Motion Planning For Autonomous Vehicles In Non-Signalized Intersections

Darshit Patel

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Mechanical Engineering

Azim Eskandarian, Chair
Saied Taheri
Kaveh Akbari Hamed

June 14, 2023
Blacksburg, Virginia

Keywords: Motion Planning, Autonomous Vehicles, Sampling-Based Algorithms, Vehicle Control

Motion Planning For Autonomous Vehicles In Non-Signalized Intersections

Darshit Patel

(ABSTRACT)

Real-time path generation, including collision checks, is vital in critical driving scenarios such as navigating non-signalized intersections. These intersections lack organized traffic flow, which raises the risk of accidents. Rapidly Exploring Random Trees (RRT) is a widely adopted algorithm in robotics for motion planning due to its simplicity and probabilistic completeness. Over the years, researchers have made modifications to the basic RRT algorithm to improve its performance in dynamic environments, making it a favored planning algorithm for autonomous driving. Among these variants, probabilistic RRT (pRRT) demonstrates promising capabilities for efficient online replanning. The first part of the thesis thoroughly studies the pRRT algorithm and compares its performance to the standard RRT and RRT* algorithms through Python simulations. The pRRT algorithm outperformed the RRT and RRT* algorithms in terms of success rate and time to find a safe trajectory. The algorithm was implemented experimentally on scaled cars for the validation of its feasibility. The experimental results show good sim-to-real transfer for this algorithm.

The second part of the thesis proposes a novel algorithm for path planning. The algorithm outperforms the standard RRT and pRRT techniques in terms of optimality and conformance to human instincts. The generated paths are much smoother and easier for the controller to track. The AV implementation combines the probabilistic RRT with the RRT-Connect algorithm to mitigate the problem of parameter tuning of the standard pRRT algorithm. The idea is to generate intermediate critical points around the obstacles to grow multiple

trees between these points which are then eventually connected if a safe trajectory is found. The algorithm was tested in simulation and showed comparatively better performance in handling obstacles.

Motion Planning For Autonomous Vehicles In Non-Signalized Intersections

Darshit Patel

(GENERAL AUDIENCE ABSTRACT)

Due to uncontrolled traffic flow, non-signalized intersections are critical for autonomous driving. Motion planning is responsible for the vehicle's decision-making and generating actions based on its surroundings. Rapidly Exploring Random Trees (RRT) is one of the most widely used algorithms for motion planning in robotics due to its simplicity and a guarantee of finding a collision-free path if it exists. Due to the randomness of the algorithm, the time to find a collision-free path increases rapidly as the surrounding environment complicates.
In this thesis, we thoroughly study a modified version of RRT called the probabilistic RRT (pRRT) for motion planning of autonomous vehicles. The pRRT algorithm reduces the randomness of the standard RRT algorithm and takes into account the destination location and the positions of the obstacles to find a path around the obstacles and toward the destination point. The algorithm was experimentally validated and confirmed the simplistic transfer from simulations to reality.
In the second part of the thesis, we propose a novel algorithm that combines the properties of pRRT and another well-known algorithm called RRT-Connect. This algorithm plans collision-free paths from the start, and the goal points towards free space around the obstacles simultaneously and then combines these fragmented paths. This reduces the overall planning time and was found to be better at providing smooth paths.

Dedication

I dedicate this master's thesis to my loving family, whose unwavering support and encouragement have been the foundation of my academic journey. To my parents, who have always believed in my abilities and provided me with endless love and guidance, I am forever grateful. Your sacrifices and unwavering belief in me have been a constant source of motivation. I also dedicate this work to my sister, whose encouragement and understanding have been invaluable throughout this process. Lastly, I dedicate this thesis to my dear friends and mentors who have provided me with unwavering support, inspiration, and intellectual stimulation.

Acknowledgments

I would like to express my profound gratitude to my advisor, Dr. Azim Eskandarian, whose unwavering support, expertise, and guidance were instrumental in the successful completion of this master's thesis. His mentorship and encouragement have been invaluable throughout this journey, and I am truly grateful for his commitment to my academic and professional growth. I would also like to extend my heartfelt appreciation to Dr. Saied Taheri and Dr. Kaveh Akbari Hamed, the esteemed members of my thesis committee. I am thankful for their expertise and insightful feedback that has improved the quality of this thesis. I am deeply indebted to my family members for their unyielding love, encouragement, and continuous belief in my abilities. Additionally, I am grateful to all the members of my lab for their collaboration, camaraderie, and valuable insights. Their contributions, discussions, and brainstorming sessions have significantly enriched my research and broadened my understanding of the subject matter. Lastly, I would like to acknowledge the countless individuals whose names may not appear in this section but have, in various ways, influenced and shaped my academic journey. Their collective impact, whether through intellectual discussions, friendship, or support during challenging times, has played a vital role in my personal and professional development.

Contents

List of Figures

List of Tables

List of Abbreviations

APF Artificial Potential Fields.

AV Autonomous Vehicles.

DL Deep Learning.

ESC Electronic Speed Control.

FHWA Federal Highway Administration.

FLOP Floating Point Operation.

FoV Field of View.

GAN Generative Adversarial Networks.

GPR Gaussian Process Regression.

GPS Global Positioning System.

IMU Inertial Measurement Unit.

MPC Model Predictive Control.

OCP Optimal Control Problem.

OV Other Vehicle.

PDF Probability Density Function.

PPM Position Probability Map.

PRM Probabilistic RoadMap.

pRRT Probabilistic Rapidly Exploring Random Trees.

RC Radio Control.

RL Reinforcement Learning.

ROS Robot Operating System.

RRT Rapidly Exploring Random Trees.

SAE Society of Automotive Engineers.

SBP Sampling Based Planning.

SLAM Simulatneuous Localization and Mapping.

SQP Sequential Quadratic Problem.

USDoT United States Department of Transportation.

V2I Vehicle to Infrastructure.

V2V Vehicle to Vehicle.

V2X Vehicle to Everything.

Symbols

Δ	Spacing between discrete points of the environment
δ	Steering input
λ	Bias Parameter
$\mathcal{X}_{free}$	Obstacle free space
$\mathcal{X}_{obs}$	Occupied space
μ^i	OV Positions
ϕ^i	Normal probability distribution around OVs
ϕ_{max}	Maximum probability for the probability distribution
ϕ_{min}	Minimum of all the ϕ_{max} for OVs
ψ	Vehicle heading angle
σ	Bias standard deviation
u	Velocity input
v	Vehicle speed
x_0	Vehicle start position
x_{goal}	Vehicle destination position

Chapter 1

Introduction

The US transportation department reported a total of 33244 fatal motor crashes during 2019, leading to the death of 36096 individuals. Federal Highway Administration (FHWA) of the United States Department of Transportation(USDoT) reported that human error accounted for 94% of all the fatalities caused by driving incidents [1]. Autonomous driving has emerged as a groundbreaking technology that promises to revolutionize the transportation industry and redefine how we perceive mobility. With the rapid advancements in artificial intelligence, sensor technologies, and connectivity, autonomous vehicles have the potential to enhance road safety, reduce traffic congestion, and provide new opportunities for improved efficiency and convenience [2].

1.1 Autonomous Vehicles

Autonomous driving research has gained a lot of traction in recent years due to better computation and the rapid development of efficient hardware. The Society of Automotive Engineers (SAE) has categorized autonomous vehicles into six types starting from level 0 through level 5 [3]. Level 0 indicates no autonomy and complete manual control of the vehicle. Although, the vehicle can provide warnings and safety messages to drivers and can be classified as a vehicle with Driver's alert system. In levels one and two, the vehicle provides steering and/or braking assistance, depending on the conditions. For example, lane

centering or cruise control mode. When either one is present, it can be said as level 1, and when both, then it is level 2. From level 3 onwards, we start getting vehicles that drive on their own. Level 3 indicates minimum self-driving capabilities and that the driver needs to be aware with hands on the steering wheel at all times, while level 5 indicates complete autonomy and does not require human supervision.

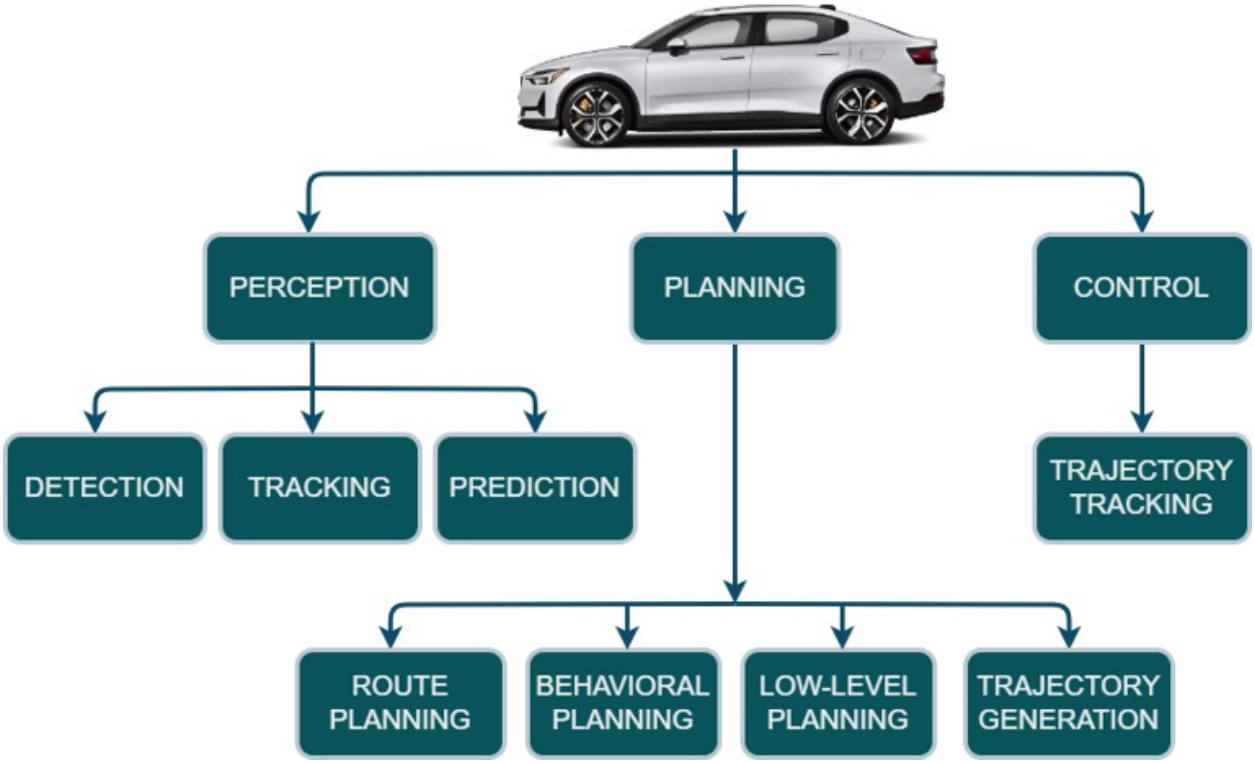


Figure 1.1: The software stack of an Autonomous Vehicle

Currently, there is no vehicle available commercially that can be classified as a level 5 self-driving car. To be classified as fully self-driving, the vehicle needs to have many capabilities to emulate a human driver. Figure 1.1 summarizes the different modules of a self driving car. Three overall components of autonomous driving are Perception, Planning, and Control [4, 5, 6]. Perception represents the eyes of a human driver, and the major task of this component is to understand the environment and break it down into objects of relevance. Recent developments have seen many new learning-based methods for environment perception like

the attention neural network [7] and reinforcement learning based (RL) methods [8]. Cameras, lidars, and radars are the most commonly used perception sensors in this component of autonomous driving. Vehicle communication technologies like V2X (Vehicle To Vehicle) and V2I (Vehicle To Infrastructure) are popular for combining the information obtained by different vehicles and remote units to share an overall understanding of the environment. [55]

The planning module can be considered the brain of autonomous driving. It uses the information obtained by the perception module and uses it to obtain a set of actions to complete the task of driving. This is one of the most active areas of research, as making safe and efficient driving decisions better than a human brain or at least emulating it is much more difficult. The path planning task can be further categorized into high-level route planning, low-level behavioral planning, path generation, and path tracking. The overall planning module satisfies the planning hierarchy in that given order. In the first step of route planning, an overall route from start to destination is obtained. This includes deciding which roads to take and optimizing the route. Once the route is obtained, the low-level planner makes decisions at each segment of the route [9]. For example, once the AV reaches an intersection, the low-level planner decides the plan to pass through the intersection [10]. Depending on the behavioral decision made by the low-level planner, a collision-free path is generated [11]. This generated path may consist of the cartesian coordinates that the vehicle needs to follow in order to pass through safely. In the trajectory generation step, a sequence of desired states and waypoints required for the vehicle to follow the path is generated. This stage provides the AV with control inputs at each time step which requires the algorithms to consider the vehicle dynamics and the road conditions [12].

The control module tracks the trajectory generated by the planning module. This module needs to account for the uncertainties and unexpected factors that may influence the vehicle's

motion. Simultaneous Localization and Mapping (SLAM) is often used to track the vehicle in the world frame and ensure the trajectory is safely followed [13].

1.2 Autonomous Vehicles at non-signalized intersections

Between 2017 and 2021, the National Highway Traffic Safety Administration (NHTSA) reported a staggering total of 31,297,306 traffic crashes. Among these crashes, there were 43,857 fatal crashes out of a total of 177,409, and 4,329,379 injury-only crashes out of a total of 9,019,571, that were reported to have occurred at intersections [14]. This highlights the significant impact of intersection-related incidents on road safety.

Furthermore, out of all the fatal crashes that occurred at intersections, approximately two-thirds were reported to have taken place at uncontrolled intersections. This indicates the importance of implementing effective intersection control measures to enhance safety. A striking statistic is that 97% of these intersection-related crashes involved drivers who were not paying proper attention. This emphasizes the critical role of driver attentiveness. Hence, for autonomous driving, it becomes important to develop methods that can handle such critical scenarios effectively.

A non-signalized intersection, also known as an unsignalized intersection, refers to an intersection where traffic movements are not regulated by traffic signals or traffic lights. Developing planning algorithms for autonomous vehicles at non-signalized intersections poses several challenges that need to be addressed. Some of these challenges include:

Uncertainty in other vehicles' intentions: At non-signalized intersections, it is difficult for autonomous vehicles to accurately predict the intentions of other vehicles, such as whether they will yield or proceed. This uncertainty adds complexity to the planning process and

requires robust algorithms to handle different possible scenarios.

Complex and dynamic traffic interactions: Non-signalized intersections often involve intricate traffic interactions, such as right-of-way rules, yielding, merging, and conflicting movements. Planning algorithms need to account for these complex interactions while ensuring safe and efficient navigation through the intersection.

Limited sensor visibility: The visibility of sensors may be limited due to occlusions, parked vehicles, or obstructed views at non-signalized intersections. This limitation makes it challenging to perceive and detect all relevant objects and accurately assess the traffic situation. Algorithms must be capable of handling incomplete or noisy sensor data.

Handling ambiguity and communication: Communication between autonomous vehicles and human drivers, pedestrians, or cyclists becomes crucial at non-signalized intersections. Algorithms need to account for potential ambiguity in communication signals, such as hand gestures or eye contact, and make appropriate decisions based on this information.

Real-time decision-making: Non-signalized intersections require rapid decision-making to navigate safely and efficiently. Planning algorithms must operate in real-time, efficiently searching and evaluating potential trajectories while considering various factors like vehicle dynamics, traffic conditions, and environmental constraints.

Hence, non-signalized intersections pose a major threat to autonomous vehicles as the chances of accidents with non-autonomous vehicles increase in such an unregulated traffic flow [15].

1.3 Contributions

The present technologies pertaining to low-level planning face several difficulties. The traditional optimization-based techniques are computationally heavy and ineffective for online

planning in complex scenarios. Some simpler algorithms, like graph-based algorithms, do not scale well with the environment and fail to handle dynamic scenarios effectively. Learning-based algorithms have gained attention in motion planning but rely heavily on previous data and tend to learn incorrect behaviors.

The probabilistic Rapidly Exploring Random Trees (pRRT) algorithm was proposed for the online planning of autonomous vehicles in intersections [16]. It adds the property of goal biasing to the standard RRT to obtain faster solutions in intersection environments. However, it is not known how well the algorithm performs compared to the widely used closed-loop RRT and RRT* algorithms in terms of efficiency and optimality. The current research on pRRT does not indicate the algorithm's scalability for bigger environments and multiple obstacles. The pRRT algorithm requires tuning of a few parameters, which makes the algorithm to be specifically designed for different scenarios. Due to this, handling multiple static obstacles is challenging.

This thesis addresses these problems, and the following major contributions are made to this field of research:

- A thorough sensitivity study of the various parameters of the pRRT algorithm to recognize a favorable range of values for autonomous driving.

- A comprehensive evaluation framework for comparing the motion planning algorithms that provide a standardized and objective way to compare the performance of different algorithms in driving scenarios.

- Developing a simulation framework to deploy any sampling-based or graph-based algorithm on non-signalized intersections with multiple lanes, multiple obstacles, and different vehicle speeds scenarios in real-time.

- Experimentally evaluating the pRRT algorithm to confirm the feasibility of the sim-to-real capabilities of the algorithm.

- Developing a new path planning algorithm for smoother paths for self-driving cars to overcome the difficulties of existing discrete path planning methods.

- Combining the strategies of the new algorithm, the standard pRRT, and the standard RRT Connect algorithms to mitigate a few difficulties encountered in the standard pRRT technique for motion planning.

1.4 Thesis Outline

The overall aim of this thesis is to obtain a good low-level local planner for autonomous driving, especially in the critical scenarios of non-signalized intersections. The thesis is organized as follows:

Chapter 2: This chapter reviews the existing low-level planning algorithms and explains the merits and demerits of those techniques

Chapter 3: This chapter introduces the popular sampling-based algorithms, RRT, RRT*, and pRRT, and their implementation for path planning. The chapter also describes the implementation of these algorithms for autonomous driving conditions.

Chapter 4: This chapter analyzes and discusses those algorithms for non-signalized intersection scenarios and makes a thorough comparison of their efficiency and safety in driving.

Chapter 5: This chapter shows the details about the experimental evaluation of the pRRT algorithm and shows the simulation-to-reality capabilities of this technique.

Chapter 6: This chapter introduces a new discrete path-planning algorithm to generate

smoother paths that are easier to track than the ones generated by traditional algorithms. Using this technique, a modification to the standard pRRT technique is introduced to mitigate some of the design difficulties of the algorithm.

Chapter 7: This chapter summarizes the overall work presented in this thesis and outlines some of the future aspects that can enhance the work done in this thesis.

Chapter 2

Review of Literature

The movement of an autonomous vehicle (AV) from its starting point to its destination is guided by a decision-making hierarchy. At the lower level, behavioral decision-making is employed to determine a short-term plan for specific situations like overtaking. Once the decision is made, the motion planner generates a collision-free path and trajectory to navigate through the traffic [17]. The process of motion planning consists of two main stages: path planning and trajectory planning. Over the years, various algorithms have been developed and tested to obtain feasible paths for mobile robots [18]. According to Zhou et al., path-planning algorithms can be classified into two broad categories: machine learning-based approaches and traditional approaches [19].

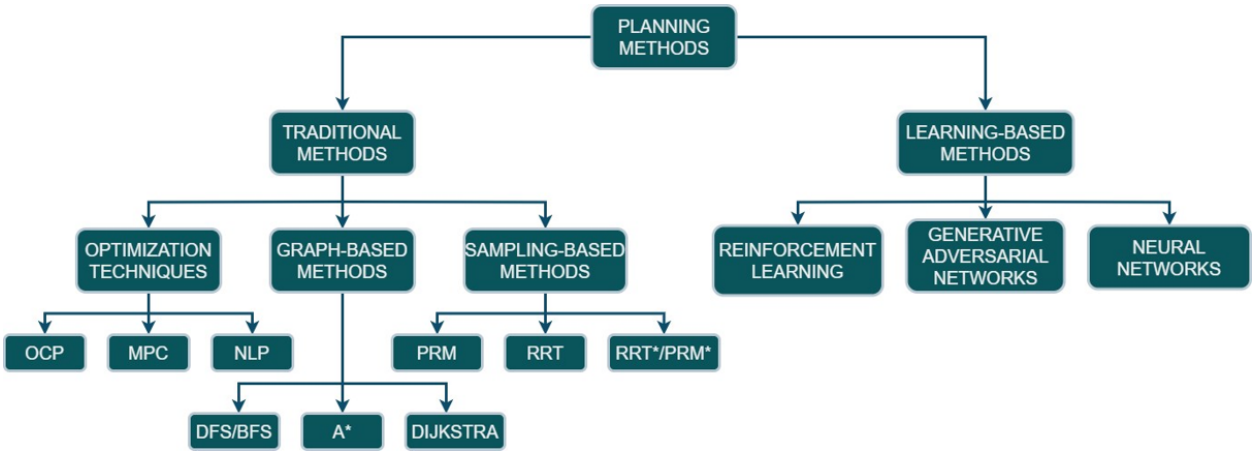


Figure 2.1: Classification of low-level planning methods

Data-driven learning-based algorithms are characterized by their reliance on extensive train-

ing in order to achieve accurate and safe planning. Neural Networks are one of the most widely used learning methods. They have been employed to generate optimal motion plans for mobile robots and manipulators using training data [20]. Within motion planning research, reinforcement learning (RL) algorithms have garnered attention due to their capacity to generate efficient policies based on previously gathered driving data [21]. Numerous Deep Reinforcement Learning (DRL) algorithms have been examined for autonomous navigation tasks [22, 23]. Typically, these methods involve offline model training, followed by the deployment of the learned policy/model on the vehicle. Generative Adversarial Networks (GANs) have also gained attention for generating optimal motion plans for autonomous driving [24]. However, such learning algorithms necessitate a substantial amount of data, which is not readily accessible for driving scenarios. Additionally, they run the risk of learning unsafe policies due to ambiguous reward functions. Consequently, the decisions made by these techniques may be unpredictable, and ascertaining what the model has learned proves to be exceptionally challenging.

In contrast to learning-based algorithms, traditional path-planning algorithms adhere to a set of predefined rules, making their implementation simpler and their behavior more predictable. This category of algorithms can be further classified into several subcategories, including geometric curves based planning, numerical optimization methods, graph-based search algorithms, and sampling-based techniques. Geometric curves have garnered attention in path planning research due to their ability to generate continuous and smooth paths. Researchers have explored the use of cubic splines [25], continuous curvature paths [26], and bezier curves [27] to generate smooth paths for autonomous navigation. These techniques are typically employed to achieve smoother paths, but they cannot generate trajectories that include control inputs at each time step. To address this, a separate trajectory planner is required to generate feasible trajectories for path tracking.

Numerical optimization techniques are often utilized to generate optimal trajectories, focusing on generating control commands rather than a specific path. These techniques are commonly employed in racing scenarios to obtain the fastest trajectories [28, 29]. They are particularly suitable for closed environments where constraints can be precisely defined. Optimal control problems (OCPs) are also popular for motion planning in urban environments. For instance, [30] introduced the autonomous parking problem as an optimal control problem, formulating it using non-holonomic constraints and employing sequential quadratic programming (SQP) to solve the optimization problem. Recent advancements in this field include the development of lightweight frameworks for optimal control problems related to autonomous parking, as demonstrated by [31]. Model Predictive Control (MPC) is widely used for planning trajectories for autonomous driving as they take into account the vehicle model and provide feasible paths [32]. But as the number of obstacles increases, the problem becomes more complex, and it becomes computationally expensive to solve for a collision-free trajectory.

Artificial Potential Fields (APF) are often integrated into optimization-based methods to introduce collision avoidance as a constraint [33]. However, these techniques are often computationally expensive and highly sensitive to the vehicle model used in the problem formulation. As a result, they are typically employed in modular planning tasks such as parking, emergency collision avoidance, and cruise control. In complex dynamic scenarios like non-signalized intersections, these techniques are not suitable for online planning due to their limitations.

Graph-based algorithms and sampling-based algorithms are extensively utilized in the path planning of various systems, including mobile robots, robotic arms, and autonomous vehicles. These techniques enable the determination of trajectories from a starting position while ensuring collision avoidance with obstacles in the environment. In comparison to

optimization-based or learning-based approaches, graph-based and sampling-based methods offer computational efficiency and a lightweight framework.

Graph-search algorithms such as A* and Dijkstra are renowned for their ability to find the shortest paths. However, they may not converge to an optimal solution quickly and thus are commonly employed for high-level route planning [34]. Examples of their implementation in autonomous driving scenarios can be found in [35, 36]. Another approach, presented in [37], involves constructing a graph comprising all possible straight-line paths between points in the environment. Siméon et al. introduced the concept of generating a graph based on the critical vertices of obstacles, ensuring that the generated graph does not intersect with any obstacles [38]. These algorithms then search the graph to find the shortest path between the starting and ending points. Graph-search algorithms are well-suited for closed environments with static objects but may not scale well to large environments and are not suitable for rapid re-planning in dynamic environments.

Sampling-based algorithms, on the other hand, take a different approach by randomly sampling points in the free space and connecting them to form a roadmap. Popular sampling-based algorithms include Rapidly-exploring Random Trees (RRT) and Probabilistic Roadmaps (PRM). These algorithms offer advantages such as scalability and adaptability to dynamic environments. RRT-based algorithms rapidly explore the configuration space, while PRM-based algorithms construct a network of feasible paths. They are particularly effective in scenarios where quick re-planning and adaptability to changes are crucial.

In summary, graph-based algorithms and sampling-based algorithms are widely utilized for path planning in various systems. Graph-search algorithms excel in finding the shortest paths but are more suitable for closed environments with static objects. Sampling-based algorithms provide scalability and adaptability to dynamic environments, making them well-suited for quick re-planning and handling changes in the environment.

Sampling-based planning (SBP) methods, such as Rapidly Exploring Random Trees (RRT) [39] and Probabilistic Roadmaps (PRM) [40], have been extensively employed in robotics for path planning purposes. These techniques offer several advantages compared to graph-based methods, particularly in terms of scalability for large environments and suitability for dynamic environments [41]. Researchers have made various modifications to these standard algorithms in order to obtain optimal trajectories, resulting in variants such as PRM* and RRT* [42].

PRM has primarily been used with holonomic mobile robots in static environments, such as industrial robots, while RRT has demonstrated greater suitability for non-holonomic dynamic systems, including autonomous driving scenarios. Optimal planning using modified versions of RRT has been employed for high-level planning tasks. However, it should be noted that these approaches are less suitable for online planning due to their computational requirements and complexity [43], [44], [45]. In complex scenarios, basic SBPs may require more time to generate feasible trajectories.

In order to improve the performance of basic sampling-based planning (SBP) methods, several modifications have been introduced to achieve faster and more feasible solutions [46]. One such modification is Bi-RRT, which reduces convergence time by simultaneously growing two trees from the start and goal positions and connecting them [47]. Another approach, presented in [48], combines the properties of Bi-RRT and PRM algorithms by creating two distinct graphs and connecting them using optimally matching nodes. It is important to note that these algorithms assume a static environment, and the generated paths may not be suitable for dynamic environments. Additionally, the connection between the two trees may result in a discontinuous path that can be challenging to track.

To address these limitations, Closed Loop RRT (CLRRT) was proposed, which considers vehicle inputs as sampling nodes instead of spatial coordinates. The tree is expanded us-

ing the forward dynamics of the controller, resulting in a smooth and dynamically feasible trajectory instead of discontinuous path segments. This modification enables the algorithm to simultaneously plan the path and trajectory [49]. Various vehicle models can be used to describe the non-holonomic dynamics of a vehicle. For example, [50] incorporates a Dubin's car model into the RRT algorithm for path finding in a mobile robot. In this context, a kinematic bicycle model is used to describe the vehicle dynamics for a simplified comparison of the algorithms.

It is worth noting that these RRT-based techniques can sometimes increase convergence time significantly, as they may explore unnecessary areas of the environment, particularly in driving scenarios.

Research efforts have been directed towards reducing unnecessary sampling in order to improve the efficiency of sampling-based planning methods. Informed RRT* is one such technique that recursively shrinks the sampling space after obtaining an initial solution using RRT, aiming to converge to an optimal trajectory [51]. Although this method is probabilistic complete, it theoretically requires exponential time to converge to a globally optimal solution.

Dynamic Domain RRT (DDRRT) addresses the problem by dividing the environment into different domains and forming trees within these domains individually. These trees are later connected to obtain a final feasible trajectory [52]. However, connecting these trees can be challenging and may result in infeasible trajectories.

NC-RRT (Node Connection RRT) aims to enhance the performance of RRT by reducing the number of unnecessary nodes in the tree and avoiding getting stuck in local minima [53]. While this technique can improve efficiency by removing unnecessary nodes, it may be more computationally expensive compared to standard RRT due to the additional steps involved.

