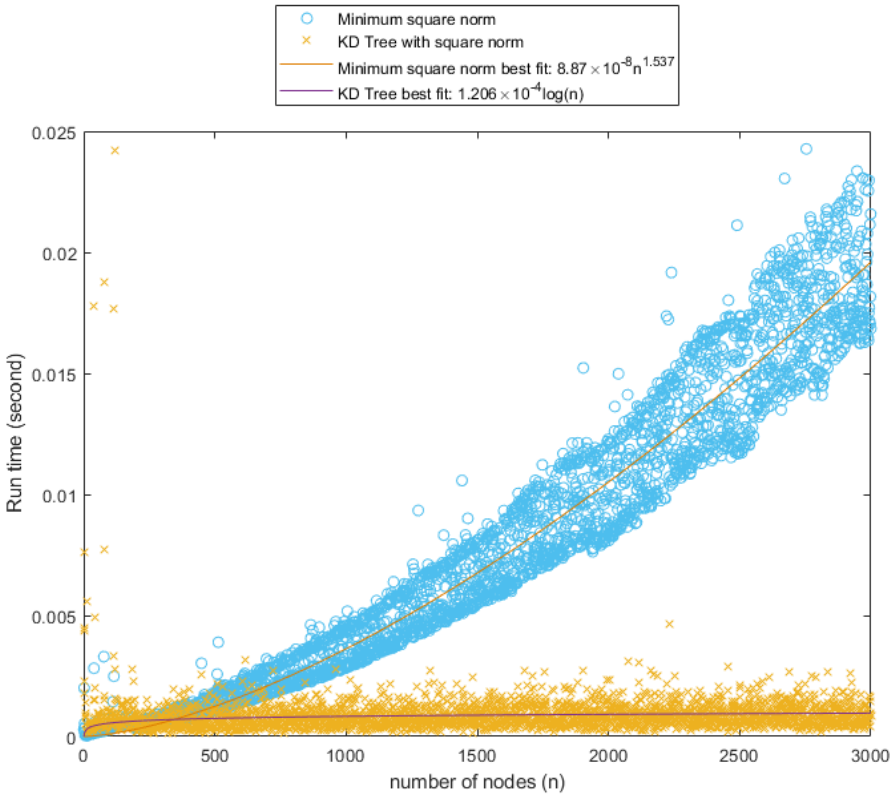


Figure 3.14: Efficiency of finding a nearest neighbour using Method 1 and Method 2

We can see that as the number of nodes increase, the run time of Method 1 exhibits a time efficiency of $\mathcal{O}(n^{1.537})$. Meanwhile, Method 2 gives a time efficiency of $\mathcal{O}(log(n))$, which is significantly faster when the number of nodes increases beyond 350 nodes.

Please note that KD Tree will be applied on TCRRT* only because of the time limit for this study.

3.4 Summary

In this chapter, the problem definition of obstacle-based multi-agent decomposed task consistent path planning is introduced. The methodology of task consistent path planning in terms of RRT* and FMT* are given. Task decomposition methods for TCFMT* and TCRRT* are proposed and acceleration using KD Tree on RRT* is introduced. In the next chapter, recent results, limitations discovered and difficulties encountered will be discussed.

Chapter 4

Evaluation and Discussion

In this chapter, the evaluation method and the result of selecting parameters for TCFMT* and testing the task decomposition methods proposed in the previous chapter will be presented and discussed at length. In addition, the limitations and difficulties will be discussed.

4.1 Evaluation Method

4.1.1 Parameters Selection for TCFMT*

As mentioned in Section 2.4.3 and Section 3.2.2, the number of samples, the neighbour searching radius and the fluctuation of the changing radius will be investigated for this study. This is done because of the following reasons:

- In terms of the radius, it actually holds the meaning of maximum edge length. This is because it defines the region of interest for a point to search for and steer to its neighbours. A very small radius leads to a small region of interest. In this case, the tree will not be able to explore well because some paths that can result in lower costs may be left unexplored. In addition, V_{open} may be frequently emptied because the region of interest is so small that it cannot find a valid neighbour and the task decomposition mechanism is triggered. On the other hand, a very large radius will induce a long computation time because we need to look at all the neighbours in range (line 12 - 22 in Algorithm 4). In terms of task consistency, FMT* using a large radius also allow connection to configurations that are very far away from the current progress. This is also hugely undesirable.

- In terms of the number of samples, a very small number of samples means the points in the space may be very far away. This creates an issue for the neighbour searching because there will be no neighbour at all if we use a small radius. Again, a very large number of samples will lead to a long computation time and therefore also not desired.

- In terms of the fluctuation of the changing radius, a large magnitude of fluctuation means the algorithm struggles to find a valid configuration to connect and vice versa. Having a large fluctuating magnitude frequently is not desirable because the resultant path may not be smooth.

The above reasons motivate the investigation for a combination of searching radius and number of samples that can minimize the runtime of the algorithm, the cost of the path and the fluctuation of the changing radius while maintaining a path that has a reasonable decomposition result.

The determined combination of parameters will be used for further experiments with obstacles. The investigation is done on the printing task shown in Figure 4.1 and in the following way. Using TCFMT*, we will test for a sampling rate from one to seven samples per progress, and for each of these sampling rates we will test for a neighbour searching radius from 0.1m to 0.5m with an increment of 0.05m. For each combination, the time, path cost and the number of resultant paths are recorded to understand the quality of the resultant path, the mean and standard deviation of the changing radius are recorded to understand the fluctuation of the radius.

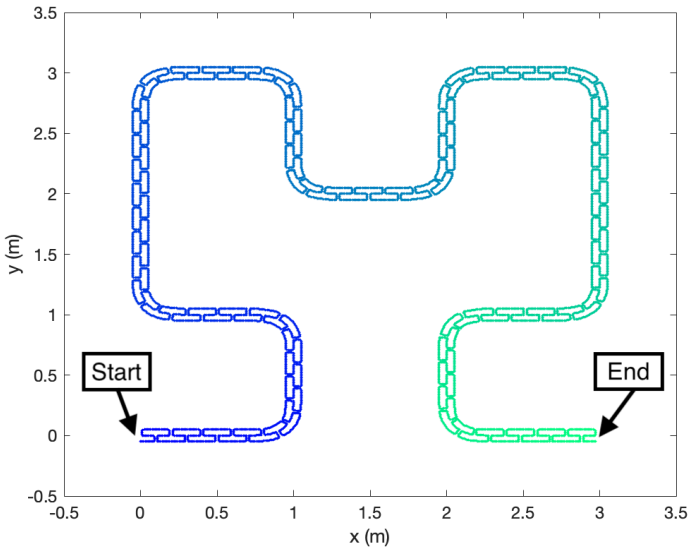


Figure 4.1: Testing printing task with starting point and end point indicated

4.1.2 Task Decomposition Assessment Method

To test the ability of the method described in Section 3.2.2 for TCRRT* and TCFMT* in obstacle based task decomposition, five obstacle configurations are designed around the case shown in Figure 4.1 to test their limits and performance. These tests are tested with a robot and a compatible base region with corresponding dimensions shown in Figure 4.2:

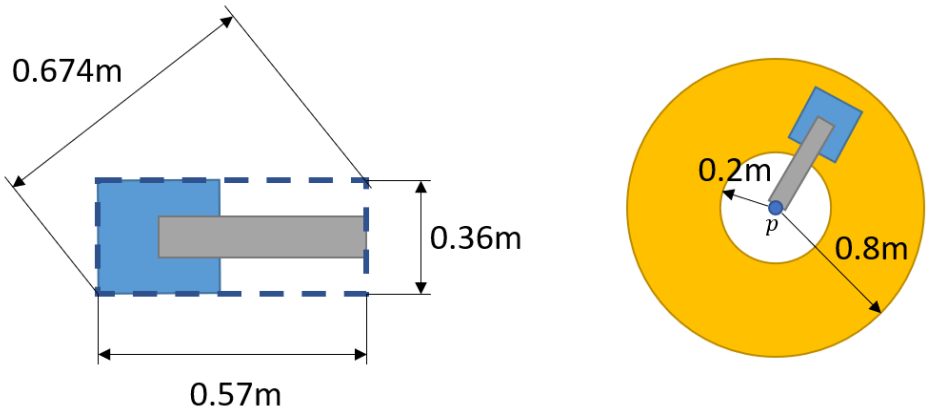


Figure 4.2: Robot and its compatible based region dimensions

1. **Obstacle configuration 1**: Short narrow gap

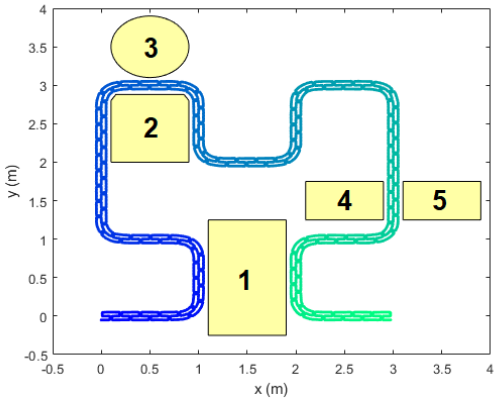


Figure 4.3: Obstacle configuration 1: Short narrow gap

In obstacle configuration 1 shown in Figure 4.3, we can see that there are five obstacles in total. The first obstacle aims to test two things:

- The obstacle detection and avoidance of the proposed method.
- To test if the method proposed for TCRRT* in Section 3.2.2 can correctly ignore the gap between obstacle 1 and obstacle 4 since there is no printing task through it.

The remaining obstacles form two cases of narrow gaps, the first gap (2 and 3) is to test how the methods will deal with the case when the task is decomposed at one single point such

that the printing task can be continued right on the other side of the obstacles. The second gap (4 and 5) is to test for the case when a small section of the printing is surrounded by obstacles that are still reachable by the robot.

For this configuration, the printing task should be decomposed into three sections. So, three exploring trees and resultant paths are expected.

2. **Obstacle configuration 2**: Long narrow gap (section not printable between obstacles)

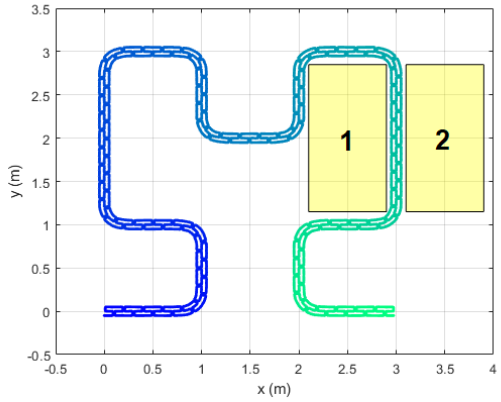


Figure 4.4: Obstacle configuration 2: Long narrow gap (section not printable between obstacles)

Figure 4.4 shows two obstacles placed on both sides of a section of the printing task in a way that the robot will not be able to pass through. In this case, we expect two points of task decomposition and two resultant paths. However, the covered section of the task is unreachable by the robot so there should be no path planned for that.

3. **Obstacle configuration 3**: Long tunnel near sharp turning (section printable between obstacles)

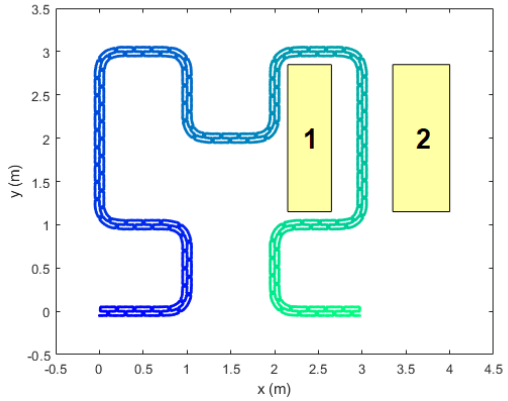


Figure 4.5: Obstacle configuration 3: Long tunnel near sharp turning (section printable between obstacles)

Figure 4.5 shows a variant of obstacle configuration 2 which has the printing task go through

a gap (width 0.7m) that can have the robot pass through with no collision. This case aims to test whether the proposed method can turn sharp corners into a narrow region and find its way through the tunnel. Especially in the case of TCFMT*, the algorithm may not be able to find its way through the tunnel. This is because the valid configurations sampled in the tunnel will have their orientations more directed in parallel with the printing task. So there may exist a sudden jump in orientation. In this case, the algorithm may trigger a false positive task decomposition. In this configuration, we expect one solution path only.

4. **Obstacle configuration 4**: Three boxes case

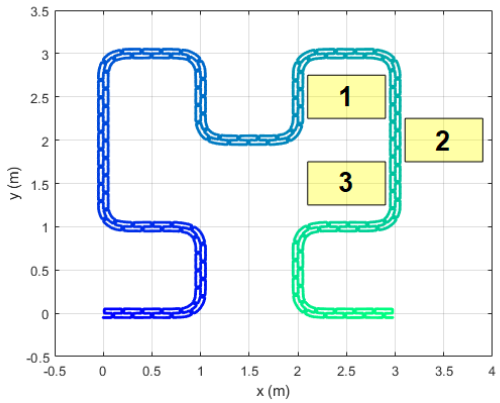


Figure 4.6: Obstacle configuration 4: Three boxes case

The three boxes case shown in Figure 4.6 can be view as a variant of narrow gaps in the first obstacle configuration. However, this is a bit different. This configuration is designed in a way that obstacle 1 and obstacle 3 are 0.5m apart. Since we are using a compatible base region with a radius of 0.8m, the section of the printing task around the point $(x = 3, y = 2)$ should be printable from above and below obstacle 2. So, two sections of the resultant path are expected in the optimal case. However, three paths are also possible.

5. **Obstacle configuration 5**: Non-Convex obstacles

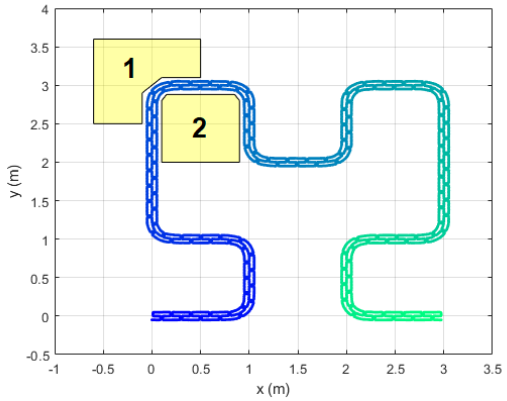


Figure 4.7: Obstacle configuration 5: Non-Convex obstacles

Obstacle configuration 5 shown in Figure 4.7 has a non-convex obstacle covering a corner of the printing task. This case aims to test how well the procedure of decomposition for TCRRT* responds to non-convex obstacles. In this case, we expect two solution paths as a result.

In this study, TCRRT* and TCFMT* are assessed with these obstacle configurations. For each configuration, five tests are done and some measurements are recorded to evaluate the path planners. They are as follows:

1. **Time (s)**: To check the time efficiency of the algorithms.
2. **Iterations**: To benchmark the efficiency of the algorithms procedure-wise. I.e. the number of iterations needed for the algorithm to perform to solve a particular obstacle configuration.
3. **Path total cost**: To evaluate the cost needed for the robot to navigate itself through the resultant paths.
4. **Number of paths**: To compare against the optimal number of decomposed sections and benchmark the task decomposition quality.
5. **Total number of path nodes**: To measure how many nodes the robot needed to follow.
6. **The trend of the furthest progress** s_{max}: To evaluate the ability of the algorithm to move forward along the task.
7. **Mean and standard deviation of the size of the changing radius (TCFMT* only)**: To measure the fluctuation of the changing radius and show insights to how struggle the TCFMT* was to find a valid configuration in a particular obstacle configuration.

For TCRRT*, we would also like to evaluate the quality of task decomposition based on the procedure proposed in Section 3.2.2 by its ability to accommodate TCRRT* to plan a path. To elaborate, we would like to know if TCRRT* can plan paths based on the decomposition scheme provided by the TCRRT* decomposition procedure. In this regard, graphics of the decomposition results will be shown along with the task planning result. The parameters used for TCRRT* are based on Šustarevas et. al[84] and are presented as in Table 4.1 along with the parameter for the task decomposition procedures:

Table 4.1: TCRRT* and task decomposition parameters

Parameters	Value
Edge length increment ϵ_{inc}	0.01
Maximum edge length ϵ_{reach}	0.05
neighbour search radius r	0.25
Compatible base region overlapping threshold $\alpha_{overlap}$	0.5 (50%)

In the case when TCRRT* get stuck at any progress s when exploring a section of path, we would allow TCRRT* to try 1000 times to search for a point that can lead to further progress. If

that cannot lead to further progress, we will terminate the search for that section and continue with other sections according to the decomposition scheme.

For TCFMT*, since the decomposition mechanism is triggered on the fly, the results of path planning will be shown and discussed with the measurements above to evaluate the quality of task decomposition. Moreover, the trend of the size of the three data sets $V_{unvisited}, V_{open}$ and V_{closed} will also be presented to verify the method discussed in Section 3.2.2. The parameters used for TCFMT* in this experiment are displayed in Table 4.2:

Table 4.2: TCFMT* parameters

Parameters	**Value**
Number of point to sample per progress n	To be determined experimentally
Maximum trials before adaptive sampling n_{sample_trials}	100
Number of points to be adaptively sampled per progress k	10
Specified neighbour searching radius r_n	To be determined experimentally
Changing radius increment $\epsilon_{r,inc}$	0.01

4.2 Results

4.2.1 TCFMT* Parameter Selection Results

As mention in Section 4.1.1, our goal here is to get a combination of specified neighbour searching radius r_n and sampling rate n that can minimize the runtime, path cost, and the fluctuation of the value of radius while running the algorithm while maintaining a path that has a reasonable decomposition result. Since the printing task in Figure 4.1 is continuous and there is no obstacle, one resultant path is expected only. First of all, time is plotted against the specified radius and the sampling rate (sampling intensity) as in Figure 4.8[1]. We can see that as n increase, the time increase gradually from no time to very time consuming ($\sim$ 16000 seconds) as long as $r_n > 0.16$. On the other hand, as r_n increases, the time consumed also increases at a similar pace in general, but exponentially increase when $n \geq 5$. This makes sense because as r_n and n increase, the algorithm will sample more points and search for neighbours in a larger region, so it needs more time to process.

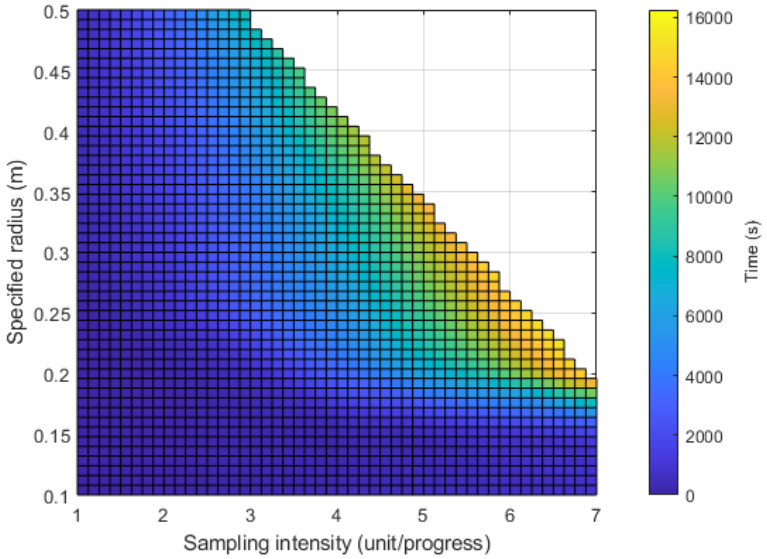


Figure 4.8: Time as a function of sampling rate and specified neighbour searching radius

Next, the number of paths in the resultant solutions is plotted against the parameters as in Figure 4.9. We can see that when the specified radius falls below 0.25m (green-yellow region) there is a high chance that the algorithm will return more than one path. This is the scenario when the radius is too small, the algorithm struggles to find a valid neighbour for a point. The task decomposition is subsequently triggered and start a new path. From Figure 4.9, a safe region is when $r_n > 0.25$ and $n > 2$, combining the result from Figure 4.8 the region where $0.25 \leq r_n \leq 0.5$ and $2 \leq n \leq 3$ seems suitable.

[1]Since the runtime for the results in the upper right triangular part of the plot becomes extremely time-consuming, they are discarded. This also applies to other plots in this experiment

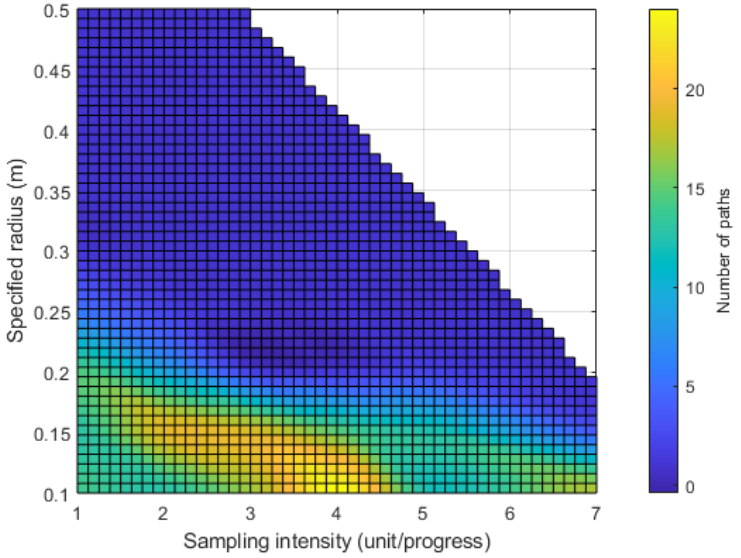


Figure 4.9: Number of paths as a function of sampling rate and specified neighbour searching radius

In terms of path cost as illustrated in Figure 4.10, we can see that the cost gradually decreases as r_n and n increase. This is because the increased radius and sampling rate feeds the algorithm with an abundance of samples. So, it is easier to find alternative low-cost paths. However, this is at an expense of time as shown in Figure 4.8. We can also see that a smaller radius will lead to a larger cost. This is because a small radius increases the bias of the path planning and there are not many alternatives to the algorithm. This is be seen in the lower part of the figure that the cost maintains around 19 units when the radius r_n fall below 0.2 no matter how high is the sampling rate n. An extreme occasion is when $r_n = 0.2$ and $n = 1$. The resultant path is shown in Figure 4.11. The path constructed is so biased that it has to choose a very long path as a result, not to mention it is decomposed into pieces. The growth of the tree is also very retarded which is a result of a low sampling rate n.

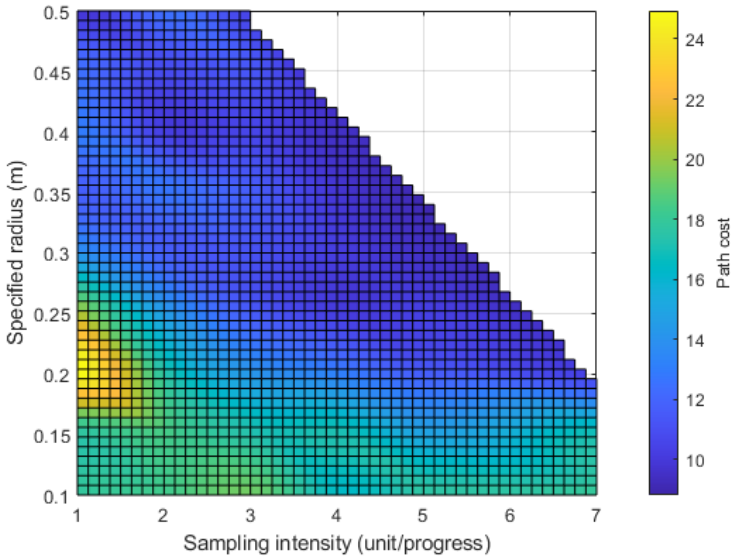


Figure 4.10: Path cost as a function of sampling rate and specified neighbour searching radius

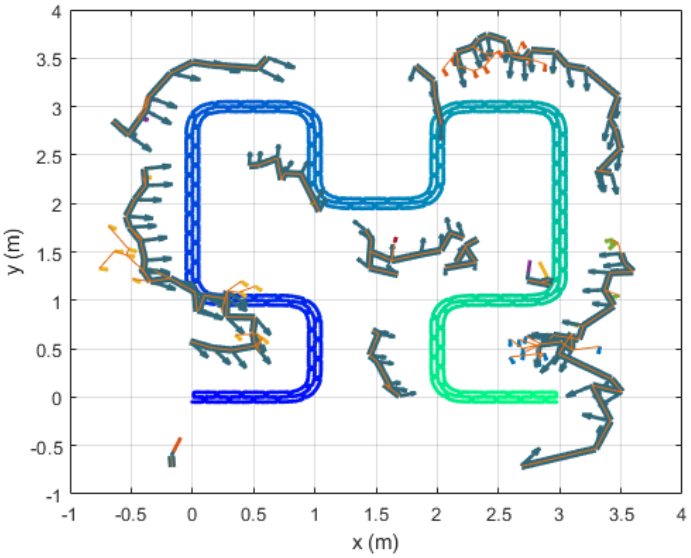


Figure 4.11: TCFMT* Path generated with $r_n = 0.2$ and $n = 1$, indicated in dark green with arrows indicating orientations. The resultant tree is indicated in orange.

Now we look at the fluctuation of the changing radius as a result of the modification mentioned in Section 4.1.1. In Figure 4.12, we can see a linearly increasing trend of mean radius starting from $r_n = 0.3$, which make sense because we are increasing it. The interesting part is the region where $r_n \leq 0.3$. We can see that as r_n decrease and sampling rate n increase, the mean of the changing radius decreases. This is because when the sampling rate is large the samples in the space will be very crowded, so it will still be easier for the algorithm to find a neighbour even though the neighbour search radius r_n is small. However, when the sampling rate is small, the

algorithm will found very hard to find a neighbour because the points are so far away. So the radius in the algorithm expands frequently to search for neighbours and leads to a high mean of the changing radius. This also reflects how struggling the algorithm is when it is working on the case shown in Figure 4.11.

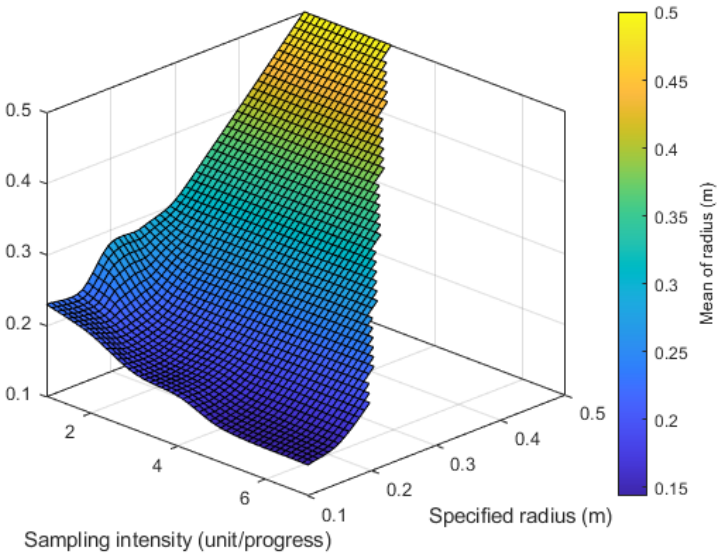


Figure 4.12: Mean radius as a function of sampling rate and specified neighbour searching radius

In terms of the standard deviation, Figure 4.13 shows a result that agrees with Figure 4.10 in the way that as both r_n and n increase, the standard deviation goes down gradually. This is because as there are more samples in the search region when finding a neighbour, the radius does not need to expand so frequently to search for a point. From another perspective, it also makes sense to have a high standard deviation when the sampling rate is low and when the searching radius is small. A low sampling rate means the samples are sparse and so the radius has to expand to find neighbours. A small searching radius also needs to expand the searching region because it cannot always find a valid configuration for connection. Therefore, these regions shall be avoided.