These techniques rely on random uniform sampling to grow trees in unexplored environments. However, the inherent randomness of the sampling process makes online re-planning using RRT difficult, particularly in larger environments where the computational burden becomes more significant [54].

In dynamic planning scenarios, it is essential for autonomous vehicles to have knowledge about the states and motion of other vehicles. Vehicle-to-Vehicle (V2V) communications enable autonomous vehicles to obtain information about the surrounding vehicles using Vehicle-to-Infrastructure (V2I) techniques [55]. By leveraging techniques like Gaussian Process Regression (GPR) [57], the current states of other vehicles can be used to predict their future states and assess the likelihood of potential collisions. GPR has been employed to predict obstacle positions in dynamic environments, enabling the definition of free spaces for sampling points in RRT away from predicted obstacle positions [58].

The ST-RRT* algorithm plans both the trajectory and the timing of a robot's movements in a 4D space-time domain, making it suitable for dynamic environments with obstacle motion prediction. However, this method is complex and computationally intensive [59]. Goal biasing techniques, initially introduced for unmanned air vehicles [60], have been applied to explore the environment more efficiently towards the destination. Theta-RRT* combines RRT with any-angle search, expanding the tree towards the goal direction with a higher probability, resulting in faster convergence to the goal [61]. However, Theta-RRT* can be computationally expensive due to the need to calculate the cost to the goal at each iteration.

Probabilistic RRT (pRRT) incorporates biasing and prediction techniques to create a non-uniform sampling of points in the environment. It generates a position probability map (PPM) that assigns a negative bias to predicted obstacle positions and a positive bias to the goal position. RRT is then used to plan a trajectory with sampling performed according to the PPM [16]. The results demonstrate significantly faster online re-planning, particularly

in intersection scenarios, when a high bias factor is applied.

These techniques, which integrate biasing and prediction, offer improved performance and faster convergence for planning in dynamic environments. However, it is important to consider the computational complexity and potential trade-offs when selecting the appropriate method for a given application.

Chapter 3

Sampling Based Planning Algorithms

This chapter introduces the low-level planning problem for autonomous driving in section 3.1. The algorithmic details of the popular sampling-based planning algorithms, RRT (Rapidly Exploring Random Trees) and RRT*, are summarized in section 3.2. To mitigate some of their drawbacks, the pRRT (Probabilistic RRT) algorithm was introduced by Wu et al. in [16]. A brief introduction to the pRRT technique is presented as well. Implementation of these algorithms for the purpose of autonomous driving is provided in section 3.3.

3.1 Problem Definition

A state-space, $\mathcal{X} \subseteq \mathbb{R}^n$, and an input-space, $U \subseteq \mathbb{R}^m$, are defined such that $x(t) \in \mathcal{X}$ and $u(t) \in U$ are the states and the inputs of the system at time t respectively. The coordinate space is divided into two disjoint sets, $\mathcal{X}_{free}(t)$ denoting obstacle-free region and $\mathcal{X}_{obs}(t)$ denoting region with obstacles, such that $\mathcal{X}_{free} \cup \mathcal{X}_{obs} = \mathcal{X}$. The purpose of the motion planning problem is to find a set of inputs $u(t)$ such that the ego vehicle follows a series of states $x(t) \in \mathcal{X}_{free}(t)$ to reach a goal state $x_{goal} \in \mathcal{X}_{free}(t)$, from an initial state $x_0 \in \mathcal{X}_{free}(t = 0)$, in a finite time interval $[0, t_f]$ while obeying the system dynamics. The $\mathcal{X}_{free}$ and $\mathcal{X}_{obs}$ are dynamic due to the moving traffic environment. This coordinate space is discretized such that the horizontal and vertical distance between two adjacent points is Δ.

3.2 Details of the algorithms

3.2.1 Rapidly Exploring Random Trees (RRT)

RRT constructs a tree in the free space $\mathcal{X}_{free}$ to connect an initial state to a goal position. It follows a heuristic approach to building this tree as described below:
First, a tree, $T(N, E)$, is initialized with x_0 as the root node. Here N represents the tree's nodes, and E represents the edges of the tree. The tree grows iteratively until the goal state x_{goal} is reached. It consecutively implements the following functions in order to generate a successful path.

Sampling: During each iteration, a point is randomly sampled such that $p(X, Y) \in \mathcal{X}$, where x and y denotes the point coordinates in the surrounding environment called x_{sample}. This sampling is uniform, and every point in the environment has an equal probability of getting sampled.
Nearest Neighbour: Once a point is sampled, this procedure finds a node state in the growing tree, $x_{nearest} \in T(N, E)$, which has the lowest Euclidean distance to the sampled point by comparing the distances to all the nodes of $T(N, E)$.
Steer: This function generates a new node x_{new} from $x_{nearest}$ towards the sampled point x_{sample}. To avoid large edges and maintain the ability to find paths around the obstacles, the maximum length of the new edges is limited to the spacing between the discrete points (Δ). Considering the above rules, the new node x_{new} is generated.
Collision Check: A new edge is created between $x_{nearest}$ and x_{new}. If this edge does not lie in $\mathcal{X}_{free}(t)$, the edge and x_{new} will be discarded, and the algorithm continues with other iterations. If the edge does lie in $\mathcal{X}_{free}(t)$, the new state is added to the tree, $T(N, E)$, as a new node with $x_{nearest}$ as its parent node.

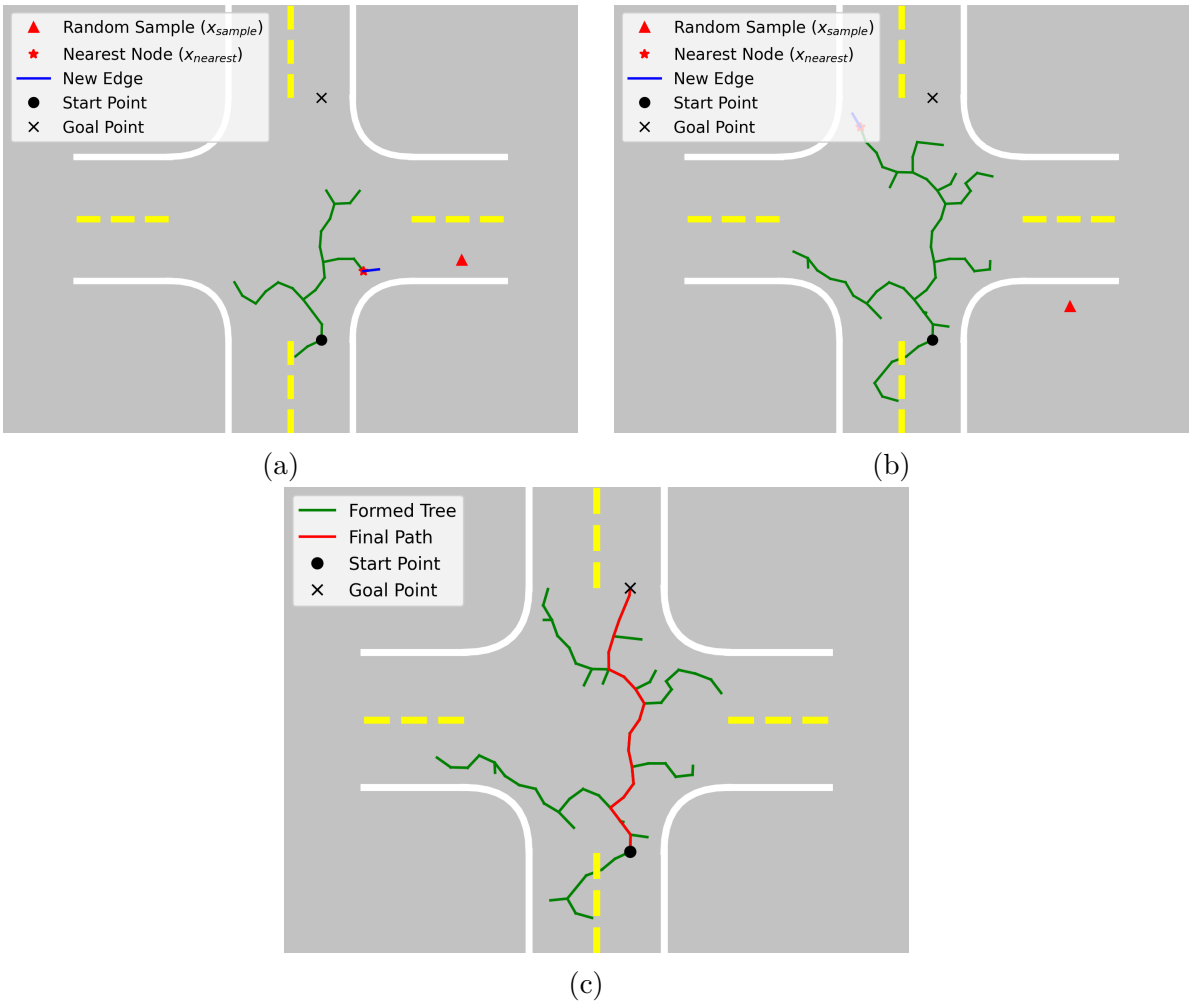


Figure 3.1: Path planning using RRT in an empty intersection environment: a) As the edge generated by the newly sampled point and the growing tree is in the free space, the edge is added to the growing tree; b) New edge is not formed by the sampled point as the edge generated by the sampled point and the growing tree results in a collision with the intersection boundary; c) The final path generated by the RRT algorithm

Convergence Criteria: A threshold radius, d_{goal}, is defined such that if the distance between the new node, x_{new}, and the goal state, x_{goal}, becomes less than d_{goal}, the iterative loop is terminated and a path is successfully obtained. Alternatively, maximum allowable iterations are defined so that the algorithm does not enter an infinite in case a feasible

Algorithm 1 Rapidly-Exploring Random Tree

$T \leftarrow InitializeTree(x_0)$
$it = 0$
while $it \leq maxIter$ **do**
  $x_{sample} \leftarrow random\_state(x_{free})$
  $x_{nearest} \leftarrow nearest\_node(x_{sample})$
  $x_{new} \leftarrow steer(x_{free}, x_{rand})$
  if Collision$(x_{nearest}, x_{new})$ = False **then**
    $T \leftarrow InsertNode(x_{new})$
    $T \leftarrow InsertEdge(x_{nearest}, x_{new})$
  if $dist(x_{new}, x_{goal}) \leq r_{goal}$ **then**
    break
  $it = it + 1$

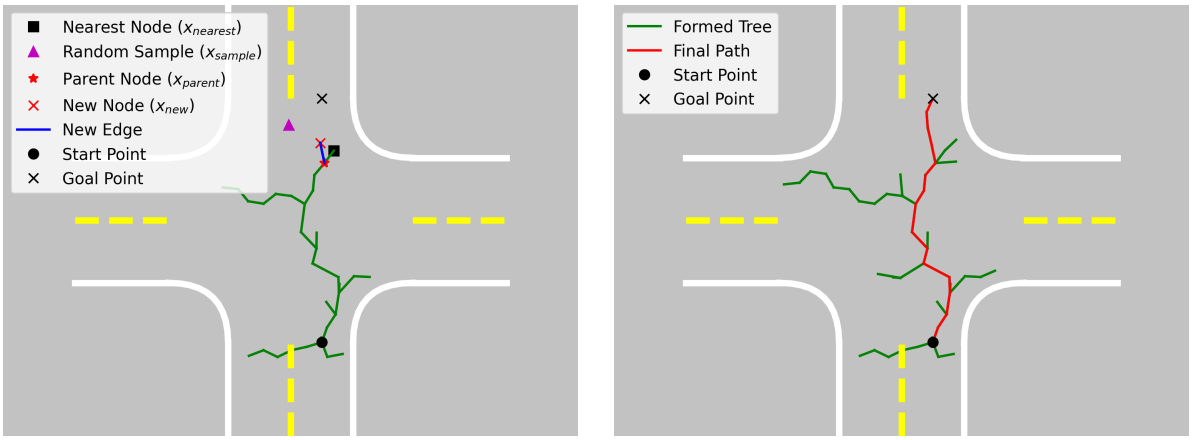
solution does not exist.

Figure 3.1 shows an example of a path generated by the RRT algorithm. The tree starts growing from the start point and explores the intersection environment until it can find a safe path to the goal point. Algorithm 1 shows the steps for the RRT motion planning method.

3.2.2 RRT*

RRT* is a modified version of the standard RRT with a few additional steps to obtain an optimal path as the time approaches infinity. A new variable, *cost*, is added to the nodes in addition to the cartesian coordinates. RRT* algorithm converges towards a reduced planning cost. To find the shortest path, we consider the cumulative Euclidean distance to travel from x_0 to x_{new} as the cost of x_{new}. For planning, a tree, $T(N, E)$, is initialized and follows the same architecture as RRT until the steering step to obtain x_{new}. After obtaining x_{new}, the algorithm performs the following tasks to obtain an optimal path to the goal node.

Finding Close Neighbors: Given a new state x_{new}, a tree $T(N, E)$, and a search radius r, this procedure creates a circle with radius r from the center x_{new} and finds all the



(a) (b)

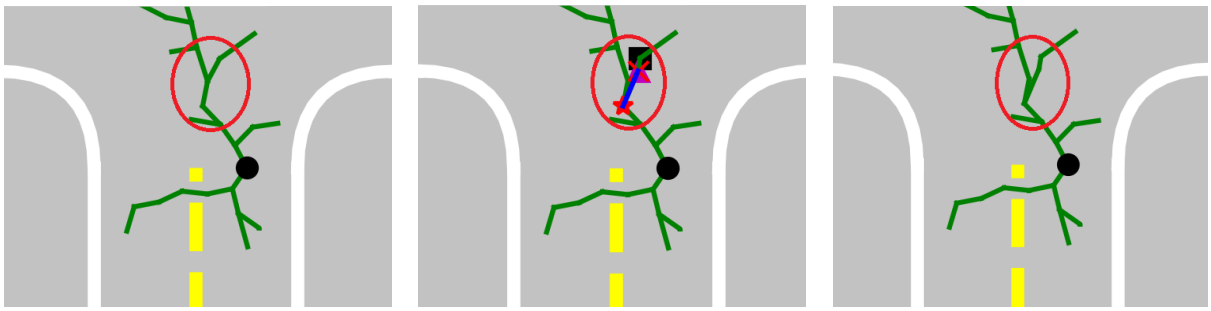
Figure 3.2: Path planning using RRT* in an empty intersection environment: a) Demonstration of the "choose parent" step of the RRT* algorithm; b) The final path generated by the RRT* algorithm

nodes of the growing tree within this circle. All these close neighboring nodes are stored in $near\_nodes$.

Choose Parent: This procedure calculates the cost of reaching x_{new} from all the nodes in $near\_nodes$ and selects the node with the lowest cost to be the parent of x_{new}. While choosing the parent node, the procedure also makes sure that the edge created between x_{new} and the parent node lies in $\mathcal{X}_{free}(t)$.

Rewire: This procedure checks each node in $near\_nodes$ to see if the overall cost for going to x_{near} in $near\_nodes$ via x_{new} is lower than going from x_{near} to x_{new}. For such a node, this procedure modifies the tree to make x_{new} the parent of x_{near} and attaches this edge to the growing tree $T(N, E)$.

Figure 3.2 shows the growth of the tree using RRT* algorithm. The algorithm converges the same way as previously described in the standard RRT algorithm. Hence, the convergence criteria we use do not let the algorithm converge to a globally optimal solution. This algorithm is probabilistic complete, meaning it can convert to a globally optimal solution as time approaches infinity. Figure 3.3 shows the rewiring stage of the RRT* algorithm. This



(a) (b) (c)

Figure 3.3: Rewiring of the tree. One branch of the tree rewires and changes its shape due to the rewiring step.

step finds that the nearest node can be reached faster from the sampled point and hence reshapes the tree in that region. The summary for the RRT* method is outlined in algorithm 2.

Algorithm 2 RRT*

$T \leftarrow InitializeTree(x_0)$
$it = 0$
while $it \leq N$ **do**
  $x_{rand} \leftarrow random\_state(x_{free})$
  $x_{nearest} \leftarrow nearest\_node(x_{rand})$
  $x_{new} \leftarrow steer(x_{free}, x_{rand}, \Delta t)$
  $near\_nodes = find\_neighbors(x_{new}, T, r)$
  $x_{parent} \leftarrow ChooseParent(x_{new}, near\_nodes, x_{nearest})$
  if Collision$(x_{nearest}, x_{new})$ = NULL **then**
    $T \leftarrow InsertNode(x_{new})$
    $T \leftarrow InsertEdge(x_{parent}, x_{new})$
  if $dist(x_{new}, x_{goal}) \leq r_{goal}$ **then**
    break
  $T \leftarrow Rewire(T, near\_nodes, x_{new})$
  $it = it + 1$

3.2.3 Probabilistic RRT

Probabilistic RRT has a similar structure as RRT. The main difference occurs in sampling a random point in the environment. For basic RRT, the distribution of random points in the environment is uniform, while for pRRT, the distribution is biased towards x_{goal} and away from obstacles using a Position Probability Map.

Position Probability Map (PPM)

It is assumed that the positions of the obstacles are known to us. Using this information along with the known goal position, the PPM creates a positive bias for random point sampling towards the destination and a negative bias towards the obstacle positions. This can significantly reduce the number of iterations as unnecessary points are not sampled, and more points are sampled toward the goal state. While sampling the points only at the goal position would drastically reduce the number of iterations, it would not preserve the alternative edges property of RRT in case replanning is required due to the sudden entry of an obstacle vehicle. This alternative edge property of RRT allows the vehicle to explore the environment more thoroughly in case a simple and straightforward path is infeasible.

This sampling bias is incorporated through a position probability map (PPM) that mixes Gaussian Probability Density Functions (PDFs) of the goal and obstacles as shown in [57]. Considering μ^{dest} as the destination location, and $\sigma^{2,dest}$ as a user-defined covariance matrix, the destination pdf is expressed as:

$$\phi^{goal} = \mathcal{N}(\mu^{goal}, \sigma^{2,goal}) \tag{3.1}$$

Similarly, the PDFs for obstacles and their predictions can be defined using their positions

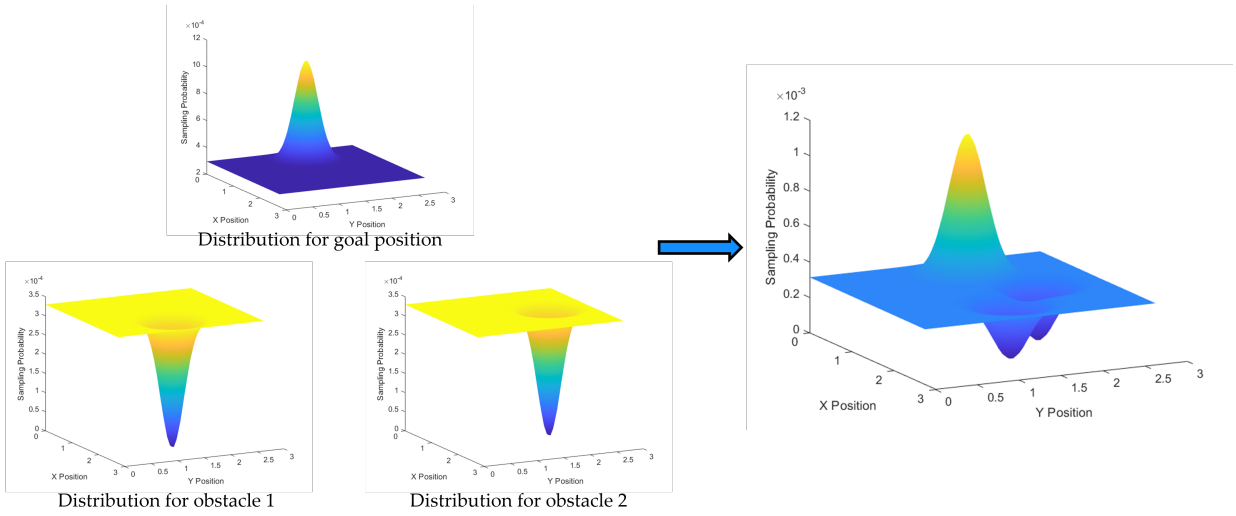


Figure 3.4: Formation of the position probability map using Gaussian distributions

μ^i and their covariance matrices $\sigma^{2,i}$ for N obstacles in the environment as follows:

$$\phi^i = \mathcal{N}(\mu^i, \sigma^{2,i}) \tag{3.2}$$

Using these PDFs, the final probability map for the intersection can be obtained as follows:

$$\phi^{map} = \lambda \cdot \frac{\phi_{min}}{\phi^{goal}_{max}} \cdot \phi^{goal} - \sum_{i=0}^{n} \lambda \cdot \frac{\phi_{min}}{\phi^{i}_{max}} \cdot \phi^i + \phi_{min} \tag{3.3}$$

where,

$$\phi_{max} = max(\phi) \qquad \text{and} \qquad \phi_{min} = min(\phi^i_{max} | i = 1, 2, .., n) \tag{3.4}$$

Here λ is the bias parameter that dictates the point sampling bias towards the goal position. To get zero probability density at the exact positions of OVs, all the PDFs were re-scaled using their maximum values. This process can be seen in figure 3.4. Here the Gaussian of obstacle positions are inverted such that the probability at the mean position is 0. Theλ parameter defines the relative scale of the goal bias while the σ parameter defines the width of the Gaussian distributions. Here uniform variance was considered but the variances and

means can be varied individually for each obstacle depending on the uncertainty of their position estimations.

Figure 3.5 shows an example of a position probability map. The probability density at

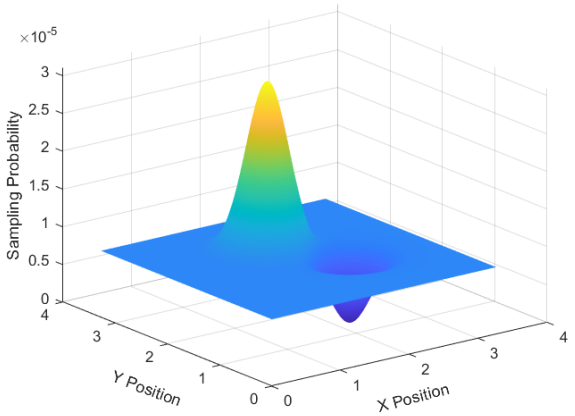
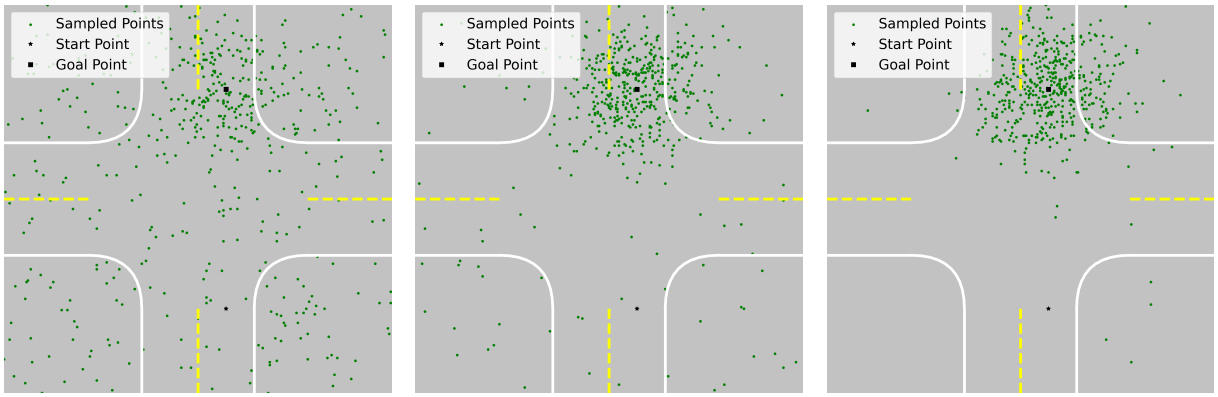


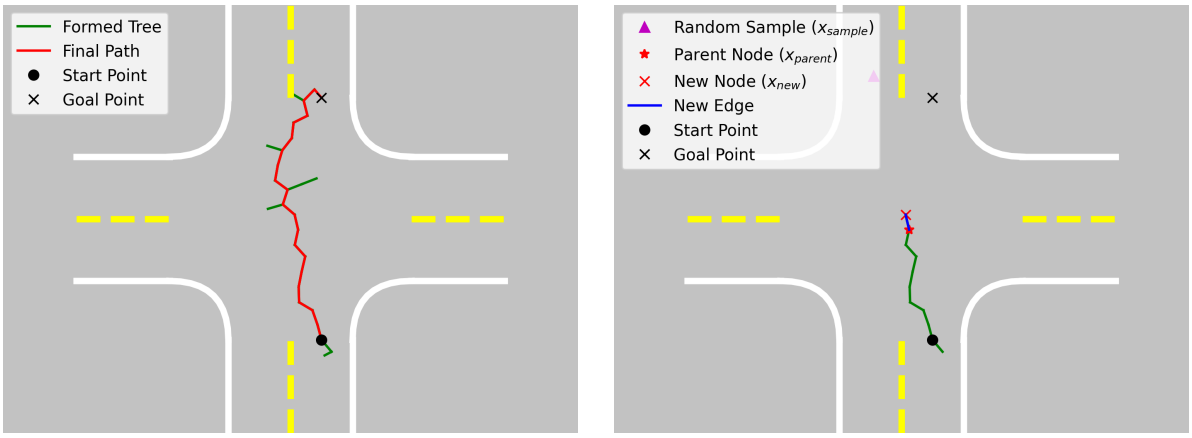
Figure 3.5: Example of a Position Probability Map with $\lambda = 5$ and $\sigma = 0.25$.



(a) (b) (c)

Figure 3.6: Sampling of 500 points using the probability position map without any obstacles in the intersection environment with $\sigma = 0.25$: a)$\lambda = 10$; b)$\lambda = 100$; c)$\lambda = 1000$

current and predicted obstacles vehicle positions is 0, while it is maximum at the goal position. The probability position map depends on the λ and σ parameters. The effects of these parameters are reported in the upcoming chapters. For the pRRT algorithm, the sampling of random points occurs according to the non-uniform distribution obtained using



(a) (b)

Figure 3.7: Path planning using pRRT in an empty intersection environment

the Probability Position Map. The rest of the algorithm's structure is similar to the standard RRT algorithm. Figure 3.7 shows the growth of the tree using the pRRT algorithm. The procedure is similar to RRT, but the sampling is done using the PPM. The step-by-step procedure is listed in Algorithm 3.

Algorithm 3 pRRT

$T \leftarrow InitializeTree(x_0)$
$it = 0$
for obstacle i **do**
  $P^i, \sigma^i \leftarrow Gaussian(obstacle)$
$PPM \leftarrow generateMAP(P^{1..N}, \sigma^{1..N})$
while $it \leq maxIter$ **do**
  $x_{rand} \leftarrow random\_state(PPM)$
  $x_{nearest} \leftarrow nearest\_node(x_{rand})$
  $x_{new} \leftarrow steer(x_{nearest}, x_{rand}, \Delta t)$
  if Collision$(x_{nearest}, x_{new})$ = NULL **then**
    $T \leftarrow InsertNode(x_{new})$
    $T \leftarrow InsertEdge(x_{nearest}, x_{new})$
  if $dist(x_{new}, x_{goal}) \leq r_{goal}$ **then**
    break
  $it = it + 1$

3.3 Implementation for autonomous driving

The previous sections described the general sampling-based algorithms. As seen before, they result in discontinuous paths, which are often not feasible for vehicle movement. To mitigate this problem, closed-loop RRT (CL-RRT) was introduced [49]. This algorithm uses vehicle dynamics in the steering step of RRT. Instead of creating a straight line from the nearest node to the sampled point, the algorithm uses the vehicle model to generate a new node and a new edge between the nearest node and the sampled point. To describe the vehicle model, we define the state space as $\mathcal{X} \subseteq \mathbb{R}^4$, and each state x consists of a local horizontal coordinate X, a local vertical coordinate Y, a yaw angle ψ, and a local velocity v. The control space is defined as $U \subseteq \mathbb{R}^2$, and every control input u includes a steering angle δ and the vehicle velocity v.

$$\begin{aligned} x &= [X, Y, \psi, v] \\ u &= [u_1, u_2] = [\delta, v_{in}] \end{aligned} \tag{3.5}$$

The node for the algorithms is comprised of its parent node, the state of the vehicle, the control input at its parent node, and the total path length from the start position to the node. Using this, the new steering step of the algorithms for autonomous driving can be defined as:

Steer: This function generates a control input, u, to drive the system from $x_{nearest}$ to x_{rand} while obeying the dynamic model constraints and the input constraints. Due to a constant time-step Δt, the system might not reach the sampled state. A new state, x_{new} would be generated using $x_{new} = x_{nearest} + f_v(x_{nearest}, u)\Delta t$ where the vehicle motion model is defined as $\dot{x} = f_v(x, u)$. In this thesis, we use the kinematic bicycle model for the vehicle model as described in [56]. The control input u for generating the new node is selected using the optimal control problem defined below:

Control input selection: This control problem tries to maximize the vehicle speed and

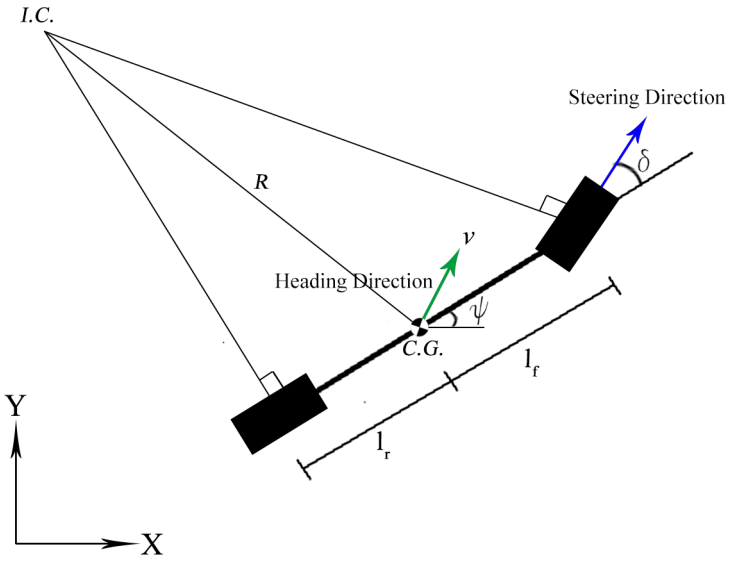


Figure 3.8: Bicycle Kinematic Model

minimize the time to reach the sampled point. The steering angle and velocity bounds were considered as the constraints to the control problem. The objective function is defined as:

$$u_{optimal} = \min_{u} J(x,u) \qquad s.t. -\delta_{max} \leq u_1 \leq \delta_{max} \\ 0 \leq u_2 \leq v_{max}. \tag{3.6}$$

Here J is the cost function and is defined to minimize the change in control inputs and lateral acceleration and maximize the distance covered in the fixed duration of Δt. The cost expression is defined below:

$$J = w_1 \left\{ \left(\frac{u_1 - u_{1,near}}{\delta_{max}} \right)^2 + \left(\frac{u_2 - u_{2,near}}{v_{max}} \right)^2 \right\} + w_2 \left(\frac{v_{near}^2}{R} \right)^2 + \\ w_3 \left(\frac{v - v_{max}}{v_{max}} \right)^2 + w_4 \left(dist_{new} - dist_{ref} \right)^2 \tag{3.7}$$

Here, R refers to the turning radius of the vehicle and can be obtained using the following expression:

$$R^2 = \frac{(l_f + l_r)^2}{\tan^2 \delta} + l_r^2 \tag{3.8}$$

Using the same velocity input as the previous point (nearest node) and 0 steering angle, $u_{ref} = [0, u_{2,near}]$ a reference state is obtained using the vehicle model, $x_{ref} = f_v(x_{near}, u_{ref})$. The Euclidean distance between the reference state and the sampled point is considered as the reference distance, $dist_{ref}$, in equation 3.7. $dist_{new}$ is the Euclidean distance between the new state and the sampled point.

Using the optimal control input to determine the new node for the algorithms results in a dynamically feasible edge of the tree. Hence the final tree and the feasible trajectory generated are smoother and without any discontinuities. The next chapter explains the trajectories generated by the algorithms in detail.

3.3.1 Prediction of obstacle vehicles' motion

$\mathcal{X}_{obs}$ for a road scenario is defined by the road boundaries and the positions of the obstacle vehicles and their predictions. The study is conducted on a non-signalized intersection; hence, the boundaries are stationary. Due to the consideration of flowing traffic, the OV space is time-dependent. Gaussian Process Regression (GPR) is used to predict the location of OVs in the environment as shown in [57] using training data. According to GPR, a model is trained using a training dataset $D = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$, where x_i is the vehicle state input and y_i is the predicted output.

$$f \sim gp(m(x), k(x, x')) \tag{3.9}$$

where $m(x)$ is the mean function and $k(x, x')$ is the covariance function which describes the correlation between the function values at two different input points x and x'. Given a new input x, the trained model f predicts the output. The horizontal coordinate X and the vertical coordinate Y are predicted using two different trained models, gp_x and gp_y, respectively. It is assumed that the current states of the OVs are known to the ego vehicle (using V2I or V2V techniques), which includes the position, yaw angle, and velocity. Given the current state of the OVs, the GPR models predict their location after a desired time gap dt

Chapter 4

Simulation results and discussion

In this chapter, we have compared the performance of RRT, RRT*, and pRRT for a non-signalized intersection environment. Section 4.1 conducts a thorough simulation study to validate the feasibility of the pRRT algorithm for autonomous driving.

4.1 Simulation Study

4.1.1 Simulation Setup

RRT, RRT*, and pRRT were compared through simulations in a Python environment. Four-way intersections were generated as shown in Figure 4.1. The ego vehicle is represented in red, while the obstacle vehicles are indicated in black. The initial position for the vehicle is fixed at the bottom entry. Depending on the scenario, the ego vehicle either goes straight toward the top exit or turns to the left or right exit. The intersection is not regulated by a traffic signal or stop signs indicating a non-systematic flow of traffic. The width of the lane was considered as 12 ft. in accordance to [62]. Average car width of 6 ft. was used for the simulation. For simulation simplicity and provision for future experimental tests, the whole setup was scaled down by a factor of 30. Hence 1 unit in the simulation represents a real dimension of 30ft.

The path of obstacle vehicles is pre-defined using a bezier curve depending on the scenario.

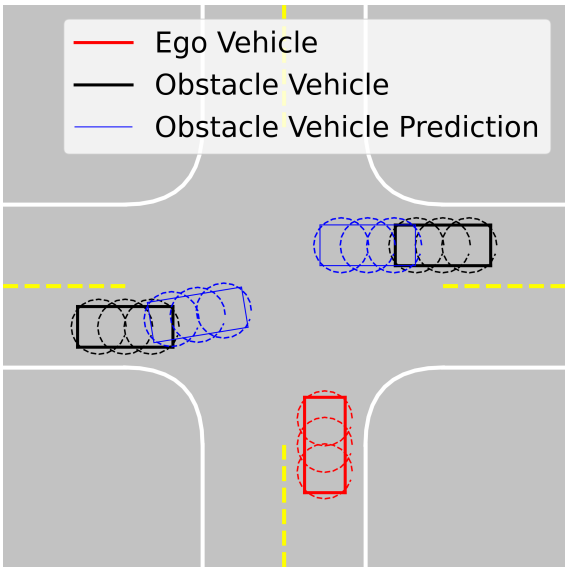
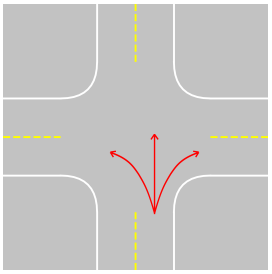
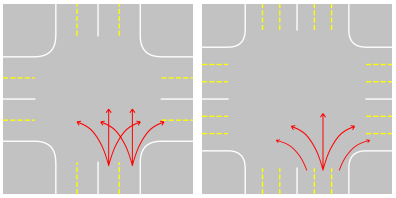


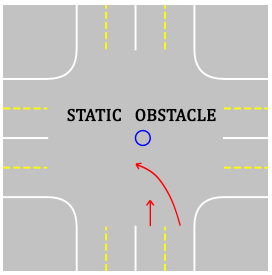
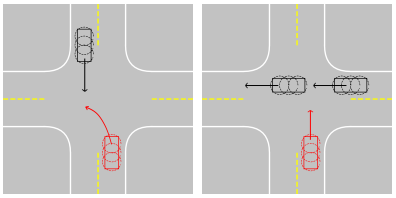
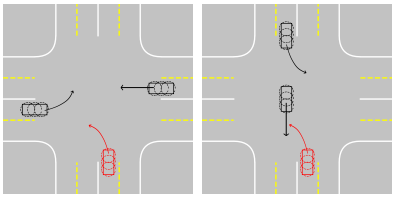
Figure 4.1: Simulation environment for non-signalized intersection

The longitudinal and lateral control for OV tracking is done using MPC and Stanley control, respectively, as described in [16]. Depending on the current state of the OV, GPR is used to predict its position for future time steps, represented by the dotted black lines in figure 4.1.

No	Description	Purpose	Cases
1	Trajectory generation in a one-lane intersection with no obstacles	Demonstrate the baseline performance of the algorithms	
2	Trajectory generation in Two and three-lane intersections without obstacles	Demonstrate the scalability of the algorithms	

Continued on next page

Table 4.1 – continued from previous page

No	Description	Purpose	Cases
3	Trajectory generation in an intersection with a static obstacle in the middle	Study the alternative path generation capability around the obstacles	
4	A one-lane intersection with OVs obstructing the ego vehicle's path at different times from different directions	Study the baseline performance for handling dynamic obstacles. Also, to conduct sensitivity study for various vehicle speeds and decelerations	
5	A two-lane intersection with two OVs entering the intersection from different directions at different times	Demonstrate the algorithms' ability to handle multiple dynamic obstacles and find safe trajectories between the OVs	

Continued on next page

Table 4.1 – continued from previous page

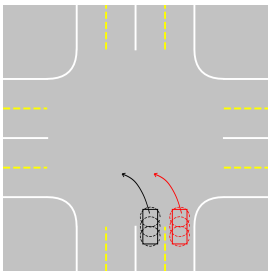
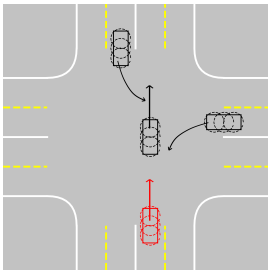
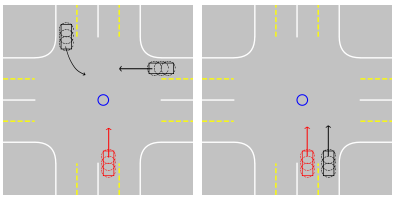
No	Description	Purpose	Cases
6	A two-lane intersection with an OV moving parallel to the ego vehicle	Demonstrate the ability of the ego vehicle to move safely and parallel to the OV	
7	A two-lane intersection with three OVs entering the intersection at different times and from different directions	To compare the performance of the algorithms to human intuition when handling multiple OVs	
8	A two-lane intersection with multiple OVs and a static obstacle in the middle of the intersection	To study algorithms' ability to handle static and dynamic obstacles simultaneously and complete safe maneuvers	

Table 4.1: Summary of the scenarios tested in simulation

4.1.2 Simulation Scenarios

A summary of all the scenarios used for simulations is provided in table 4.1

No obstacle vehicle at the intersection

To evaluate the path-finding capability of each algorithm, they were tested on an empty intersection without any OVs. Such scenarios indicate the baseline performance of the algorithms. Moreover, comparing the obtained trajectories with human intuition is easier when finding a path in an empty intersection.

The objective was to find a feasible path for left, right, and straight maneuvers from the initial position. To evaluate the scalability of the algorithms, they were tested in one-lane, two-lane, and three-lane intersections. A total of three scenarios for a one-lane intersection, six scenarios for a two-lane intersection, and five scenarios for a three-lane intersection were tested. To test the algorithms' ability to handle static objects hindering the usual motion of vehicles, a static obstacle was introduced in the middle of a 2-lane intersection. Left turn and straight maneuver were considered for this environment. It is necessary to study the sensitivity of various parameters of the pRRT algorithm and how they affect the solution trajectories. [16] showed the trajectories generated by using high bias and low variance values, which make the algorithm too greedy towards the goal and lose the quality of randomness of RRTs. Hence, this analysis was done to find values for these parameters that balance randomness and quick planning. This study used the outside left turn maneuver in the two-lane intersection. The parameter bias value (λ) was varied from 10^{-1} to 10^{8}. The bias variance (σ^2) was varied from 10^{-6} to 1.

With traffic flow

The most important test for a motion planning algorithm is a safe maneuver in traffic situations with other vehicles. Simulations were carried out for different scenarios involving the planning of ego vehicle amongst the movement of obstacle vehicles. The simulation

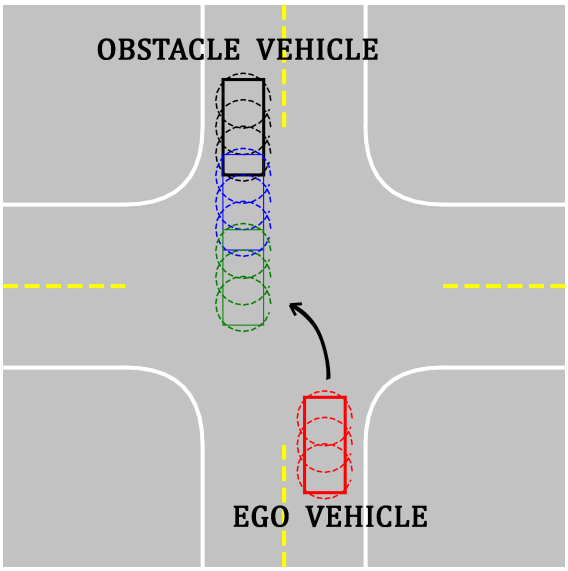


Figure 4.2: Left turn maneuver for ego vehicle

Algorithm 4 End to end planning steps for intersection

$Ego \leftarrow InitializeEgoVehicle(x_0)$
while goal not reached **do**
  for OV i **do**
    $P^i \leftarrow$ Current OV State
    $P^i_{predict}, \sigma^i_{predict} \leftarrow GPRpredict(P^i)$
  if No $Path$ found **then**
    $Path \leftarrow Planning(x_0, x_{goal}, P^{1..N_v}, P^{1..N_v}_{predict})$
  if $Collision(Path, P^{1..N_v}, P^{1..N_v}_{predict}) == True$ **then**
    $Path \leftarrow NULL$
    $Ego \leftarrow decelerate(Ego)$
  else
    $Ego \leftarrow ExecutePath(Ego, dt)$

considered the ego vehicle to be at rest at its starting position. Using the information about the states of other vehicles, their future states after 0.75s and 1.5s are predicted. Using this information, the planning algorithm generates a trajectory toward the goal position. Once the trajectory is generated, the path is checked for collision with OVs at each timestep of $dt = 0.1s$. The collision with predicted states is checked for the trajectory after the prediction period. This process is summarized in algorithm 4.

One obstacle vehicle A single obstacle vehicle was used to test all the algorithms initially. The scenario where the ego vehicle takes a left turn, and the obstacle vehicle makes a straight line maneuver from top entry to exit is shown in Figure 4.2. This scenario was used to study different obstacle vehicle speeds and ego vehicle maximum speeds. The maximum speeds used for ego and obstacle vehicles were 10 and 20 miles per hour. Different vehicles have different decelerating capabilities, so $15ft/s^2$ and $30ft/s^2$ were tested. The bias value for the pRRT algorithm was selected to be $\lambda = 10^3$ and the standard deviation as $\sigma = 0.05$. Maximum allowable iterations for the algorithms were limited to 250, translating to a CPU time of 0.5s for RRT and pRRT algorithms. If the planner cannot converge to a solution, it clears the formed tree and restarts the tree formation from the initial position.

Multiple obstacle vehicles Figure 4.1 shows an example of an intersection with two obstacle vehicles. Scenarios with multiple lanes were also tested. Non-signalized intersections are prone to critical situations where other vehicles might suddenly appear in intersections, causing a collision threat. Hence, scenarios with multiple vehicles with a higher risk of collision have been tested and presented in the results section. The intersection with a static obstacle was also used for simulating traffic scenarios to evaluate if the algorithms can handle static and dynamic obstacles effectively.

4.2 Results and Analysis

4.2.1 Static Environment

The three algorithms were run 100 times each in the empty intersection scenarios described in the table 4.1. As these algorithms are probabilistic complete, we used a threshold of 2000

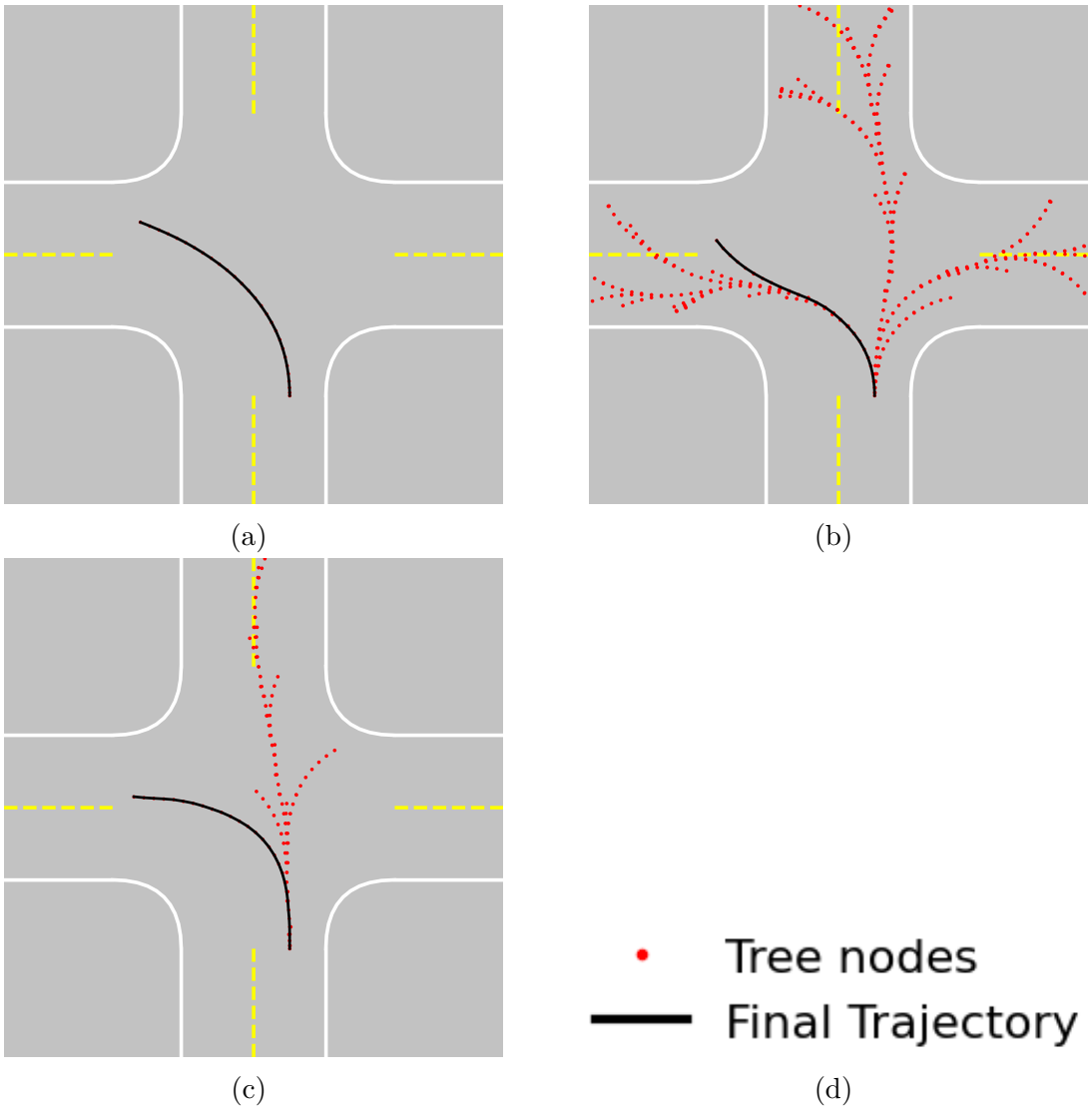
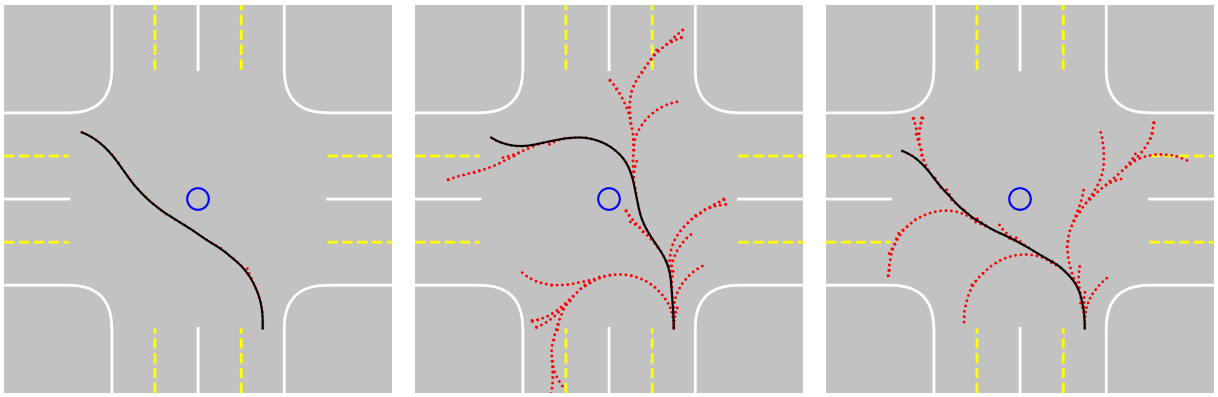


Figure 4.3: Left turn on a one-lane intersection: a) pRRT, b) RRT, c) RRT*, d) Legend

iterations for the algorithms. The algorithms were evaluated using the following metrics: success rate out of 100 runs for each scenario, the average number of CPU-based floating point operations (FLOPs), average CPU time, and the average number of iterations. The optimality of the trajectories was evaluated using the path length which was considered as the cost function for RRT*. The sample point spacing (Δ) was kept at 0.05m for all the scenarios. The pRRT algorithm used $\lambda = 10^3$ and $\sigma = 0.01$ as its parameters.

Figure 4.3 shows the trajectories generated by the three algorithms for a left turn scenario on a single-lane intersection. Due to a relatively high bias value and there being no obstacles in the intersection, pRRT generates a very intuitive trajectory with few iterations. There is no need to find alternative paths and hence the tree formed by it is fairly simplistic and minimal. Because there is no bias in the standard RRT algorithm, it also explores unnecessary regions and returns a huge tree with unimportant branches.

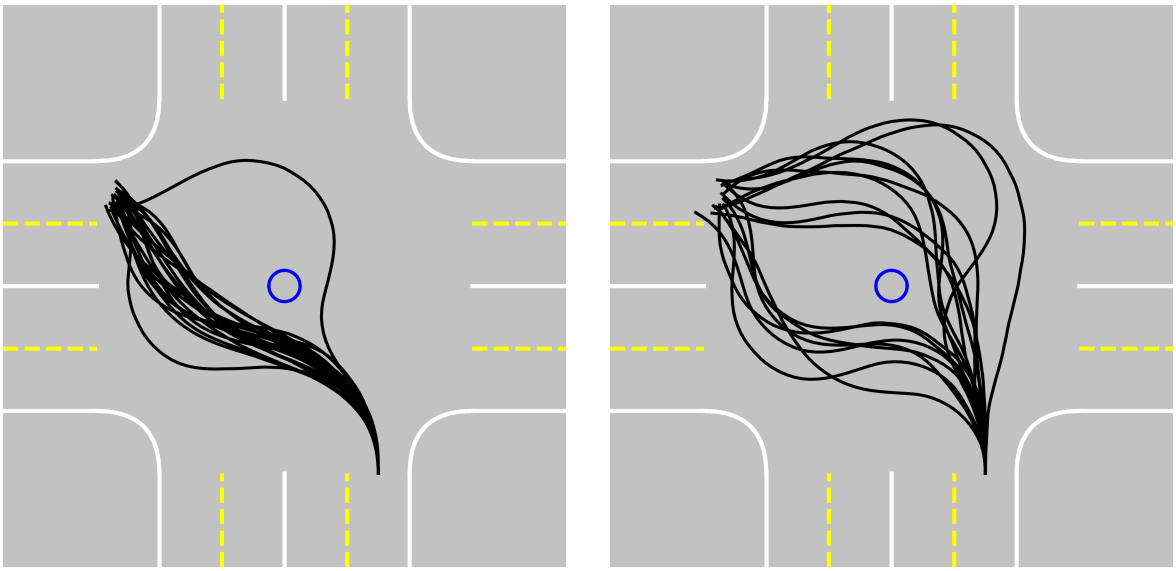


(a) (b) (c)

Figure 4.4: Trajectory generation around static objects: a) pRRT, b) RRT, c) RRT*

When a static object, like a pothole or a traffic cone, blocks the usual driving path, the randomness of RRT and pRRT helps find an alternative path around the obstacle, as shown in figure 4.4. Compared to RRT, pRRT gives more optimal trajectories. For a left turn around a static object, the trajectories obtained by pRRT and RRT for 30 trials are shown in figure 4.5. The uniform sampling results in longer and more distorted trajectories for RRT, while the pRRT trajectories are fairly consistent and predictable.

The RRT* algorithm works similarly to RRT but tries to find an optimal solution. Due to the threshold to maximum allowable iterations, RRT* could not reach a globally optimal solution. In figure 4.6, we can see that the overall path length obtained by RRT* is smaller than the one by RRT. The RRT* algorithm tries to minimize the overall path length for a



(a) (b)

Figure 4.5: Comparing the successful trajectories generated by pRRT and RRT around a static obstacle over 30 trials: a) pRRT, b) RRT

Table 4.2: Average path length for trajectories (in simulation units)

	1-Lane	**2-Lane**	**3-Lane**	**Static Object**
RRT	1.280	2.076	2.513	2.989
RRT*	**1.268**	1.908	2.503	**2.60**
pRRT	1.301	**1.838**	**2.379**	2.750

Table 4.3: Average number of FLOPS for successful trajectories (x10^6)

	1-Lane	**2-Lane**	**3-Lane**	**Static Object**
RRT	20.73	24.18	50.10	39.47
RRT*	279.0	210.3	266.1	220.6
pRRT	**0.940**	**1.508**	**2.332**	**18.67**

sampled point, while RRT connects it directly to the nearest node. Due to this property, RRT often generates longer paths as shown in figure 4.6c. Table 4.2 summarizes the average path lengths obtained by each algorithm. Because the algorithms solve an optimal steering problem towards the sampled point, pRRT tends to give an optimal solution for higher bias values. Hence, we can see similar performance between pRRT and RRT* for optimality.