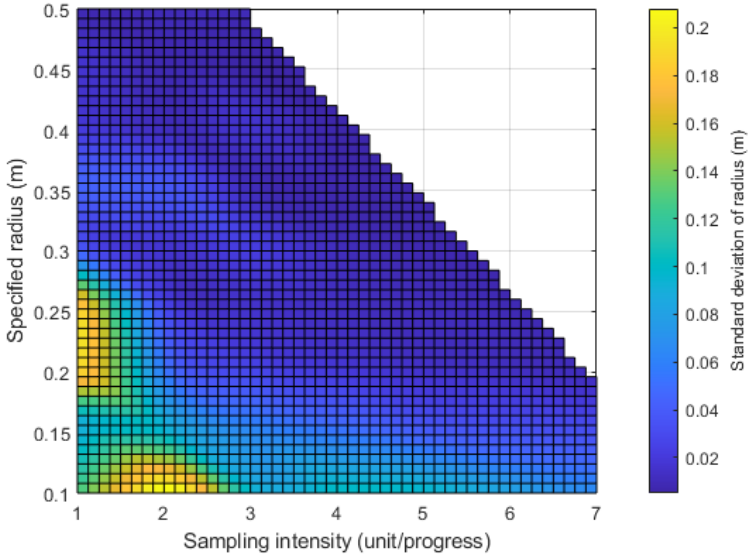


Figure 4.13: Standard deviation of radius as a function of sampling rate and specified neighbour searching radius

Concluding the results above, a neighbour searching radius $r_n = 0.3$ and a sampling rate $n = 2$ is chosen for further studies. This combination is chosen because on one hand an increment of n from two to three increases the runtime from around 1300 seconds to 3700 seconds but with an insignificant decrease in the path cost. On the other hand, if we choose an $r_n = 0.25$ the path cost will have a noticeable increase from around 14 to 18. And more importantly, it will hit the boundary where the path can easily decompose by itself. With this, the resultant path when $r_n = 0.3$ and $n = 2$ is presented in Figure 4.14. This result is done in 1358.1 seconds, have a path cost = 11.6121. and with a changing radius standard deviation of around 0.03 with reference to Figure 4.13.

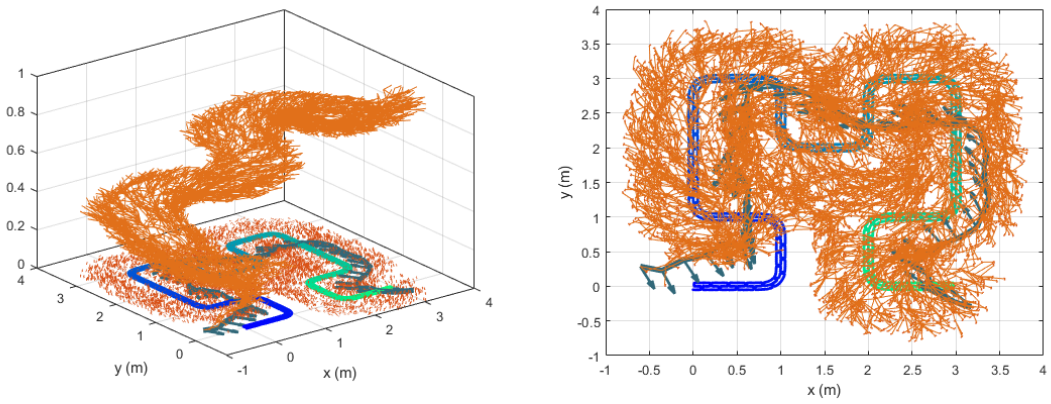


Figure 4.14: TCFMT* Path generated with $r_n = 0.3$ and $n = 2$, indicated in dark green with arrows indicating orientations of base poses. The resultant tree is indicated in orange. The vertical axis of the left figure indicates the progress from 0 to 1, and shows the tree gradually grows from progress $s = 0$ to $s = 1$

In the next section, we will benchmark and compare the performance of TCFMT* and TCRRT* on task decomposition when dealing with obstacles and path planning.

4.2.2 Task Decomposition and Path Planning Results

In this section, TCFMT* and TCRRT* are tested using the configuration of the obstacles mentioned in Section 4.1.2. In the following, the results[2] will be presented one by one. Before going into the results, we would like to introduce a scheme for the reader to identify different parts of the results in Table 4.3.

Table 4.3: Indicators of different parts of visualizations in results

Category	**Objects**	**Indicator**
Path planning	Exploring tree	Bright orange lines
	Resultant path	Dark green lines with arrows indicating base poses of the robot
Environment	Printing task	As in Figure 4.1
	Obstacles	Light yellow polygons
Task decomposition for TCRRT*	Edges involved in task decomposition procedure	Orange lines around polygons
	Checklines	Green dotted lines
	Rough break points	Orange unfilled circle markers
	Point of decomposition	Yellow filled circle markers

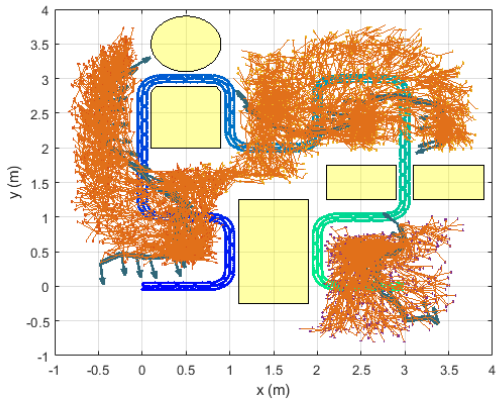
[2]Since the test results in each experiment follow similar trends, only selected results will be presented here. The remaining results are included in the Appendix B for the sake of completeness.

Obstacle Configuration 1: Short narrow gap

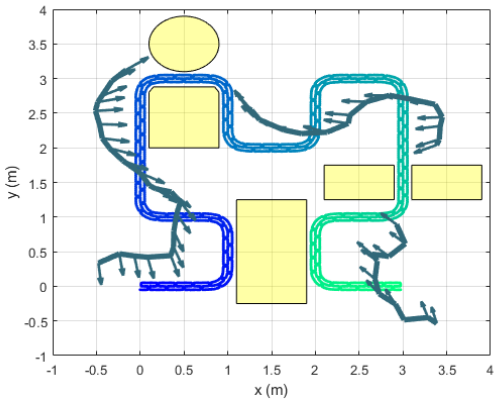
Expected goal:

In this configuration, there are two narrow gaps where the robot is not able to pass through. So, the task is expected to be decomposed into three sections.

TCFMT* Result:

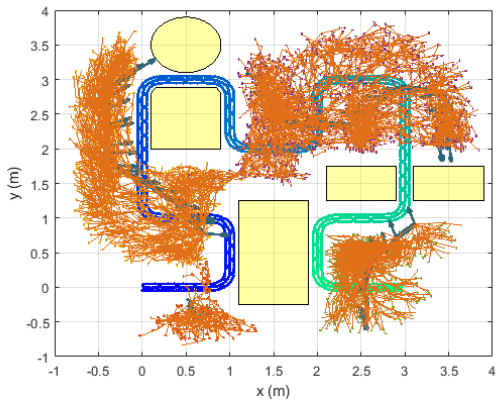


(a) Test one result showing the tree and the path

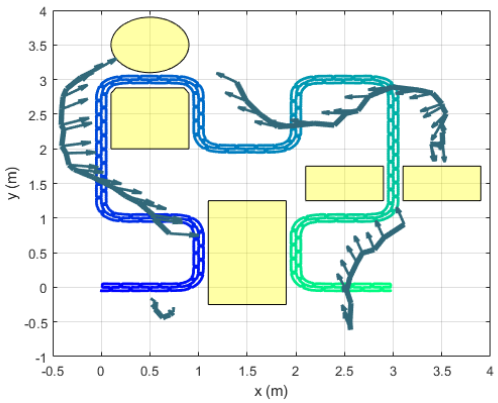


(b) Test one result showing the path only

Figure 4.15: TCFMT* results: Test one on obstacle configuration 1. Path found in 1634 seconds and with a path cost of 10.9598, decomposed into three sections

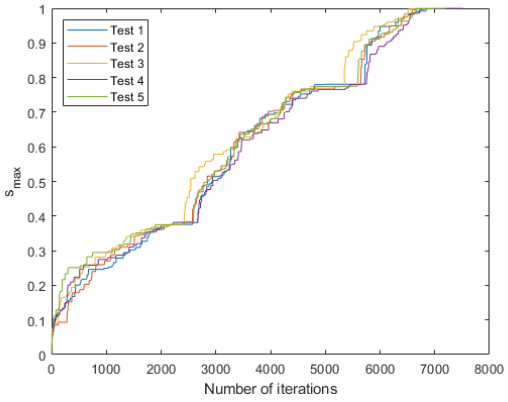


(a) Test two result showing the tree and the path

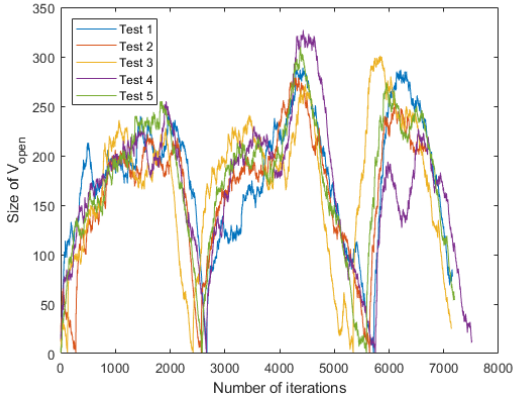


(b) Test two result showing the path only

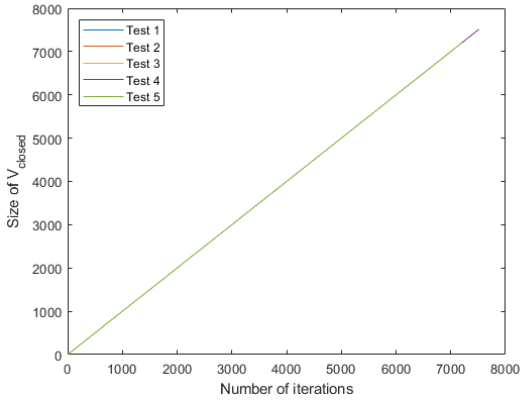
Figure 4.16: TCFMT* results: Test two on obstacle configuration 1. Path found in 1497.3 seconds and with a path cost of 10.8347, decomposed into four sections



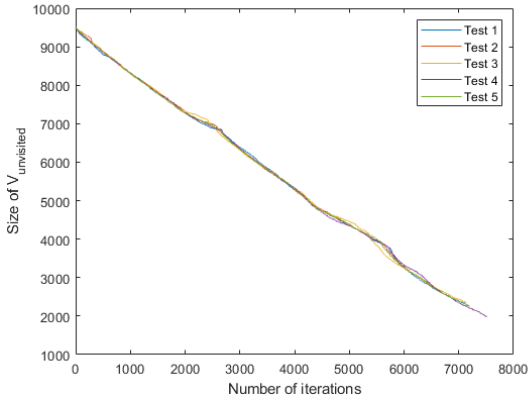
(a) Trends of s_{max} vs. number of iterations



(b) Trends of the size of V_{open} vs. number of iterations



(c) Trends of the size of V_{closed} vs. number of iterations



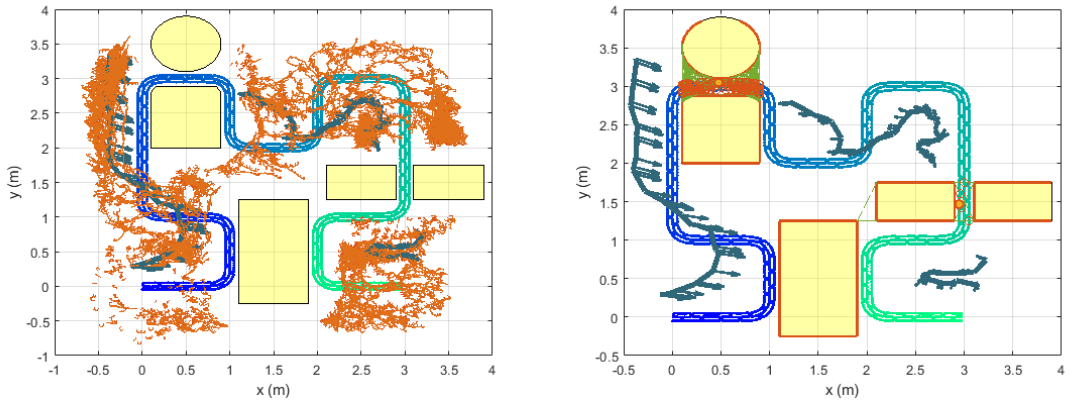
(d) Trends of the size of $V_{unvisited}$ vs. number of iterations

Figure 4.17: TCFMT* results of the five tests for obstacle configuration 1

From Figure 4.15 and Figure 4.16, we can see that tasks are effectively decomposed around the gaps. However, in Figure 4.16b we can see that there is a small section of path planned around $(x = 0.5, y = -0.5)$ and is separated from the main path. The reason for this is because the algorithm cannot find a point that it can extend to, which might ultimately be due to the pre-sampling procedure. In fact, this happened throughout the experiments and this issue will be further discussed in Section 4.3.

Figure 4.17 shows the trends of s_{max}, size of V_{open}, V_{closed} and $V_{unvisited}$. In general, all five tests followed similar trends. In particular, the linearly increasing trend of V_{closed} and the decreasing trend of $V_{unvisited}$ are quite consistent. These trends make sense because we are adding one point to V_{closed} and getting points away from $V_{unvisited}$ every iteration. In Figure 4.17d, the decreasing rate slowed down a bit around 2800^{th} and 5800^{th} iteration. The reason for this is because the algorithm is fed with points that it cannot make a connection. Cross-referencing Figure 4.17a and Figure 4.17b, we can see a stagnation of increase in s_{max} and the size of V_{open} drop to zero and then bounce back around the mentioned region. These two regions are exactly where the gaps present. This suggests that the method proposed in Section 3.2.2 is working.

TCRRT* Result:



(a) Test two result showing the tree and the path

(b) Test two result showing the path only

Figure 4.18: TCRRT* results: Test two on obstacle configuration 1. Path found in 9091.2 seconds and with a path cost of 13.7082, decomposed into three sections.

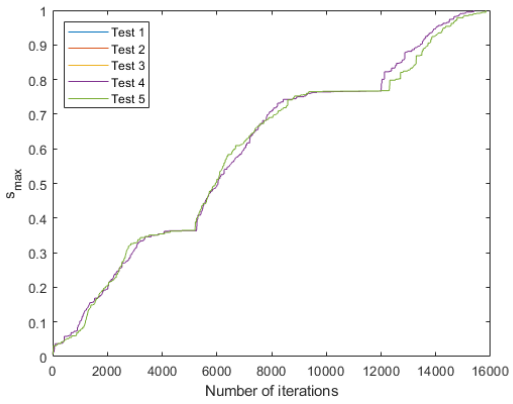


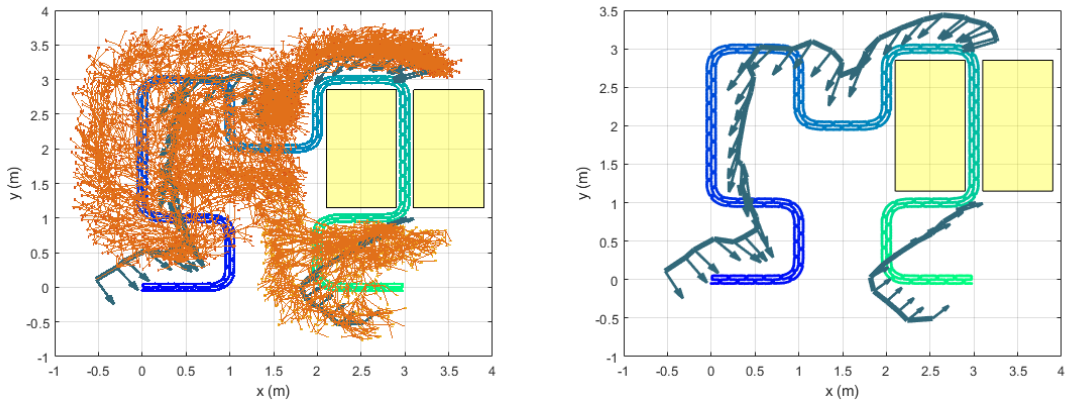
Figure 4.19: TCRRT* results: s_{max} vs. iterations of the five tests for obstacle configuration 1

In the context of TCRRT*, we can see in Figure 4.18a that the tree is very well explored and three paths are planned successfully. Together with Figure 4.18b, green checklines and orange rouge breakpoints between obstacles are observed. We can also notice the yellow point of decomposition is determined at approximately the middle of each gap. Moving to Figure 4.19, we can see that the first goal at around $s_{max} = 0.37$ is reached under 1000 iterations of trials. However, the second gap require all given number of trials at around $s_{max} = 0.76$. This suggested that TCRRT* has been forced to stop and start a new section of path planning which consumes a huge amount of time. So, the point of decomposition provided by the task decomposition procedures is not always reachable. The reason for this might be due to fact that the procedures do not take the shape of the task into account. This also happens in other experiments and will be further discussed in Section 4.3.

Obstacle Configuration 2: Long narrow gap (section not printable between obstacles)

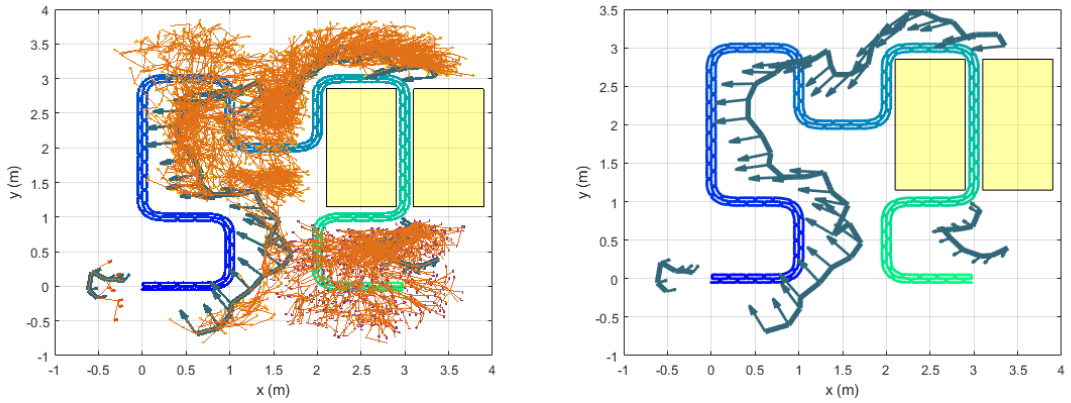
Expected goal:
Configuration 2 has its obstacles arranged in a way that a section of the printing task is covered by the obstacles. So the algorithms are expected to skip that section and produce two paths in total.

TCFMT* Result:



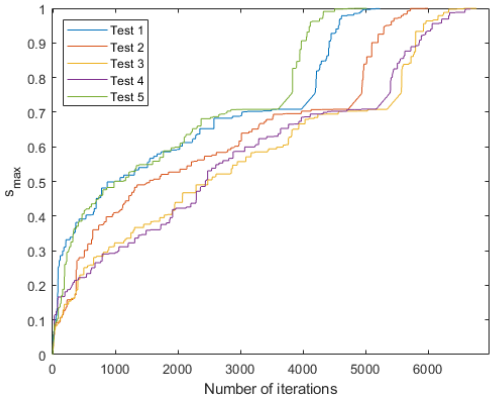
(a) Test four result showing the tree and the path (b) Test four result showing the path only

Figure 4.20: TCFMT* results: Test four on obstacle configuration 2. Path found in 2630.9 seconds and with a path cost of 11.1528, decomposed into two sections

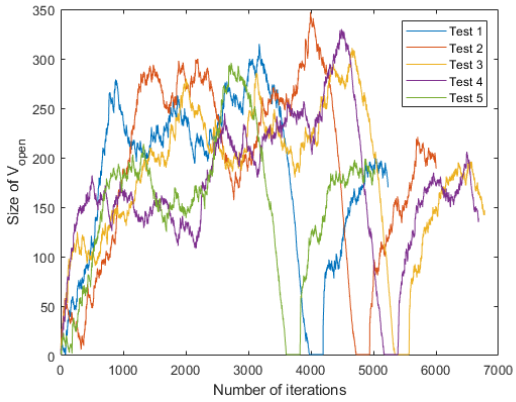


(a) Test five result showing the tree and the path (b) Test five result showing the path only

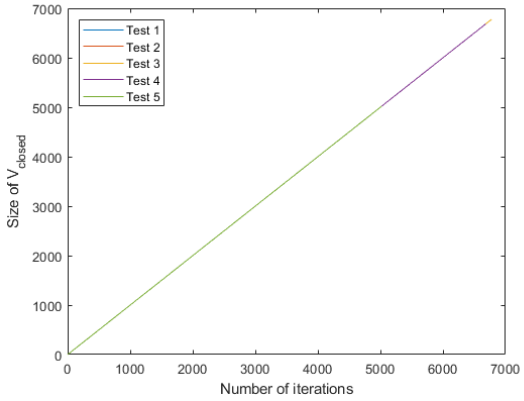
Figure 4.21: TCFMT* results: Test five on obstacle configuration 2. Path found in 2158.2 seconds and with a path cost of 13.7476, decomposed into three sections



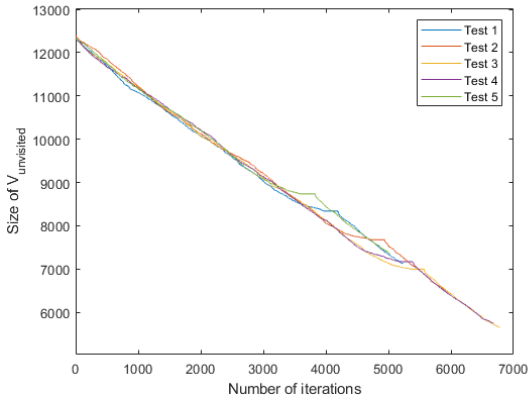
(a) Trends of s_{max} vs. number of iterations



(b) Trends of the size of V_{open} vs. number of iterations



(c) Trends of the size of V_{closed} vs. number of iterations



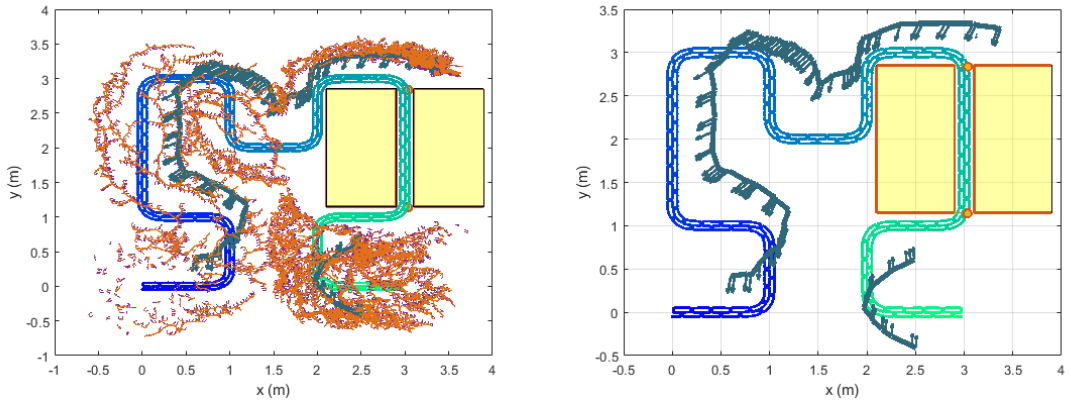
(d) Trends of the size of $V_{unvisited}$ vs. number of iterations

Figure 4.22: TCFMT* results of the five tests for obstacle configuration 2

Figure 4.20 and Figure 4.21 showcased test four and five on configuration two. We can see that the task is successfully decomposed on the two sides of the gap. The printing task section hidden by the obstacles is successfully ignored and path planning continues after skipping the section. However, there is a small section of path planned around $(x = -0.5, y = 0)$ and is isolated from the main path. The reason for this is the same as the one mentioned in the previous experiment.

The trends shown in Figure 4.22 shows a similar result to that of obstacle configuration 1, in the way that the characteristics of the curves are the same when obstacles are encountered. A slight difference in this experiment compared to the first one is that some tests finished faster than the others. For example, in Figure 4.22a we can see that test five is the fastest while test three is the slowest. To illustrate, we can see that test four in Figure 4.20a the tree explored well into the left side of the printing task ($x \leq 0$), which takes time and so the overall increase of progress s_{max} is slower. In contrast, as shown in Figure 4.21a, the mentioned region is not explored at all. Therefore, test five is faster than test four. The reason for this is due to the task consistency constraints. This will be further discussed in Section 4.3.

TCRRT* Result:



(a) Test one result showing the tree and the path

(b) Test one result showing the path only

Figure 4.23: TCRRT* results: Test one on obstacle configuration 2. Path found in 31268 seconds and with a path cost of 10.4026, decomposed into three sections.