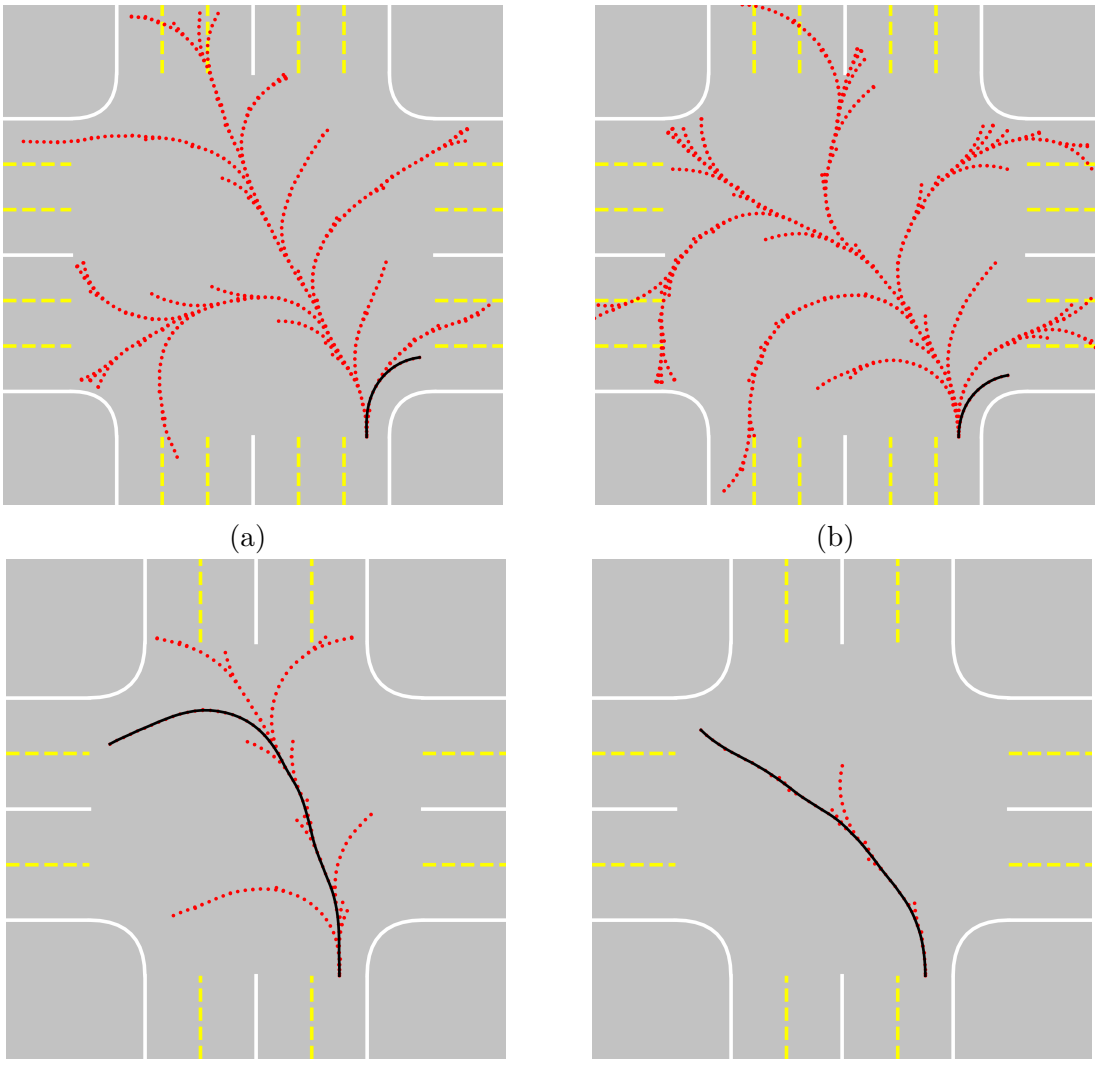
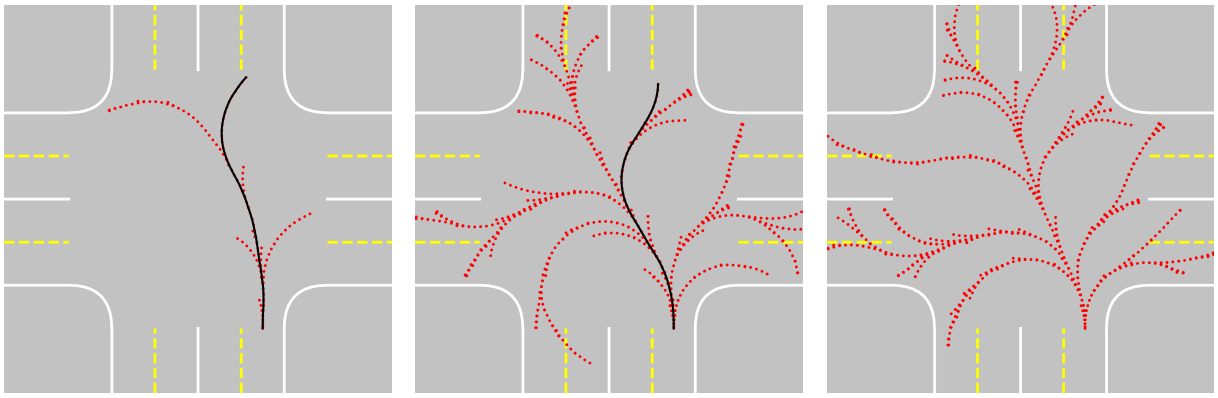


Figure 4.6: Outside left turn on two-lane and three-lane intersections: a) RRT, b) RRT*, c) RRT, d) RRT*

Table 4.4: Average number of iterations for successful trajectories

	1-Lane	**2-Lane**	**3-Lane**	**Static Object**
RRT	679.7	665.3	847.2	938.0
RRT*	671.3	691.7	841.8	866.3
pRRT	**34.00**	**35.83**	**39.83**	**352.1**

Table 4.3 summarizes the average number of floating point operations (FLOPs) required by each of the three algorithms. These FLOPs are counted between tree initialization at the



(a) (b) (c)

Figure 4.7: Trees formed by RRT for the same scenario for different runs

Table 4.5: Average CPU time for successful trajectories (in seconds)

	1-Lane	**2-Lane**	**3-Lane**	**Static Object**
RRT	1.476	1.473	1.958	2.816
RRT*	54.31	34.55	40.71	38.58
pRRT	**0.067**	**0.073**	**0.093**	**1.268**

start position and feasible trajectory generation. The number of operations performed by the pRRT algorithm is lower than RRT and RRT* as it reduces the sampling of unnecessary points. The number of operations is higher for a similar maneuver in larger environments for the pRRT algorithm. This was not found to be true for RRT and RRT*. Because of the inherent randomness and uniform sampling, the algorithm might converge quickly to a solution or not converge in a finite time, hence unpredictable. This can be seen in figure 4.7. The average number of iterations for RRT and RRT* are similar as seen in table 4.4. The computational load for each iteration of the algorithms can be assessed through the number of FLOPs required for each algorithm per iteration. Using tables 4.3 and 4.4, FLOPs per iteration for RRT were found to be 42962, whereas, for RRT*, they were 317801 as it performs additional steps to optimize the total cost. FLOPs per iteration for pRRT were found to be 50783, which is comparable to that of RRT. This indicates that the additional step to form a PPM and sample points through the nonlinear distribution is not computationally

expensive. Table 4.5 summarizes the average CPU time required for getting a successful trajectory. The average time for pRRT is about 0.09s for empty intersections and about 1 second for intersections with static objects in the center. Hence, it can be used for online planning.

Sensitivity analysis for pRRT

As explained previously, the pRRT algorithm creates a position probability map (PPM) using the Gaussian distributions of goal and obstacle positions. The major factors affecting the PPM are the bias factor (λ) and the variance (σ^2). The bias factor deviates the algorithm from RRT to pRRT. As this value lowers, the algorithm becomes closer to the standard RRT. This can be seen in figure 4.8.

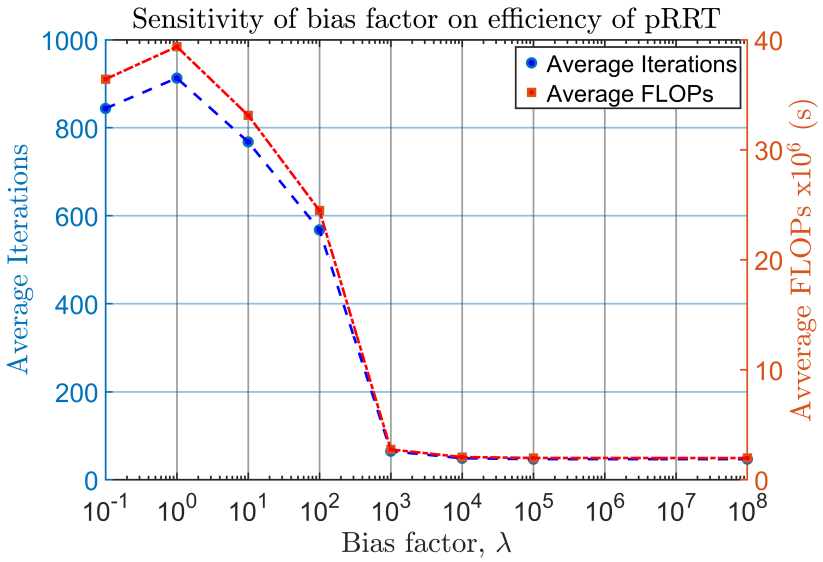
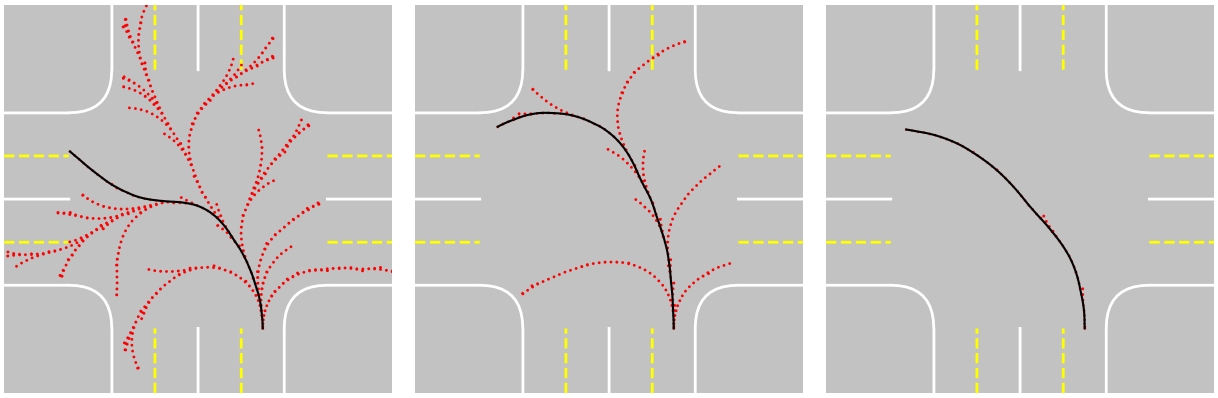


Figure 4.8: Efficiency vs Bias

As the value of this bias factor increases, the number of iterations and hence the number of floating point operations decrease significantly. For a low bias value of 0.1 - 10, the algorithm performs similarly to the standard RRT algorithm and takes a similar time to converge to a

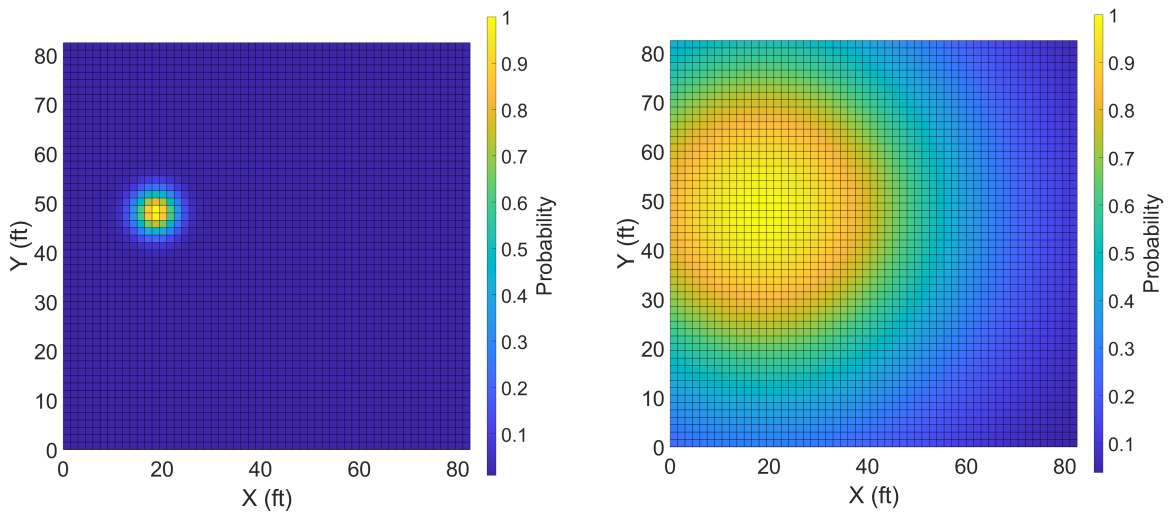


(a) (b) (c)

Figure 4.9: Effect of bias value λ on explored space: a) $\lambda = 10$, b) $\lambda = 100$, c) $\lambda = 1000$

solution. There is a big jump from RRT behavior to pRRT behavior as the λ value changes from 10^1 to 10^3. This indicates that point sampling becomes more concentrated at the goal location at $\lambda = 10^3$. The probability densities for every point apart from the goal location become near zero as the bias value increases from 10^4. Hence the algorithm performs quite similarly, and there is not much effect of increasing the value. Even for lower bias values, the branches of the generated tree grow towards the goal due to the goal bias, as seen in figure 4.9. Due to this, unnecessarily longer paths are not generated. As the variance increases, the sampling space around the goal location widens, as shown in figure 4.10. The PPMs shown in the figure were for a single-lane intersection with a goal bias towards the coordinates (18, 47).

For higher values of sigma, the sampling space becomes more distributed, resulting in a higher number of FLOPs and iterations as seen in figure 4.11. Even though the sampling space is distributed, the goal bias makes the algorithm converge to a solution faster. The number of FLOPs for $\sigma = 1$ is much lower than that of RRT, indicating better performance than RRT even for higher variance values. Lower variances lead the sampling space to be more concentrated towards the goal.



(a) (b)

Figure 4.10: Effect of standard deviation σ on sampling space: a) $\sigma = 0.1$, b) $\sigma = 1$

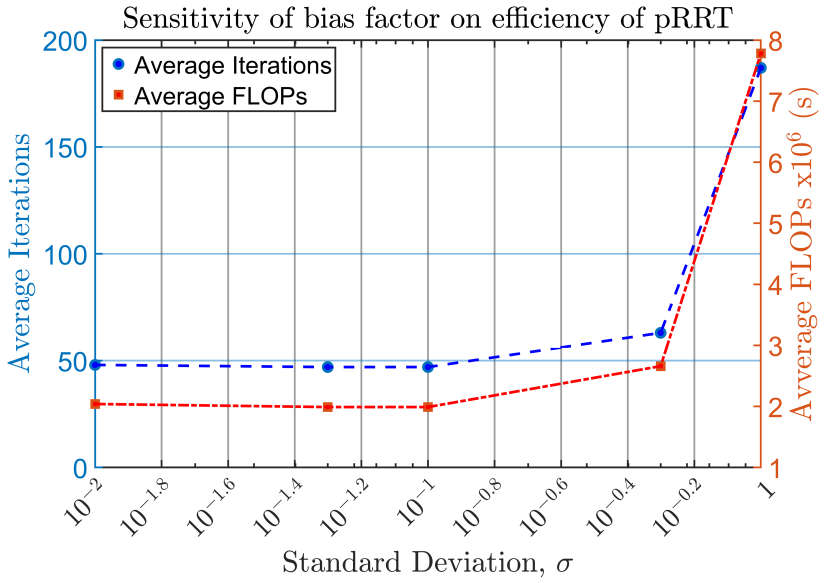


Figure 4.11: Efficiency vs Sigma

The overall success rate for pRRT was nearly 100% for an empty intersection. Even for scenarios with a static object, the pRRT algorithm outperformed RRT and RRT* with $\lambda = 10^3$. The success rates over 100 runs for each scenario for the algorithms and varying bias values for pRRT are summarized in table 4.6.

Table 4.6: Success rate for intersection with no OVs with maximum iterations limited to 2000

	Empty intersections	**Static Object**
RRT	77%	40%
RRT*	72%	63%
pRRT ($\lambda = 1$)	67%	48%
pRRT ($\lambda = 10$)	74%	54%
pRRT ($\lambda = 100$)	81%	65%
pRRT ($\lambda = 1000$)	**99%**	**84%**

4.2.2 Dynamic environment

One obstacle vehicle

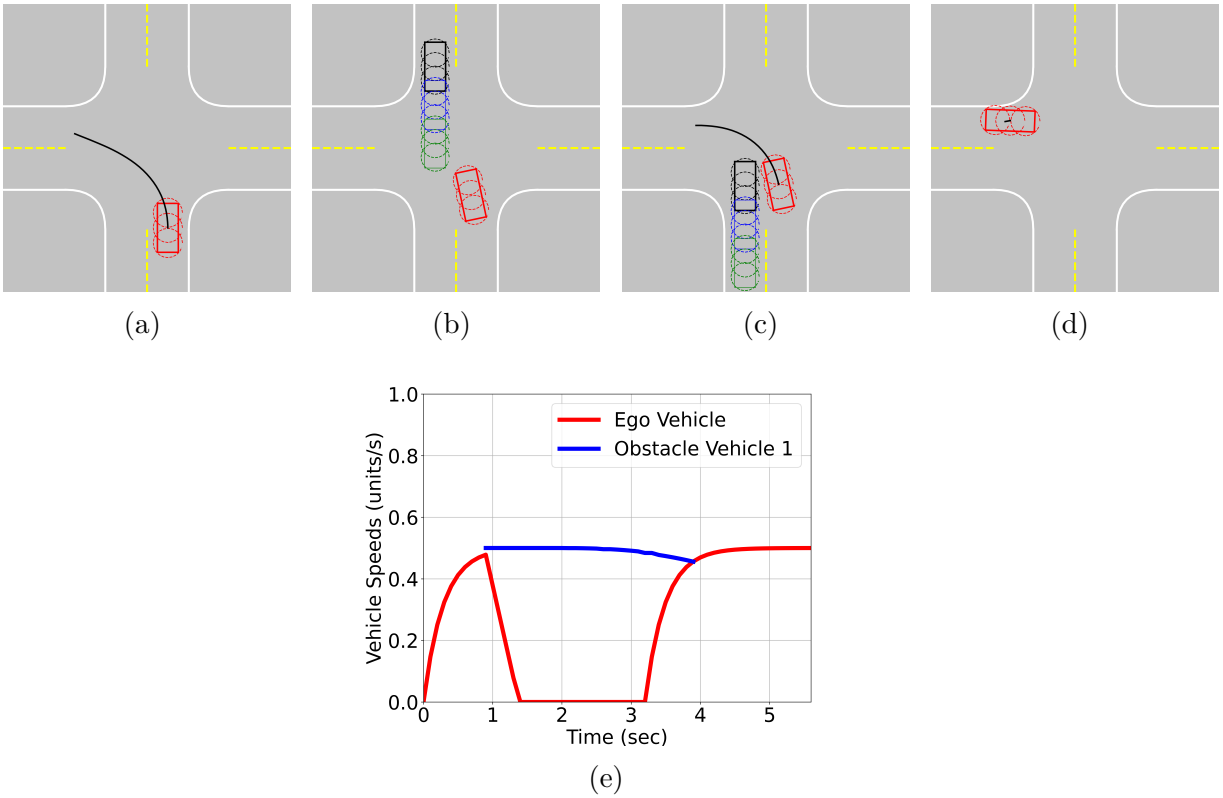
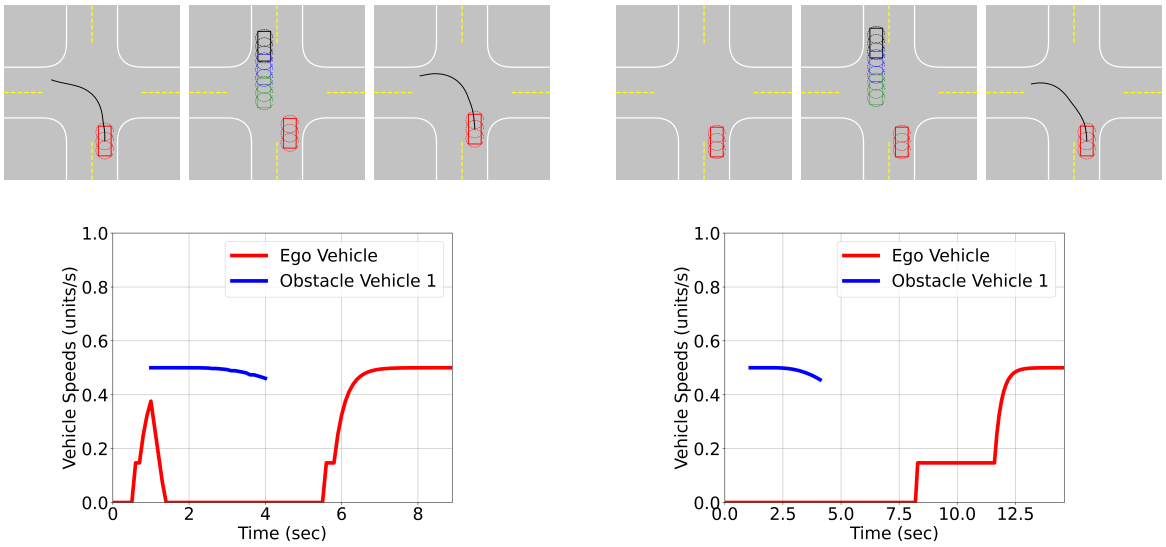


Figure 4.12: Left turn on a single lane intersection using pRRT

Figure 4.12 shows the ego vehicle turning left in a single-lane intersection. The ego vehicle

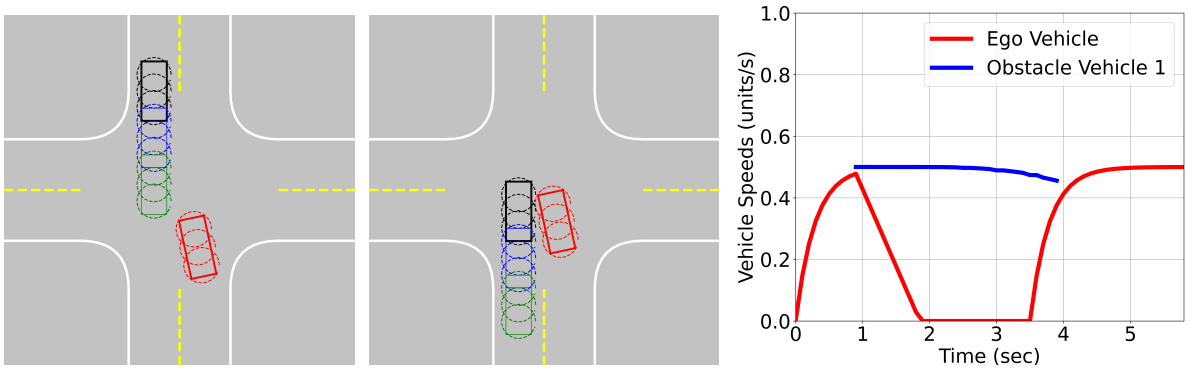
starts moving as an initial solution is obtained from the start position. The blue vehicle shows the predicted position of the obstacle vehicle after 0.75 seconds and the green vehicle shows the prediction after 1.5 seconds. After 1 second of the ego vehicle entering the intersection, an obstacle vehicle suddenly enters the intersection from the top entry as shown in figure 4.12b. The collision check predicts a collision and hence breaks the tree. In order to avoid the collision, the pRRT algorithm tries to quickly re-plan but cannot find any feasible trajectory, and hence it slows down and comes to a halt. Once the OV passes, as shown in figure 4.12c, the OV re-plans the path and completes the move. The speed plot shows the ego and obstacle vehicle speeds during the maneuvers. The same scenario with RRT and RRT* is shown in



(a) (b)

Figure 4.13: Left turn on a single lane intersection using a) RRT and b) RRT*

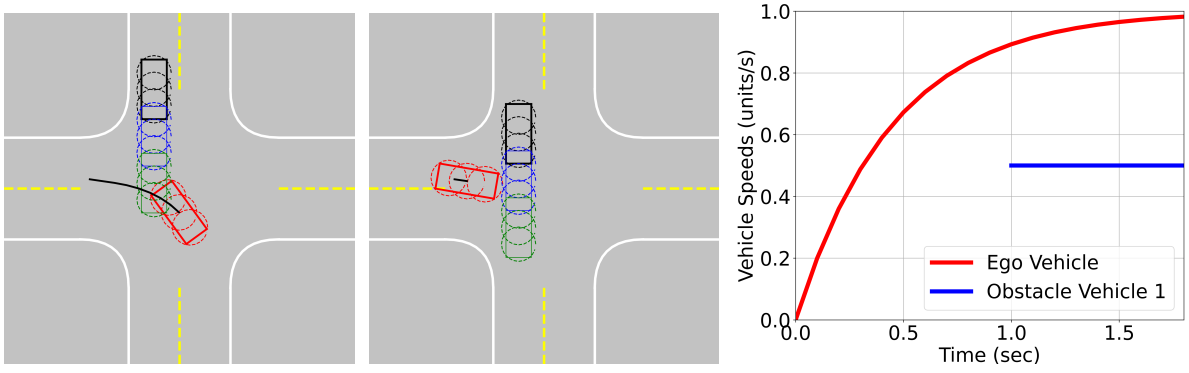
figure 4.13. The RRT algorithm requires more time to converge to a solution; hence, it stops and waits until a new plan is obtained. Hence, the RRT algorithm takes 3 seconds longer to pass through the intersection for the same maneuver. The speed plot for RRT shows slight discontinuities in the ego vehicle speeds. This occurs because the randomly generated point might be very close to its nearest node and hence does not require an increase in the velocity



(a) (b) (c)

Figure 4.14: Left turn on a single lane intersection using pRRT with the deceleration of $15ft/s^2$

resulting in the velocity control input being 0. Also, the resulting trajectory is not optimized. The path generated by pRRT seems to be more intuitive to humans. The path generated by RRT* is the shortest, but it takes the longest time to generate a feasible trajectory. The period of constant velocity in the speed plot for RRT* is due to the defined cost function of minimizing path length. The cost function could be defined as minimizing fuel used as well. The previous scenario uses the maximum deceleration of $30ft/s^2$ which causes rapid



(a) (b) (c)

Figure 4.15: Left turn on a single lane intersection using pRRT with the maximum ego vehicle speed of 30 mph

deceleration of the ego vehicle. When the deceleration was reduced to $15ft/s^2$, the vehicle

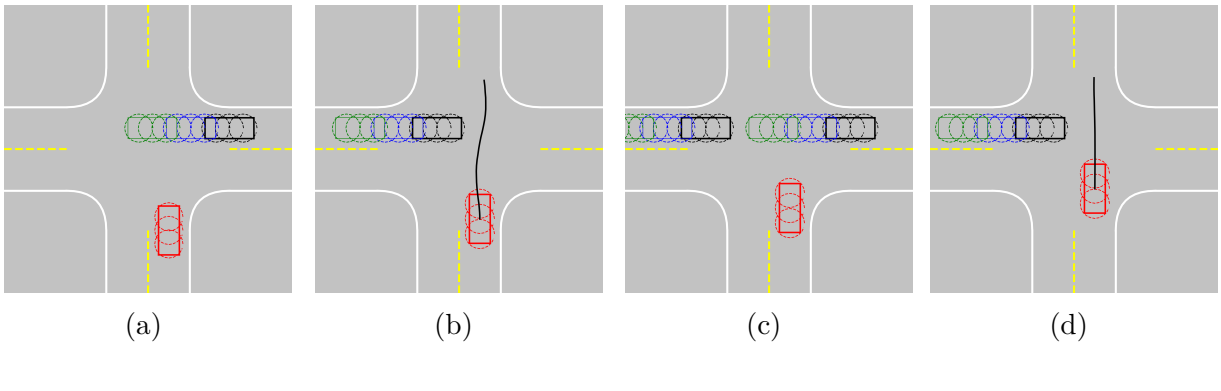
comes very close to the obstacle vehicle but was still able to stop. This can be seen in figure 4.14. As 1 unit in simulation represents 30ft, the maximum speeds of 0.5 unit/s imply 15 ft/s or 10 mph. The pRRT algorithm responds well with higher vehicle speeds as well. When the maximum ego vehicle speed was increased to 20mph, the ego vehicle passed through the intersection before the obstacle vehicle hindered the path. This can be seen in figure 4.15. Even though the predicted states of the OV hinder the planned path, the algorithm checks for collision with predicted states after 0.75 and 1.5 seconds of the planned trajectory. This indicates that pRRT finds a different solution depending on the ego vehicle speeds, obstacle vehicle speeds, and the environment structure to complete the maneuver quickly.

Multiple obstacle vehicles:

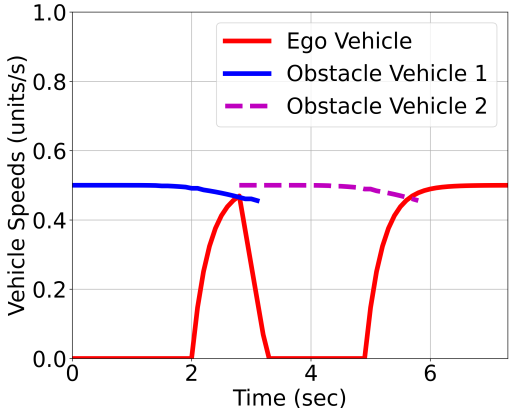
One of the major advantages of using RRT is its ability to obtain alternate paths by exploring the environment randomly. Using a lower bias value in pRRT leads to better environment explorations to give alternative paths while being faster than RRT. Figure 4.16 shows the scenario when a second obstacle vehicle enters the intersection while the ego vehicle moves.

Figure 4.17b shows the distorted path obtained by the algorithm after the first obstacle vehicle goes out of the ego vehicle's path. The convergence time for RRT* is higher than RRT, increasing its idle time in the intersection.