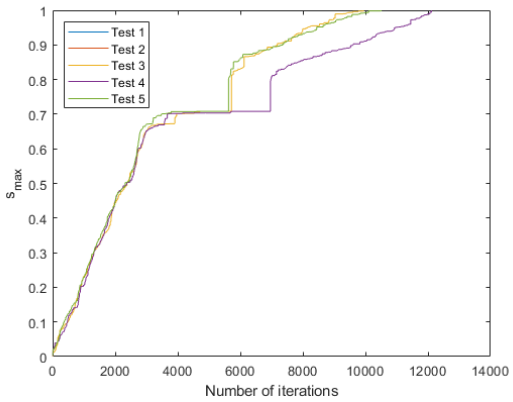


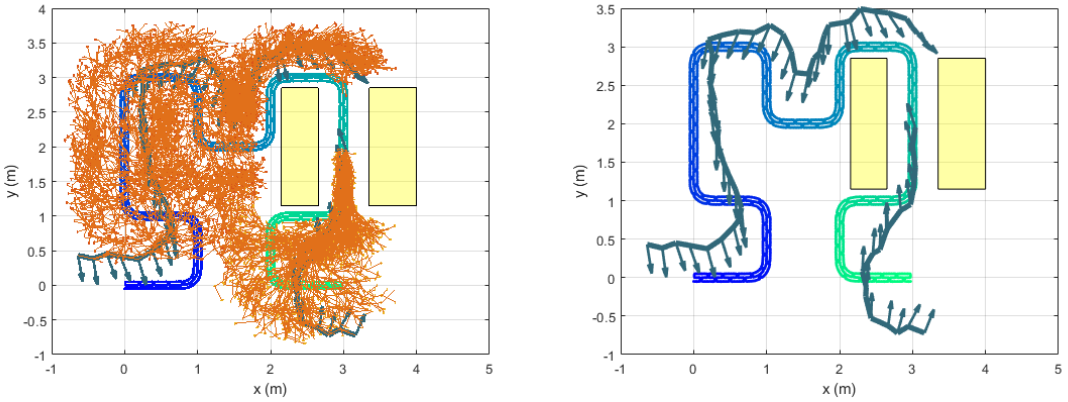
Figure 4.24: TCRRT* results: s_{max} vs. iterations of the five tests for obstacle configuration 2

In the case of TCRRT*, all results have successfully decomposed the task into two complete sections as shown in Figure 4.23. The points of decomposition are located at the openings of the gap, which correctly notify TCRRT* to start exploring new paths at sensible locations. However, a huge load of effort was spent to look for the point of decomposition in the first path and exploring the second path, which is not able to proceed at all. Looking into the position around $(x = 3.2, y = 3.3)$, we can see a tiny bit of path planned for the second section. In Figure 4.24, we can notice that TCRRT* spent more than 1000 iterations to search for the first and the second point of decomposition at around $s_{max} = 0.71$. After that, the planner was terminated and start planning the third section at $s_{max} = 0.81$. The reason why the algorithm is spending this much time is that it did not know that the second section was in fact unprintable. This suggests a future development to validate if a section of the task is printable or not.

Obstacle Configuration 3: Long tunnel near sharp turning (section printable between obstacles)

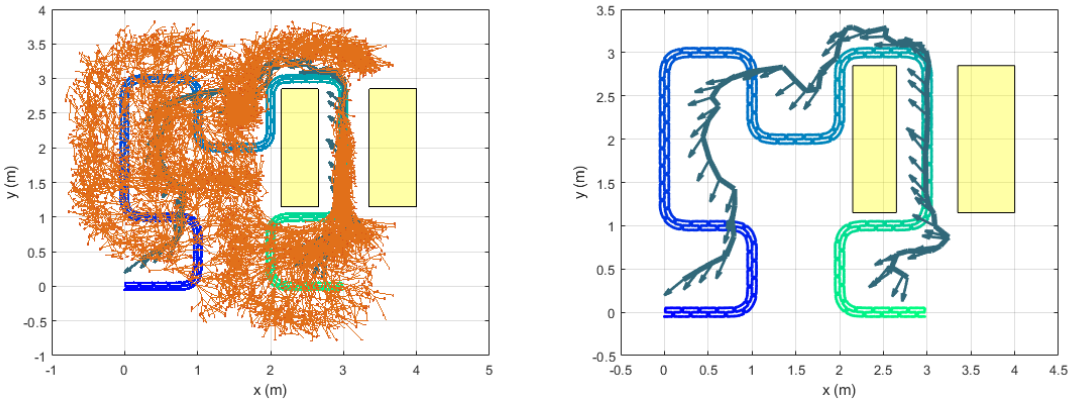
Expected goal:
In configuration 3, since we have one gap (0.7m) that is wide enough for the robot to pass through with any orientations, one path is expected to be produced by the algorithms and no decomposition should occur.

TCFMT* Result:



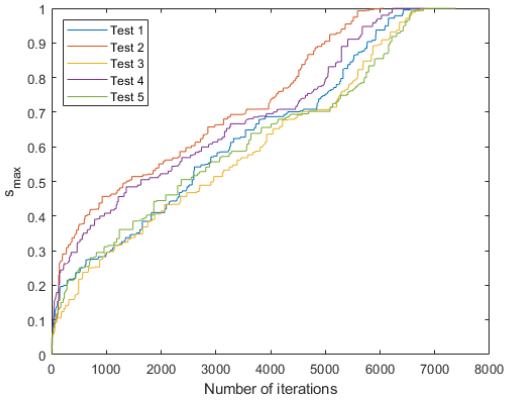
(a) Test one result showing the tree and the path (b) Test one result showing the path only

Figure 4.25: TCFMT* results: Test one on obstacle configuration 3. Path found in 1432.9 seconds and with a path cost of 12.4366, decomposed into two sections

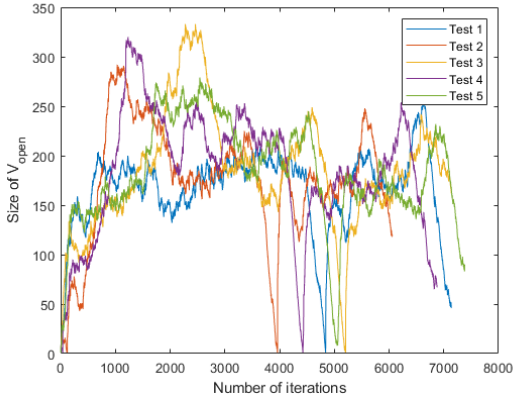


(a) Test five result showing the tree and the path (b) Test five result showing the path only

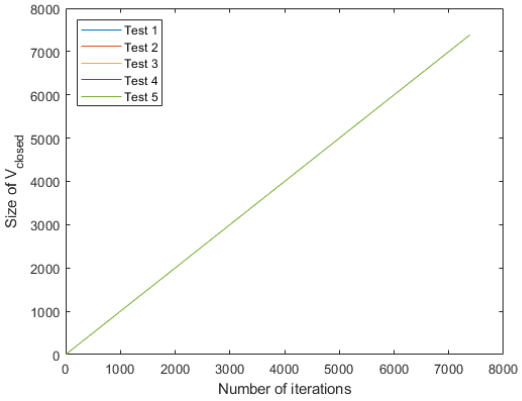
Figure 4.26: TCFMT* results: Test five on obstacle configuration 3. Path found in 1390.2 seconds and with a path cost of 12.8242, decomposed into three sections



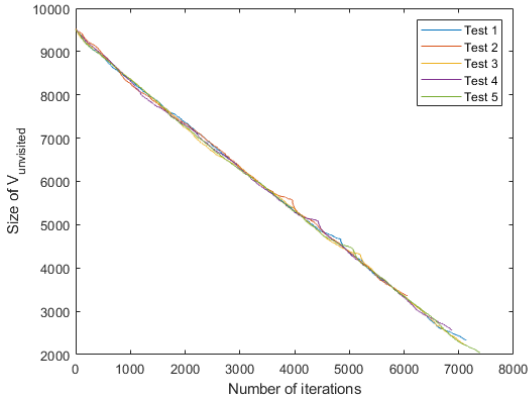
(a) Trends of s_{max} vs. number of iterations



(b) Trends of the size of V_{open} vs. number of iterations



(c) Trends of the size of V_{closed} vs. number of iterations



(d) Trends of the size of $V_{unvisited}$ vs. number of iterations

Figure 4.27: TCFMT* results of the five tests for obstacle configuration **3**

The results of test one and test five are displayed in Figure 4.25 and Figure 4.26 respectively. We can see that a complete path from the start to the end is formed for test five as shown in Figure 4.26b. However, the path is decomposed in test one as we can see in Figure 4.25b.

Together with Figure 4.27, we can see all the indications that implies the trigger of the task decomposition mechanism in Figure 4.27a, 4.27b and 4.27d in test one to four. Together with the result presented in Figure 4.25 and 4.26, this experiment reveals a limitation of the TCFMT* algorithm on getting the robot into narrow regions and suggested a possible future improvement on finding feasible configurations to connect in such scenarios.

Looking into the orientations (arrows) of the planned path in Figure 4.25b, we can see that the last point of the first section of the path at around $(x = 3.2, y = 3.2)$ and the first point of the second section of the path at around $(x = 3, y = 2)$ have a huge difference in orientation. This suggests that the algorithm cannot find a path so that the robot can drift into the space between the obstacles. If we look at the entrance of the tunnel around $(x = 3, y = 3)$, we can see that the printing task can block the robot from entering the tunnel if the robot is placed behind the printing. This is further illustrated in Figure 4.28.

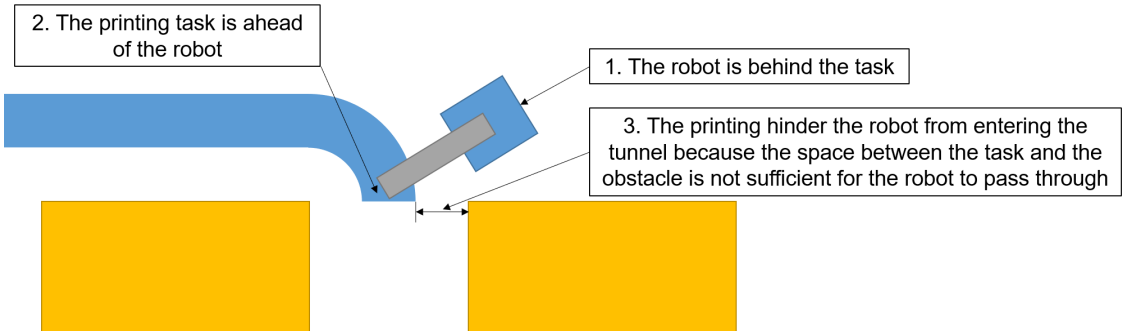


Figure 4.28: Illustration showing the scenario that the robot is printing from behind the task (blue) and cannot get into the tunnel because the obstacles (yellow) together with the task block the robot from entering it, which forced a decomposition.

To get into the tunnel, the algorithm needs to find a sequence of points such that the robot is ahead of the printing task as shown in Figure 4.29, which is also the result shown in Figure 4.26b This reveals that the pre-sampled points may not be able to provide a sequence of points that allows the transition of orientation to help the robot get into the tunnel without colliding with anything.

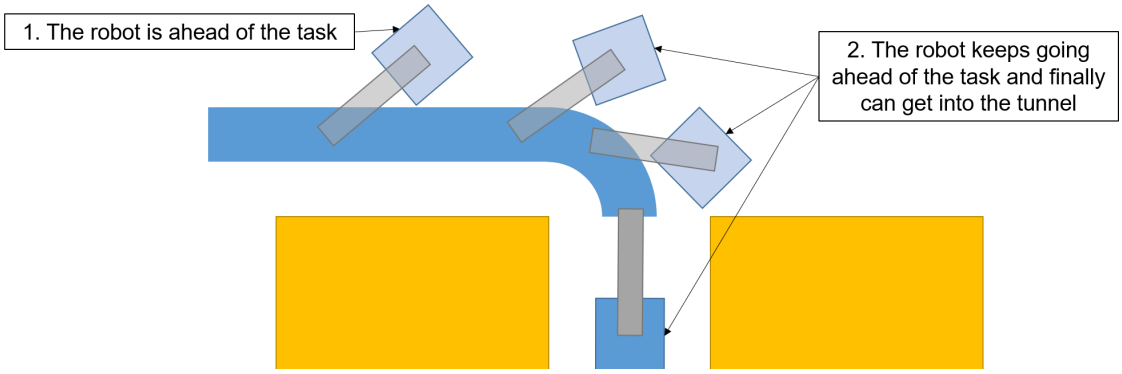
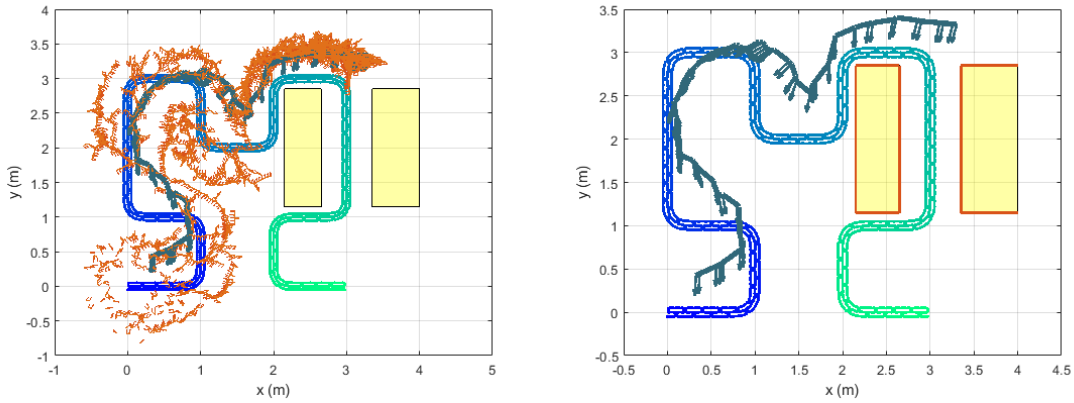


Figure 4.29: Illustration showing the scenario that the robot is printing in front of the task (blue) for a certain section before getting into the tunnel between the obstacles (yellow)

This is also be seen from the tree. In Figure 4.25a we can see that there is no exploration at all into the tunnel from the first section. This indicates that the algorithm cannot find a suitable configuration from the sampled points at the beginning of the tunnel ($x = 2.8, y = 3$). Therefore, a new tree is initialized at around ($x = 3, y = 2$). In contrast, in test five, since the robot has already printing ahead of the task at some point before encountering the obstacles (around $x = 2.5, y = 3$), a compatible configuration can bring the robot into the tunnel without collision can be found easier.

TCRRT* Result:



(a) Test three result showing the tree and the path (b) Test three result showing the path only

Figure 4.30: TCRRT* results: Test three on obstacle configuration **3**. Complete path search unsuccessful

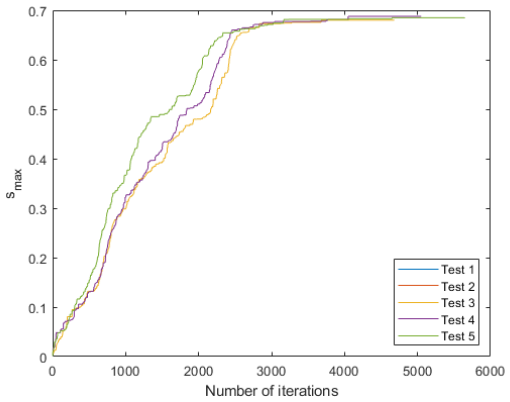


Figure 4.31: TCRRT* results: s_{max} vs. iterations of the five tests for obstacle configuration **3**

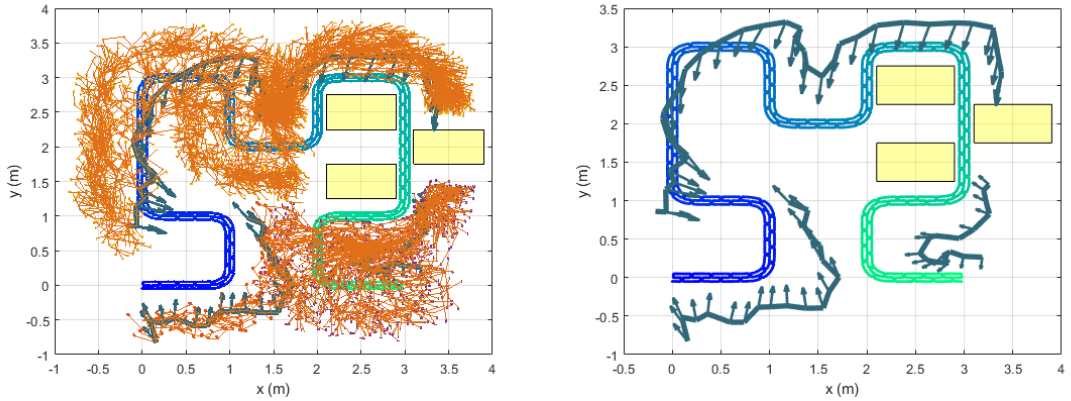
All tests with TCRRT* have consistently failed to provide a path that navigates the robot to print the full task in obstacle configuration 3. Figure 4.30 shows that the tree and the path ends at around $(x = 3.3, y = 3.4)$. There is no point of decomposition calculated because the obstacles are more than one size of the robot apart (the diagonal length of the robot). Moving to Figure 4.31, we can observed that all the curves are upper-bounded by $s_{max} = 0.7$. This reveals that there would be around 30% of the task left unprinted if we use the result from TCRRT*. There are two reasons why TCRRT* stopped planning. The first one is the same as the one mentioned in the TCFMT* section of this experiment. This implies TCRRT* also find it difficult to navigate a robot into narrow regions with correct orientations. The second one is because the decomposition procedures did not treat the evolving task as an obstacle. So, it failed to realize that the entry of the tunnel was indeed a difficult area for the planner which may need a task decomposition. This will be further discussed in Section 4.3.

In this case, TCFMT* may be slightly better than TCRRT* with the proposed decomposition procedures because TCFMT* will always find another point to start a new tree provided that there are valid points in $V_{unvisited}$. Although this may result in task decompositions that we may not want to happen, the printing task will still be able to carry out.

Obstacle Configuration 4: Three boxes case

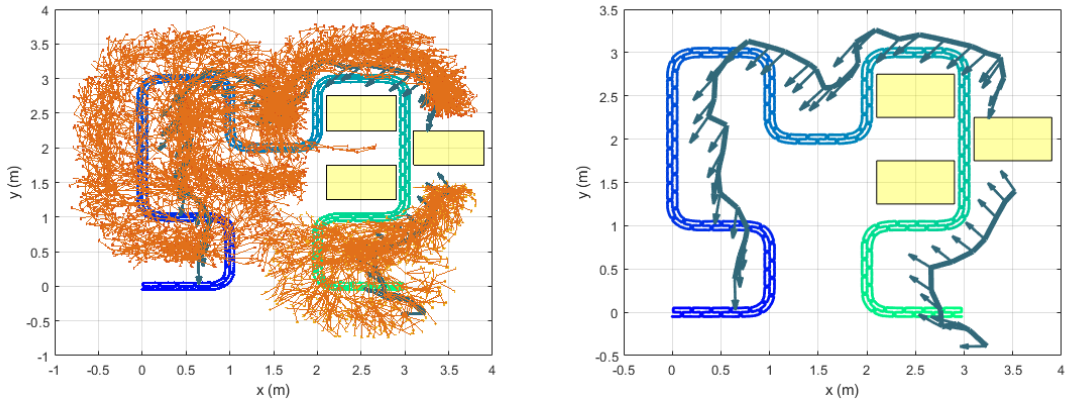
Expected goal:
In configuration 4, as mentioned in Section 4.1.2, we expect the algorithms to return two paths in the optimal case, with the possibility of getting three paths.

TCFMT* Result:



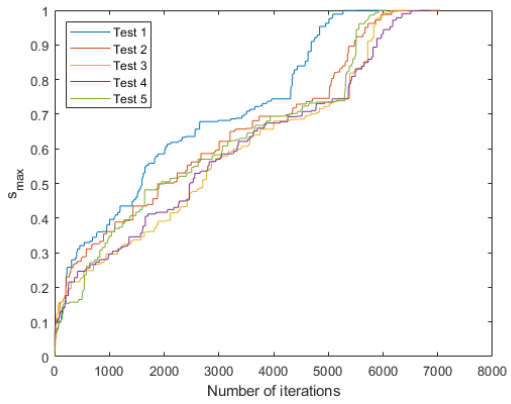
(a) Test five result showing the tree and the path (b) Test five result showing the path only

Figure 4.32: TCFMT* results: Test one on obstacle configuration 4. Path found in 1150.9 seconds and with a path cost of 15.9652, decomposed into three sections

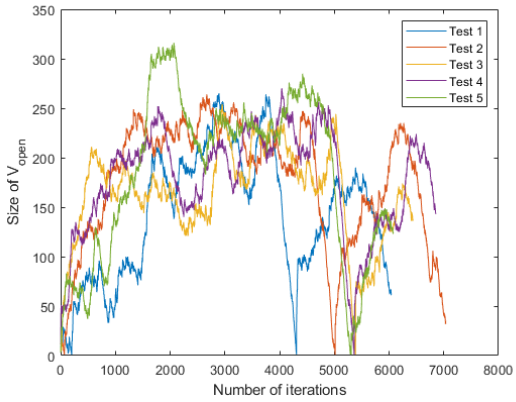


(a) Test four result showing the tree and the path (b) Test four result showing the path only

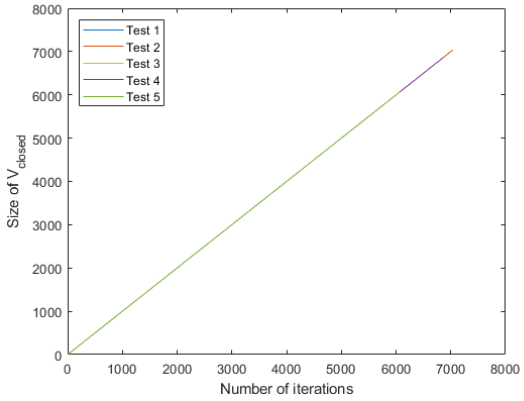
Figure 4.33: TCFMT* results: Test four on obstacle configuration 4. Path found in 1290.7 seconds and with a path cost of 10.4335, decomposed into two sections



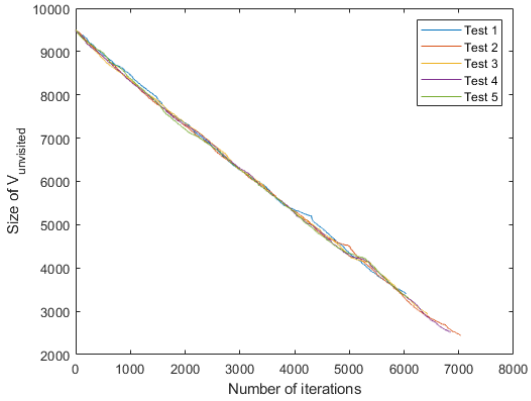
(a) Trends of s_{max} vs. number of iterations



(b) Trends of the size of V_{open} vs. number of iterations



(c) Trends of the size of V_{closed} vs. number of iterations

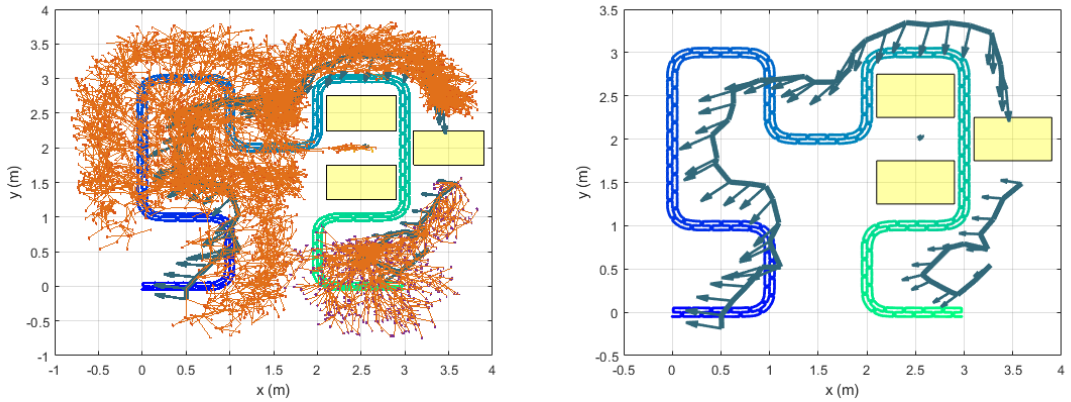


(d) Trends of the size of $V_{unvisited}$ vs. number of iterations

Figure 4.34: TCFMT* results of the five tests for obstacle configuration 4

Figure 4.32 and 4.33 shows the result of test one and four of the experiment on configuration 4. Again, we can see that the decomposition mechanism is successfully triggered around the second block of obstacle with reference to Figure 4.6 and two sections of paths are produced in test four. However, there is a false positive trigger of task decomposition at around $(x = 1.25, y = 1)$ which resulted in the third section of the path. Again, the reason for this is mentioned in the result of obstacle configuration 1. Moving to Figure 4.34, we can observe a similar result when TCFMT* meets obstacles around 4300^{th} to 5400^{th} iteration for all the five tests.

Out of the five tests, there is one test result that happened to have three valid decompositions as shown in Figure 4.35 below:



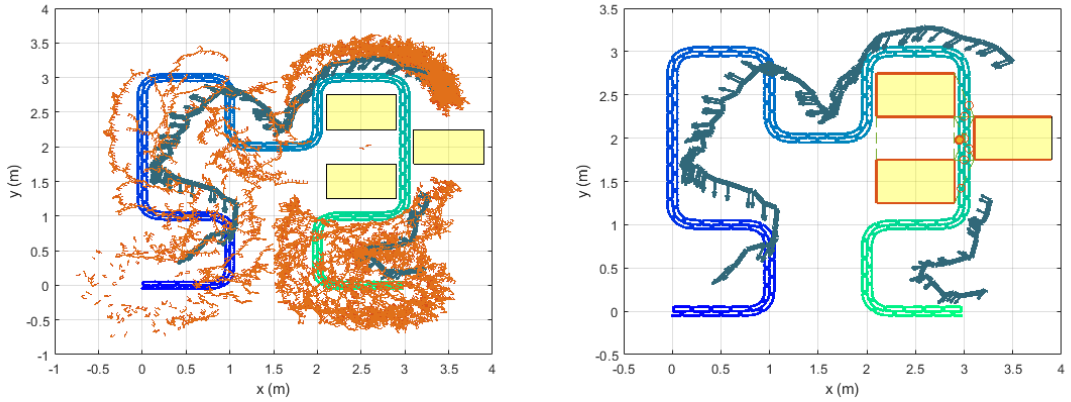
(a) Test five result showing the tree and the path

(b) Test five result showing the path only

Figure 4.35: TCFMT* results: Test five on obstacle configuration 4. Path found in 1105.9 seconds and with a path cost of 11.8777, decomposed into three sections

We can see the very short second section is located at around $(x = 2.5, y = 2)$. Although this is completely normal from the perspective of the algorithm, we actually do not want this to happen. The reason for this is because the region is unreachable by the robot. To further elaborate, the printed task at around $(x = 2, y = 2)$ will block the robot from entering the region. Although we can allocate that part of the task to the robot first, there may still be cases that the region is not reachable. An example would be sticking a wall to the two boxes on the left side of the three boxes case. This suggested a limitation of the algorithm which needs assistance from the task allocation module mentioned in Section 2.3.

TCRRT* Result:



(a) Test one result showing the tree and the path

(b) Test one result showing the path only

Figure 4.36: TCRRT* results: Test one on obstacle configuration 4. Path found in 4992.7 seconds and with a path cost of 13.9108, decomposed into two sections

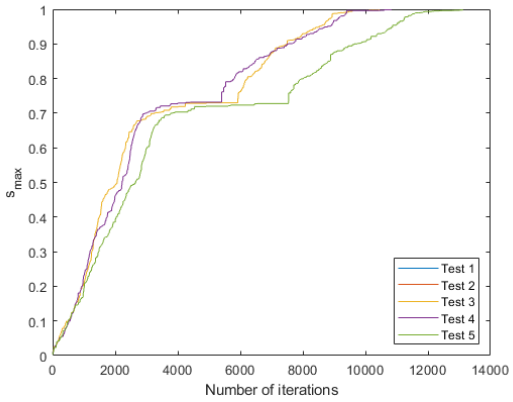


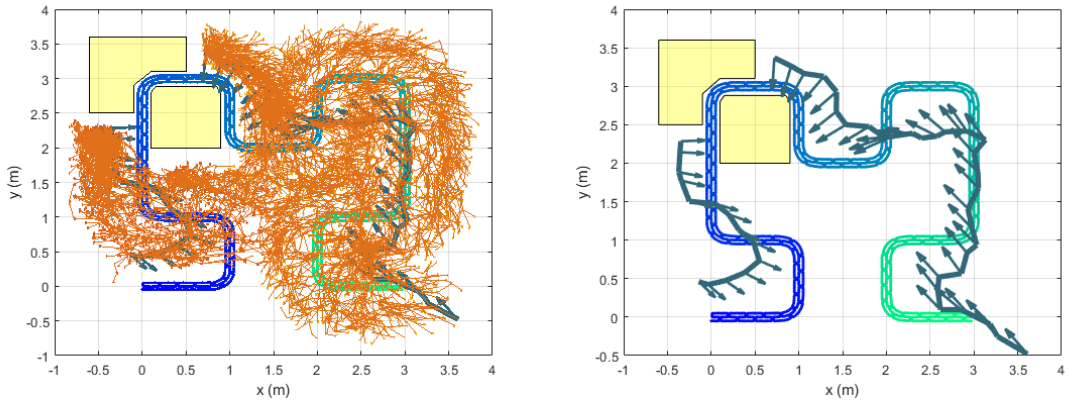
Figure 4.37: TCRRT* results: s_{max} vs. iterations of the five tests for obstacle configuration 4

In the case of TCRRT*, a point of decomposition is successfully identified at around ($x = 3, y = 2$) and two sections of paths are successfully planned in all five tests as shown in Figure 4.36. Figure 4.37 shows the rising trends of s_{max} in all five tests from 0 to 1. However, similar to previous experiments, the progress has been held for many iterations around $s_{max} = 0.73$ because TCRRT* struggle to reach the point of decomposition and the reason for this is mentioned in obstacle configuration 1.

Obstacle Configuration 5: Non-Convex obstacles

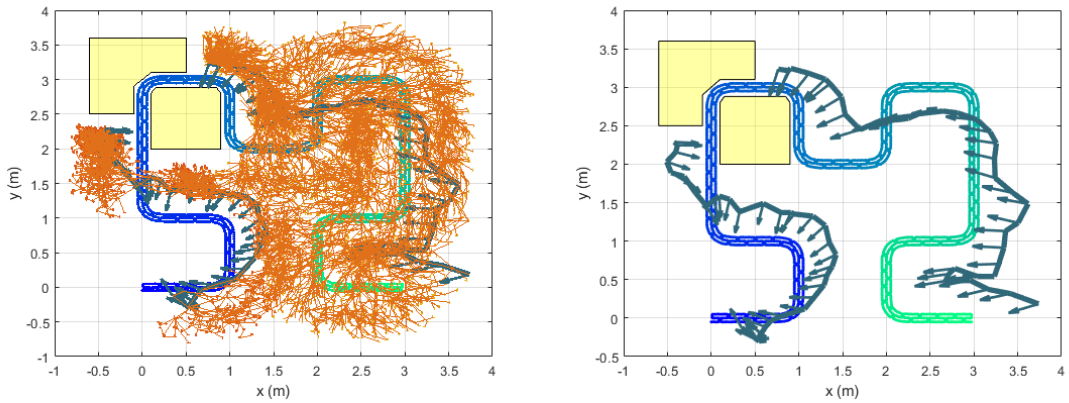
Expected goal:
In this experiment, there is a small section of the printing task covered up by the obstacles, so the algorithms are expected to skip that portion and produce two separated paths in total.

TCFMT* Result:



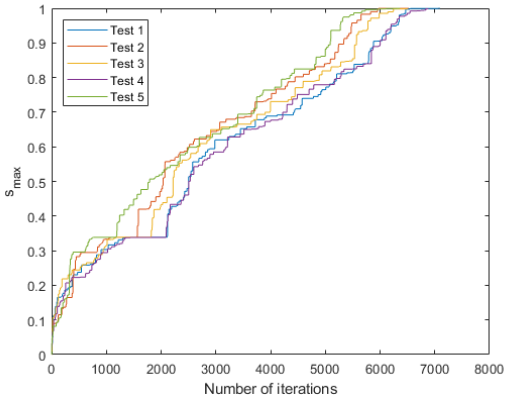
(a) Test one result showing the tree and the path (b) Test one result showing the path only

Figure 4.38: TCFMT* results: Test one on obstacle configuration 5. Path found in 1440.1 seconds and with a path cost of 11.3393, decomposed into two sections

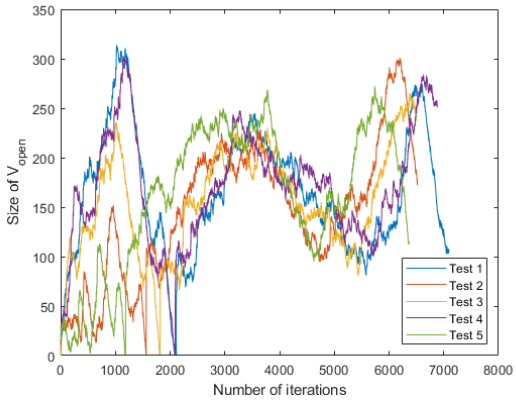


(a) Test two result showing the tree and the path (b) Test two result showing the path only

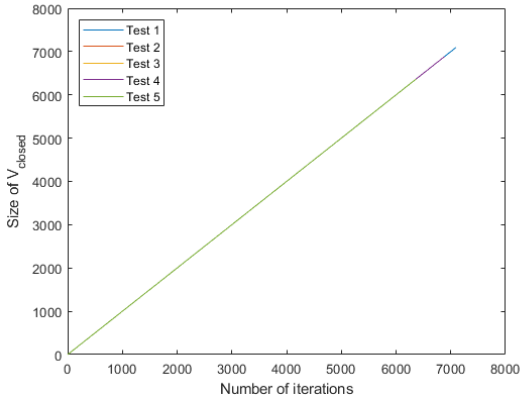
Figure 4.39: TCFMT* results: Test two on obstacle configuration 5. Path found in 1388.4 seconds and with a path cost of 14.977, decomposed into three sections



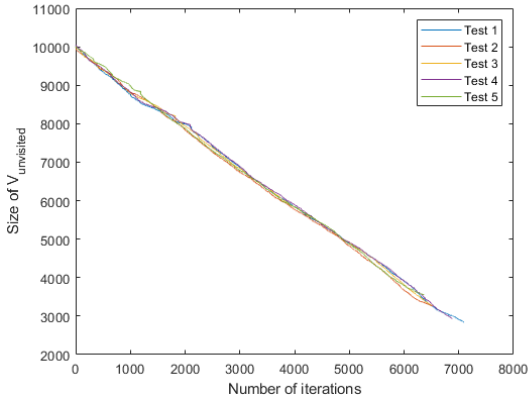
(a) Trends of s_{max} vs. number of iterations



(b) Trends of the size of V_{open} vs. number of iterations



(c) Trends of the size of V_{closed} vs. number of iterations



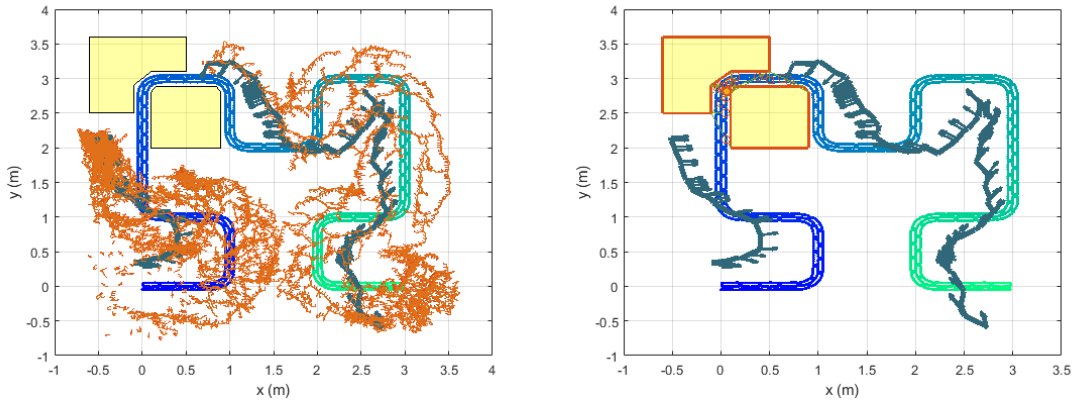
(d) Trends of the size of $V_{unvisited}$ vs. number of iterations

Figure 4.40: TCFMT* results of the five tests for obstacle configuration 5

Figure 4.38 and 4.39 shows the results of test one and two of the experiment on configuration 5. Similar to the results before, two paths are successfully produced in each test and the decomposition mechanism is successfully triggered when TCFMT* encountered the obstacles.

In this set of experiments, there is no unintended decomposition. However, from the record of the past experiments, we believe it is just a coincidence that there is no false positive decomposition five times in a row and it remains an issue that needs to be resolved.

TCRRT* Result:



(a) Test three result showing the tree and the path (b) Test three result showing the path only

Figure 4.41: TCRRT* results: Test three on obstacle configuration 5. Path found in 11232 seconds and with a path cost of 13.4886, decomposed into two sections

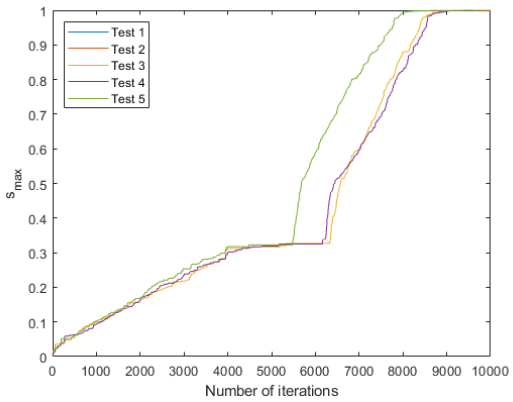


Figure 4.42: TCRRT* results: s_{max} vs. iterations of the five tests for obstacle configuration 5

In configuration 5, similar to the previous experiment, a point of decomposition is calculated and two paths are successfully planned in each test. We can see that the decomposition procedures do work obstacles configurations that involve non-convex shapes as displayed in Figure 4.41. However, a significant amount of time was used to reach the point of decomposition at around $s_{max} = 0.33$. Moreover, although there is no unprintable part, in this case, there may be cases that the procedures will fail. The reason for this is because the procedure assumes the obstacles are defined by their corners. This shows another limitation of the proposed TCRRT* task decomposition procedures and will be further discussed in Section 4.3.

4.2.3 Comparison of TCFMT* and TCRRT*

In this section, we will compare the performance of TCFMT* and TCRRT* using the measurements mentioned in Section 4.1.2. First of all, the resultant statistics of TCFMT* and TCRRT* are presented in mean ± standard deviation in Table 4.4 and Table 4.5 respectively.

Table 4.4: TCFMT* Experimental results on the five obstacle configurations

Obstacle Configuration	1	2	3	4	5
Time (s)	1599.947 ± 68.396	2483.687 ± 216.063	1386.337 ± 75.179	1208.722 ± 81.966	1446.310 ± 60.619
Iterations	7138.600 ± 300.883	5944.600 ± 805.636	6921.000 ± 513.258	6489.400 ± 449.363	6673.400 ± 303.635
Path total cost	12.979 ± 2.186	11.642 ± 1.258	12.777 ± 1.204	12.611 ± 2.129	13.047 ± 2.672
Number of paths	3.600 ± 0.894	2.600 ± 0.894	2.000 ± 0.707	2.600 ± 0.548	2.000 ± 0.000
Total number of path nodes	59.400 ± 9.072	54.200 ± 4.604	57.600 ± 3.647	56.400 ± 9.397	56.800 ± 11.777
Mean size of the changing radius (m)	0.303 ± 0.000	0.302 ± 0.000	0.303 ± 0.000	0.303 ± 0.000	0.302 ± 0.000
Standard deviation of the changing radius (m)	0.034 ± 0.005	0.019 ± 0.003	0.030 ± 0.004	0.031 ± 0.012	0.020 ± 0.005

Table 4.5: TCRRT* Experimental results on the five obstacle configurations

Obstacle Configuration	1	2	3	4	5
Time (s)	9476.785 ± 251.504	52296.564 ± 20131.351	2034.827 ± 496.680	5105.629 ± 1016.676	9540.640 ± 1725.864
Iterations	15656.400 ± 240.450	10924.200 ± 1105.892	5022.800 ± 395.963	11130.000 ± 1123.043	9763.000 ± 216.769
Path total cost	13.772 ± 0.058	11.715 ± 1.222	8.824 ± 0.970	13.348 ± 0.808	13.909 ± 0.393
Number of paths	3.000 ± 0.000	3.000 ± 0.000	1.000 ± 0.000	2.000 ± 0.000	2.000 ± 0.000
Total number of path nodes	110.800 ± 1.643	135.600 ± 13.372	102.200 ± 16.223	139.200 ± 11.606	141.200 ± 7.429

From the tables above, we can see that TCFMT* can finish the path planning much faster than TCRRT*. The reason for this is the same as mentioned before - TCRRT* spent much time on finding the points of decomposition, which are sometimes unreachable at all. This also explains why there are much more iterations in TCRRT* across all the experiments.

Moving to path total cost, although TCFMT* has higher standard deviations, it produces a more optimal path than TCRRT* as the costs are universally lower except obstacle configuration 3. However, this is due to the failure of TCRRT* on producing a valid path from the start to the end for all five tests. Concerning the number of paths TCRRT* did a better job in the sense that there is no unintended task decomposition that leads to extra paths, whereas there exist some false positive task decomposition in the case of TCFMT*. Regarding the total number of path nodes, since we are using different searching radius for TCFMT* and TCRRT*, a direct comparison will not be fair. But since TCRRT* can use a smaller radius, the path generated are smoother in general since there are more nodes in the paths. Lastly, the changing radius of TCFMT* has a mean around 0.303m in general and a minor fluctuation of about 2 - 3cm, which are very closed to the value that we determined in Section 4.2.1. This means the algorithm is comfortable with this size of the neighbour searching radius.

To sum up, TCFMT* has a better performance overall in the sense that it can produce valid,

lower-cost paths in a shorter period of time. Although TCFMT* suffers false-positive decompositions and sampling issues, TCRRT* will fail to provide a complete path whenever it cannot find a point with further progress under 1000 trials, which is a more serious drawback.

4.3 Limitations and Difficulties

4.3.1 Limitations and Difficulties Encountered in TCFMT*

False Positive Task Decompositions

Throughout the experiments, we can observe unintended decomposition happens from time to time. This indicates that the algorithm cannot always find a point for connections. To further elaborate, since we have an elastic neighbour searching radius, finding a neighbour is not a problem. The problem is the points pre-sampled cannot provide the algorithm to find a neighbour such that there is no collision with anything and at the same time satisfied the task consistency constraint and has a larger progress value s for the connection. So, V_{open} was eventually emptied and a new tree is started. This shows a limitation of the method proposed.

Limited Explorations

In all the experiments, we can observe that the TCFMT* tree tends to search for samples on one side of the task until there is a turning. For example, in Figure B.9, B.11 and B.15, we can see that the trees start and stay on the left side of the beginning section of the task (around $x = 0.5, y = 0.5$). On the other hand, in Figure B.6, B.10 and B.16, the trees stay on the right side of the task (around $x = 0.5, y = -0.5$).

The reason why it is not exploring on the other side of the task actually boils down to the task consistency imposed on sampling. With task consistency constraints, the base poses we sampled all have their orientations approximately pointing towards the task, which means that the orientations of the points on both sides of the task are pointing approximately at opposite directions. If the tree starts around the boundary of the compatible base region and is on either side of the task (which are the cases in the mentioned example above), jumping to the other side of the task will be difficult even an edge without collision is found. This is because searching neighbours with our task consistency distance metrics will upper-bound the change in both positions and orientations. So, a huge change in orientation is not permitted. Therefore, with task consistency constraints, points on the same side of the task are a more feasible choice for the algorithm to make progressive connections in such cases. This happens until there is a turning of the task that put the robot in a position that moving to the other side of the task is feasible. This suggests a further development to extend the algorithm with multiple starting points and will be further discussed in Chapter 6.

Pre-sampling: Lack of Flexibility

In obstacle configuration 3, we have shown the reason why a decomposition occurred in test one. This result suggests a limitation that is grounded in TCFMT*. Since we have all the base poses pre-sampled, what TCFMT* actually trying to do is to explore a fixed set of poses and find a continuous path to navigate the robot to the goal. This means if no pose in the set can lead

to a possible connection with the existing tree, the algorithm will fail to provide such a path. Compared TCFMT* with TCRRT*, TCRRT* sample poses when the algorithm is building the exploring tree. So, TCRRT* has the freedom to sample until a feasible pose is found. In contrast, TCFMT* has limited options to connect. This suggests a development of adaptive sampling when such scenarios are encountered and will be further discussed in Chapter 6.

Unreachable Regions

In obstacle configuration 4, we have shown that the algorithm can be paths planned in regions where it can regardless of whether it is reachable by the robot or not. Since the method proposed will only work on decomposing the tasks, this situation has to be handled by the task allocation module mentioned in Section 2.3. In this way, the unreachable paths will not be distributed to the robots.

4.3.2 Limitations and Difficulties Encountered in TCRRT*

With the experiments, the following limitations has been discovered for the task decomposition procedure.

Task Shape and Evolvement: Not taken into account

In obstacle configuration 1, we have shown that the point of decomposition is unreachable. This is because the task decomposition procedure just blindly merging compatible base regions that were overlapped for more than 50%. So the procedures do not know that the printing task can block the robot from reaching the resultant point of decomposition. To elaborate, Before the robot trying to reach the point of decomposition, it has already printed a part of the printing task that will block its way from printing getting onto the decomposition point. This significantly dragged the efficiency of the overall algorithm because the TCRRT* will have to trial and error 1000 times before proceeding to the next section of path planning.

In obstacle configuration 3, although the tunnel was designed to allow the robot to pass through, TCRRT* still cannot plan the path into the tunnel. On one hand, it is because a valid configuration cannot be sampled. On the other hand, it is because the task decomposition procedure did not realize that the entrance of the tunnel is a point that decomposition is needed. If we take the evolving printing task into account as an obstacle, the procedure should be able to realize that the gap was indeed a narrow one for the robot and decomposition should occur.

Limitation on the variety of obstacles

In obstacle configuration 5, we have raised that some obstacle configurations cause failure in providing a valid point of decomposition. In the task decomposition procedures, we consider all vertices and edges of the obstacles around the printing task and merge rough breakpoints that have their compatible base region highly overlapped. In this regard, obstacles can be structured in a way that the points of decomposition end up in places that the robot is not reachable. Such an example is shown in Figure 4.43.

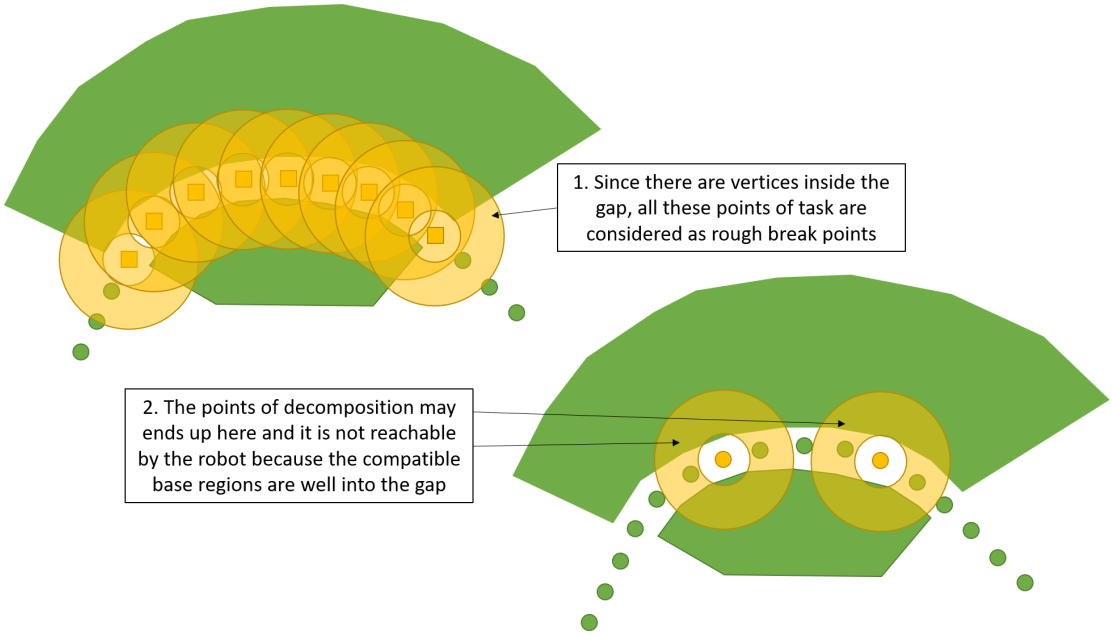


Figure 4.43: An scenario showing an obstacle configuration with the points of decomposition in places where the robot cannot reach

If the obstacles in Figure 4.43 are defined by the corner vertices, then the points of decomposition should be at the openings of the gap. However, this is under the assumption that curves and edges are continuous pieces, which is not true in many cases. So the set of task decomposition procedures actually has very limited applications.

4.4 Summary

In this chapter, the evaluation method was first introduced, the parameters for TCFMT* was determined. The results of using TCFMT and TCRRT* with task decomposition procedures are also presented with five obstacle configuration cases. Limitations and difficulties of pre-sampling and the variety of obstacles in TCFMT* and TCRRT* decomposition procedures are also discussed.

Chapter 5

Conclusion

5.1 Objective Recap

There are four main objectives for this study. The first objective is to investigate and perform navigation of single mobile manipulator with task consistent path planning. The second objective is to investigate and propose methods for decomposing construction 3D printing tasks. The third objective is to integrate the proposed methods for task decomposition with task consistent path planning for multiple robots to work in parallel. The final objective is to evaluate the feasibility and performance of the integrated systems.

5.2 Major Findings

The integrated systems of task decomposition enabled task consistent path planning, TCFMT* and TCRRT*, which were evaluated by five obstacle configuration experiments. In these experiments, the systems were able to identify obstacles that block robots from finishing the task and plan separated sections of paths. With this, the resultant solution can be distributed for one or more robots to work on. The findings are as below:

- In terms of TCRRT*, since the decomposition procedures did not consider the shape and the evolvement of the printing task and they rely on narrowly scoped assumptions of the obstacles, the overall performance is rather inefficient. This is because the points of decomposition can easily end up in places that are unreachable which may result in unfinished paths.
- In terms of TCFMT*, the task decomposition mechanism was successfully triggered around the obstacles throughout the experiments. However, there were occurrences of false-positive decomposition. This suggests future improvement on finding feasible configurations when entering narrow regions, exploring with more than one starting point and adaptive sampling. These will be further discussed in Chapter 6.
- The statistics of TCRRT* and TCFMT* have shown that TCFMT* has a better overall performance such that it can return valid, lower-cost paths in a shorter period of time.

All of the above indicates that TCFMT* is a better choice for task decomposition in construction 3D printing with suitable future developments.

5.3 Conclusions

In this study, the literature of recent technologies in construction 3D printing, task decomposition, path planning and task consistent path planning were reviewed. The need for task decomposition around obstacles is identified. Then, RRT* and FMT* were selected to extend with task consistency constraints due to their optimality in relinking points and the efficiency in dealing with points in unreachable regions respectively.

Moreover, two methods of task decomposition for construction 3D printing are proposed and discussed in detail. The methods were then integrated with corresponding task consistent path planners to enable task decomposition. The combined systems were then evaluated by five obstacle configurations.

In general, the objectives of this study were achieved. Some future works have to be conducted to overcome the limitations stated in this study to polish TCFMT*. This study has proven the possibility of task decomposition in construction 3D printing. As the demand for 3D printing in construction continues to expand, the task decomposition method proposed in this study could be a plausible solution to decompose tasks in large scale construction 3D printing with appropriate improvements.

Chapter 6

Future Work

6.1 Two Stage Pose Sampling

From the result of obstacle configuration 3 in Section 4.2.2, the issue of not being able to navigate the robot into narrow passages has triggered unnecessary tasks decomposition, which limited the performance of TCFMT*. Since the ultimate root cause is not being able to find a series of poses that have correct orientations for the robot to enter the region, we can attempt a two-stage sampling scheme. Firstly, we can sample the Cartesian coordinates in the pre-sampling stage. After that, we can determine the orientation of the point that the algorithm trying to connect when it is extending. To elaborate, suppose we are trying to extend the exploring tree from point A to point B and point A has an orientation θ_A. When the algorithm try to do the extension, it use the distance metrics (2.3) and (3.1). With the Cartesian coordinate known, we will have a degree of freedom in the dimension of orientation, which means we can choose the orientation θ_B of point B from a range of orientations. With this, we can turn this into a problem of finding a suitable orientation so that the calculated distance equals to ϵ_{reach} from the range of orientation subject to the task consistent constraint that the orientations have to be pointing toward the task at most θ_{thres} away. If there is no valid orientation under this constraint, we can skip point B. With this, the occurrences of triggering task decomposition in similar scenarios should be minimized.

6.2 Exploring with Multiple Starting Points

In Section 4.3, we have shown that navigating the exploring tree across the printing task is difficult because of the task consistency constraints. This issue limited TCFMT* from providing shorter paths and may lead to false-positive task decompositions. In this regard, extending TCFMT* to have multiple starting points may help mitigate the issue. Using multiple starting points, the exploring tree will be able to grow from multiple locations on both sides of the task at progress $s = 0$. Since TCFMT* will always choose the point that has the lowest cost in V_{open} for every iteration, the higher cost paths will have a higher cost all the way along the exploring process. So, they will automatically phase out at the end. Therefore, using multiple starting points will improve the path searching of TCFMT* and further minimize false-positive task decomposition episodes.

6.3 Adaptive Sampling

We have also discussed in Section 4.3 that there can be no points in the pre-sampled set that can lead to a possible connection with the existing tree. In this regard, allow a limited number of adaptive sampling will also help the algorithm to explore feasible connections. With this, TCFMT* will have a higher chance to explore possible poses and the effect of rigidity in the pre-sampled set of poses can be further diminished. In addition, if the operations of the program can be optimized to be faster, we can choose a larger neighbour searching radius and a higher sampling rate per progress. We believe this will further improve the success rate of finding feasible connections and further reduce the chance of having false-positive decompositions.