Hence, it is unsuitable for online planning through intersections, as seen in figure 4.13b. pRRT and RRT were deployed at various scenarios with multiple obstacle vehicles in a larger intersection for studying comparative performance. It is possible for the ego vehicle not to need to stop and let the obstacle vehicle pass. As shown in figure 4.17b, the pRRT algorithm quickly finds a plan and completes the left turn before the second obstacle vehicle threatens a collision. Figure 4.17b shows the same scenario using RRT. The algorithm takes



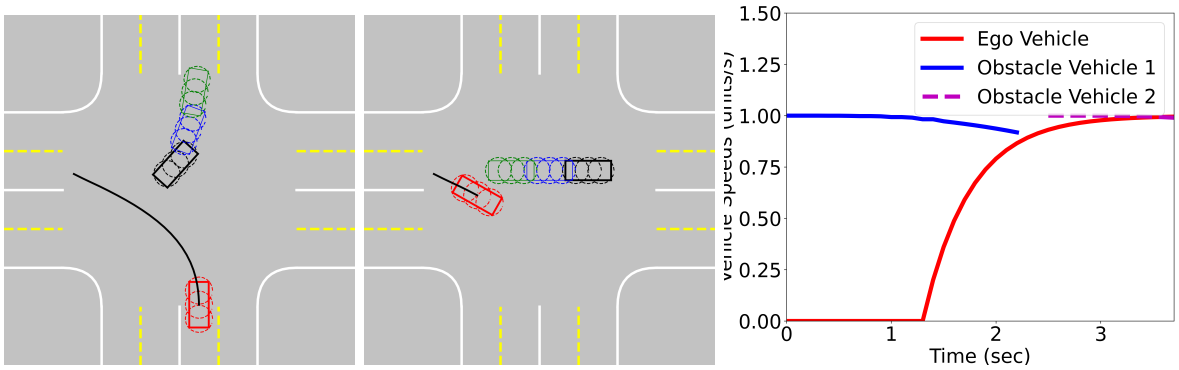
(a) (b) (c) (d)



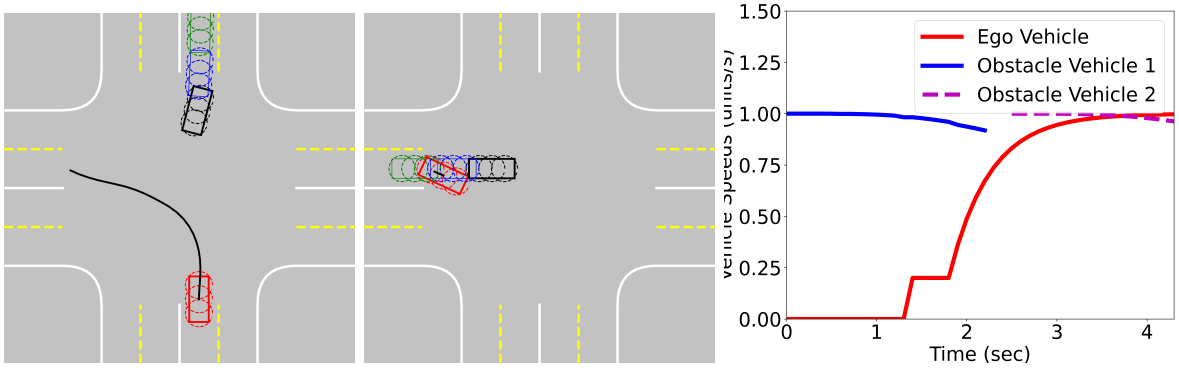
(e)

Figure 4.16: Ego vehicle passing straight through the intersection with two obstacle vehicles using pRRT. The trajectory obtained after the first OV passed through the intersection is obstructed by another OV entering the intersection.

longer to find a solution after the first obstacle vehicle moves out of the way. This newly generated path does not detect a collision with the other obstacle vehicle as its states are predicted to be up to 1.5s. Hence the ego vehicle continues the motion, which leads to a collision. One of the main challenges with pRRT is to find optimal values for the bias and variance parameters that provide a good balance between a faster convergence and the ability to find alternative paths. Figure 4.18 shows a scenario when two obstacle vehicles are in the intersection simultaneously. The pRRT algorithm with $\lambda = 10^3$ is able to find good trajectories around the OVs, as seen in figure 4.18a. The effect of slower convergence of RRT can be witnessed in figure 4.18b.



(a)

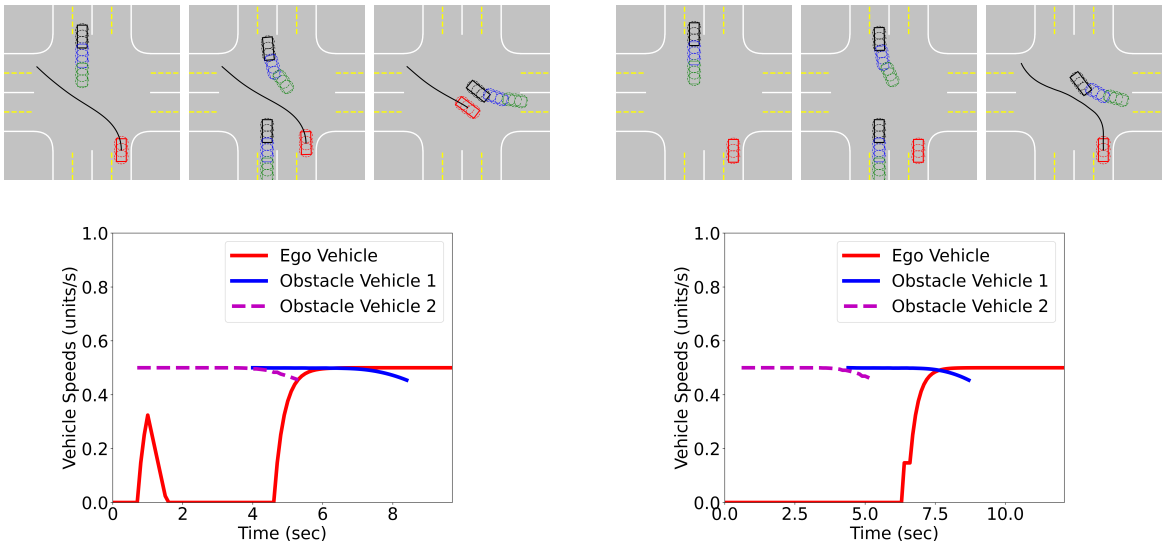


(b)

Figure 4.17: Ego vehicle making a left turn on a 2-lane intersection at 30 mph: a) pRRT: Safely passes through b) RRT: Causes a collision

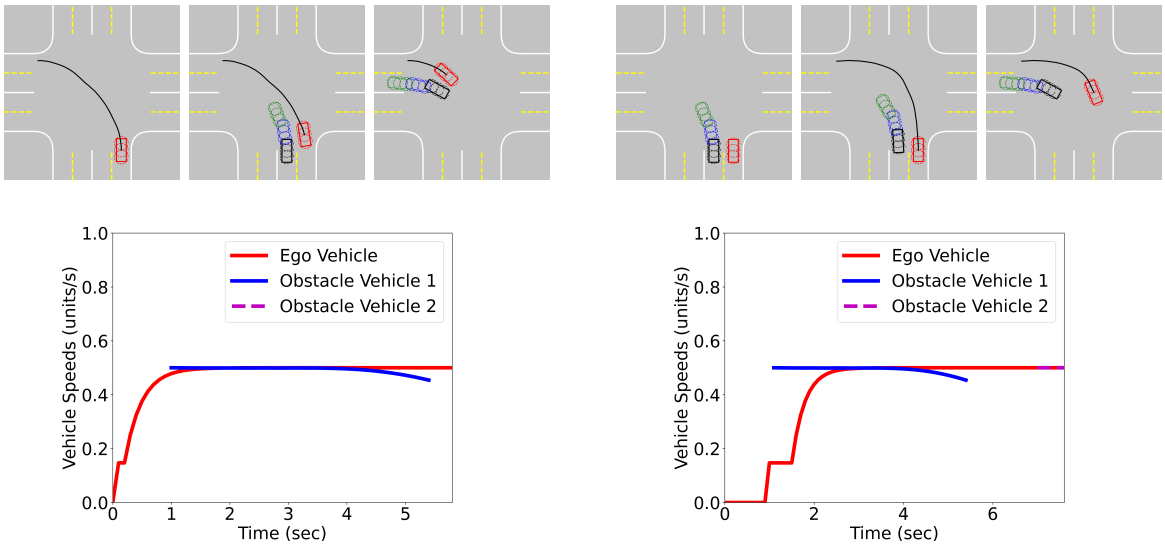
RRT tends to generate conservative trajectories around an obstacle. Figure 4.19 shows a scenario when the motion of the OV does not affect the motion of the ego vehicle. RRT forms a longer trajectory while being slower to generate the trajectory than pRRT.

Figure 4.20 shows a scenario when three OVs enter the intersection from different directions in short intervals. As pRRT replans a trajectory faster, the ego vehicle completes the maneuver before the third OV obstructs the ego vehicle's path. Using the RRT algorithm, the ego vehicle remains stationary in the middle of the intersection and moves after the third OV frees the intersection. The decisions made by the pRRT algorithm conform more to human intuition than the ones made by RRT. Due to its complete randomness and slow convergence,



(a) (b)

Figure 4.18: Finding a safe path through multiple obstacles vehicles: a) pRRT using $\lambda = 10^3$ and $\sigma = 0.05$, b) RRT with uniform sampling

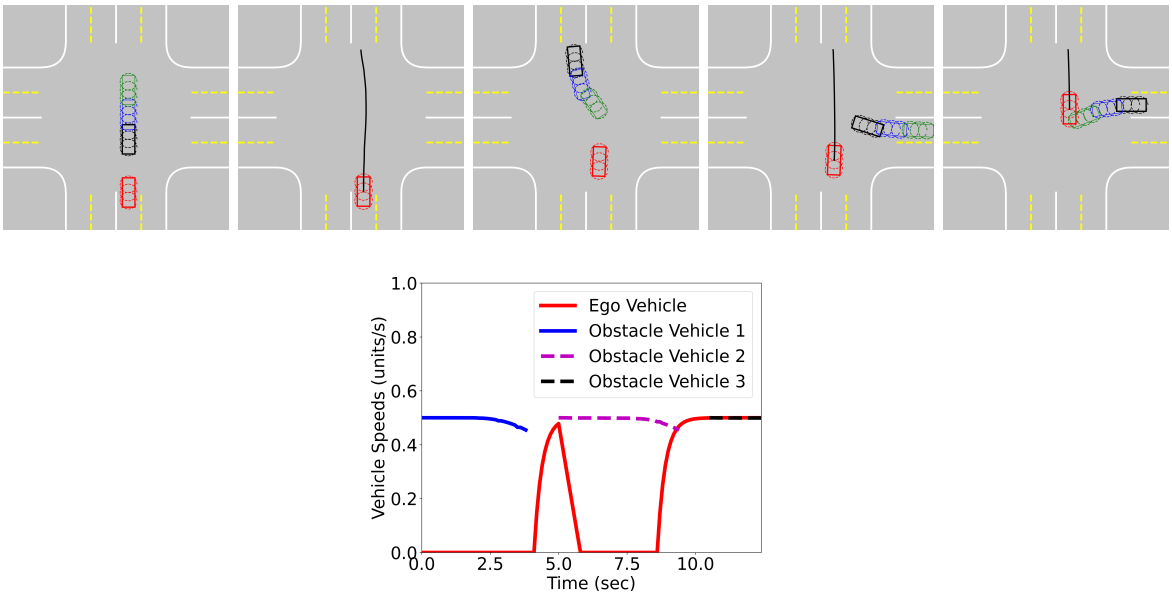


(a) (b)

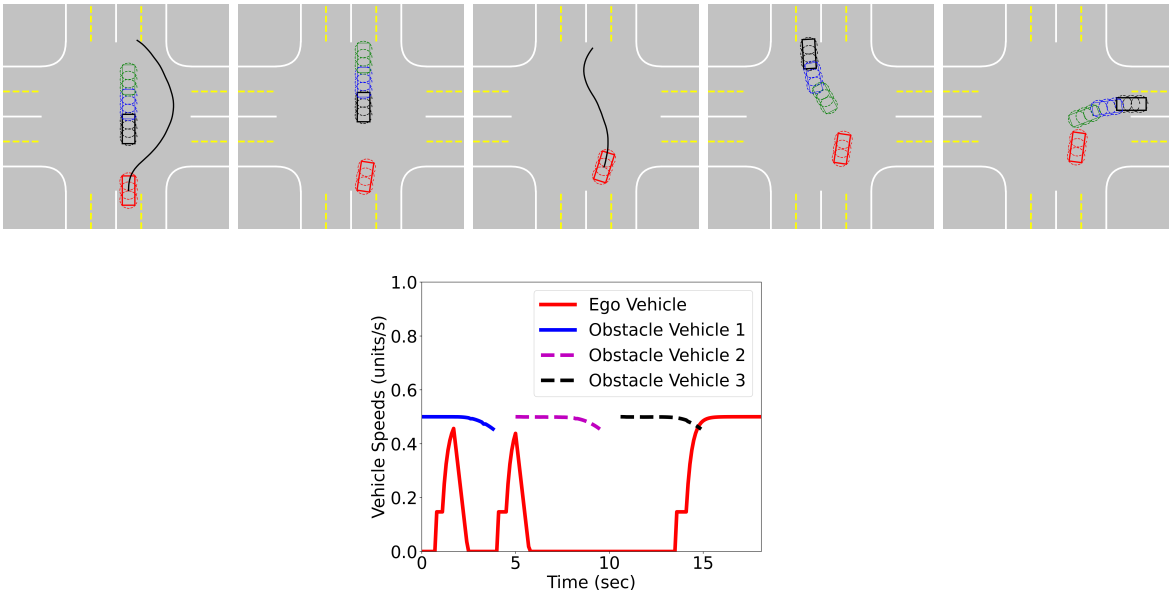
Figure 4.19: Ego vehicle making a left turn parallel to an OV: a) pRRT using $\lambda = 10^3$ and $\sigma = 0.05$, b) RRT with uniform sampling

the trajectories generated by RRT are unconventional.

Figures 4.21 and 4.22 show the scenarios when there is a static obstacle in the middle of



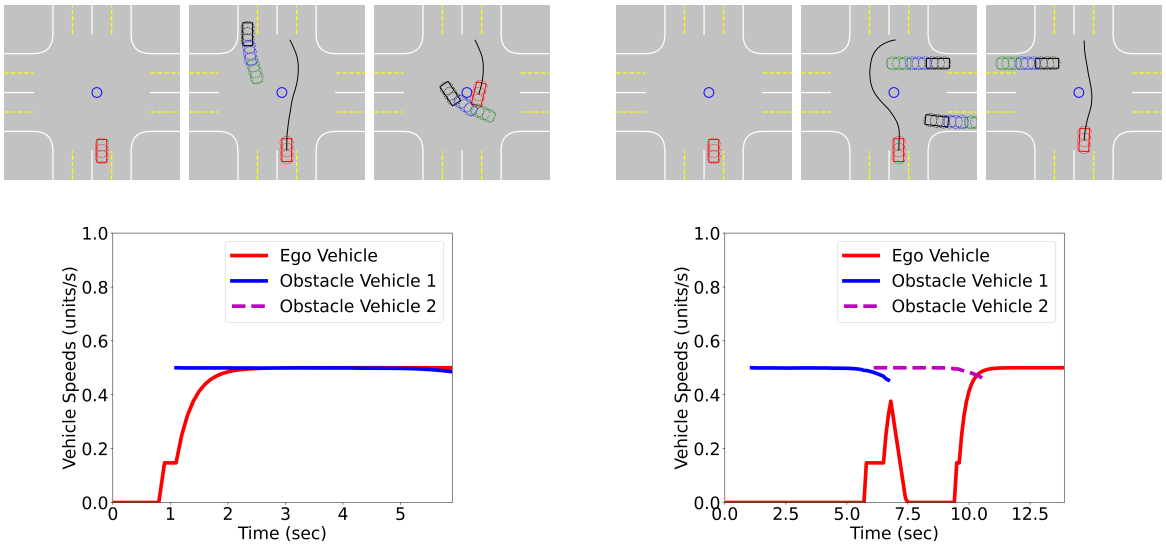
(a)



(b)

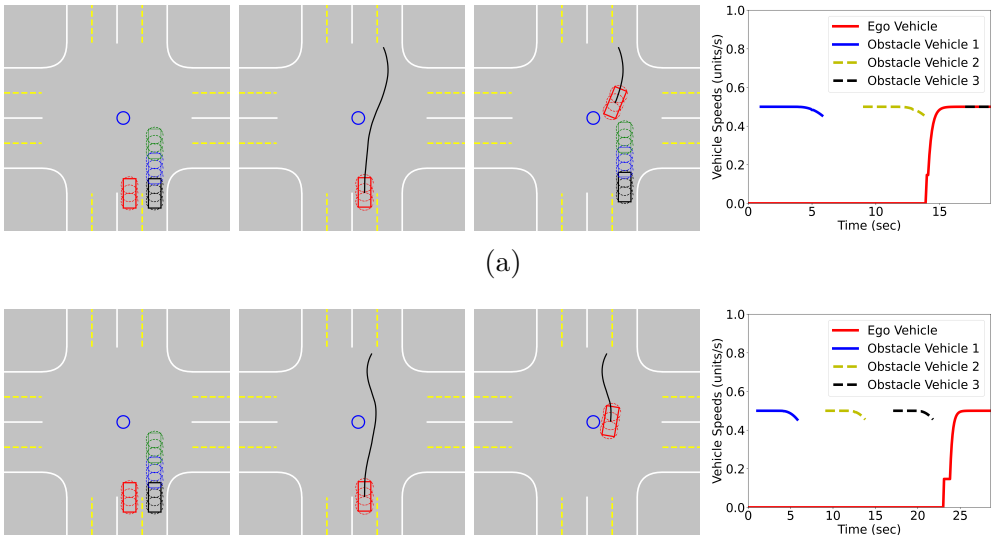
Figure 4.20: Scenario with three obstacle vehicles: a) pRRT: Safely passes through with the quick planning, b) RRT: Due to slow planning, the ego vehicle stays in the middle of the intersection

the intersection along with moving traffic. The pRRT algorithm finds a path quickly after entering the intersection using pRRT around the static object. In both these cases, RRT



(a) (b)

Figure 4.21: Ability for ego vehicle to move around static obstacles: a) pRRT using $\lambda = 10^3$ and $\sigma = 0.05$, b) RRT with uniform sampling



(a)

(b)

Figure 4.22: Ability for ego vehicle to move around static obstacles with parallel moving traffic: a) pRRT using $\lambda = 10^3$ and $\sigma = 0.05$, b) RRT with uniform sampling

finds a solution much slower and completes the maneuver about twice the time that pRRT requires.

RRT and pRRT can be used to find a collision-free path for online motion planning in intersections. But the pRRT algorithm is much quicker and reduces the idle time of the ego vehicle significantly when compared to RRT. RRT* is much slower than RRT and pRRT, so using it for online planning is not feasible. Due to its probabilistic completeness, it can be used for offline route planning.

Chapter 5

Experimental evaluation of pRRT

To validate the feasibility of the probabilistic RRT in real life, the algorithm was deployed on scaled autonomous cars for experimental studies. Section 5.1 described the scaled autonomous car platform used for experimentation.

5.1 Test Platform

The Autonomous Systems and Intelligent Machines (ASIM) lab has built a 1/10th proportionally scaled model of an autonomous vehicle for testing various algorithms called the Xtenth car. The Xtenth car is equipped with various sensors to emulate a full-sized self-driving car. The model uses a Traxxas Slash 4X4 VXL all-terrain Remote Controlled (RC) car as the base vehicle. The vehicle is driven by one brushless DC motor resulting in a top speed of 60 mph. The steering system comprises an Ackermann steering mechanism driven by one servo motor. These actuators are controlled by an open source Electronic Control Unit (ESC) called the VESC.

5.1.1 Sensing and Processing

The model is equipped with an Nvidia Jetson Orin AGX board for computations and processing. The board is compact and powerful, making it suitable for robotics applications.

Figure 5.1: Picture of the experimental test platform

The board is made of a 12-core Arm Cortex-A78AE v8.2 CPU, 2048-core NVIDIA Ampere architecture GPU with 64 tensor Cores, 32 GB of RAM, and 1TB of external NVMe SSD storage. This makes the computer that drives the model car extremely powerful for implementing computationally heavy planning and control algorithms.

The sensor suite on the vehicle platform comprises a camera and a lidar. The Zed2 stereo camera made by Stereolabs was used for visual perception. This stereo camera has a field of view of 120° and a depth estimation range of 20m. The camera also has built an inertial measurement unit (IMU), a barometer, and a magnetometer which can be used for SLAM purposes.

For object detection, the G2 lidar made by YDLIDAR was used. It is a 2D lidar with a 360° field of view and a maximum range of 12m. The lidar can scan 505 points in its FoV with a frequency ranging between 5 and 12 Hz. The lidar has a very low power and computation demand, making it very suitable for scaled experimental platforms.

5.1.2 Software implementation

Robot Operating System (ROS) is implemented on the scaled car that provides a set of tools and frameworks for easy integration of software and hardware. The noetic version of ROS, which was released in 2020, is implemented on the vehicle.

To implement the pRRT algorithm on the vehicle, the program was rewritten to create a ROS-compatible script. The VESC was programmed such that the actuators require the velocity and steering angle commands as input. The pRRT algorithm results in a trajectory with desired control commands at each time step. Hence the trajectory generated by the Python implementation shown in 4.1 can be used for experimental evaluation.

The algorithm requires the obstacle positions to check for collisions while planning a path. It does not require vision information and the type of object, and hence the camera was not used for the experiments. The information about the obstacle positions was obtained using the lidar. Due to limited space in the lab environment, the ego vehicle speed was limited to 1 m/s ($\sim$ 2 mph). The VESC requires a minimum of 0.5 m/s as the velocity input for the motor to actuate. The servo motor limits the steering angle to have a maximum angle of 20.5° on either side. Hence for the optimal control problem to decide the control inputs at each time step, the limits to the control commands were selected as follows:

$$\begin{aligned} -20.5^\circ \leq u_1 \leq 20.5^\circ \\ 0.5 \text{ m/s} \leq u_2 \leq 1.0 \text{ m/s} \end{aligned} \tag{5.1}$$

The pRRT algorithm

In this thesis, we do not take into consideration the perception module and the high-level planning module. Hence the environment details, like the lane boundaries, are pre-fed to

the algorithm as known values. As we focus on low-level trajectory generation, it is assumed that the destination position is known. The starting position of the ego vehicle is considered the origin of the global reference frame.

In the simulations, we used the location of the obstacle vehicles and static obstacles for generating the PPM for pRRT. For experimentation, we rely just on the lidar information and hence don't know the type of obstacle. Hence each detected point is considered an individual obstacle and used for generating the PPM. The lidar gives the location in terms of distance from the lidar center and the angle from the zeroth beam. The detection is limited to the front field of view of lidar, spanning a total of 70°. These are converted to the cartesian coordinates in reference to the lidar's local coordinate frame. A transformation matrix between the lidar position and the center of gravity of the vehicle gives the obstacle position in the vehicle's local frame. In the method described in simulations, the algorithm considers everything in the global reference frame. In the initial position, the local frame and the global frame coincide, and hence no need for transformation is required to get obstacle locations in the global frame to generate the PPM.

Once the obstacle locations are determined, the environment details, the goal location, and the obstacle points are fed into the pRRT algorithm, which tries to find a feasible solution. The resulting trajectory gives a sequence of control inputs at a timestep of 0.1s. The lidar is set to run at the frequency of 10Hz, and hence it updates the lidar readings at every time increment of 0.1s. The location of the vehicle is tracked using the VESC odometry. The VESC returns the distance and the direction covered by the vehicle at each time step. Using this, the current location and heading angle are determined in the global coordinate frame. The obstacle locations observed by the lidar are converted from the vehicle's local frame to the global frame. At each time step, a collision check is performed with the obtained trajectory and the new obstacle positions. If no collision is detected, the vehicle implements

the control input of the node, which is nearest to the current vehicle position in the tree. If, at any point, a collision is detected, the trajectory is cleared, replanning takes place, and the process is repeated until the vehicle reaches the goal position.

5.2 Experiments

5.2.1 Intersection Environment

Due to limited space in the laboratory, a one-lane intersection environment was created for experiments. The environment was similar to the intersection environment used in simulations. A lane width of 22 inches was used for the environment. As seen in figure 5.2, instead of curved curbs, right angles were used, making the right turn very sharp. Pedestrians, buildings, and other cars were used as obstacles.

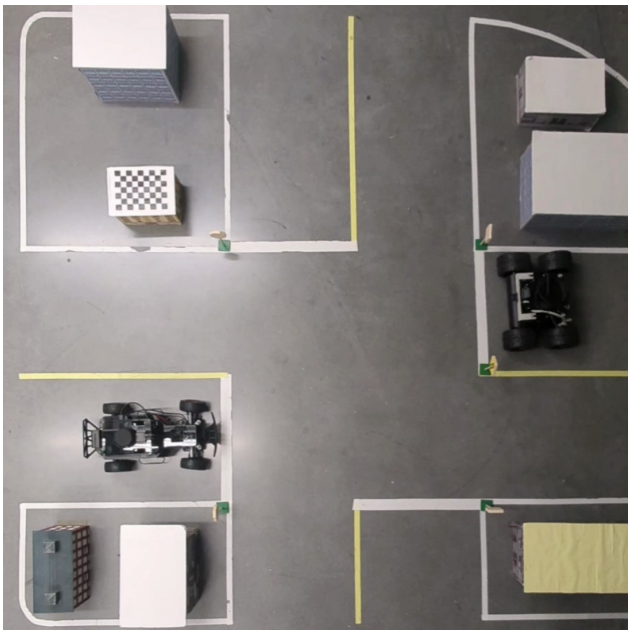


Figure 5.2: Single Lane Intersection Environment

For testing the scalability, a 2-lane intersection was also constructed with the same lane

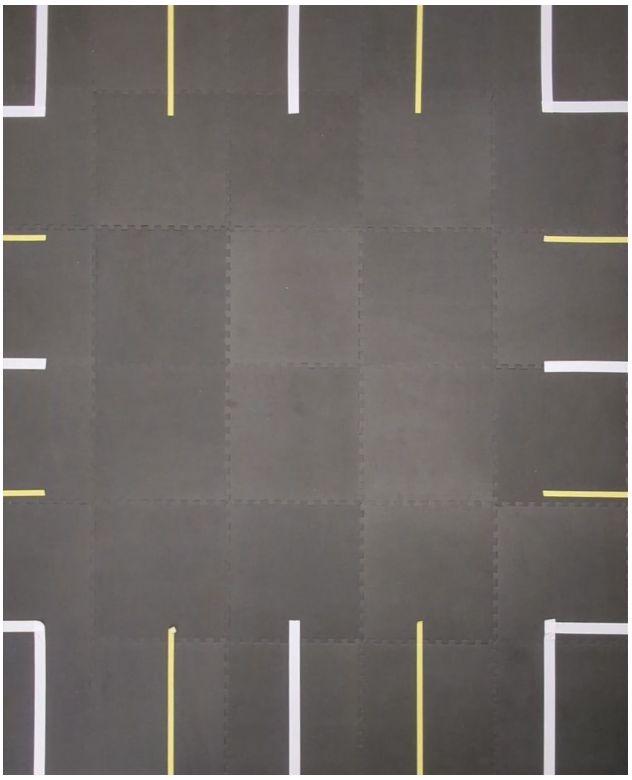


Figure 5.3: Double Lane Intersection Environment

widths. Figure 5.3 shows the 2-lane intersection. Due to limited space, the entry and exit lengths of the lanes were limited. A rubber pad was used for the construction of this environment, which had a higher static friction coefficient as compared to the tiled floor.

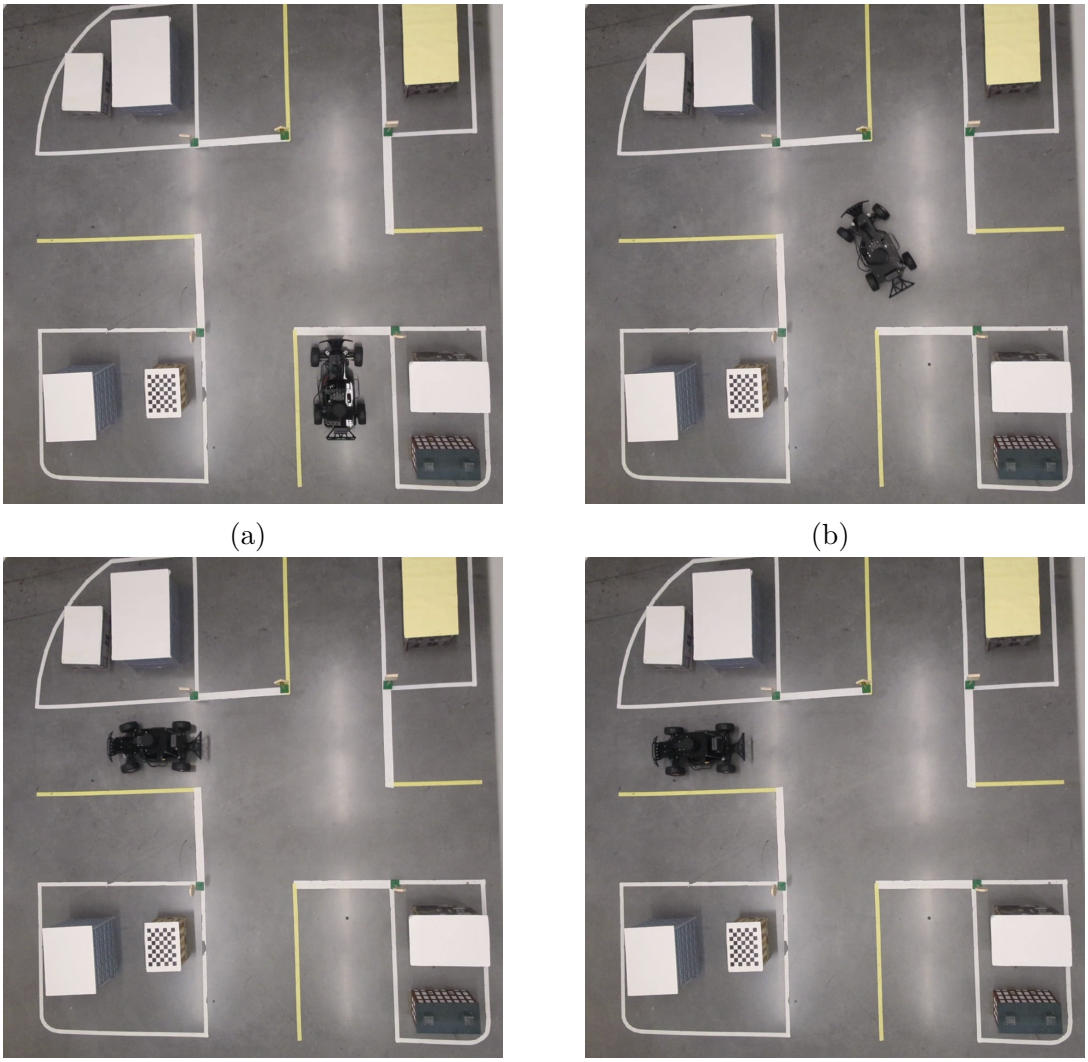
5.2.2 Testing scenarios

The algorithm was tested on an empty intersection for a left turn and straight maneuvers. The limit on the maximum steering angle and the minimum required vehicle speed of the Xtenth platform makes it impossible for it to make a right turn in the environment. Hence only the left turn and straight maneuvers were tested. To observe the performance with static obstacles, a static pedestrian was added in the middle of the intersection. For dynamic obstacles, the path of the vehicle was obstructed by a human once the ego vehicle started

moving. For all the scenarios, the bias value (λ) was chosen to be 100, the standard deviation (σ) was chosen to be 0.1, and the spacing (Δ) was chosen to be 0.1 as well.