Bibliography

[1] *What is additive manufacturing?* [Online]. Available: https://www.ge.com/additive/additive-manufacturing.

[2] M. Chinthavali, "3d printing technology for automotive applications," in *2016 International Symposium on 3D Power Electronics Integration and Manufacturing (3D-PEIM)*, IEEE, 2016, pp. 1–13.

[3] L. Hao, D. Raymond, G. Strano, and S. Dadbakhsh, "Enhancing the sustainability of additive manufacturing," in *5th International Conference on Responsive Manufacturing-Green Manufacturing (ICRM 2010)*, IET, 2010, pp. 390–395.

[4] C.-Y. Liaw and M. Guvendiren, "Current and emerging applications of 3d printing in medicine," *Biofabrication*, vol. 9, no. 2, p. 024 102, 2017.

[5] J. Norman, R. D. Madurawe, C. M. Moore, M. A. Khan, and A. Khairuzzaman, "A new chapter in pharmaceutical manufacturing: 3d-printed drug products," *Advanced drug delivery reviews*, vol. 108, pp. 39–50, 2017.

[6] S. C. Joshi and A. A. Sheikh, "3d printing in aerospace and its long-term sustainability," *Virtual and Physical Prototyping*, vol. 10, no. 4, pp. 175–185, 2015.

[7] S. Lim, R. Buswell, T. Le, S. Austin, A. Gibb, and T. Thorpe, "Developments in construction-scale additive manufacturing processes," *Automation in Construction*, vol. 21, pp. 262–268, 2012, ISSN: 0926-5805. DOI: https://doi.org/10.1016/j.autcon.2011.06.010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0926580511001221.

[8] A. H. Espera, J. R. C. Dizon, Q. Chen, and R. C. Advincula, "3d-printing and advanced manufacturing for electronics," *Progress in Additive Manufacturing*, pp. 1–23, 2019.

[9] A. Vanderploeg, S.-E. Lee, and M. Mamp, "The application of 3d printing technology in the fashion industry," *International Journal of Fashion Design, Technology and Education*, vol. 10, no. 2, pp. 170–179, 2017.

[10] F. M. Mwema and E. T. Akinlabi, "Basics of fused deposition modelling (fdm)," in *Fused Deposition Modeling*, Springer, 2020, pp. 1–15.

[11] R. Pradhan, S. Rahman, A. Qureshi, and A. Ullah, "Chapter 12 - biopolymers: Opportunities and challenges for 3d printing," in *Biopolymers and their Industrial Applications*, S. Thomas, S. Gopi, and A. Amalraj, Eds., Elsevier, 2021, pp. 281–303, ISBN: 978-0-12-819240-5. DOI: https://doi.org/10.1016/B978-0-12-819240-5.00012-2. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128192405000122.

[12] *The advantages and disadvantages of fdm*. [Online]. Available: https://www.cs.cmu.edu/~rapidproto/students.98/susans/project2/pros.html.

[13] *3d printers for aerospace industry*. [Online]. Available: https://purpleplatypus.com/resources/industries/aerospace/.

[14] B. Khoshnevis, "Automated construction by contour crafting—related robotics and information technologies," *Automation in Construction*, vol. 13, no. 1, pp. 5–19, 2004, The best of ISARC 2002, ISSN: 0926-5805. DOI: https://doi.org/10.1016/j.autcon.2003.08.012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0926580503000736.

[15] X. Zhang, M. Li, J. H. Lim, Y. Weng, Y. W. D. Tay, H. Pham, and Q.-C. Pham, "Large-scale 3d printing by a team of mobile robots," *Automation in Construction*, vol. 95, pp. 98–106, 2018, ISSN: 0926-5805. DOI: https://doi.org/10.1016/j.autcon.2018.08.004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0926580518304011.

[16] J. Pegna, "Exploratory investigation of solid freeform construction," *Automation in Construction*, vol. 5, no. 5, pp. 427–437, 1997, ISSN: 0926-5805. DOI: https://doi.org/10.1016/S0926-5805(96)00166-5. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0926580596001665.

[17] S. Lim, R. A. Buswell, P. J. Valentine, D. Piker, S. A. Austin, and X. De Kestelier, "Modelling curved-layered printing paths for fabricating large-scale construction components," *Additive Manufacturing*, vol. 12, pp. 216–230, 2016, Special Issue on Modeling Simulation for Additive Manufacturing, ISSN: 2214-8604. DOI: https://doi.org/10.1016/j.addma.2016.06.004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214860416301154.

[18] E. Dini. (Accessed 2021). "D-shape," [Online]. Available: https://d-shape.com/.

[19] G. Cesaretti, E. Dini, X. De Kestelier, V. Colla, and L. Pambaguian, "Building components for an outpost on the lunar soil by means of a novel 3d printing technology," *Acta Astronautica*, vol. 93, pp. 430–450, 2014, ISSN: 0094-5765. DOI: https://doi.org/10.1016/j.actaastro.2013.07.034. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0094576513002889.

[20] D. Ozdemir. (2021). "Germany's first 3d-printed residential building is near completion," [Online]. Available: https://interestingengineering.com/germanys-first-3d-printed-residential-building-is-near-completion.

[21] R. van Woensel, T. van Oirschot, M. J. H. Burgmans, P. D. Masi Mohammadi, and K. Hermans, "Printing architecture: An overview of existing and promising additive manufacturing methods and their application in the building industry," *The International Journal of the Constructed Environment*, vol. 9, no. 1, pp. 57–81, 2018. DOI: 10.18848/2154-8587/cgp/v09i01/57-81. [Online]. Available: https://doi.org/10.18848/2154-8587/cgp/v09i01/57-81.

[22] M. T. Souza, I. M. Ferreira, E. Guzi de Moraes, L. Senff, and A. P. Novaes de Oliveira, "3d printed concrete for large-scale buildings: An overview of rheology, printing parameters, chemical admixtures, reinforcements, and economic and environmental prospects," *Journal of Building Engineering*, vol. 32, p. 101 833, 2020, ISSN: 2352-7102. DOI: https://doi.org/10.1016/j.jobe.2020.101833. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352710220334665.

[23] J. M. Gere and B. J. Goodno, *Mechanics of materials. eight edition*, 2013.

[24] A. Albar, M. Chougan, M. J. Al- Kheetan, M. R. Swash, and S. H. Ghaffar, "Effective extrusion-based 3d printing system design for cementitious-based materials," *Results in Engineering*, vol. 6, p. 100 135, 2020, ISSN: 2590-1230. DOI: https://doi.org/10.1016/j.rineng.2020.100135. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2590123020300414.

[25] B. Bayle, J.-Y. Fourquet, and M. Renaud, "From manipulation to wheeled mobile manipulation: Analogies and differences," *IFAC Proceedings Volumes*, vol. 36, no. 17, pp. 97–104, 2003, 7th IFAC Symposium on Robot Control (SYROCO 2003), Wroclaw, Poland, 1-3 September, 2003, ISSN: 1474-6670. DOI: https://doi.org/10.1016/S1474-6670(17)33376-1. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1474667017333761.

[26] *Spot arm: Boston dynamics*, Accessed 2021. [Online]. Available: https://www.bostondynamics.com/spot-arm.

[27] P. Singh and D. Dutta, "Multi-direction slicing for layered manufacturing," *Journal of Computing and Information Science in Engineering*, vol. 1, no. 2, pp. 129–142, Mar. 2001, ISSN: 1530-9827. DOI: 10.1115/1.1375816. eprint: https://asmedigitalcollection.asme.org/computingengineering/article-pdf/1/2/129/5626303/129_1.pdf. [Online]. Available: https://doi.org/10.1115/1.1375816.

[28] N. A. Spielberg, J. Klein, N. Oxman, and S. J. Keating, "Digital construction platform: A compound arm approach," 2014. [Online]. Available: https://dspace.mit.edu/handle/1721.1/91574.

[29] T. P. Iran, T. L. Chung, H. K. Kim, S. B. Kim, and M. S. Oh, "Trajectory tracking of mobile manipulator for welding task using sliding mode control," in *30th Annual Conference of IEEE Industrial Electronics Society, 2004. IECON 2004*, vol. 1, 2004, 407–412 Vol. 1. DOI: 10.1109/IECON.2004.1433345.

[30] M. D. Ngo, N. T. Phuong, V. H. Duy, H. K. Kim, and S. B. Kim, "Control of two wheeled welding mobile manipulator," *International Journal of Advanced Robotic Systems*, vol. 4, no. 3, p. 32, Sep. 2007. DOI: 10.5772/5685. [Online]. Available: https://doi.org/10.5772/5685.

[31] W.-S. Yoo, J.-D. Kim, and S.-J. Na, "A study on a mobile platform-manipulator welding system for horizontal fillet joints," *Mechatronics*, vol. 11, no. 7, pp. 853–868, 2001, ISSN: 0957-4158. DOI: https://doi.org/10.1016/S0957-4158(00)00036-2. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957415800000362.

[32] A. Gawel, H. Blum, J. Pankert, K. Krämer, L. Bartolomei, S. Ercan, F. Farshidian, M. Chli, F. Gramazio, R. Siegwart, M. Hutter, and T. Sandy, "A fully-integrated sensing and control system for high-accuracy mobile robotic building construction," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 2300–2307. DOI: 10.1109/IROS40897.2019.8967733.

[33] S. E. Jenny, E. Lloret-Fritschi, F. Gramazio, and M. Kohler, "Crafting plaster through continuous mobile robotic fabrication on-site," *Construction Robotics*, vol. 4, no. 3-4, pp. 261–271, Nov. 2020. DOI: 10.1007/s41693-020-00043-8. [Online]. Available: https://doi.org/10.1007/s41693-020-00043-8.

[34] N. Hack, T. Wangler, J. Mata-Falcón, K. Dörfler, N. Kumar, A. N. Walzer, K. Graser, L. Reiter, H. Richner, J. Buchli, *et al.*, "Mesh mould: An on site, robotically fabricated, functional formwork," in *Second Concrete Innovation Conference (2nd CIC)*, vol. 19, 2017, pp. 1–10.

[35] G. Dudek, M. Jenkin, E. Milios, and D. Wilkes, "A taxonomy for swarm robots," in *Proceedings of 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '93)*, vol. 1, 1993, 441–447 vol.1. DOI: 10.1109/IROS.1993.583135.

[36] R. De Keyser, "Formation control of ugvs using an uav as remote vision sensor," 2014.

[37] L. Yongshen, Y. Xuerong, Y. Yajun, and P. Shengdong, "Formation control of ugvs based on artificial potential field," in *2018 37th Chinese Control Conference (CCC)*, IEEE, 2018, pp. 6830–6835.

[38] Y. Qi, S. Zhou, Y. Kang, and S. Yan, "Formation control for unmanned aerial vehicles with directed and switching topologies," *International Journal of Aerospace Engineering*, vol. 2016, 2016.

[39] H. Liu, Y. Tian, F. L. Lewis, Y. Wan, and K. P. Valavanis, "Robust formation control for a team of satellites," in *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, IEEE, 2018, pp. 30–35.

[40] B. Das, B. Subudhi, and B. B. Pati, "Cooperative formation control of autonomous underwater vehicles: An overview," *International Journal of Automation and computing*, vol. 13, no. 3, pp. 199–225, 2016.

[41] X. Chen and R. W. Brockett, "Centralized and decentralized formation control with controllable interaction laws," in *53rd IEEE Conference on Decision and Control*, IEEE, 2014, pp. 601–606.

[42] A. Ajorlou, A. Momeni, and A. G. Aghdam, "A class of bounded distributed control strategies for connectivity preservation in multi-agent systems," *IEEE Transactions on Automatic Control*, vol. 55, no. 12, pp. 2828–2833, 2010.

[43] E. Şahin, "Swarm robotics: From sources of inspiration to domains of application," in *Swarm Robotics*, Springer Berlin Heidelberg, 2005, pp. 10–20. DOI: 10.1007/978-3-540-30552-1_2. [Online]. Available: https://doi.org/10.1007/978-3-540-30552-1_2.

[44] M. Bakhshipour, M. Jabbari Ghadi, and F. Namdari, "Swarm robotics search rescue: A novel artificial intelligence-inspired optimization approach," *Applied Soft Computing*, vol. 57, pp. 708–726, 2017, ISSN: 1568-4946. DOI: https://doi.org/10.1016/j.asoc.2017.02.028. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1568494617301072.

[45] N. Lincoln and S. M. Veres, "Decision methods for non-tethered deep space interferometry," *IFAC Proceedings Volumes*, vol. 44, no. 1, pp. 60–65, 2011.

[46] M. Dogar, R. A. Knepper, A. Spielberg, C. Choi, H. I. Christensen, and D. Rus, "Multi-scale assembly with robot teams," *The International Journal of Robotics Research*, vol. 34, no. 13, pp. 1645–1659, Jul. 2015. DOI: 10.1177/0278364915586606. [Online]. Available: https://doi.org/10.1177/0278364915586606.

[47] C. Zhang, A. Hammad, and H. Bahnassi, "Collaborative multi-agent systems for construction equipment based on real-time field data capturing," *Journal of Information Technology in Construction (ITcon)*, vol. 14, no. 16, pp. 204–228, 2009.

[48] S. Kang and E. Miranda, "Planning and visualization for automated robotic crane erection processes in construction," *Automation in Construction*, vol. 15, no. 4, pp. 398–414, 2006, The first conference on the Future of the AEC Industry (BFC05), ISSN: 0926-5805. DOI: https://doi.org/10.1016/j.autcon.2005.06.008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0926580505000713.

[49] K.-L. Lin and C. T. Haas, "Multiple heavy lifts optimization," *Journal of construction engineering and management*, vol. 122, no. 4, pp. 354–362, 1996.

[50] M. A. D. Ali, N. R. Babu, and K. Varghese, "Collision free path planning of cooperative crane manipulators using genetic algorithm," *Journal of computing in civil engineering*, vol. 19, no. 2, pp. 182–193, 2005.

[51] P. á. Sivakumar, K. Varghese, and N. R. Babu, "Automated path planning of cooperative crane lifts using heuristic search," *Journal of computing in civil engineering*, vol. 17, no. 3, pp. 197–207, 2003.

[52] C. Wu, C. Dai, G. Fang, Y.-J. Liu, and C. C. L. Wang, "General support-effective decomposition for multi-directional 3-d printing," *IEEE Transactions on Automation Science and Engineering*, vol. 17, no. 2, pp. 599–610, 2020. DOI: 10.1109/TASE.2019.2938219.

[53] V. Moysiadis, N. Tsolakis, D. Katikaridis, C. G. Sørensen, S. Pearson, and D. Bochtis, "Mobile robotics in agricultural operations: A narrative review on planning aspects," *Applied Sciences*, vol. 10, no. 10, 2020, ISSN: 2076-3417. DOI: 10.3390/app10103453. [Online]. Available: https://www.mdpi.com/2076-3417/10/10/3453.

[54] L. Sabattini, V. Digani, C. Secchi, and C. Fantuzzi, "Hierarchical coordination strategy for multi-agv systems based on dynamic geodesic environment partitioning," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016, pp. 4418–4423. DOI: 10.1109/IROS.2016.7759650.

[55] C. S. Kong, N. A. Peng, and I. Rekleitis, "Distributed coverage with multi-robot system," in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, 2006, pp. 2423–2429. DOI: 10.1109/ROBOT.2006.1642065.

[56] K. Karpe, A. Sinha, S. Raorane, A. Chatterjee, P. Srinivas, and L. Sabattini, *Towards optimized distributed multi-robot printing: An algorithmic approach*, 2021. arXiv: 2102.12026 [cs.RO].

[57] J. Sustarevas, K. X. Benjamin Tan, D. Gerber, R. Stuart-Smith, and V. M. Pawar, "Youwasps: Towards autonomous multi-robot mobile deposition for construction," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 2320–2327. DOI: 10.1109/IROS40897.2019.8967766.

[58] R. Zlot and A. Stentz, "Complex task allocation for multiple robots," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, 2005, pp. 1515–1522. DOI: 10.1109/ROBOT.2005.1570329.

[59] D. Rathbun, S. Kragelund, A. Pongpunwattana, and B. Capozzi, "An evolution based path planning algorithm for autonomous motion of a uav through uncertain environments," in *Proceedings. The 21st Digital Avionics Systems Conference*, IEEE, vol. 2, 2002, pp. 8D2–8D2.

[60] A. E. Eiben, J. E. Smith, *et al.*, *Introduction to evolutionary computing*. Springer, 2003, vol. 53.

[61] M. Gerke, "Genetic path planning for mobile robots," in *Proceedings of the 1999 American Control Conference (Cat. No. 99CH36251)*, IEEE, vol. 4, 1999, pp. 2424–2429.

[62] C. Hajiyev and S. Y. Vural, "Lqr controller with kalman estimator applied to uav longitudinal dynamics," 2013.

[63] R. Tedrake, *Chapter 8, Linear Quadratic Regulators, Underactuated Robotics: Algorithms for Walking, Running, Swimming, Flying, and Manipulation (Course Notes for MIT 6.832)*. [Online]. Available: http://underactuated.mit.edu/lqr.html.

[64] S. M. LaValle *et al.*, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[65] L. Janson and M. Pavone, "Fast marching trees: A fast marching sampling-based method for optimal motion planning in many dimensions - extended version," *CoRR*, vol. abs/1306.3532, 2013. arXiv: 1306.3532. [Online]. Available: http://arxiv.org/abs/1306.3532.

[66] M. Elbanhawi and M. Simic, "Sampling-based robot motion planning: A review," *Ieee access*, vol. 2, pp. 56–77, 2014.

[67] Y. Tusi and H.-Y. Chung, "Using abc and rrt algorithms to improve mobile robot path planning with danger degree," in *2016 Fifth International Conference on Future Generation Communication Technologies (FGCT)*, IEEE, 2016, pp. 21–26.

[68] J. Wang, B. Li, and M. Q.-H. Meng, "Kinematic constrained bi-directional rrt with efficient branch pruning for robot path planning," *Expert Systems with Applications*, vol. 170, p. 114 541, 2021.

[69] D. Lin, B. Shen, Y. Liu, F. E. Alsaadi, and A. Alsaedi, "Genetic algorithm-based compliant robot path planning: An improved bi-rrt-based initialization method," *Assembly Automation*, 2017.

[70] R. Pepy and A. Lambert, "Safe path planning in an uncertain-configuration space using rrt," in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2006, pp. 5376–5381.

[71] H. Zhang, Y. Wang, J. Zheng, and J. Yu, "Path planning of industrial robot based on improved rrt algorithm in complex environments," *IEEE Access*, vol. 6, pp. 53 296–53 306, 2018.

[72] H. Ryu and Y. Park, "Improved informed rrt* using gridmap skeletonization for mobile robot path planning," *International Journal of Precision Engineering and Manufacturing*, vol. 20, no. 11, pp. 2033–2039, 2019.

[73] J. Qi, H. Yang, and H. Sun, "Mod-rrt*: A sampling-based algorithm for robot path planning in dynamic environment," *IEEE Transactions on Industrial Electronics*, vol. 68, no. 8, pp. 7244–7251, 2020.

[74] I. Noreen, A. Khan, Z. Habib, *et al.*, "Optimal path planning using rrt* based approaches: A survey and future directions," *Int. J. Adv. Comput. Sci. Appl*, vol. 7, no. 11, pp. 97–107, 2016.

[75] 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.

[76] I. Noreen, A. Khan, and Z. Habib, "A comparison of rrt, rrt* and rrt*-smart path planning algorithms," 2016.

[77] M. Jordan and A. Perez, "Optimal bidirectional rapidly-exploring random trees," 2013.

[78] A. H. Qureshi and Y. Ayaz, "Intelligent bidirectional rapidly-exploring random trees for optimal motion planning in complex cluttered environments," *Robotics and Autonomous Systems*, vol. 68, pp. 1–11, 2015.

[79] J. Gammell, S. Srinivasa, and T. Barfoot, "Bit*: Sampling-based optimal planning via batch informed trees," in *RSS 2014*, 2014.

[80] S. M. LaValle and J. James J. Kuffner, "Randomized kinodynamic planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001. DOI: 10.1177/02783640122067453. eprint: https://doi.org/10.1177/02783640122067453. [Online]. Available: https://doi.org/10.1177/02783640122067453.

[81] D. Aarno, F. Lingelbach, and D. Kragic, "Constrained path planning and task-consistent path adaptation for mobile manipulators," in *ICAR '05. Proceedings., 12th International Conference on Advanced Robotics, 2005.*, 2005, pp. 268–273. DOI: 10.1109/ICAR.2005.1507423.

[82] K. Nagatani, T. Hirayama, A. Gofuku, and Y. Tanaka, "Motion planning for mobile manipulator with keeping manipulability," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 2, 2002, 1663–1668 vol.2. DOI: 10.1109/IRDS.2002.1043994.

[83] G. Oriolo and C. Mongillo, "Motion planning for mobile manipulators along given end-effector paths," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, 2005, pp. 2154–2160. DOI: 10.1109/ROBOT.2005.1570432.

[84] S. J. Julius Sustarevas Dimitrios Kanoulas, "Task-consistent path planning for mobile 3d printing (published, not released)," 2021.

[85] J. Jo, J. Seo, and J.-D. Fekete, "A progressive k-d tree for approximate k-nearest neighbors," in *2017 IEEE Workshop on Data Systems for Interactive Analysis (DSIA)*, 2017, pp. 1–5. DOI: 10.1109/DSIA.2017.8339084.

Appendix A

Calculations

A.1 Calculation of shortest point to line distance

The shortest distance of a point to a line has three cases and is illustrated in Figure A.1.

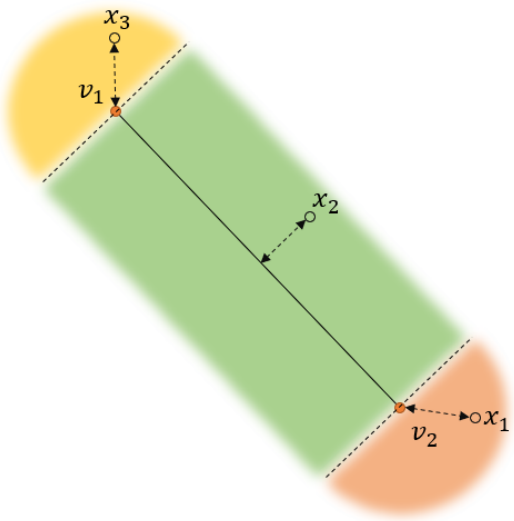


Figure A.1: Illustration of the three cases calculating the shortest distance between an edge $(\mathbf{v}_1, \mathbf{v}_2)$ and the points $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$

In Figure A.1, $\mathbf{v}_1, \mathbf{v}_2, \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3 \in \mathbb{R}^2$ are points. Utilizing the points, we have the obstacle edge $\mathbf{e}_{obs}$ and the vector $\mathbf{v}_{task}$ of a printing task point x_i and the point v_2 on the obstacle defined as (A.1) and (A.2) respectively.

$$\mathbf{e}_{obs} = \mathbf{v}_1 - \mathbf{v}_2 \tag{A.1}$$

$$\mathbf{v}_{task} = \mathbf{v}_2 - \mathbf{x}_i \tag{A.2}$$

where $i \in \{1, 2, 3\}$. Now, consider the inner product of $\mathbf{e}_{obs}$ and $\mathbf{v}_{task}$, we have:

$$\begin{aligned} \|\mathbf{v}_{task}\| \, \|\mathbf{e}_{obs}\| \cos\theta &= -\langle \mathbf{v}_{task}, \mathbf{e}_{obs} \rangle \\ \frac{\|\mathbf{v}_{task}\|}{\|\mathbf{e}_{obs}\|} \cos\theta &= -\frac{\langle \mathbf{v}_{task}, \mathbf{e}_{obs} \rangle}{\|\mathbf{e}_{obs}\|^2} \longrightarrow \mu \end{aligned} \tag{A.3}$$

This gives us a value μ to indicate where the point $\mathbf{x}_i$ is. As $\mathbf{x}_i$ is in different regions, the calculation for the shortest distance between the point and the edge is different. When $\mathbf{x}_i$ is in the yellow ($\mu > 1$) and orange region ($\mu < 0$), they are calculated as in (A.4).

$$d = \begin{cases} \|\mathbf{x}_1 - \mathbf{v}_2\| & \mu < 0 \\ \|\mathbf{x}_3 - \mathbf{v}_1\| & \mu > 1 \end{cases} \tag{A.4}$$

When x_i is in the green region ($\mu \in [0, 1]$), we can leverage the ratio μ to calculate the shortest distance as in (A.5):

$$\begin{aligned} d &= \|\mathbf{x}_2 - [\mathbf{v}_2 - \mu(\mathbf{e}_{obs})]\| \\ d &= \|\mathbf{x}_2 - [\mathbf{v}_2 - \mu(\mathbf{v}_1 - \mathbf{v}_2)]\| \end{aligned} \tag{A.5}$$

Combining (A.4) and (A.5), we have (A.6) which gives the shortest distance between the point $\mathbf{x}_i$ and the line $(\mathbf{v}_1, \mathbf{v}_2)$:

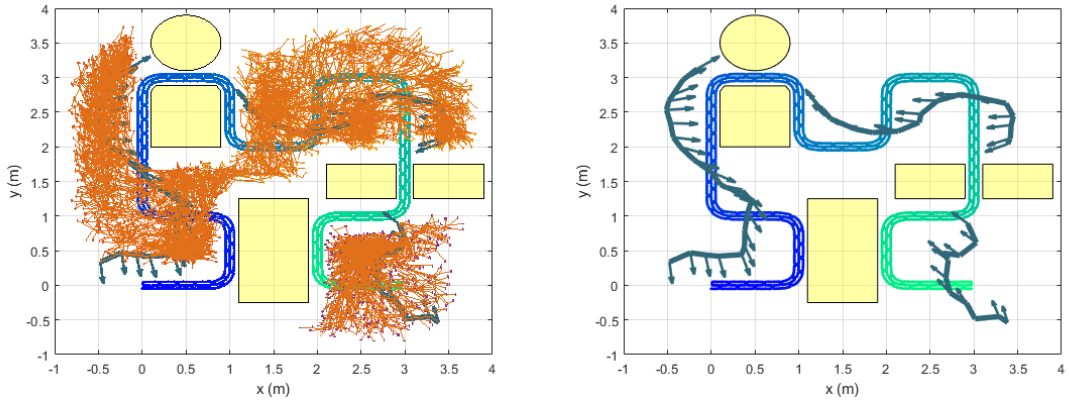
$$d = \left\| \mathbf{x}_i - \left\langle \begin{bmatrix} \mathbb{1}[\mu \leq 1] \\ \mathbb{1}[\mu \in [0, 1]] \\ \mathbb{1}[\mu > 1] \end{bmatrix}, \begin{bmatrix} \mathbf{v}_2 \\ \mu(\mathbf{v}_1 - \mathbf{v}_2) \\ \mathbf{v}_1 \end{bmatrix} \right\rangle \right\| \tag{A.6}$$

Appendix B

Visual Data

B.1 Full set of TCFMT* results

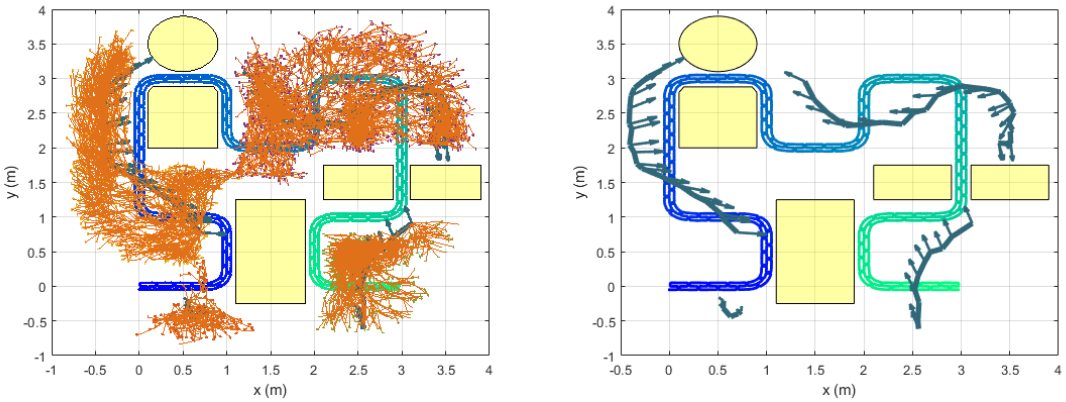
B.1.1 Obstacle Configuration 1



(a) Test one result showing the tree indicated in orange and the path indicated in dark green with arrows indicating orientations of base poses

(b) Test one result showing the path only, indicated in dark green with arrows indicating orientations of base poses

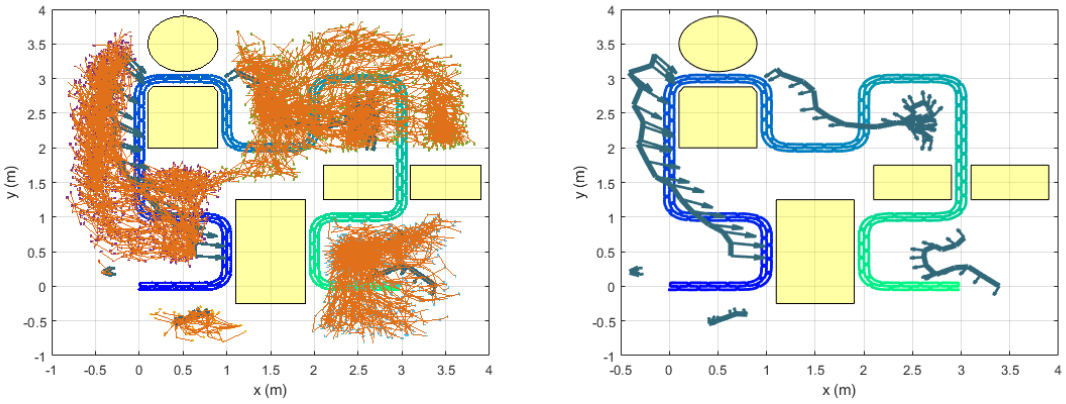
Figure B.1: TCFMT* results: Test one on obstacle configuration 1



(a) Test two result showing the tree indicated in orange and the path indicated in dark green with arrows indicating orientations of base poses

(b) Test two result showing the path only, indicated in dark green with arrows indicating orientations of base poses

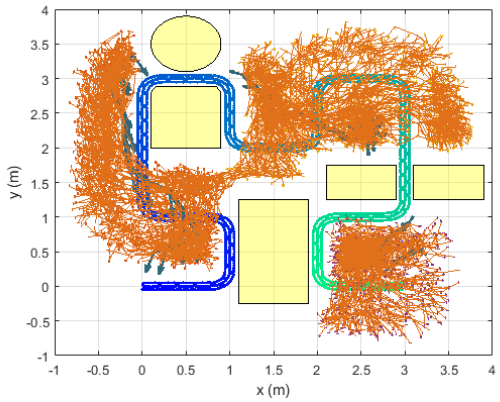
Figure B.2: TCFMT* results: Test two on obstacle configuration 1



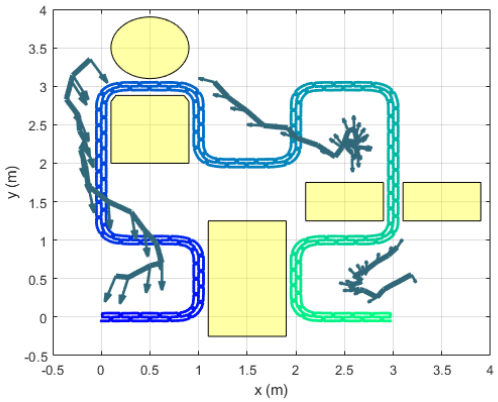
(a) Test three result showing the tree indicated in orange and the path indicated in dark green with arrows indicating orientations of base poses

(b) Test three result showing the path only, indicated in dark green with arrows indicating orientations of base poses

Figure B.3: TCFMT* results: Test three on obstacle configuration 1

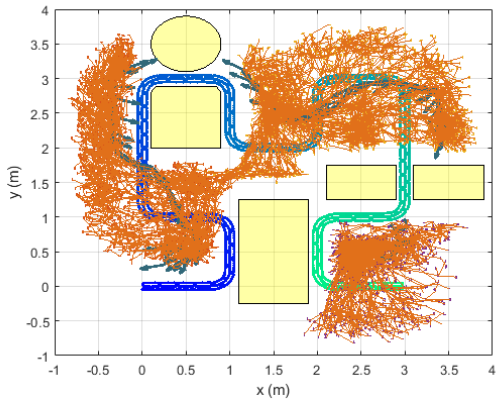


(a) Test four result showing the tree indicated in orange and the path indicated in dark green with arrows indicating orientations of base poses

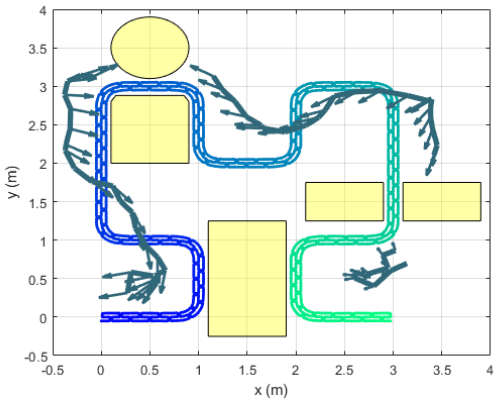


(b) Test four result showing the path only, indicated in dark green with arrows indicating orientations of base poses

Figure B.4: TCFMT* results: Test four on obstacle configuration 1



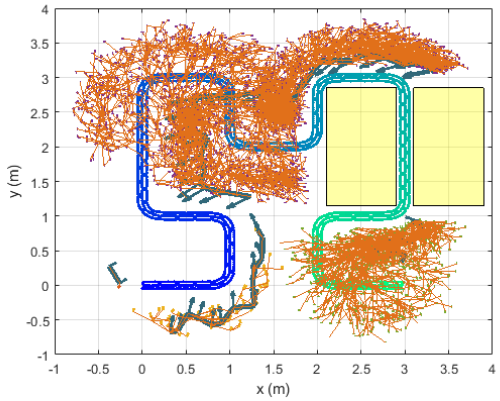
(a) Test five result showing the tree indicated in orange and the path indicated in dark green with arrows indicating orientations of base poses



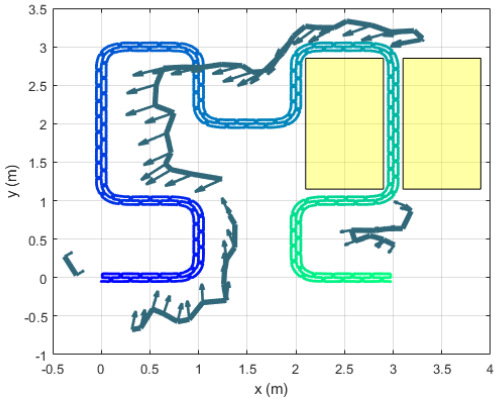
(b) Test five result showing the path only, indicated in dark green with arrows indicating orientations of base poses

Figure B.5: TCFMT* results: Test five on obstacle configuration 1

B.1.2 Obstacle Configuration 2

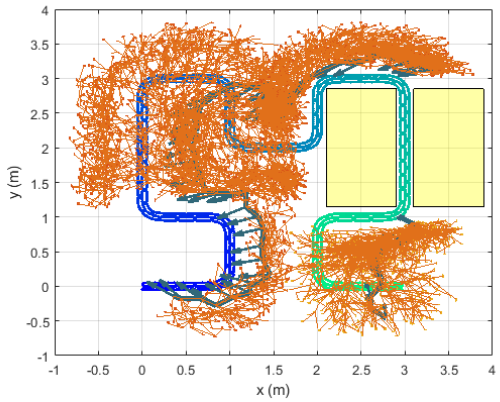


(a) Test one result showing the tree indicated in orange and the path indicated in dark green with arrows indicating orientations of base poses

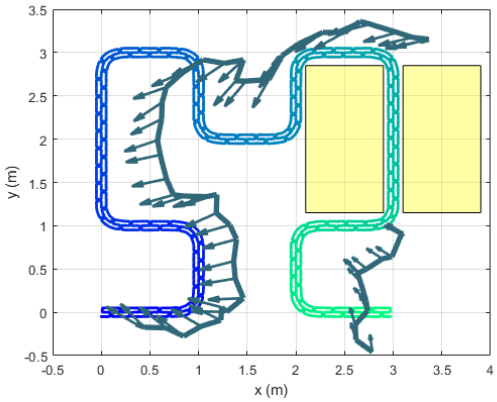


(b) Test one result showing the path only, indicated in dark green with arrows indicating orientations of base poses

Figure B.6: TCFMT* results: Test one on obstacle configuration 2

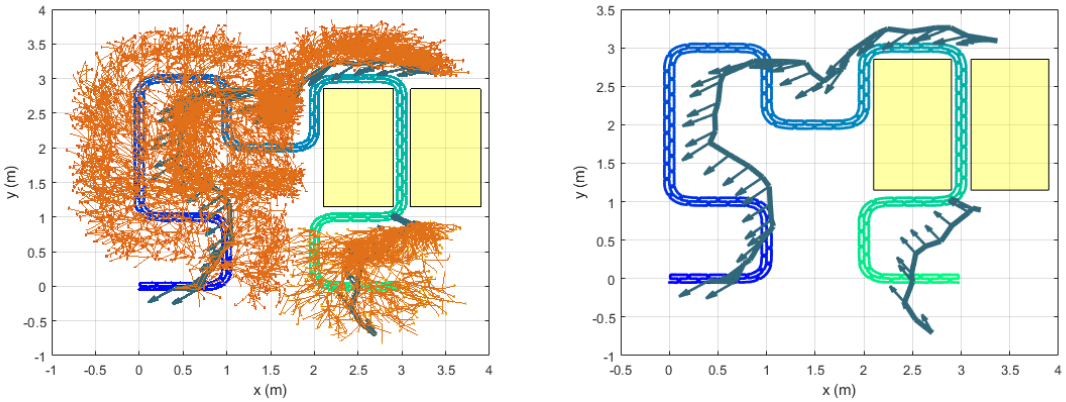


(a) Test two result showing the tree indicated in orange and the path indicated in dark green with arrows indicating orientations of base poses



(b) Test two result showing the path only, indicated in dark green with arrows indicating orientations of base poses

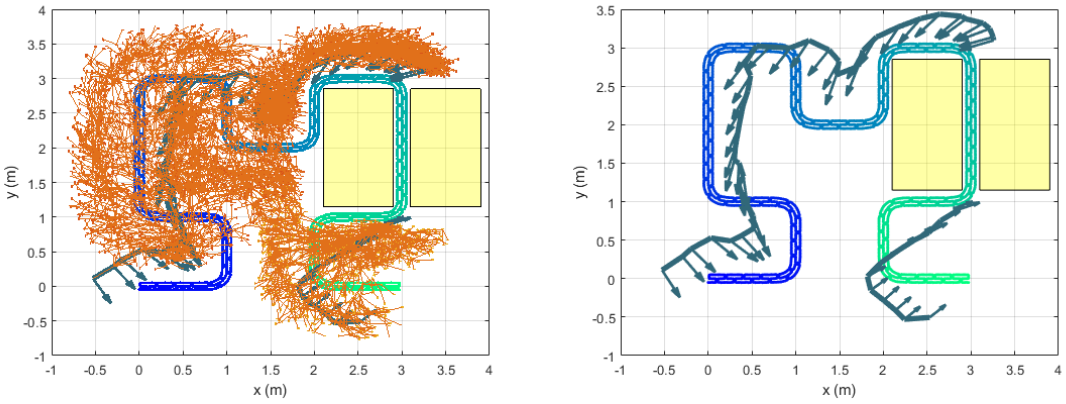
Figure B.7: TCFMT* results: Test two on obstacle configuration 2



(a) Test three result showing the tree indicated in orange and the path indicated in dark green with arrows indicating orientations of base poses

(b) Test three result showing the path only, indicated in dark green with arrows indicating orientations of base poses

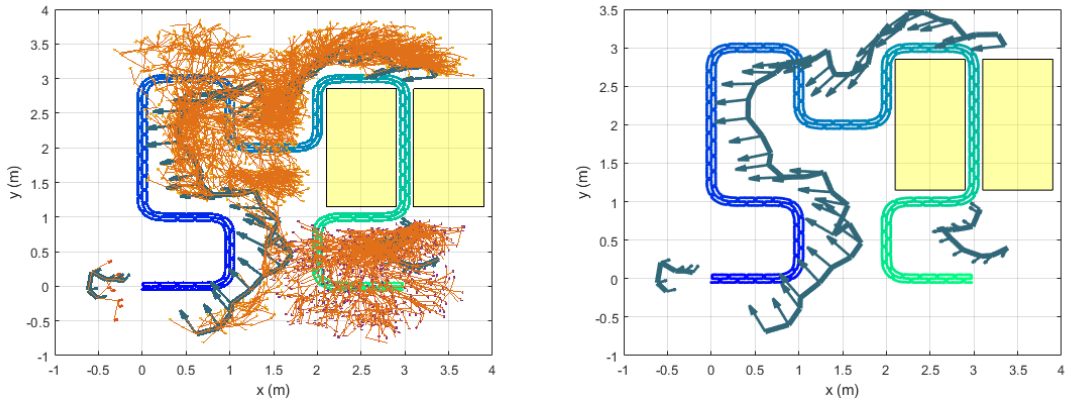
Figure B.8: TCFMT* results: Test three on obstacle configuration 2



(a) Test four result showing the tree indicated in orange and the path indicated in dark green with arrows indicating orientations of base poses

(b) Test four result showing the path only, indicated in dark green with arrows indicating orientations of base poses

Figure B.9: TCFMT* results: Test four on obstacle configuration 2

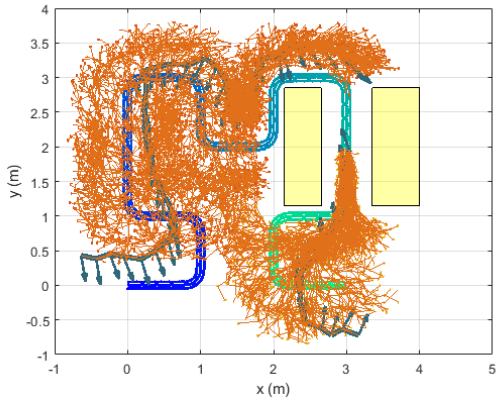


(a) Test five result showing the tree indicated in orange and the path indicated in dark green with arrows indicating orientations of base poses

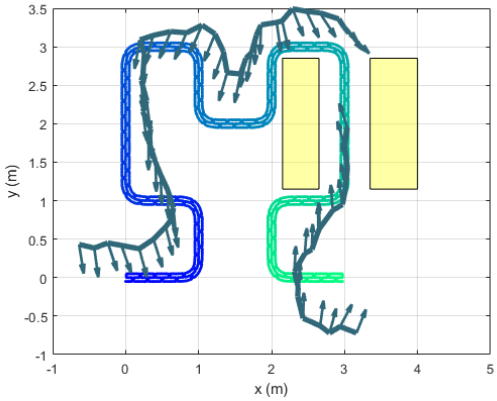
(b) Test five result showing the path only, indicated in dark green with arrows indicating orientations of base poses

Figure B.10: TCFMT* results: Test five on obstacle configuration 2

B.1.3 Obstacle Configuration 3

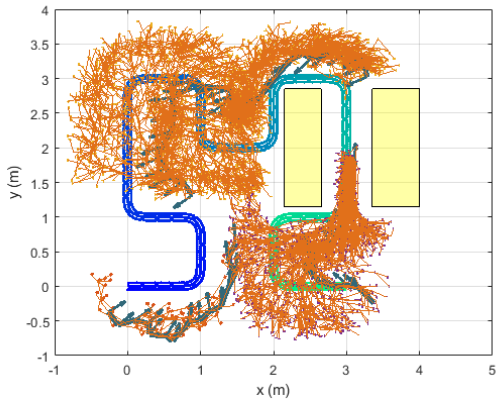


(a) Test one result showing the tree indicated in orange and the path indicated in dark green with arrows indicating orientations of base poses

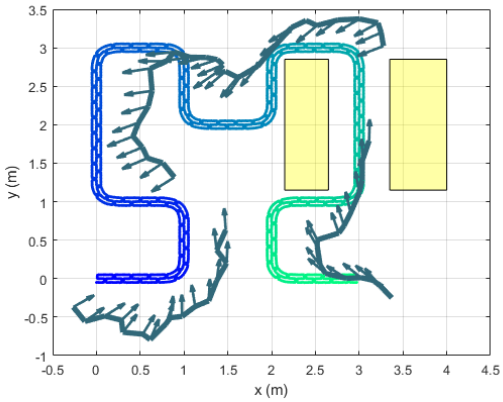


(b) Test one result showing the path only, indicated in dark green with arrows indicating orientations of base poses

Figure B.11: TCFMT* results: Test one on obstacle configuration 3

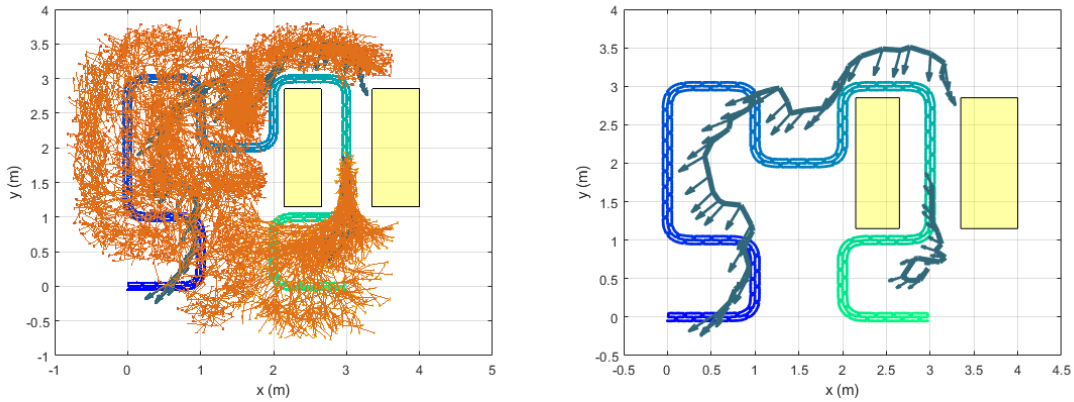


(a) Test two result showing the tree indicated in orange and the path indicated in dark green with arrows indicating orientations of base poses



(b) Test two result showing the path only, indicated in dark green with arrows indicating orientations of base poses

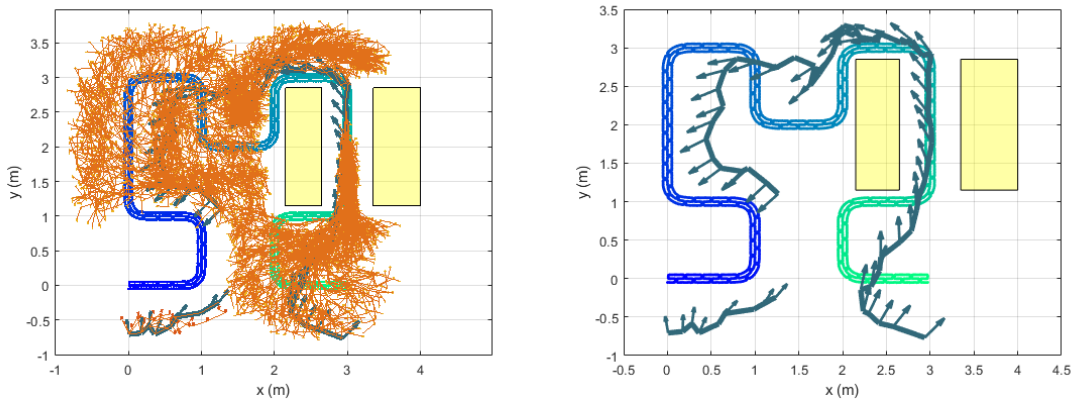
Figure B.12: TCFMT* results: Test two on obstacle configuration 3



(a) Test three result showing the tree indicated in orange and the path indicated in dark green with arrows indicating orientations of base poses

(b) Test three result showing the path only, indicated in dark green with arrows indicating orientations of base poses

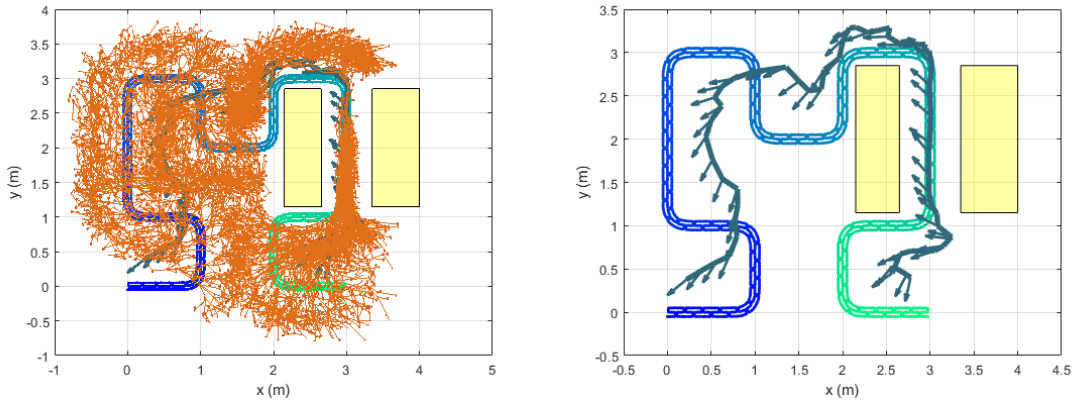
Figure B.13: TCFMT* results: Test three on obstacle configuration 3



(a) Test four result showing the tree indicated in orange and the path indicated in dark green with arrows indicating orientations of base poses

(b) Test four result showing the path only, indicated in dark green with arrows indicating orientations of base poses

Figure B.14: TCFMT* results: Test four on obstacle configuration 3

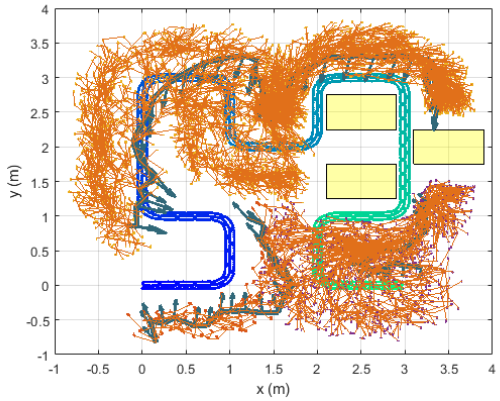


(a) Test five result showing the tree indicated in orange and the path indicated in dark green with arrows indicating orientations of base poses

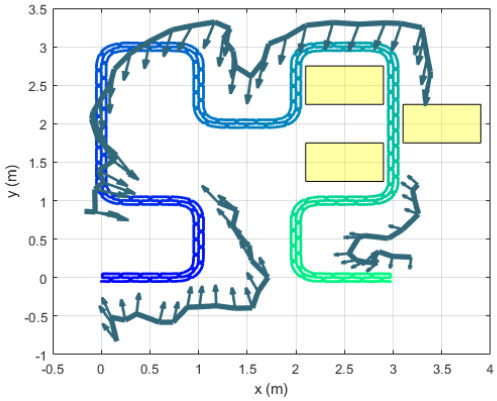
(b) Test five result showing the path only, indicated in dark green with arrows indicating orientations of base poses

Figure B.15: TCFMT* results: Test five on obstacle configuration 3

B.1.4 Obstacle Configuration 4

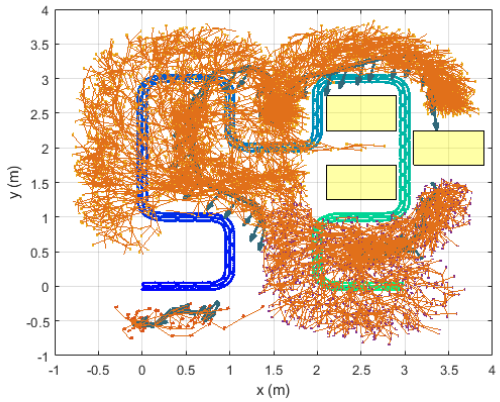


(a) Test one result showing the tree indicated in orange and the path indicated in dark green with arrows indicating orientations of base poses

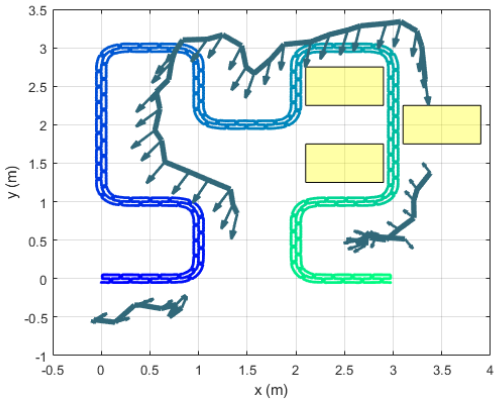


(b) Test one result showing the path only, indicated in dark green with arrows indicating orientations of base poses