5.2.3 Results and discussions

Empty Intersection

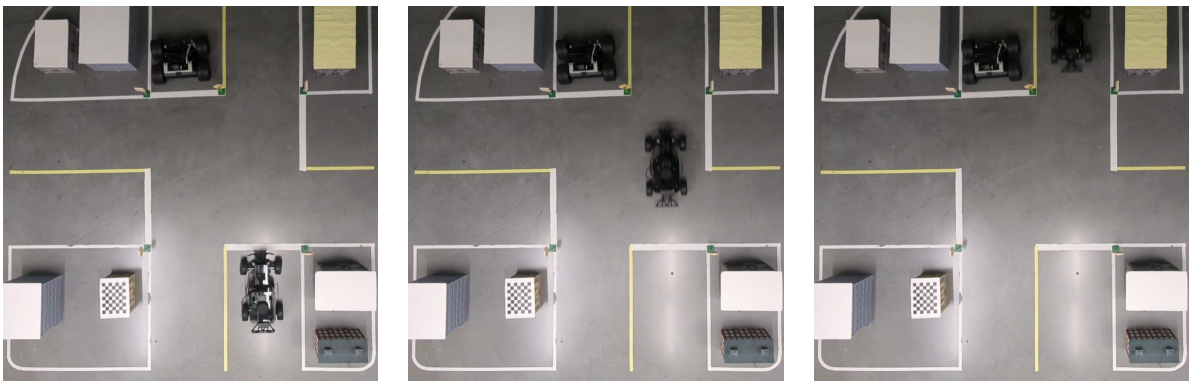


(a) (b)

(c) (d)

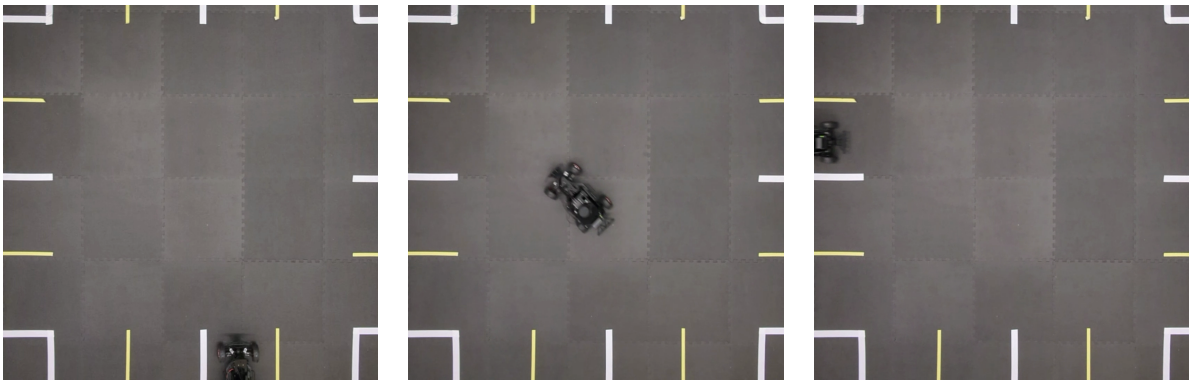
Figure 5.4: Left turn on an empty intersection

Figure 5.4 summarizes the left turn maneuver by the scaled-down vehicle on the empty intersection. The path planned by the pRRT algorithm conforms to human intuition and makes a good left turn. In the end, due to the vehicle's inertia, it overshoots the goal position and ends up moving a little bit further than the destination.



(a) (b) (c)

Figure 5.5: Straight on an empty intersection



(a) (b) (c)

Figure 5.6: Left on an empty 2-lane intersection

Figures 5.5 and 5.6 show more cases with an empty intersection. Even with a larger intersection, the trajectory planned by the algorithm is quick and smooth.

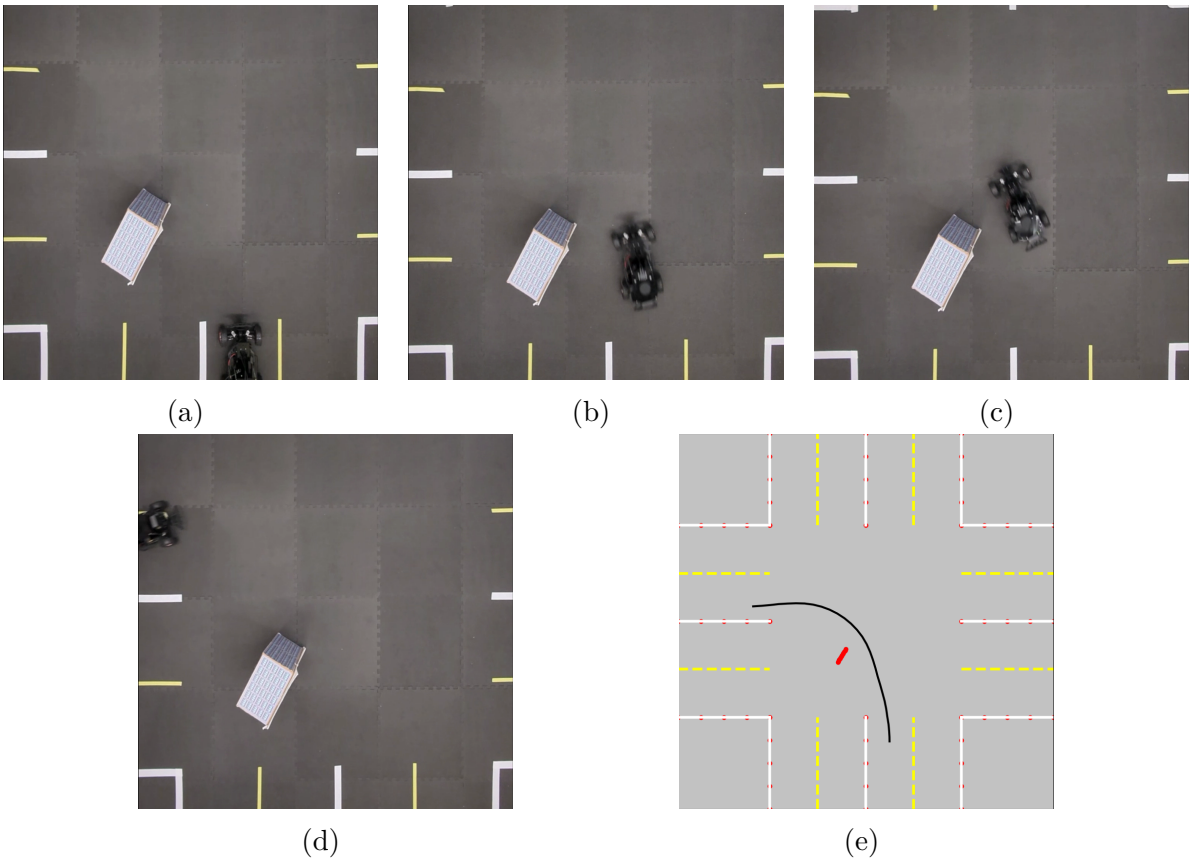


Figure 5.7: Left turn trajectory planning around a static obstacle in an intersection

Static Obstacle

For a static obstacle, a cardboard object was placed in the middle of the intersection, which can be detected using the lidar. The obstacle was positioned such that if the algorithm did not consider an obstacle, the resulting trajectory would result in a collision. Due to the relatively lower value of the bias factor, the algorithm was able to find a distorted collision-free trajectory around the static obstacle.

Figure 5.7 and 5.8 show the trajectories generated by the pRRT algorithm for a left turn with different positioning of the obstacle. Figures 5.7e and 5.8e show the path generated by the algorithm. As observed, the algorithm is flexible enough to find paths around the

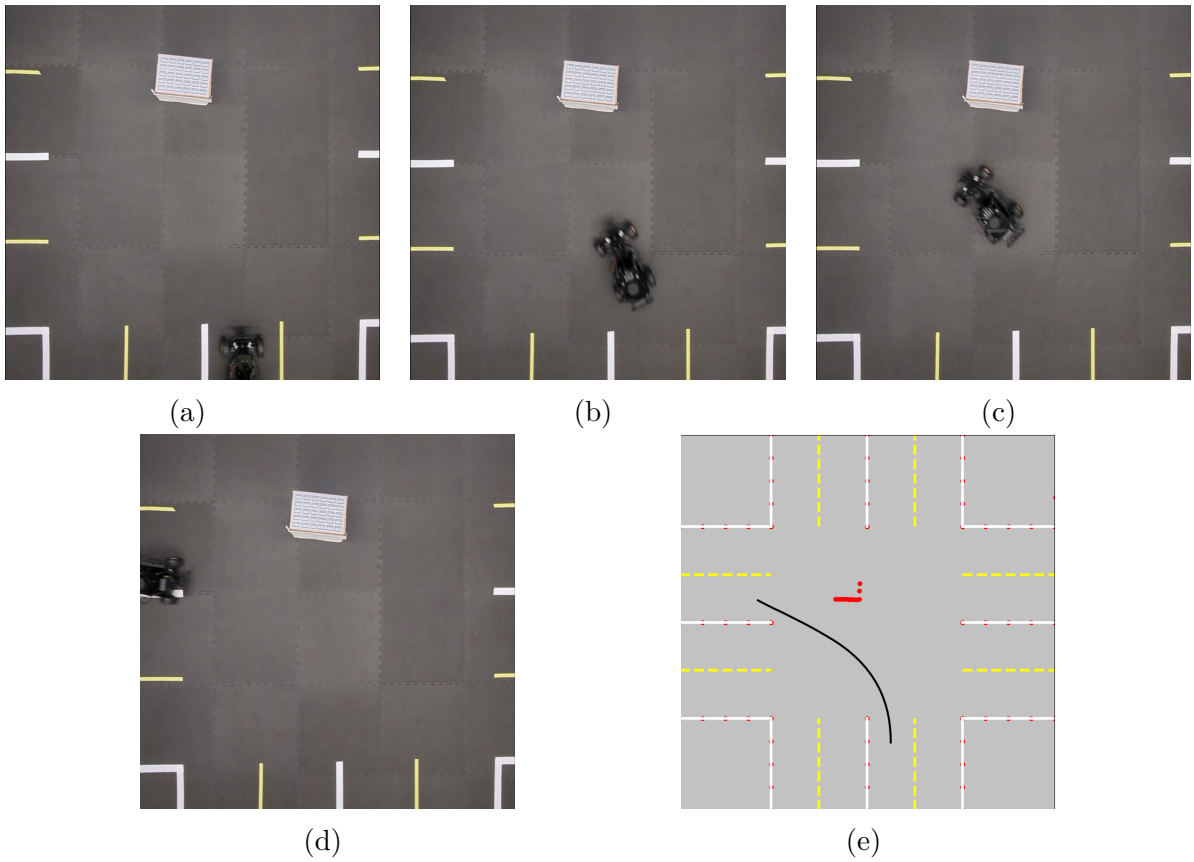


Figure 5.8: Left turn trajectory planning around a static obstacle in an intersection

obstacle and completes the maneuver without collisions.

Figure 5.9 shows the case when the ego vehicle wants to go straight, but there is a static obstacle blocking that path. The algorithm finds a path quickly to go around the static obstacle and avoid collisions. Figure 5.9g shows the path generated by the algorithm. As we can observe from the movement of the vehicle in figures 5.9a to 5.9f, the vehicle does not follow this generated path very accurately. This error in tracking was attributed to the weak localization of the experimental platform. The VESC odometry uses the control signals and the bicycle kinematic model for estimating the vehicle's current position. This results in an inaccurate estimation of the position and heading as it cannot compensate for the motion of the vehicle due to inertia/momentum and for dynamic factors like static friction. Hence the

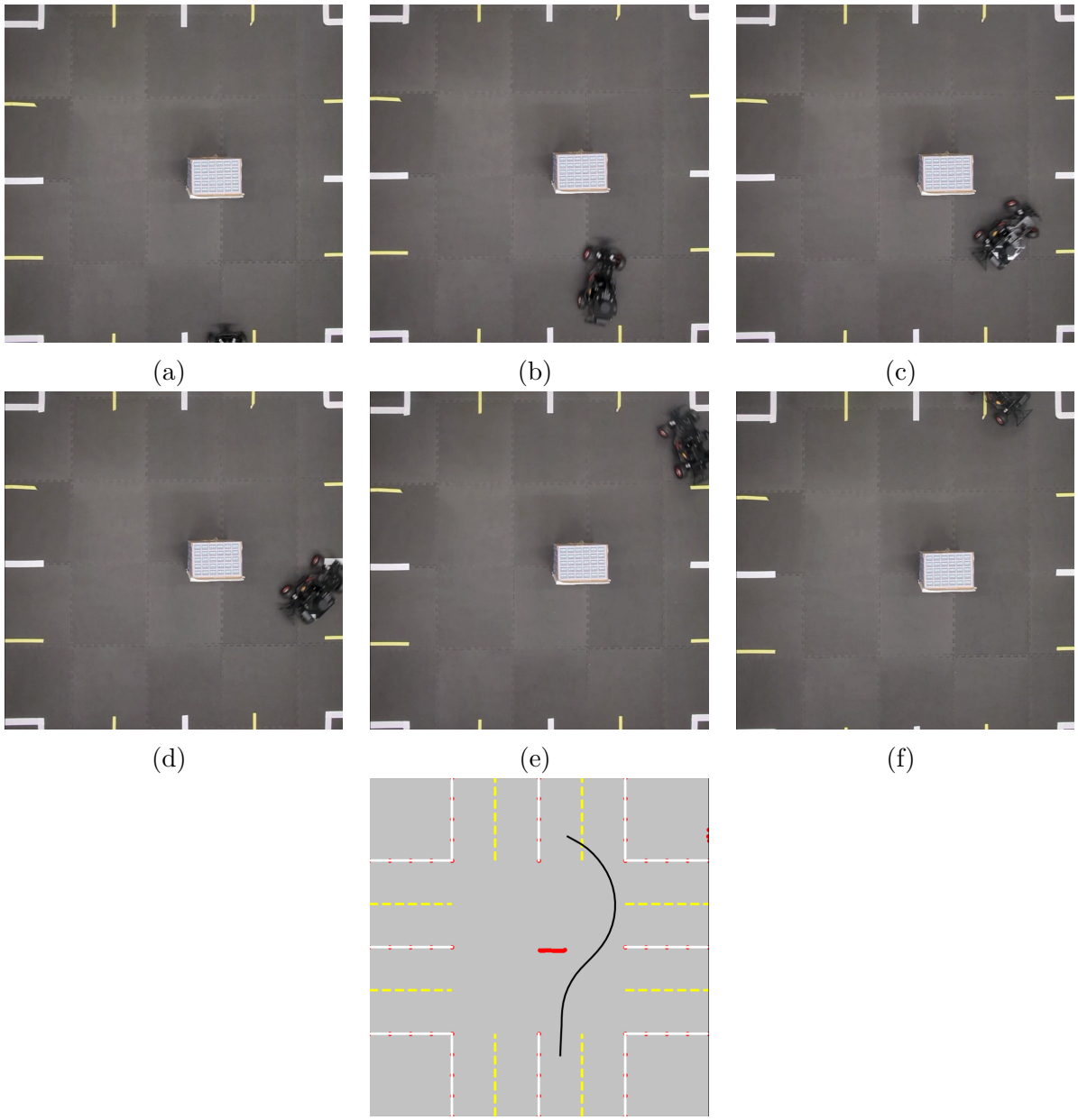


Figure 5.9: Straight trajectory planning around a static obstacle in an intersection

vehicle is not able to keep track of which control input from the tree needs to be executed next.

Dynamic Obstacle

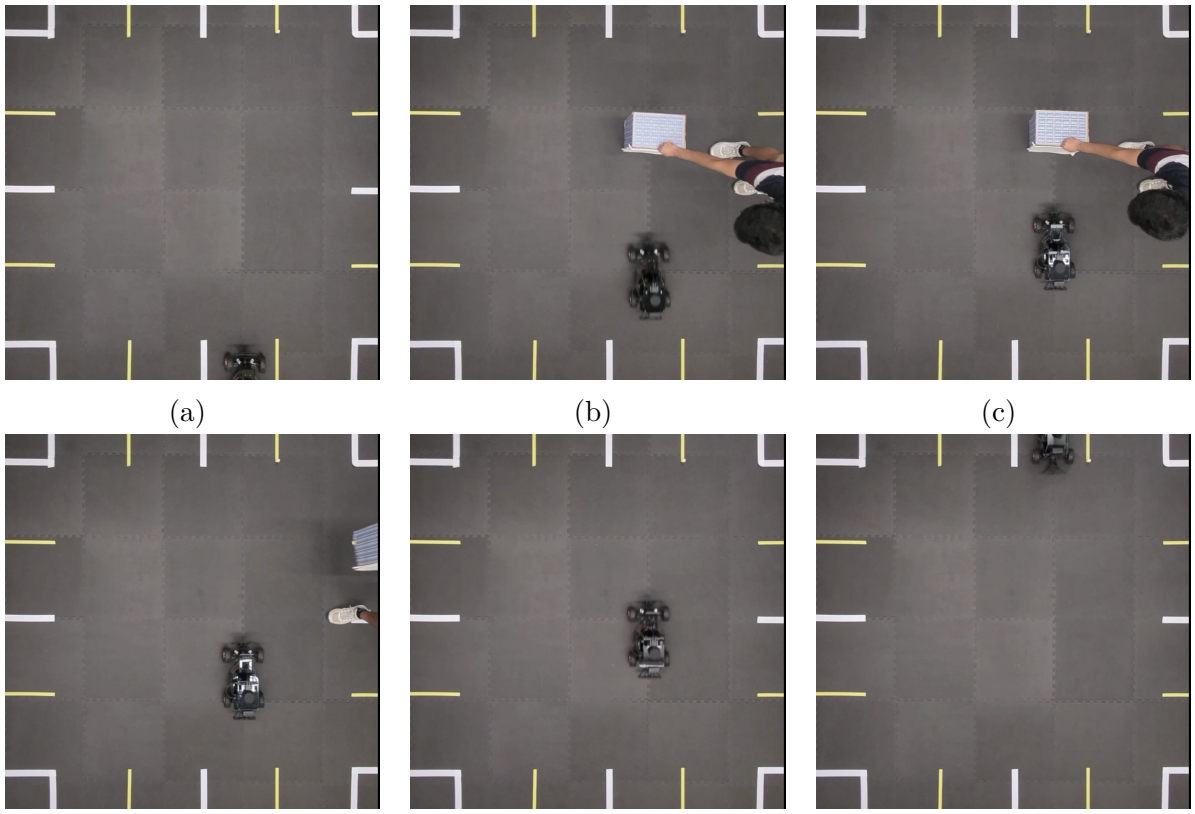


Figure 5.10: Ability of the algorithm to handle dynamic obstacles with the following timestamps: a) t = 0 s; b) t = 1.5 s; c) t = 2 s; d) t = 3 s; e) t = 5 s; f) t = 8 s

Figure 5.10 shows the scenario when the ego vehicle goes straight and an obstacle suddenly hinders its motion. The vehicle detects the obstacle at t = 1.5s and detects a collision. It starts stopping, but due to its momentum, it goes a little further, as seen in figure 5.10c. The algorithm tries to find a trajectory with the obstacle present. Due to a limit on the maximum iterations allowed and the position of the obstacle, it is unable to find any alternate path. Once the obstacle moves away, the vehicle finds a path from its current location to the goal position quickly and starts moving. As seen in figure 5.10d, as soon as the obstacle moves away, the vehicle starts moving. The ego vehicle completes the whole move within 8 seconds.

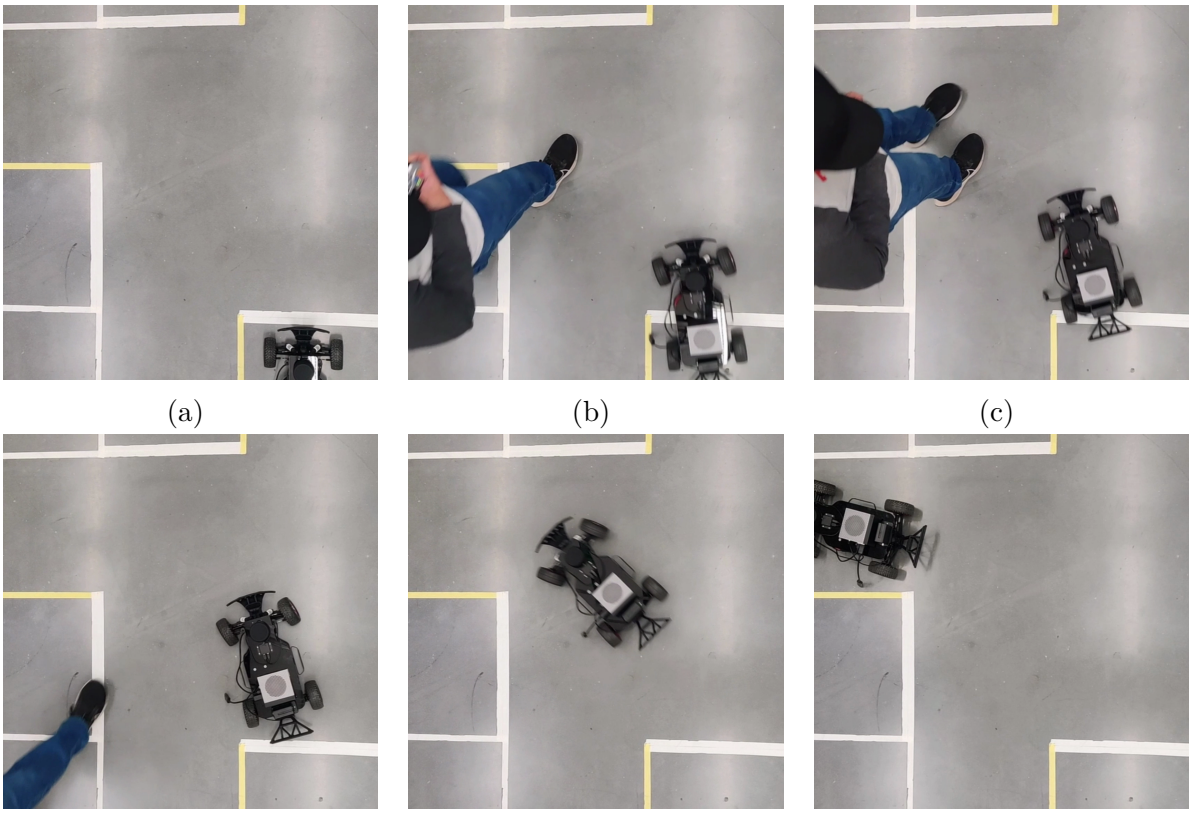
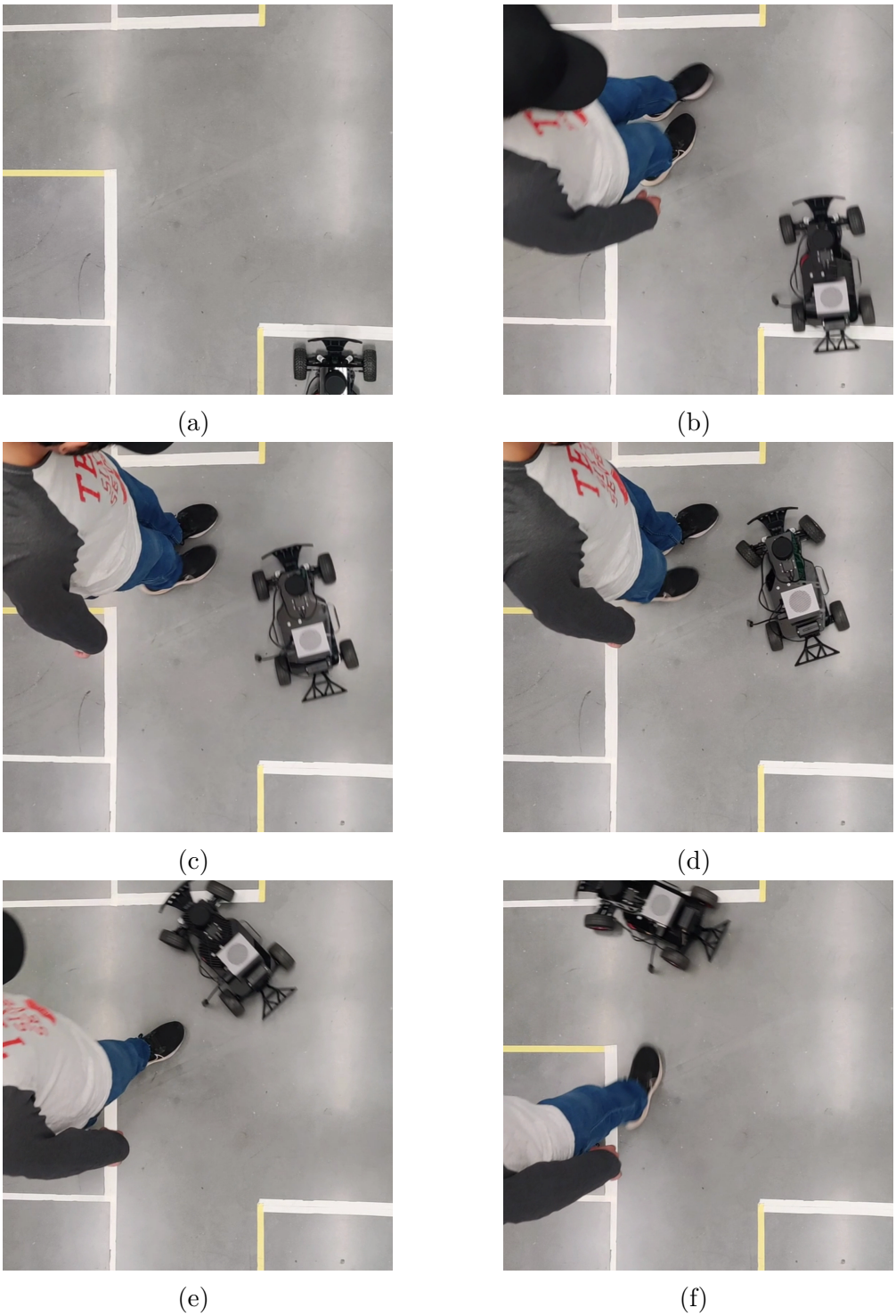


Figure 5.11: Ability of the algorithm to handle dynamic obstacles with the following timestamps: a) t = 0 s; b) t = 3 s; c) t = 4 s; d) t = 5 s; e) t = 6 s; f) t = 8 s

Figure 5.11 shows the scenario when the ego vehicle makes a left turn and an obstacle suddenly hinders its motion. A real human stepped into the environment as an obstacle for the ego vehicle. The vehicle detects the obstacle at t = 3s and detects a collision. It starts stopping, but due to its momentum, it goes a little further, as seen in figure 5.11c. The algorithm tries to find a trajectory with the obstacle present. Due to a limit on the maximum iterations allowed and the position of the obstacle, it is unable to find any alternate path. Once the obstacle moves away, the vehicle finds a path from its current location to the goal position quickly and starts moving. As seen in figure 5.11d, as soon as the obstacle moves away, the vehicle gets a steering input and starts moving. The ego vehicle completes the whole move within 8 seconds.