Figure B.16: TCFMT* results: Test one on obstacle configuration 4

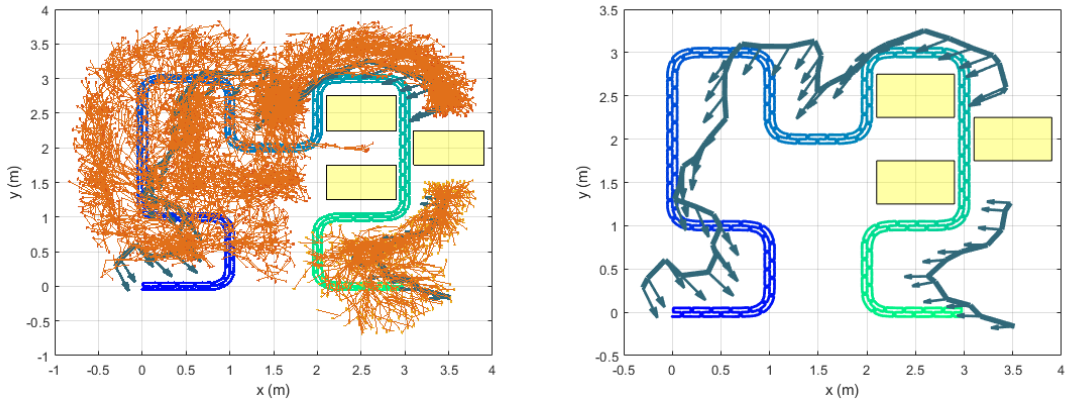


(a) Test two result showing the tree indicated in orange and the path indicated in dark green with arrows indicating orientations of base poses



(b) Test two result showing the path only, indicated in dark green with arrows indicating orientations of base poses

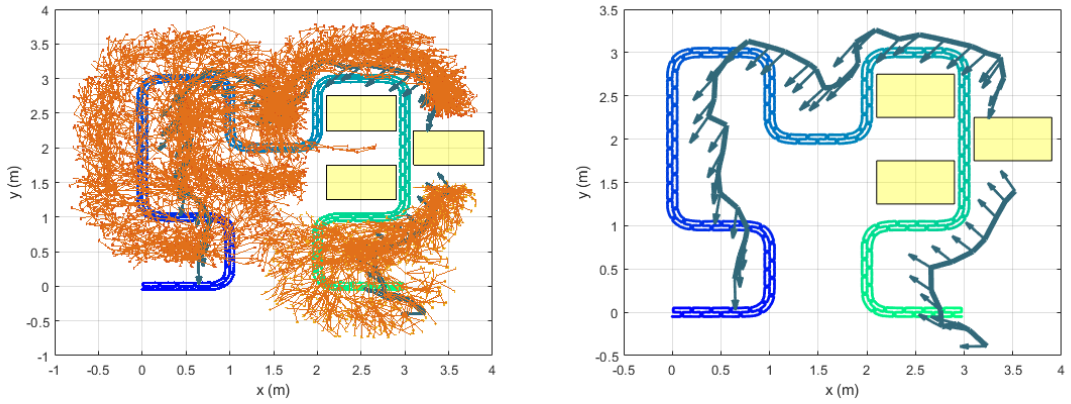
Figure B.17: TCFMT* results: Test two on obstacle configuration 4



(a) Test three result showing the tree indicated in orange and the path indicated in dark green with arrows indicating orientations of base poses

(b) Test three result showing the path only, indicated in dark green with arrows indicating orientations of base poses

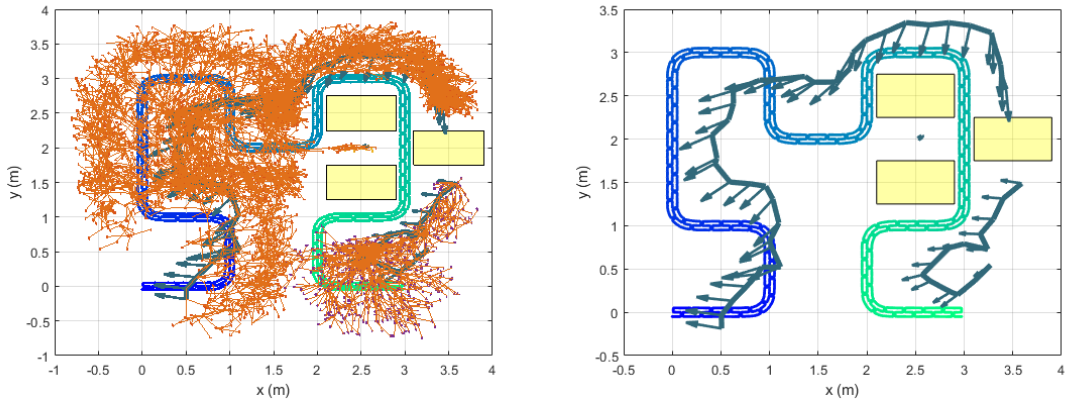
Figure B.18: TCFMT* results: Test three on obstacle configuration 4



(a) Test four result showing the tree indicated in orange and the path indicated in dark green with arrows indicating orientations of base poses

(b) Test four result showing the path only, indicated in dark green with arrows indicating orientations of base poses

Figure B.19: TCFMT* results: Test four on obstacle configuration 4

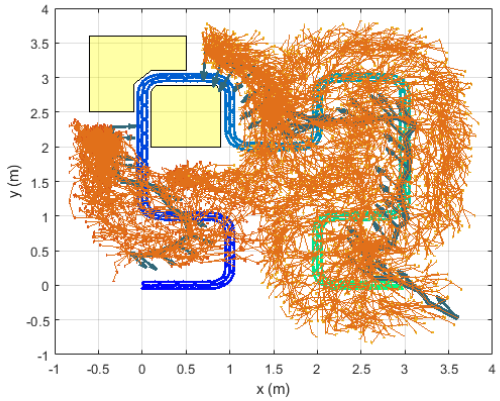


(a) Test five result showing the tree indicated in orange and the path indicated in dark green with arrows indicating orientations of base poses

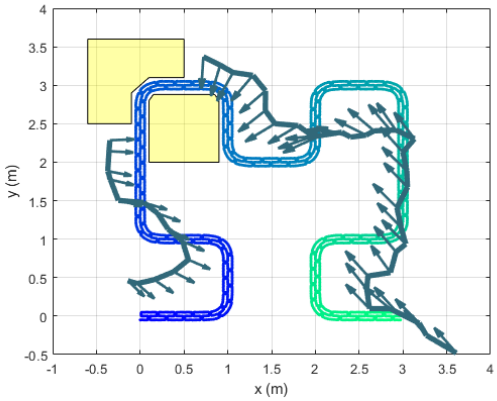
(b) Test five result showing the path only, indicated in dark green with arrows indicating orientations of base poses

Figure B.20: TCFMT* results: Test five on obstacle configuration 4

B.1.5 Obstacle Configuration 5

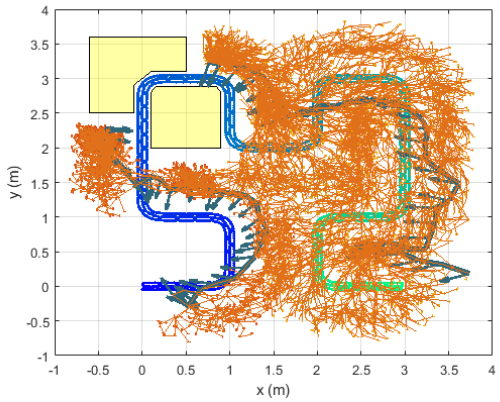


(a) Test one result showing the tree indicated in orange and the path indicated in dark green with arrows indicating orientations of base poses

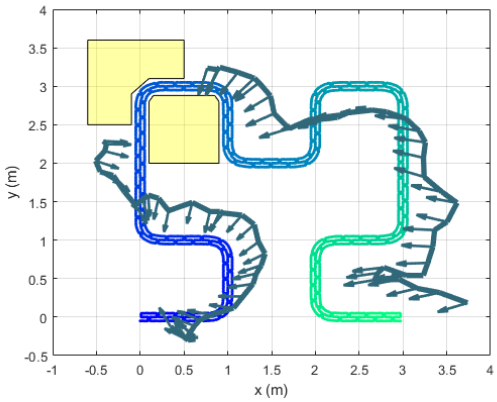


(b) Test one result showing the path only, indicated in dark green with arrows indicating orientations of base poses

Figure B.21: TCFMT* results: Test one on obstacle configuration 5

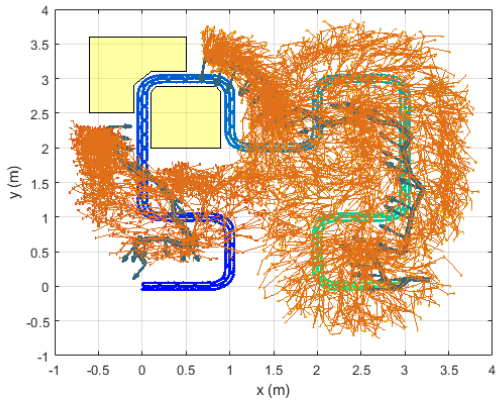


(a) Test two result showing the tree indicated in orange and the path indicated in dark green with arrows indicating orientations of base poses

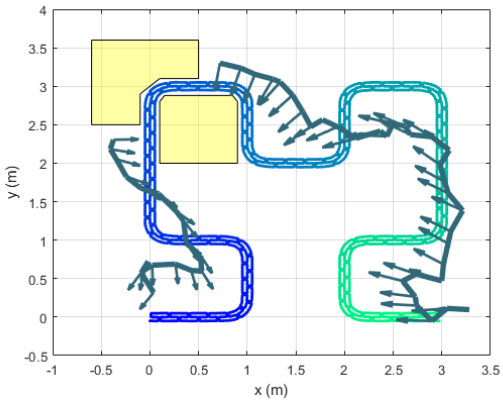


(b) Test two result showing the path only, indicated in dark green with arrows indicating orientations of base poses

Figure B.22: TCFMT* results: Test two on obstacle configuration 5

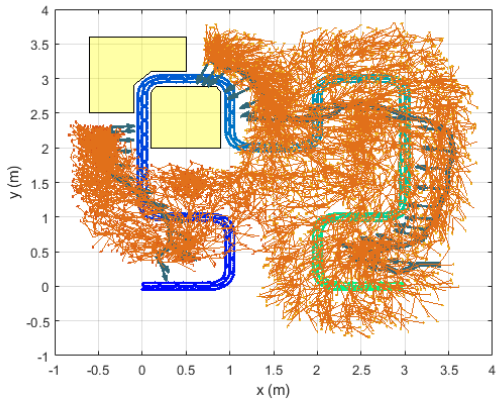


(a) Test three result showing the tree indicated in orange and the path indicated in dark green with arrows indicating orientations of base poses

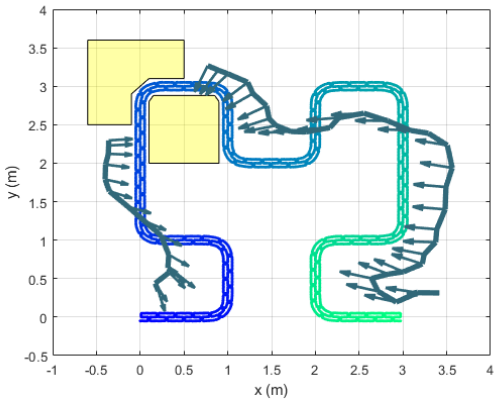


(b) Test three result showing the path only, indicated in dark green with arrows indicating orientations of base poses

Figure B.23: TCFMT* results: Test three on obstacle configuration 5

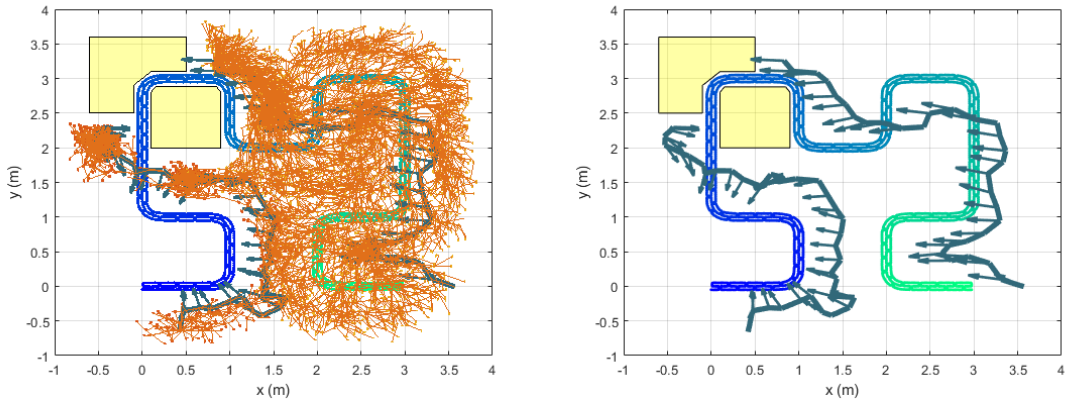


(a) Test four result showing the tree indicated in orange and the path indicated in dark green with arrows indicating orientations of base poses



(b) Test four result showing the path only, indicated in dark green with arrows indicating orientations of base poses

Figure B.24: TCFMT* results: Test four on obstacle configuration 5



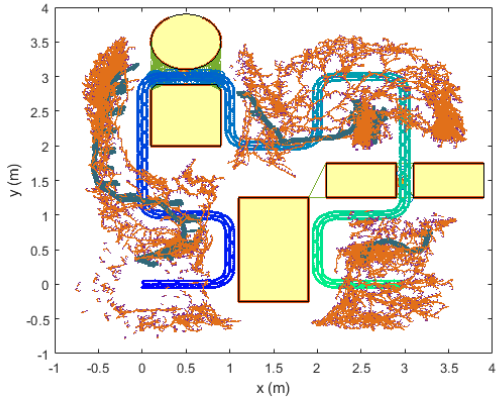
(a) Test five result showing the tree indicated in orange and the path indicated in dark green with arrows indicating orientations of base poses

(b) Test five result showing the path only, indicated in dark green with arrows indicating orientations of base poses

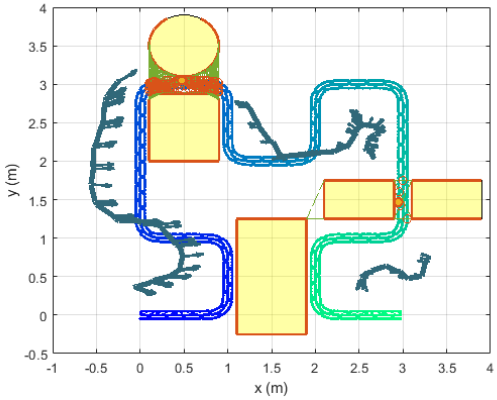
Figure B.25: TCFMT* results: Test five on obstacle configuration 5

B.2 Full set of TCRRT* results

B.2.1 Obstacle Configuration 1

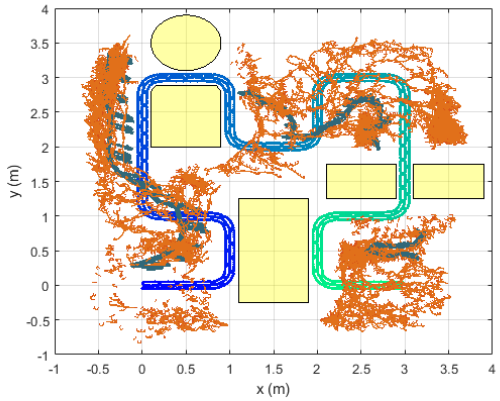


(a) Test one result showing the tree indicated in orange and the path indicated in dark green with arrows indicating orientations of base poses

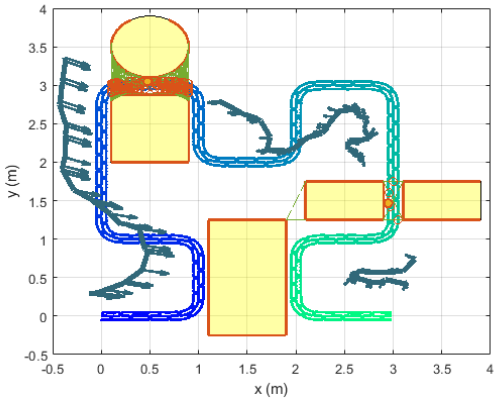


(b) Test one result showing the path only, indicated in dark green with arrows indicating orientations of base poses

Figure B.26: TCRRT* results: Test one on obstacle configuration 1

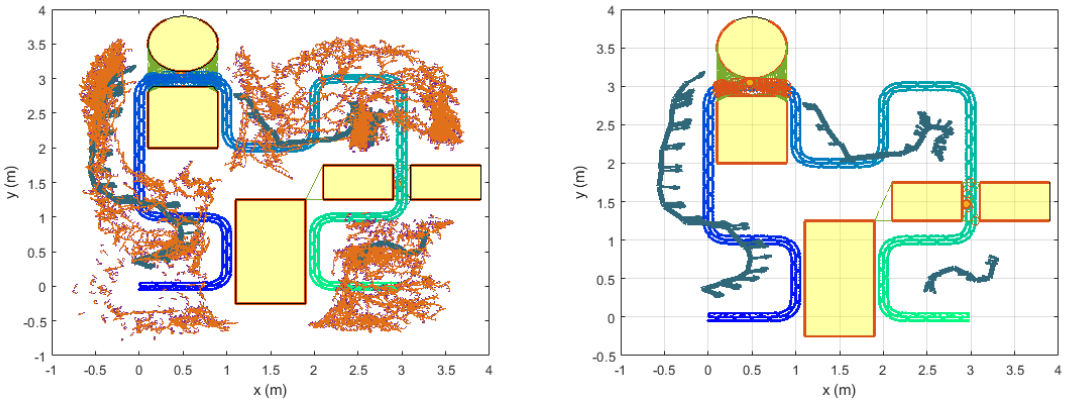


(a) Test two result showing the tree indicated in orange and the path indicated in dark green with arrows indicating orientations of base poses



(b) Test two result showing the path only, indicated in dark green with arrows indicating orientations of base poses

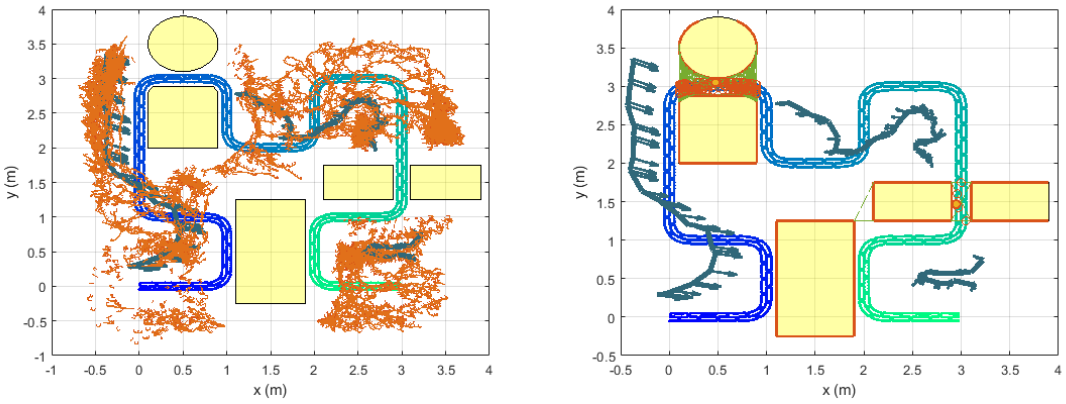
Figure B.27: TCRRT* results: Test two on obstacle configuration 1



(a) Test three result showing the tree indicated in orange and the path indicated in dark green with arrows indicating orientations of base poses

(b) Test three result showing the path only, indicated in dark green with arrows indicating orientations of base poses

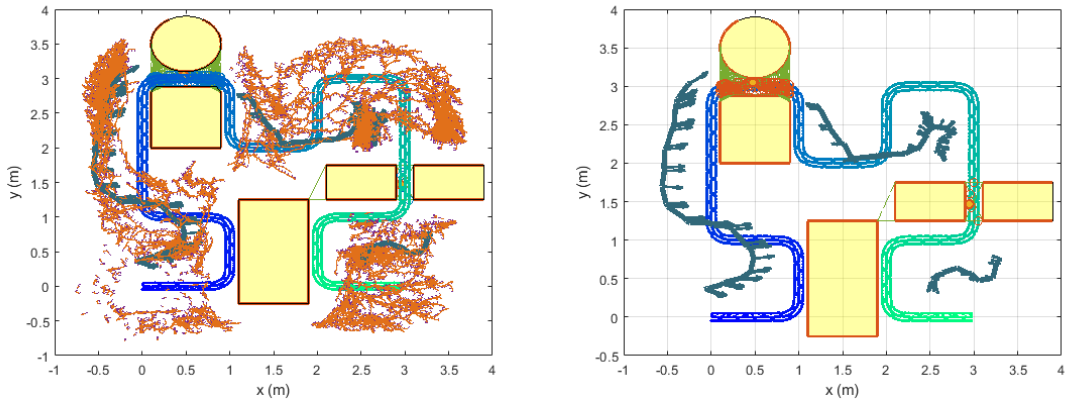
Figure B.28: TCRRT* results: Test three on obstacle configuration 1



(a) Test four result showing the tree indicated in orange and the path indicated in dark green with arrows indicating orientations of base poses

(b) Test four result showing the path only, indicated in dark green with arrows indicating orientations of base poses

Figure B.29: TCRRT* results: Test four on obstacle configuration 1

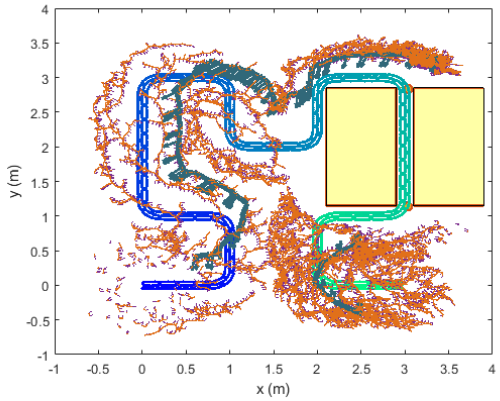


(a) Test five result showing the tree indicated in orange and the path indicated in dark green with arrows indicating orientations of base poses

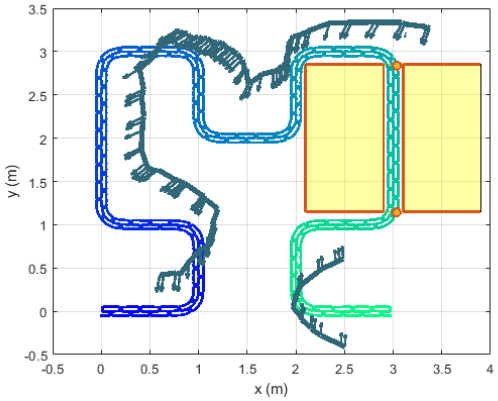
(b) Test five result showing the path only, indicated in dark green with arrows indicating orientations of base poses

Figure B.30: TCRRT* results: Test five on obstacle configuration 1

B.2.2 Obstacle Configuration 2

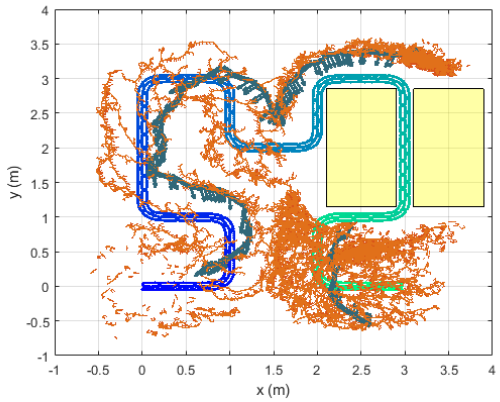


(a) Test one result showing the tree indicated in orange and the path indicated in dark green with arrows indicating orientations of base poses

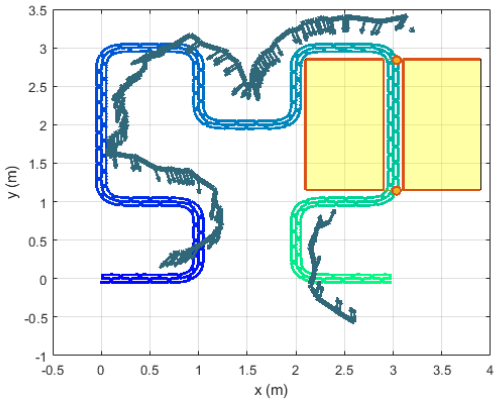


(b) Test one result showing the path only, indicated in dark green with arrows indicating orientations of base poses

Figure B.31: TCRRT* results: Test one on obstacle configuration 2

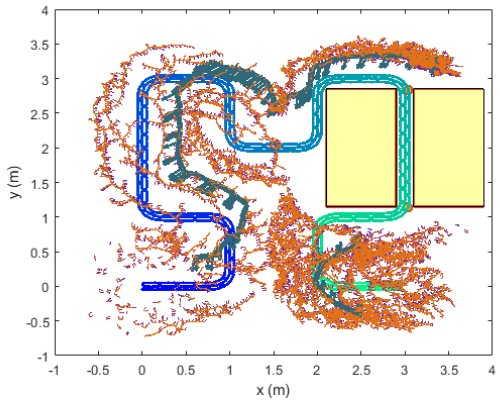


(a) Test two result showing the tree indicated in orange and the path indicated in dark green with arrows indicating orientations of base poses

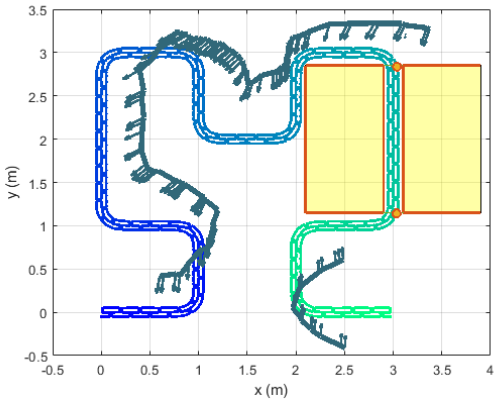


(b) Test two result showing the path only, indicated in dark green with arrows indicating orientations of base poses

Figure B.32: TCRRT* results: Test two on obstacle configuration 2

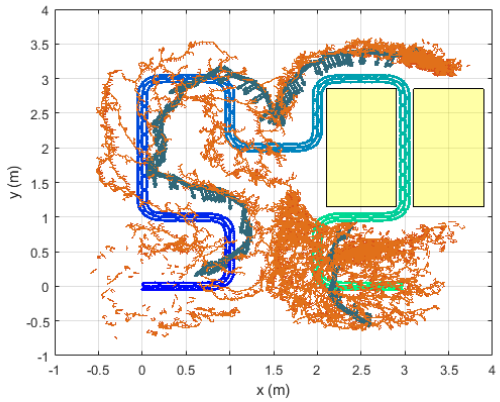


(a) Test three result showing the tree indicated in orange and the path indicated in dark green with arrows indicating orientations of base poses

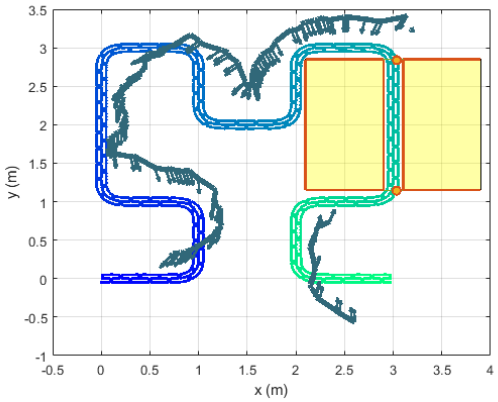


(b) Test three result showing the path only, indicated in dark green with arrows indicating orientations of base poses