(a) (b) (c) (d) (e) (f)

Figure 5.12: Dangerous trajectory tracking due to inaccurate localization: a) t = 0 s; b) t = 4 s; c) t = 5 s; d) t = 6 s; e) t = 7 s; f) t = 7 s

Figure 5.12 shows the same scenario with the entry of the obstacle being 1 second late. As shown in figure 5.12b and 5.12c, the vehicle attained a higher velocity and progressed more than the previous case when the obstacle entered the intersection. This resulted in the vehicle stopping further than it should have. Due to bad SLAM, the current position of the vehicle is assumed to be the position at which the vehicle stopped giving inputs and not compensate for the movement due to inertia and as shown in the second subfigure, 5.12b.

Due to the inaccurate current position, the obstacle positions obtained in the global frame are inaccurate as well. In our case, the algorithm decides that the observed obstacles do not interfere with a generated trajectory and hence starts moving. As seen in figures 5.12e and 5.12f, the vehicle crosses the road boundaries due to inaccurate estimates of their locations. If the vehicle had actually stopped in the position indicated in the second subfigure, the motion of the vehicle would have been accurate and collision-free.

This can be mitigated by having an accurate SLAM module for the pose estimation of the ego vehicle. For a full-sized vehicle, GPS, along with the IMU, works as a reliable source for pose estimation. With the lab environment being indoors and the movements being too small for accurate GPS readings, GPS becomes infeasible. Motion capture cameras can be used inside the indoor lab environment for accurate localization of the ego vehicle. If an accurate pose is determined for the ego vehicle, the planning and tracking becomes more accurate reducing the risks of collision.

Chapter 6

Probabilistic RRT-Connect

Standard pRRT creates a bias for sampling more points towards the goal location. This bias is governed by the bias parameter and the variance for the position probability map (PPM). And hence, it becomes necessary to tune these parameters depending on the scenario. A higher bias value leads to faster convergence in intersections with only moving traffic and no static obstacles. As seen in the previous chapter, the time for convergence and the number of iterations increased by a factor of 10 when a static obstacle was placed in the middle of the intersection. in this chapter, we propose a new motion planning method for rapid planning around obstacles.

6.1 Base algorithm

While driving, the basic human instinct lets the driver to go around any obstacle that it sees on its path. It may not be necessary to consider all the elements present in the environment if they do not hinder an initial path. In the scenario shown in figure 6.1, amongst multiple obstacles, only 1 hinders the path of the ego vehicle. Hence in this case, it might not be necessary to consider the far away obstacles for path planning. Hence, in our, we use a heuristic approach where the algorithm discovers obstacles in an initial path and re-routes around it to obtain a collision-free path.

Initial Path: The algorithm finds an initial path by directly connecting the start and goal

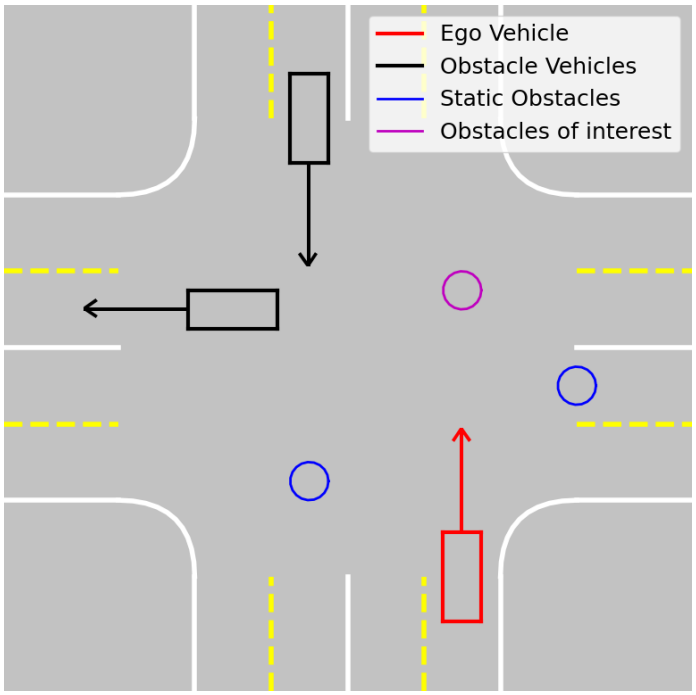


Figure 6.1: Intersection with multiple static and dynamic obstacles

points with a straight line by considering the environment to be empty and without any obstacles.

Encountered obstacles: Collision check for the path is done sequentially with all the obstacles starting from the nearest to the farthest. As soon as a collision is detected with any obstacle, that obstacle is returned as the '**obstacle of interest**' and the collision flag is marked as true. This obstacle is then added to a list of all the encountered obstacles. Hence, this list keeps track of all the obstacles that the algorithm encounters.

Obstacle Points of interest: The obstacles are discretized into distinct points as shown in figure 6.1. Using all points for intermediate goal selection is redundant and increases the convergence time. Hence only the points with a distance more than half the width of the ego vehicle are selected. This step gives the minimum number of points required for describing the obstacle from the perspective of the ego vehicle. These points are shown in black color

in figure 6.1.

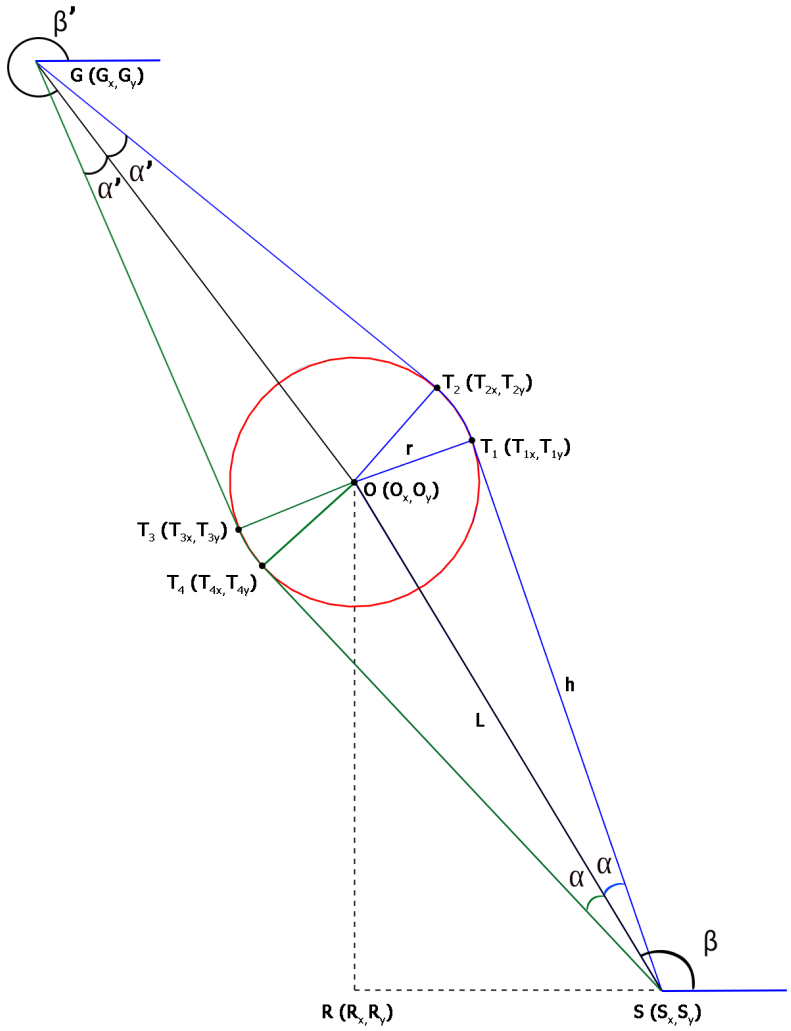


Figure 6.2: Schematic for finding the path around a selected obstacle point

Candidate paths: For each of the selected points, a safety circle with a threshold radius is generated. Using a threshold radius equal to the half-width of the vehicle results in a safe path with no safety factor. Hence this threshold radius should be selected to be more than half-width of the vehicle. In this work, we will use this threshold to be 0.75 times the width of the car. For each circle, tangent lines to it from the start point and the goal point are

determined. The two tangents drawn on the same side of the circle are connected by an arc of the circle. This process is demonstrated in figure 6.2.

Here $S(S_x, S_y)$ represents the start point and $G(G_x, G_y)$ represents the goal point. Two tangents are drawn from both these points to the safety circle. Considering tangents from the start point, $\overline{PT_1}$ and $\overline{PT_3}$, the points of intersection (T_1 and T_3) between the circle and the tangents can be obtained as follows:

$$\begin{aligned} h &= \sqrt{L^2 - r^2} \\ \beta &= atan(O_y - P_y, O_x - P_x) \\ \alpha &= atan(r, h) \\ T_{1x} &= S_x + h * cos(\beta - \alpha) \\ T_{1y} &= S_y + h * sin(\beta - \alpha) \\ T_{2x} &= S_x + h * cos(\beta + \alpha) \\ T_{2y} &= S_y + h * sin(\beta + \alpha) \end{aligned} \tag{6.1}$$

Similarly the tangent-circle intersection points from the goal point, T_2 and T_4, can be obtained as well. A path is generated by connecting $\overline{PT_1}$, $\widehat{T_1T_2}$ and $\overline{PT_2}$. Similarly another path is generated using the other pair of tangents. Hence, for each selected obstacle point, two candidate paths are obtained. These paths are checked for collision with the **encountered obstacles**. All the collision-free paths generated by all the obstacle points of interest are collected in a list of candidate paths.

Intermediate start points: The point of intersection of the tangents from the goal point with the safety circle (indicated as T_2 and T_4 in figure 6.2) are then considered as intermediate start and are associated with their respective candidate paths.

From each intermediate start points the algorithm repeats recursively until a safe collision-

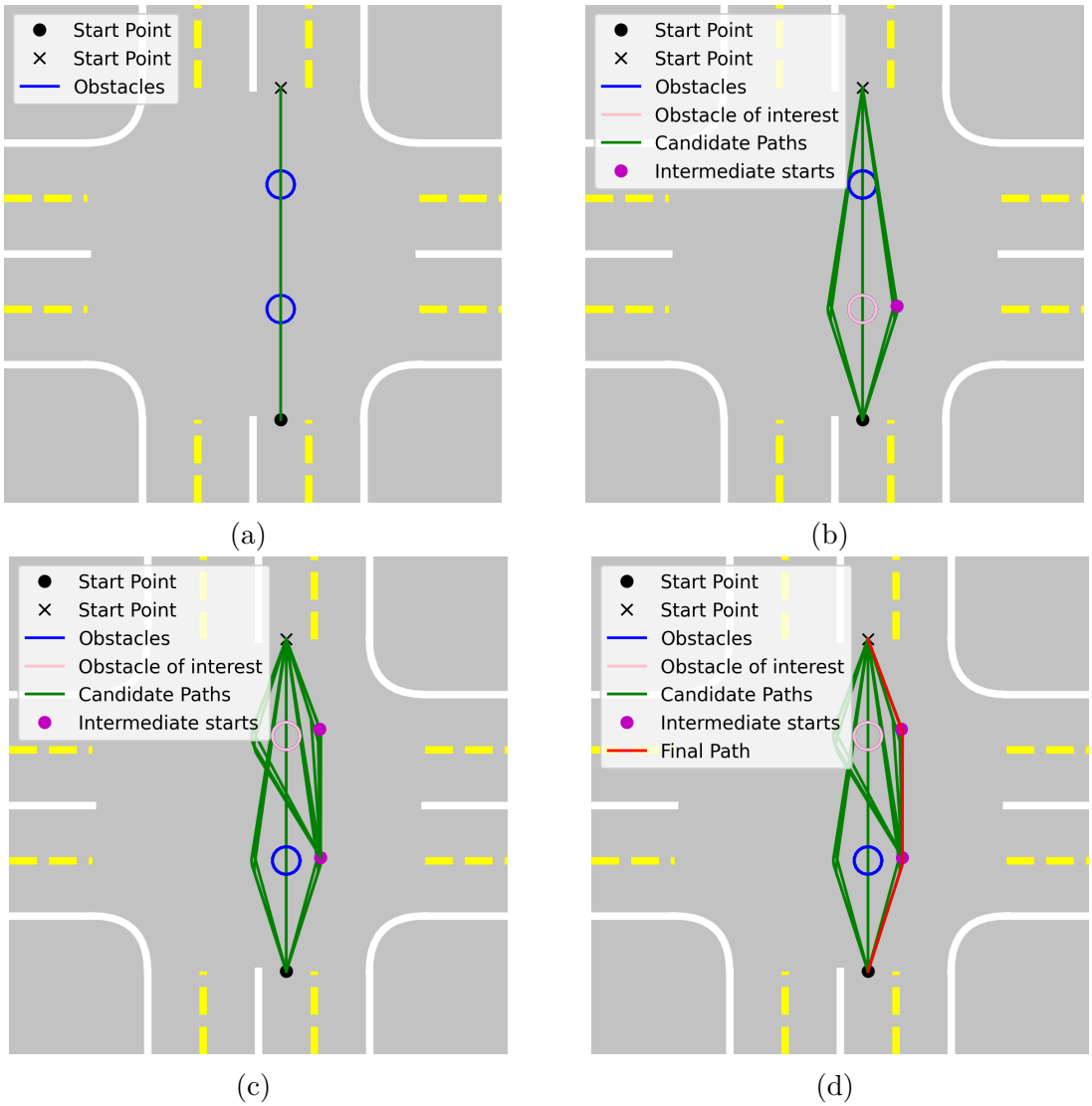


Figure 6.3: Progression of the algorithm to find a path using our method. a) An initial straight line path is obstructed; b) Candidate paths using first obstacle as the obstacle of interest; c) Repeating the algorithm from the first intermediate point; d) Obtaining a final path from the intermediate point of the second obstacle of interest

free path from start point to the goal point is obtained. This process is explained using the figure 6.3 As seen in figure 6.3a, an initial path is determined by directly connecting the start and end points. This path encounters a collision with the nearest obstacle. This obstacle becomes the obstacle of interest and is added to the **encountered obstacles** list. Candidate paths and their corresponding intermediate starts are obtained around the ob-

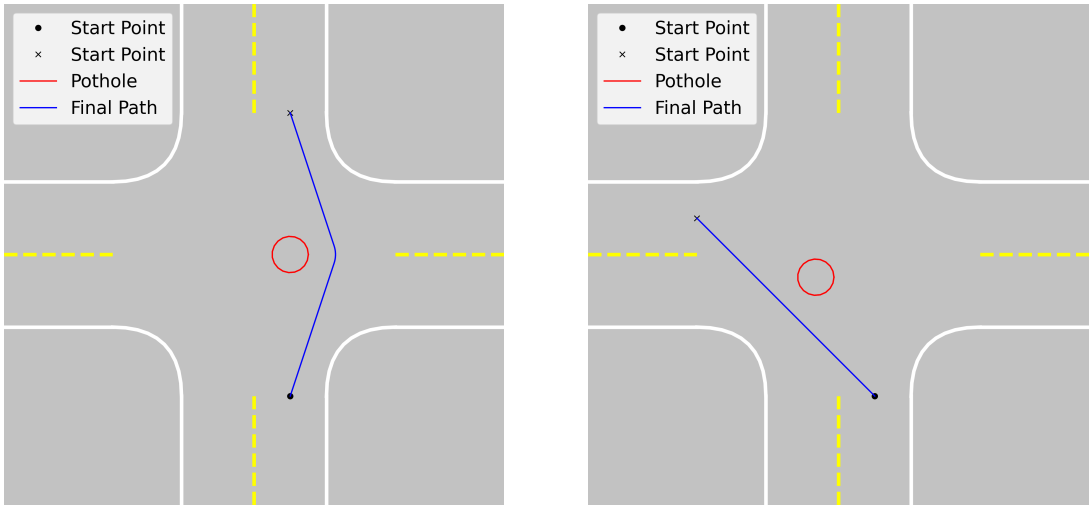
Algorithm 5 Base algorithm

```
1: Traj ← Connect(x_0, x_g)
2: Flag, obs ← Collision(Traj, All Obstacles)
3: if Flag is True then
4:     EncounteredObs.append(obs)
5:     Success, Traj ← FindPath(x_0, x_g, obs, EncounteredObs)
6:     if Success is False then
7:         for obs in All Obstacles do
8:             Success, Traj ← FindPath(x_0, x_g, obs, EncounteredObs)
9:             if Success is True then
10:                Break
11: function FindPath(x_0, x_g, obs, EncounteredObs)
12:    paths ← FindCandidatePaths(x_0, x_g, obs)
13:    for path in paths do
14:        tmpStart ← IntermediateStart(path)
15:        Flag, obs ← Collision(path, EncounteredObs)
16:        if Flag is True then
17:            return 0, NULL
18:        Flag, obs ← Collision(path, All Obstacles)
19:        if Flag is False then
20:            return 1, path
21:        else
22:            EncounteredObs.append(obs)
23:            Success, Traj ← FindPath(x_0, x_g, obs, EncounteredObs)
24:            if Success is True then
25:                path[2] ← Traj
26:                return 1, path
27:            else
28:                for obs in All Obstacles do
29:                    Success, Traj ← FindPath(x_0, x_g, obs, EncounteredObs)
30:                    if Success is True then
31:                        path[2] ← Traj
32:                        return 1, path
33:    return 0, NULL
```

stacle of interest, as shown in figure 6.3b. Using the intermediate start corresponding to the first candidate path, the algorithm repeats itself. Another obstacle obstructs the straight-line path from the intermediate start to the goal point. This obstacle is now considered an obstacle of interest and has been added to the encountered obstacles list. Candidate paths and intermediate starts from this intermediate start point are determined. The algorithm repeats again from these new candidate paths, and their corresponding intermediate starts as shown in figure 6.3c. The straight line from the new intermediate start to the goal point results in a collision-free path, and hence the final collision-free path is obtained as seen in figure 6.3d. The stepwise details of the algorithm are summarized in algorithm 5

6.2 Implementation for autonomous driving:

As seen in the previous section, the obtained trajectories using the base algorithm consist of paths and not the control inputs. Hence, it might be difficult for the AV to track the path. An initial formulation of this method was presented in [64].



(a) (b)

Figure 6.4: Path generated by our method in a one-lane intersection with static obstacles

As seen in figure 6.4, the path generated by the algorithm might be difficult for an AV to track in the beginning if it is positioned such that it faces straight towards the top exit. Hence in order to add a component to check for the feasibility of the path generated, we use the strategies of pRRT and RRT-Connect to obtain a path that is easier to track. The concept of selecting intermediate goals and start points using the base algorithm is introduced. Using these intermediate points, multiple trees are formed using a highly biased closed-loop pRRT algorithm that uses the bicycle kinematic model and optimal steering control to obtain kinematically feasible trees. These multiple trees are then connected to each other to form a final collision-free path which is easier to track using tracking techniques.

Initial Trajectory: Initially, we consider the environment to be free of obstacles, which means $\mathcal{X}_{occ} = \phi$ and $\mathcal{X}_{free} = \mathcal{X}$. A trajectory is generated from x_0 to x_g in this environment using pRRT with a high bias value. A collision check is performed to check if any detected obstacles hinder the generated trajectory.

Finding intermediate points: If a collision is detected on the initial path, the base algorithm is used to find intermediate points. In the base algorithm, the intersection points between the tangents and the safety circle are selected, as shown in figure 6.2. The point of intersection between the safety circle and the tangent from the start point is selected as an intermediate goal point. The point of intersection between the safety circle and the tangent from the goal point is selected as an intermediate start point. Hence, the points T_1 and T_2 are selected as the intermediate goal and start points, respectively, in figure 6.2.

Growing multiple trees: Once all the intermediate points are found, the pRRT algorithm tries to find feasible trajectories between pairs of intermediate start and goal points using a high bias value. For example, in figure 6.2, two trajectories are found; one between the start point, S, and the intermediate goal point, T_1, and the second between the intermediate start point, T_2, and the goal point, G. For each of the trees, a collision check is performed

with all the obstacles. If a collision is detected, the base algorithm is used to find another set of intermediate points, and the process is repeated. Once all the feasible trajectories are generated, the paths obtained from them are connected using a bezier curve.

Path Smoothing: The final path obtained might have small discontinuities, and hence the path is smoothened using bezier curves as described by [**?**]. A bezier curve is a parametric curve defined by the following expression:

$$P(t) = \sum_{i=0}^{n} B_i^n(t) \cdot P_i \tag{6.2}$$

B_i^n is known as Bernstein polynomial, and its expression is given by:

$$B_i^n(t) = \binom{n}{i}(1-t)^{n-1}t^i \tag{6.3}$$

Here P_i are the control points, and n is the number of control points. In our problem, we consider the positions of the nodes generated by the two successful trajectories as the control points. The parameter t varies from 0 to 1. Hence, $t \in [0, 1]$.

Control commands for path tracking: The final path obtained by bezier curves only consists of the spatial coordinates for the path. Model Predictive Control (MPC) is used for generating longitudinal control command which is the velocity ($u_2 = v$) and a basic Stanley controller is implemented for lateral control command which is the steering angle ($u_1 = \delta$). A timestep of $\Delta t = 0.1s$ was selected for tracking the path. For the MPC controller, a prediction horizon of 20 time-steps and a control horizon of 15 time-steps were selected. The following objective function was used:

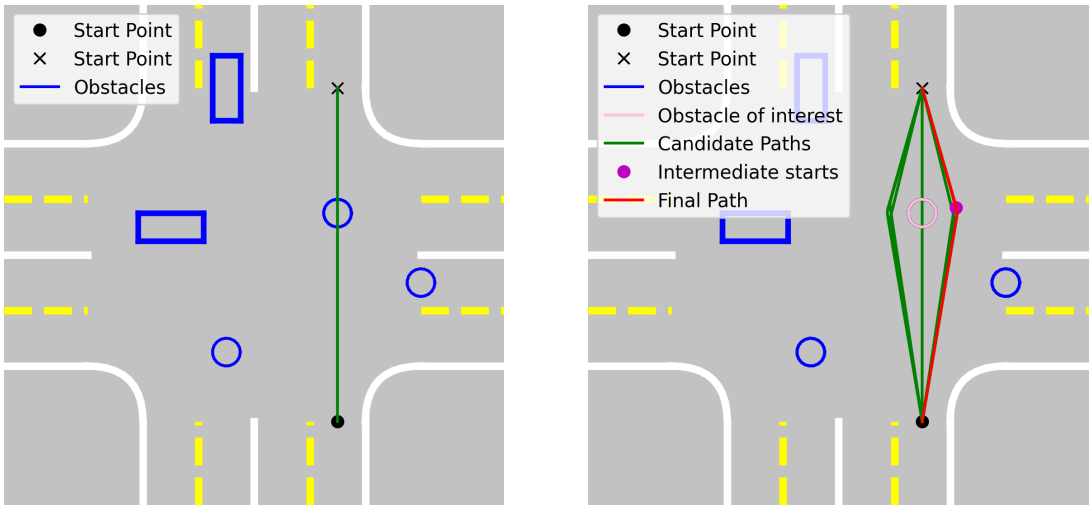
$$\begin{aligned} J(v) = \quad & (P - P_{ref})^T \cdot w_1 \cdot (P - P_{ref}) + w_2 \cdot Kv^2 \\ & + (v - v_{max})^T \cdot w_3 \cdot (v - v_{max}) \end{aligned} \tag{6.4}$$

Here, P indicates the ego vehicle position, and P_{ref} indicates the reference path. K defines the curvature of the reference path at each location. v is the control variable which is the output of the controller. v is a vector of vehicle speeds over the control period of 15 timesteps. w_1, w_2, w_3 are the weights for each minimizing variable.

6.3 Results and Discussion:

6.3.1 Base algorithm

In this subsection, we analyze our novel base path planning algorithm with simulations and compare the performance with the standard RRT and pRRT algorithms. A bias parameter of $\lambda = 100$ and a standard deviation of $\sigma = 0.1$ were chosen for the discrete pRRT simulations.



(a) (b)

Figure 6.5: Amongst multiple obstacles, the algorithm finds a path quickly without the need to explore the environment thoroughly.

Figure 6.5 shows the ability of the algorithm to minimize computation and discard unnec-

essary parts of the environment. The environment shown in figure 6.1 does not require the algorithm to consider far-away obstacles and find a path quickly amongst the obstacles as only one of the obstacles becomes critical.

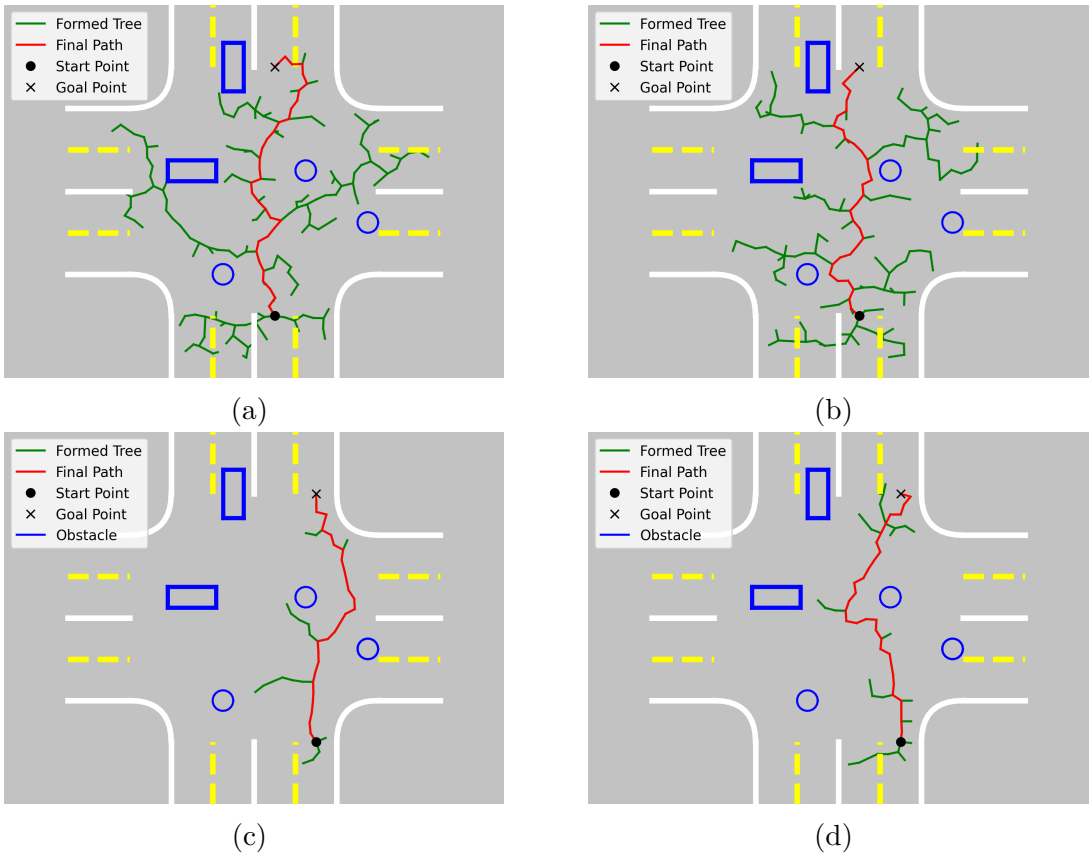
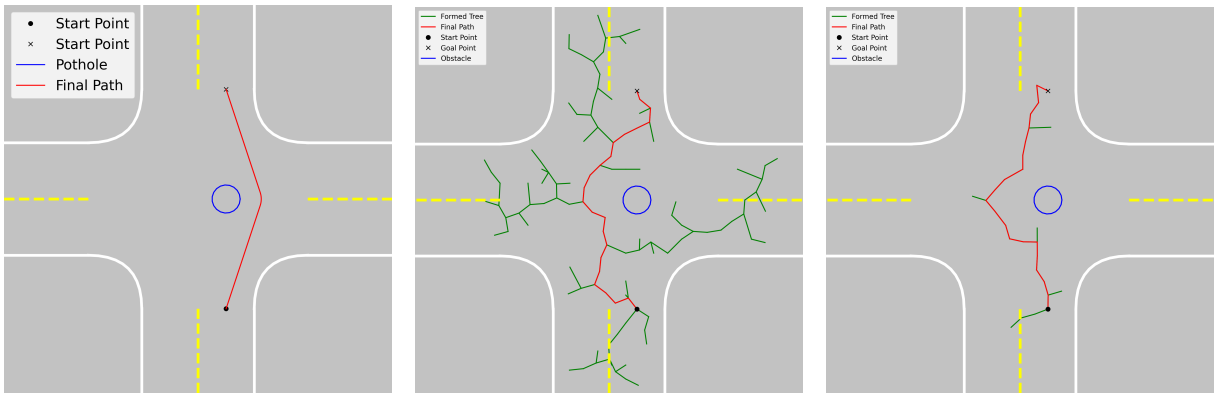


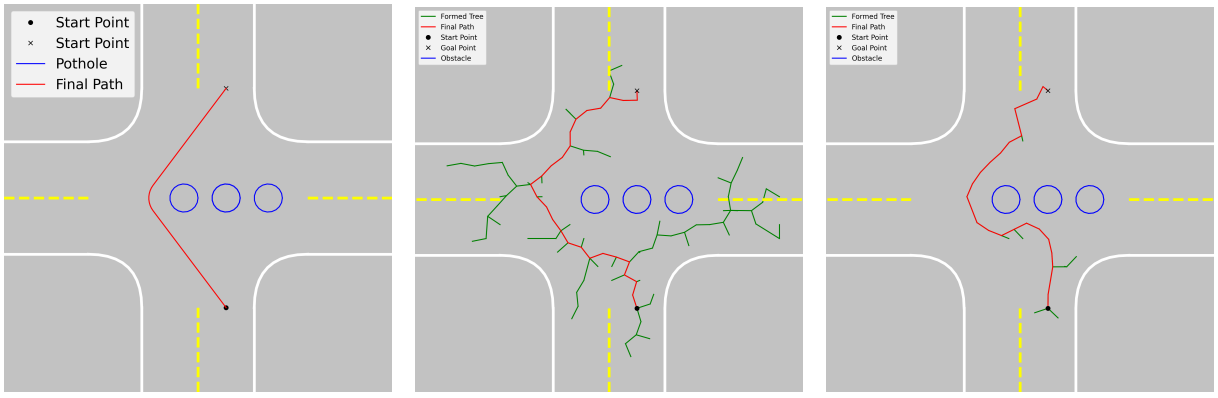
Figure 6.6: Paths obtained from standard algorithms for the given scenario: a,b) RRT; c,d) pRRT

As seen in figure 6.6, the paths formed by RRT and pRRT are highly discontinuous and random. Tracking such paths is more difficult than the one obtained by our method. Such paths are hence not very suitable for autonomous driving, which requires the vehicle to follow the vehicle dynamics model. The sharp discontinuities in these paths are infeasible for vehicle dynamics. The path obtained by our method is fairly smooth and does not have many discontinuities. This helps in efficient tracking of the generated path.