Figure B.33: TCRRT* results: Test three on obstacle configuration 2

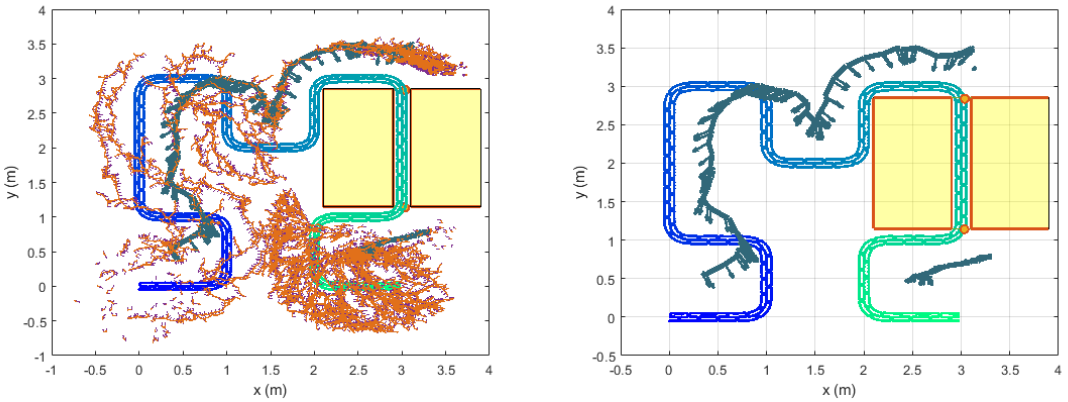


(a) Test four result showing the tree indicated in orange and the path indicated in dark green with arrows indicating orientations of base poses



(b) Test four result showing the path only, indicated in dark green with arrows indicating orientations of base poses

Figure B.34: TCRRT* results: Test four on obstacle configuration 2

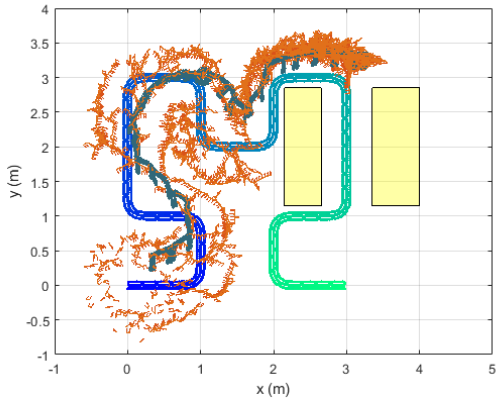


(a) Test five result showing the tree indicated in orange and the path indicated in dark green with arrows indicating orientations of base poses

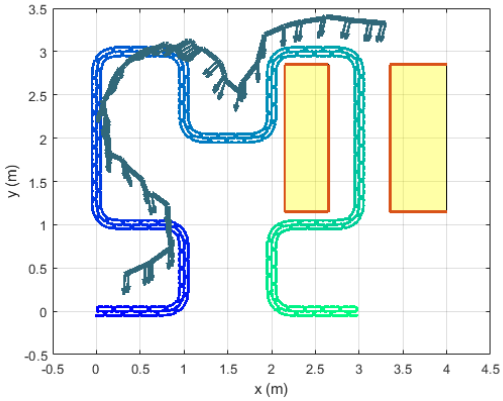
(b) Test five result showing the path only, indicated in dark green with arrows indicating orientations of base poses

Figure B.35: TCRRT* results: Test five on obstacle configuration 2

B.2.3 Obstacle Configuration 3

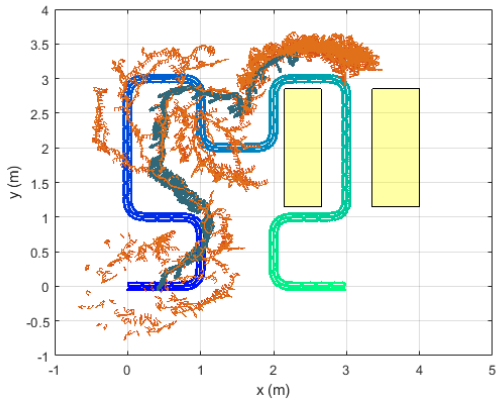


(a) Test one result showing the tree indicated in orange and the path indicated in dark green with arrows indicating orientations of base poses

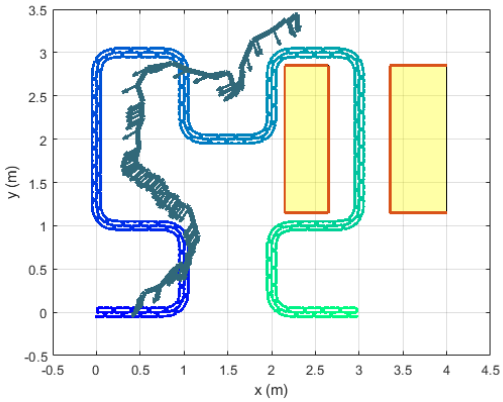


(b) Test one result showing the path only, indicated in dark green with arrows indicating orientations of base poses

Figure B.36: TCRRT* results: Test one on obstacle configuration 3

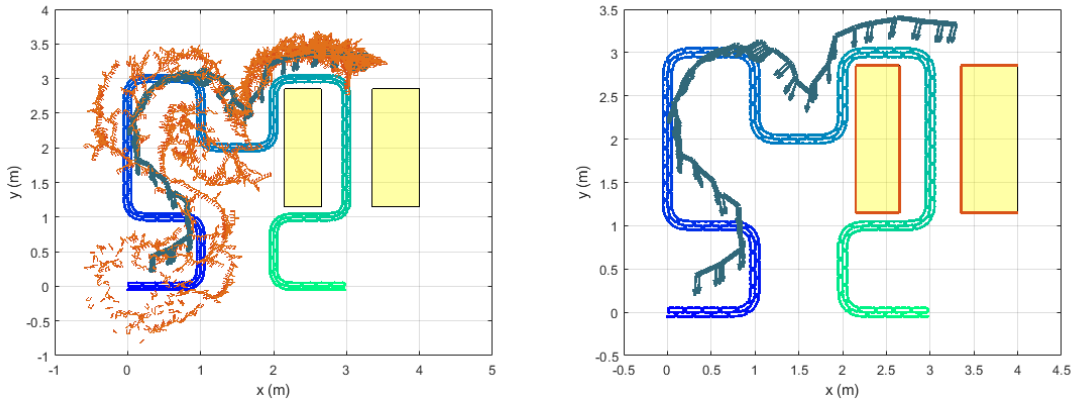


(a) Test two result showing the tree indicated in orange and the path indicated in dark green with arrows indicating orientations of base poses



(b) Test two result showing the path only, indicated in dark green with arrows indicating orientations of base poses

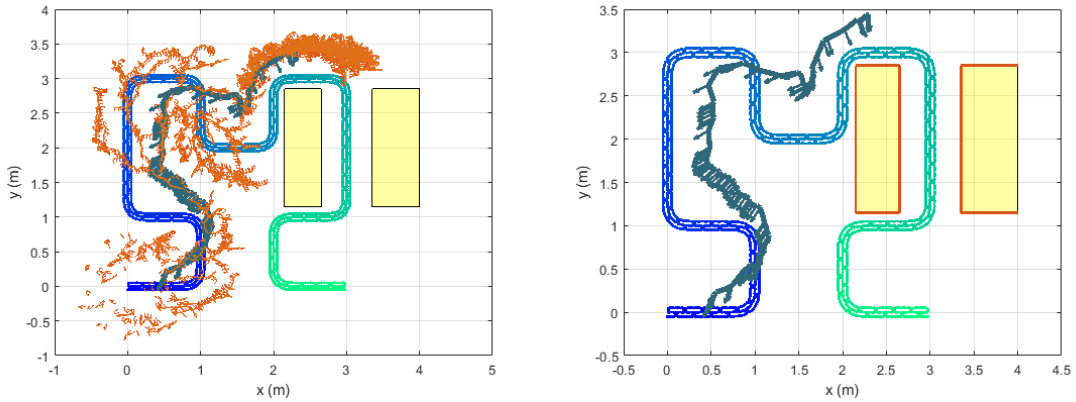
Figure B.37: TCRRT* results: Test two on obstacle configuration 3



(a) Test three result showing the tree indicated in orange and the path indicated in dark green with arrows indicating orientations of base poses

(b) Test three result showing the path only, indicated in dark green with arrows indicating orientations of base poses

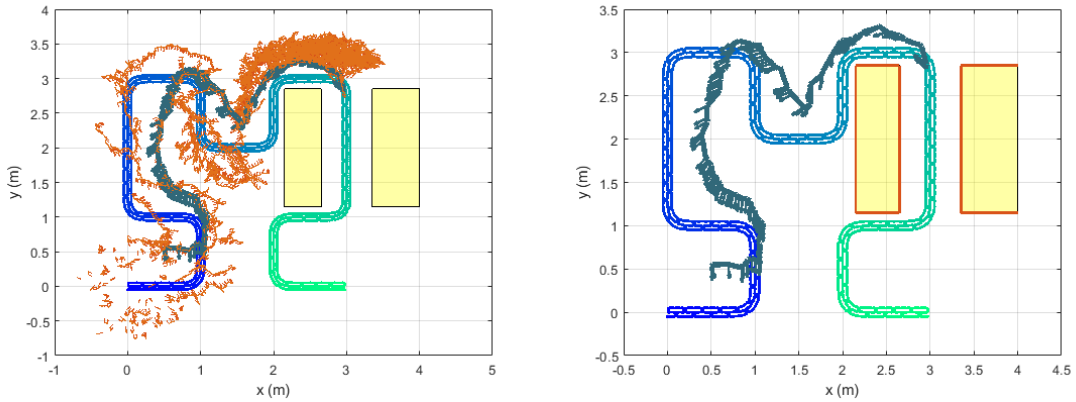
Figure B.38: TCRRT* results: Test three on obstacle configuration 3



(a) Test four result showing the tree indicated in orange and the path indicated in dark green with arrows indicating orientations of base poses

(b) Test four result showing the path only, indicated in dark green with arrows indicating orientations of base poses

Figure B.39: TCRRT* results: Test four on obstacle configuration 3

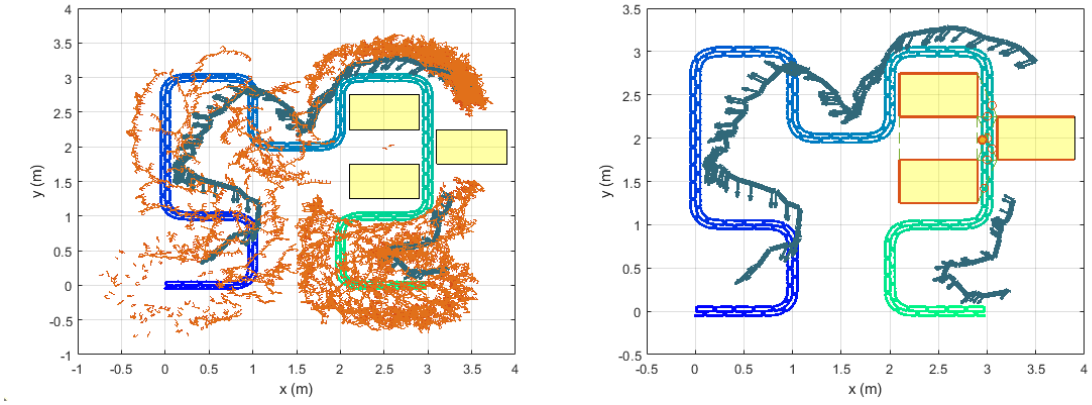


(a) Test five result showing the tree indicated in orange and the path indicated in dark green with arrows indicating orientations of base poses

(b) Test five result showing the path only, indicated in dark green with arrows indicating orientations of base poses

Figure B.40: TCRRT* results: Test five on obstacle configuration 3

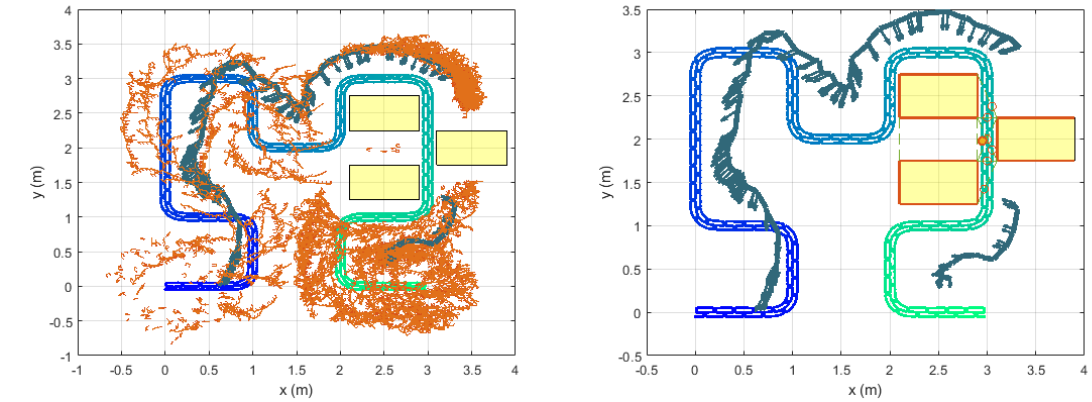
B.2.4 Obstacle Configuration 4



(a) Test one result showing the tree indicated in orange and the path indicated in dark green with arrows indicating orientations of base poses

(b) Test one result showing the path only, indicated in dark green with arrows indicating orientations of base poses

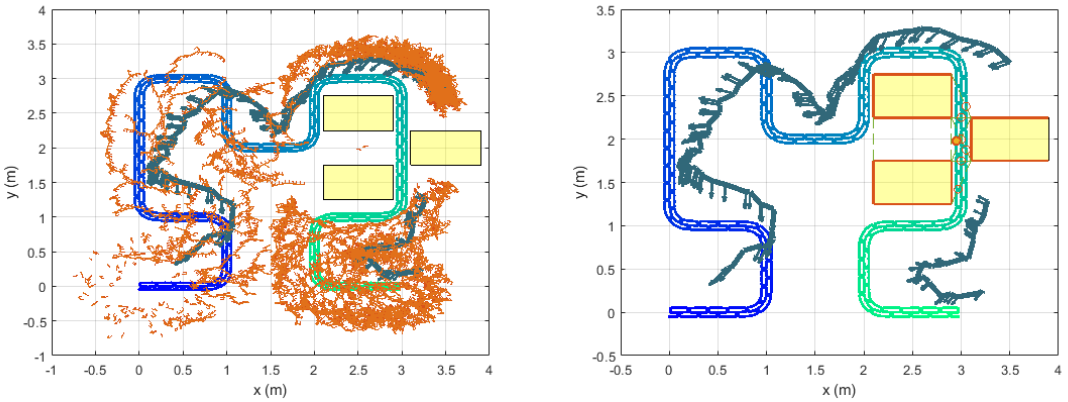
Figure B.41: TCRRT* results: Test one on obstacle configuration 4



(a) Test two result showing the tree indicated in orange and the path indicated in dark green with arrows indicating orientations of base poses

(b) Test two result showing the path only, indicated in dark green with arrows indicating orientations of base poses

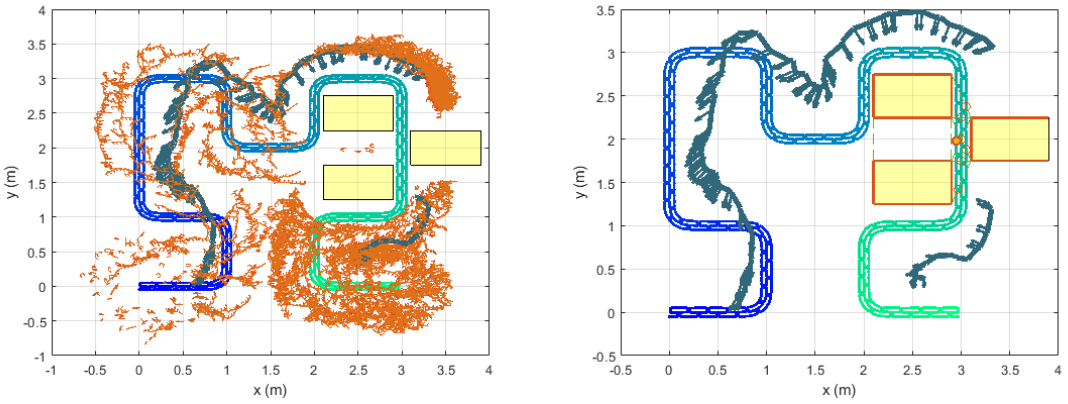
Figure B.42: TCRRT* results: Test two on obstacle configuration 4



(a) Test three result showing the tree indicated in orange and the path indicated in dark green with arrows indicating orientations of base poses

(b) Test three result showing the path only, indicated in dark green with arrows indicating orientations of base poses

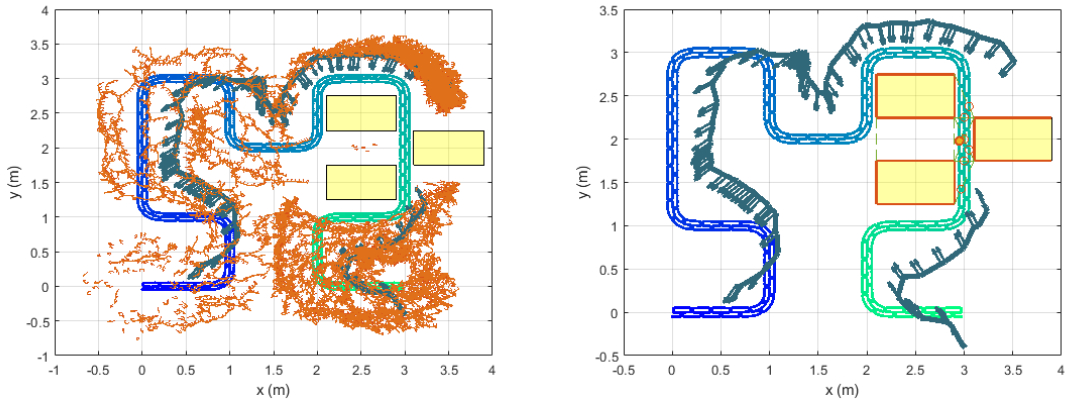
Figure B.43: TCRRT* results: Test three on obstacle configuration 4



(a) Test four result showing the tree indicated in orange and the path indicated in dark green with arrows indicating orientations of base poses

(b) Test four result showing the path only, indicated in dark green with arrows indicating orientations of base poses

Figure B.44: TCRRT* results: Test four on obstacle configuration 4

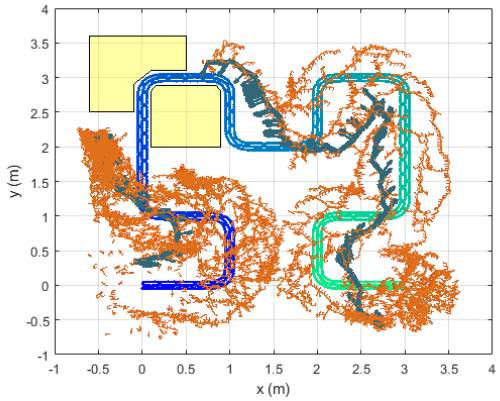


(a) Test five result showing the tree indicated in orange and the path indicated in dark green with arrows indicating orientations of base poses

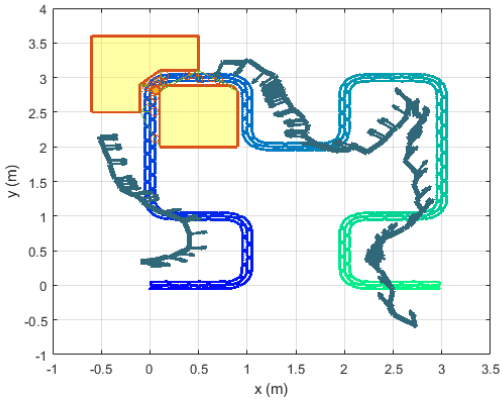
(b) Test five result showing the path only, indicated in dark green with arrows indicating orientations of base poses

Figure B.45: TCRRT* results: Test five on obstacle configuration 4

B.2.5 Obstacle Configuration 5

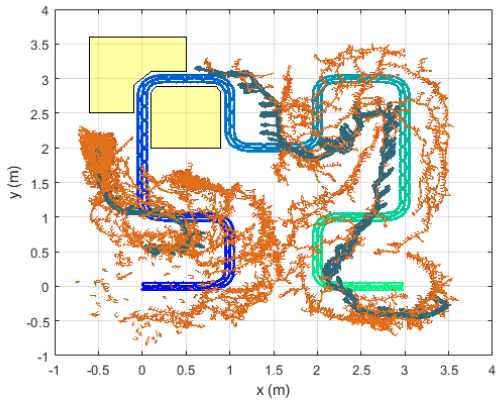


(a) Test one result showing the tree indicated in orange and the path indicated in dark green with arrows indicating orientations of base poses

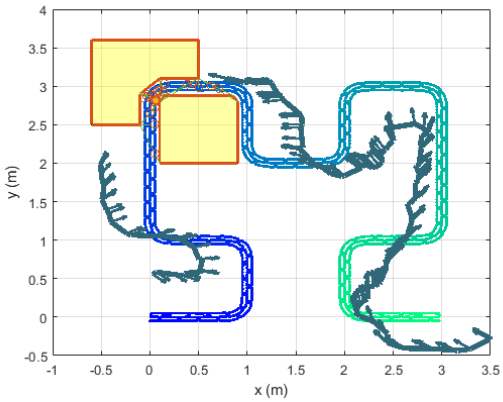


(b) Test one result showing the path only, indicated in dark green with arrows indicating orientations of base poses

Figure B.46: TCRRT* results: Test one on obstacle configuration 5

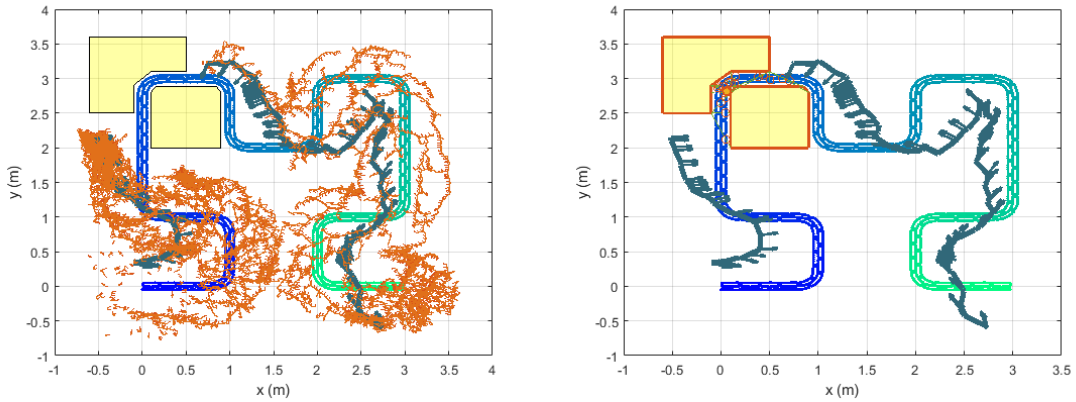


(a) Test two result showing the tree indicated in orange and the path indicated in dark green with arrows indicating orientations of base poses



(b) Test two result showing the path only, indicated in dark green with arrows indicating orientations of base poses

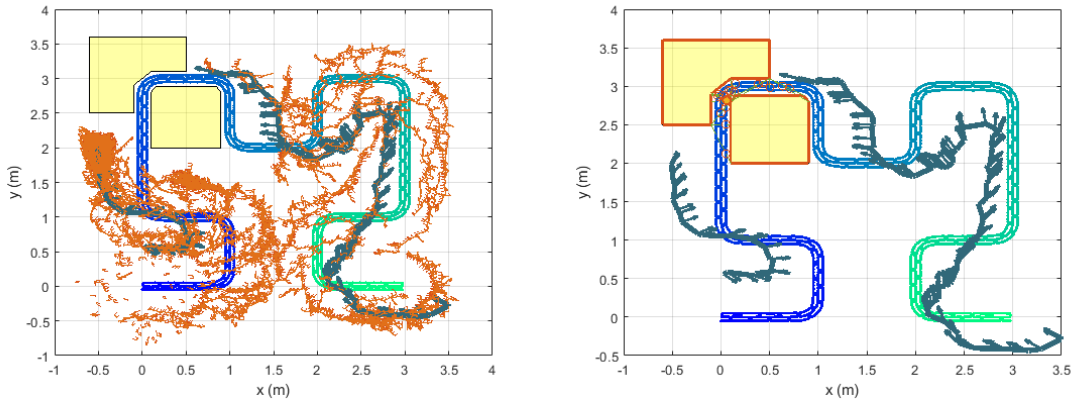
Figure B.47: TCRRT* results: Test two on obstacle configuration 5



(a) Test three result showing the tree indicated in orange and the path indicated in dark green with arrows indicating orientations of base poses

(b) Test three result showing the path only, indicated in dark green with arrows indicating orientations of base poses

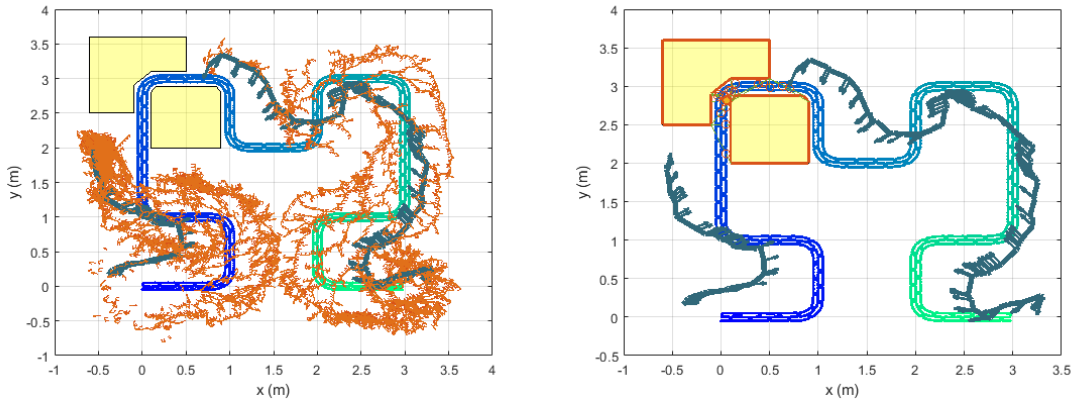
Figure B.48: TCRRT* results: Test three on obstacle configuration 5



(a) Test four result showing the tree indicated in orange and the path indicated in dark green with arrows indicating orientations of base poses

(b) Test four result showing the path only, indicated in dark green with arrows indicating orientations of base poses

Figure B.49: TCRRT* results: Test four on obstacle configuration 5



(a) Test five result showing the tree indicated in orange and the path indicated in dark green with arrows indicating orientations of base poses

(b) Test five result showing the path only, indicated in dark green with arrows indicating orientations of base poses

Figure B.50: TCRRT* results: Test five on obstacle configuration 5