(a) (b) (c)

Figure 6.7: Paths obtained from algorithms for a scenario with one static obstacle: a) Our Method; B) RRT; c) pRRT



(a) (b) (c)

Figure 6.8: Paths obtained from algorithms for a scenario with three static obstacles: a) Our Method; B) RRT; c) pRRT

Figures 6.7 and 6.8 show the comparison of the outputs of the three algorithms in different scenarios with a different number of static obstacles. It can be observed that the paths generated by RRT algorithms explore unnecessary space and result in extended paths. The paths generated by pRRT are better as they explore minimal environments but still are able to find alternate paths around the obstacles. The paths generated by our method are the best according to human intuition.

To compare the performance of the algorithms, all the algorithms were run 100 times for

	Average CPU time (sec)	Average Cost	Success Rate
Our Method	0.0713	**1.6397**	**100%**
pRRT	**0.0544**	1.9060	**100%**
RRT	0.1355	2.1315	88%

Table 6.1: Performance comparison for the scenario with one static obstacle

	Average CPU time (sec)	Average Cost	Success Rate
Our Method	0.2771	**1.9212**	**100%**
pRRT	**0.1506**	2.3615	**100%**
RRT	0.1790	2.2081	82%

Table 6.2: Performance comparison for the scenario with three static obstacles

both of these scenarios. The average time taken by the CPU to obtain a successful trajectory in seconds was compared to check the speed of the algorithm and the rate of convergence. For each algorithm, the length of the successful trajectory was considered to be the cost of the path. To compare the optimality of the algorithms, the average costs of the paths generated by the algorithms were compared. The success rate indicates the number of times the algorithms obtained a successful trajectory out of 100 trials. The quantitative comparison is listed in tables 6.1 and 6.2

As expected, our method performs the best in terms of optimality. The pRRT algorithm is the fastest among the three, but the total time taken by each algorithm is minimal and suitable for online planning. Due to the limit on maximum iterations allowed, the RRT algorithm was not able to converge all the time and hence resulted in a lower success rate. Hence, based on human intuition, and path optimality, our method performs the best. Even though the success rate and time required by pRRT are better than our method for discrete path planning, they might be different when they are used for autonomous driving conditions. In the following section, the three algorithms were deployed to obtain kinematically feasible paths using a closed-loop structure.

6.3.2 Probabilistic RRT Connect

When our new algorithm is implemented for autonomous driving using the strategies of pRRT and RRT Connect, we call that method probabilistic RRT Connect (pRRT-Connect). In this subsection, we analyze the technique with simulations and compare the performance with the standard closed-loop RRT and pRRT algorithms.

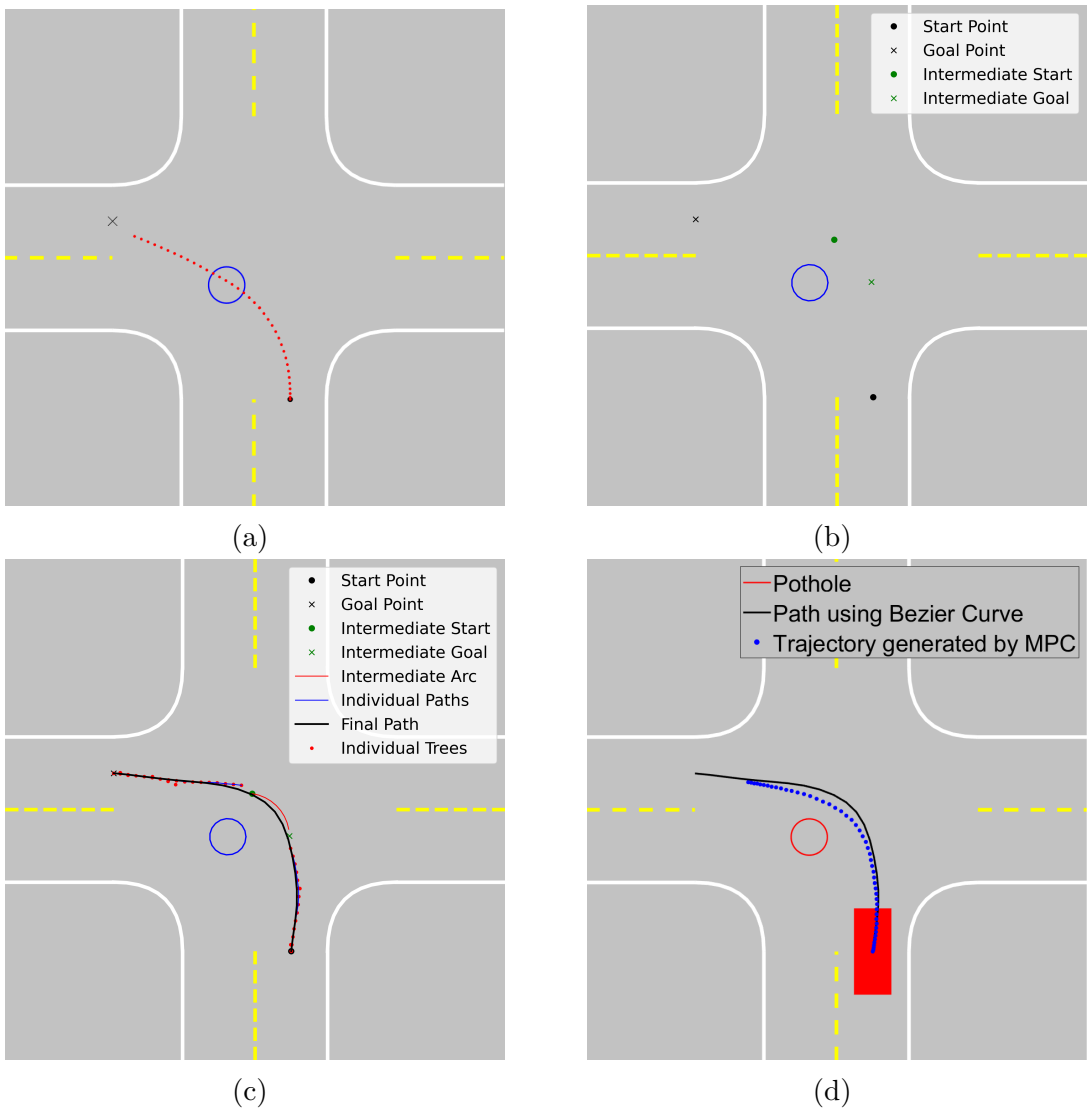
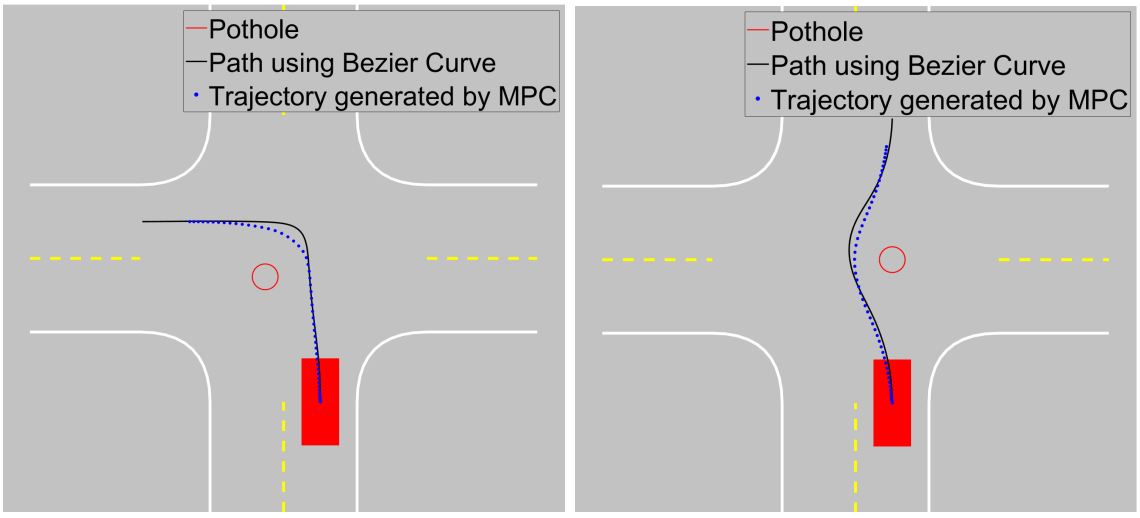


Figure 6.9: Left turn maneuver hindered by an obstacle vehicle: a) Initial trajectory generated by pRRT; b) Obtaining Intermediate points using the abse algorithm; c) Formation of multiple trees; d) Final trajectory obtained using bezier curve and the generated trees

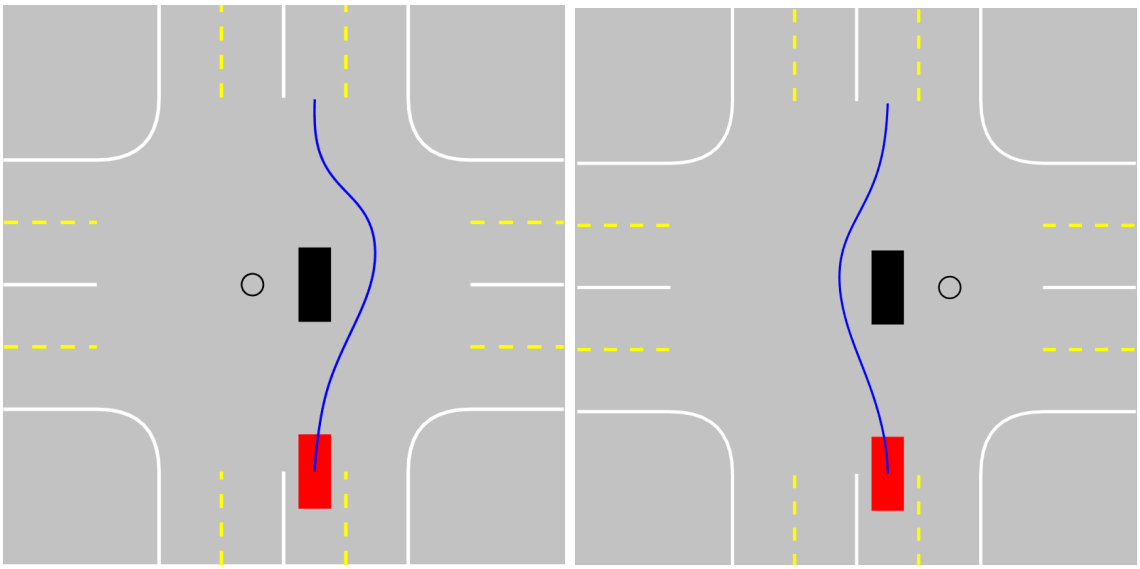
Figure 6.9 shows the case when the ego vehicle tries to make a left turn in a one-lane intersection. Initially, the algorithm considers the environment to be free of obstacles and generates a trajectory using pRRT. Collision check with the detected obstacles show a collision with an obstacle vehicle as shown in figure 6.9a. Intermediate points are obtained using the base algorithm, as mentioned in the previous sections, as shown in figure 6.9b. Feasible trajectories from the start points and goal points are obtained as shown in figure 6.9c Both the trees are connected by an arc between the intermediate start and goal points associated with the obstacle of interest. Using the bezier curves, a final path is obtained, which is tracked by the MPC controller as shown in figure 6.9d.

The algorithm efficiently finds a trajectory around the potholes as shown in figure 6.10. The trajectory generated by MPC depends on its design. Hence an error can be observed in the generated trajectory versus the generated path. For this reason, it is important to have a larger radius of safety for determining the intermediate goal points. In our study, we used the radius equal to the width of the ego vehicle giving an error margin equal to half the width of the ego vehicle.



(a) Left turn (b) Straight maneuver

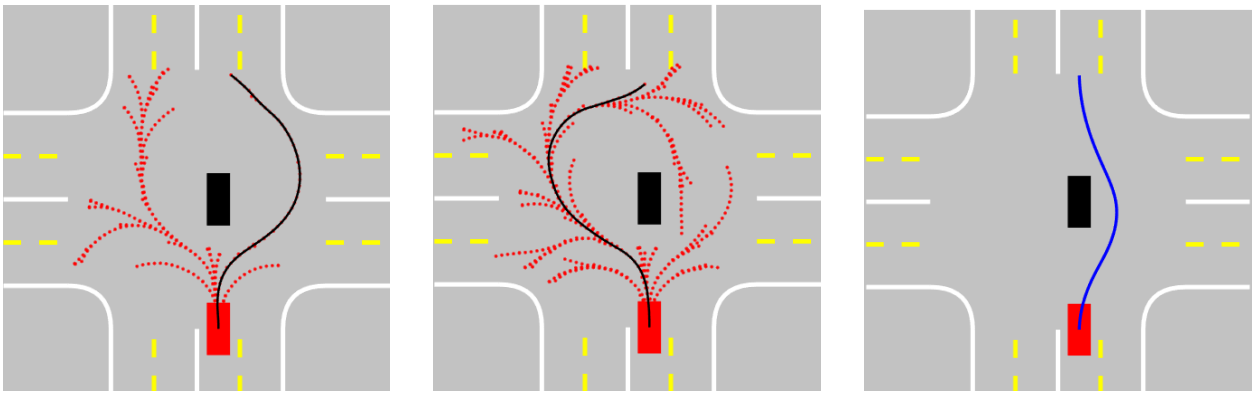
Figure 6.10: Finding a trajectory around a pothole



(a) Pothole on the left of the OV (b) Pothole on the right of the OV

Figure 6.11: Finding a trajectory around multiple obstacles in a larger environment

The algorithm can handle multiple obstacles well, as seen in figure 6.11. The scenario in the figure shows the case when an obstacle vehicle breaks down in the middle of the intersection and is unable to move. In such cases, it is important for the motion planner to find an alternate route around the vehicle. To make the scenario more complex, potholes were added to the intersection. As seen from the results, the algorithm generates trajectories similar to human intuition.



(a) Standard pRRT ($\lambda = 10^3$) (b) Standard RRT (c) Our method ($\lambda = 10^3$)

Figure 6.12: Comparison of our method with other standard methods

Figure 6.12 shows the performance of our algorithm compared to the standard methods. The trajectories obtained by the standard algorithms are too conservative and do not conform to human intuition when compared to the trajectory obtained by our method. The standard algorithms have higher convergence time as they sample points in unnecessary regions which results in large trees, as shown in the figure. This scenario was run 100 times using each of the three techniques to evaluate the performance based on the number of sampling iterations. The average number of iterations required for generating the above trajectory successfully

	Average Iterations
RRT	925
pRRT	637
pRRT-Connect	285

Table 6.3: Performance Comparison

is summarized in the table. In the proposed method, the pRRT algorithm is implemented multiple times for the intermediate points until feasible trajectories are obtained. In the best-case scenario, the first set of intermediate points can result in feasible trajectories. In the worst case, the last set of intermediate points can be fruitful. Hence it is important to reduce the obstacle points of interest, which reduces the number of intermediate goals.

Chapter 7

Conclusions and Future Works

7.1 Conclusions

In the first part of this work, we thoroughly studied the probabilistic RRT method as a good option for online motion planning in non-signalized intersections. The algorithm was compared to the standard closed-loop RRT and RRT* techniques and was found to be significantly better for online planning.

A baseline performance study was conducted for 14 maneuvers in 1, 2, and 3-lane empty intersections and two maneuvers in a 2-lane intersection with a static obstacle in the middle. pRRT was found to be more efficient and faster than closed-loop RRT and RRT* techniques. pRRT showed a higher success rate in obtaining a feasible trajectory even with low bias values than the other two techniques. The average number of floating point operations performed by pRRT for generating a successful trajectory was 15 times less than that of RRT. Regarding optimality, the algorithm provides similar results to RRT* algorithm. Due to additional steps at each iteration in RRT*, the time required for RRT* to converge was found to be 100 times more than the pRRT algorithm. The convergence time and the resulting RRT and RRT* trajectories are highly random. Adding a bias factor adds some predictability to the results. Sensitivity analysis shows that a bias value of more than 10 improves over RRT while maintaining its quality to obtain alternative paths. A framework was developed to deploy any sampling-based planning algorithm for maneuvering through an unsignalized intersection

with incoming traffic. pRRT algorithm showed the best performance for avoiding the moving traffic and safely reaching its destination. The average time to complete the motion task in larger intersections and with multiple obstacles showed that pRRT and RRT scale well with the environment. But, the risks of collision using RRT were higher due to slower replanning. Modular planning, like intersections, does not require optimal trajectories. In such scenarios, getting a safe and feasible trajectory is more important. In these metrics, the pRRT algorithm was found to be the best among its counterparts. A lower bias parameter value of 10^3 ensures rapid planning and good environment exploration.

In the second part of this work, we proposed a new algorithm for the path planning of mobile robots. Even though the base algorithm works slightly slower than the standard pRRT algorithm, the paths generated by it are much better and smoother for autonomous vehicle tracking. Using the strategies from this new algorithm, the standard pRRT, and the standard RRT-Connect algorithms, we developed a new methodology for the motion planning of self-driving cars. The new method called the probabilistic RRT-Connect, allows us to use high values for the bias parameter to obtain quick plans around the static obstacles. Compared with the performance of standard RRT and pRRT algorithms, our method showed superior performance for finding a safe trajectory with less number of iterations. Adding steps to generate trajectories by selecting intermediate goals around the obstacles was found to be more efficient than randomly sampling points in the environment to form a tree from start to destination.

7.2 Future Works:

For the first part of the thesis, the algorithms were tested for low-level planning in the specific scenario of intersection. These algorithms can be tested for more generalized cases for

deploying them for overall low-level planning in autonomous vehicles. More complex vehicle models can be used instead of the kinematic bicycle model for more realistic simulations. Instead of GPR for prediction, better models could be used for accurate and longer predictions [65]. The prediction module can be experimentally implemented to test the whole end-to-end module of pRRT motion planning.

The experimental evaluations can be implemented alongside a good SLAM module for efficient tracking of the trajectory generated by the pRRT algorithm. To implement this algorithm on a full-scaled autonomous vehicle, the algorithm needs to be scaled to incorporate dynamic characteristics, like static friction and moving inertia, that play a major role in real-life situations.

For the new algorithm, the trajectories generated were tracked using a basic MPC tracker. To further improve the method, a better MPC tracker can be implemented to track the generated path more accurately. This can effectively reduce the radius of the safety circle and hence resulting in more optimal trajectories. The base algorithm developed in this work was tested on intersection environments for autonomous driving purposes. The algorithm can be tested for the purpose of path planning for mobile robots, especially for industrial robots that work in a closed environment. The method can be experimentally implemented in the scaled autonomous vehicle to validate the real-life feasibility of the algorithm. Experimentally, the ego vehicle can be subjected to more complex scenarios, and the algorithm can be evaluated based on dynamic feasibility, optimality, and closeness to human intuition.

Bibliography

[1] Federal Highway Administration (2020). "About Intersection Safety," (website) Washington, DC. Available online: https://highways.dot.gov/safety/intersection-safety/about, last updated Feb 1, 2023.

[2] Azim Eskandarian, "Handbook of Intelligent Vehicles", Springer, 2012

[3] Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles, J3016_202104, 2021 https://www.sae.org/standards/content/j3016_202104/

[4] W. Shi, M. B. Alawieh, X. Li, and H. Yu, "Algorithm and hardware implementation for visual perception system in autonomous vehicle: A survey," Integration, vol. 59, pp. 148–156, Sep. 2017, doi: https://doi.org/10.1016/j.vlsi.2017.07.007.

[5] W. Schwarting, J. Alonso-Moraand D. Rus, "Planning and Decision-Making for Autonomous Vehicles", Annual Review of Control, Robotics, and Autonomous Systems, vol. 1, no. 1, pp. 187–210, 2018, doi: 10.1146/annurev-control-060117-105157.

[6] J. Guanetti, Y. Kim, and F. Borrelli, "Control of connected and automated vehicles: State of the art and future challenges," Annual Reviews in Control, vol. 45, pp. 18–40, 2018, doi: https://doi.org/10.1016/j.arcontrol.2018.04.011.

[7] C. Zhang, A. Eskandarian and X. Du, "Attention-based Neural Network for Driving Environment Complexity Perception," 2021 IEEE International Intelligent Transportation Systems Conference (ITSC), Indianapolis, IN, USA, 2021, pp. 2781-2787, doi: 10.1109/ITSC48978.2021.9564709.

[8] A. Gupta, A. Anpalagan, L. Guan, and A. S. Khwaja, “Deep learning for object detection and scene perception in self-driving cars: Survey, challenges, and open issues,” Array, vol. 10, p. 100057, Jul. 2021, doi: https://doi.org/10.1016/j.array.2021.100057.

[9] S. Edelkamp and S. Schrödl, “Route Planning and Map Inference with Global Positioning Traces”, in Advances in Visual Computing, Advances in Visual Computing, 2003, pp. 128–151. doi: 10.1007/3-540-36477-3_10.

[10] O. Sharma, N. C. Sahoo, and N. B. Puhan, “Recent advances in motion and behavior planning techniques for software architecture of autonomous vehicles: A state-of-the-art survey,” Engineering Applications of Artificial Intelligence, vol. 101, p. 104211, May 2021, doi: https://doi.org/10.1016/j.engappai.2021.104211.

[11] S. Teng et al., ”Motion Planning for Autonomous Driving: The State of the Art and Future Perspectives,” in IEEE Transactions on Intelligent Vehicles, doi: 10.1109/TIV.2023.3274536.

[12] L. Claussmann, M. Revilloud, D. Gruyer and S. Glaser, ”A Review of Motion Planning for Highway Autonomous Driving,” in IEEE Transactions on Intelligent Transportation Systems, vol. 21, no. 5, pp. 1826-1848, May 2020, doi: 10.1109/TITS.2019.2913998.

[13] G. Bresson, Z. Alsayed, L. Yu and S. Glaser, ”Simultaneous Localization and Mapping: A Survey of Current Trends in Autonomous Driving,” in IEEE Transactions on Intelligent Vehicles, vol. 2, no. 3, pp. 194-220, Sept. 2017, doi: 10.1109/TIV.2017.2749181.

[14] ”Fatality and Injury Reporting System Tool (FIRST),” NHTSA, https://cdan.dot.gov/query

[15] L. Wei, Z. Li, J. Gong, C. Gong, and J. Li, “Autonomous Driving Strategies at Intersections: Scenarios, State-of-the-Art, and Future Outlooks”, 2021.

[16] Wu, X., Nayak, A., and Eskandarian, A., "Motion Planning of Autonomous Vehicles under Dynamic Traffic Environment in Intersections Using Probabilistic Rapidly Exploring Random Tree," SAE Intl. J CAV 4(4):383-399, 2021, https://doi.org/10.4271/12-04-04-0029.

[17] B. Paden, M. Čáp, S. Z. Yong, D. Yershov and E. Frazzoli, "A Survey of Motion Planning and Control Techniques for Self-Driving Urban Vehicles," in IEEE Transactions on Intelligent Vehicles, vol. 1, no. 1, pp. 33-55, March 2016, doi: 10.1109/TIV.2016.2578706.

[18] D. González, J. Pérez, V. Milanés and F. Nashashibi, "A Review of Motion Planning Techniques for Automated Vehicles," in IEEE Transactions on Intelligent Transportation Systems, vol. 17, no. 4, pp. 1135-1145, April 2016, doi: 10.1109/TITS.2015.2498841.

[19] C. Zhou, B. Huangand P. Fränti, "A review of motion planning algorithms for intelligent robots", Journal of Intelligent Manufacturing, vol. 33, no. 2, pp. 387–424, 2022, doi: 10.1007/s10845-021-01867-z.

[20] S. X. Yang and M. Meng, "An efficient neural network approach to dynamic robot motion planning," Neural Networks, vol. 13, no. 2, pp. 143–148, Mar. 2000, doi: https://doi.org/10.1016/s0893-6080(99)00103-3.

[21] K. Zhu and T. Zhang, "Deep reinforcement learning based mobile robot navigation: A review," Tsinghua Science and Technology, vol. 26, no. 5, pp. 674–691, 2021.

[22] S. Sivashangaran and A. Eskandarian, "Deep Reinforcement Learning for Autonomous Ground Vehicle Exploration Without A-Priori Maps," arXiv preprint arXiv:2301.04036

[23] P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa,D. Silver, and D. Wierstra, "Continuous control with deep reinforce-ment learning,"arXiv preprint arXiv:1509.02971, 2015

[24] M. Mohammadi, A. Al-Fuqahaand J.-S. Oh, "Path Planning in Support of Smart Mobility Applications Using Generative Adversarial Networks", 2018. doi: 10.1109/cybermatics_2018.2018.00168.

[25] J. Connors and G. Elkaim, "Analysis of a Spline Based, Obstacle Avoiding Path Planning Algorithm", 2007. doi: 10.1109/vetecs.2007.528.

[26] A. Scheuer and T. Fraichard, "Continuous-curvature path planning for car-like vehicles". doi: 10.1109/iros.1997.655130.

[27] J.-W. Choi, R. Curryand G. Elkaim, "Path Planning Based on Bezier Curve for Autonomous Ground Vehicles", 2008. doi: 10.1109/wcecs.2008.27.

[28] Sivashangaran, S., Patel, D., and Eskandarian, A.,2022, "Nonlinear model predictive control for op-timal motion planning in autonomous race cars,"IFAC-PapersOnLine,55(37), pp. 645–650.

[29] J. L. Vazquez, M. Bruhlmeier, A. Liniger, A. Rupenyanand J. Lygeros, "Optimization-Based Hierarchical Motion Planning for Autonomous Racing", 2020. doi: 10.1109/iros45743.2020.9341731.

[30] K. Kondak and G. Hommel, "Computation of time optimal movements for autonomous parking of non-holonomic mobile platforms". doi: 10.1109/robot.2001.933030.

[31] B. Li, "Optimization-Based Trajectory Planning for Autonomous Parking With Irregularly Placed Obstacles: A Lightweight Iterative Framework", IEEE Transactions on Intelligent Transportation Systems, vol. 23, no. 8, pp. 11970–11981, 2022, doi: 10.1109/tits.2021.3109011.

[32] X. Qian, I. Navarro, A. De La Fortelleand F. Moutarde, "Motion planning for urban autonomous driving using Bézier curves and MPC", 2016. doi: 10.1109/itsc.2016.7795651.

[33] X. Shang, A. Eskandarian, "Emergency Collision Avoidance and Mitigation Using Model Predictive Control and Artificial Potential Function," arXiv preprint arXiv:2211.06574

[34] Rachmawati, Dian and Gustin, Lysander. (2020). Analysis of Dijkstra's Algorithm and A* Algorithm in Shortest Path Problem. Journal of Physics: Conference Series. 1566. 012061. 10.1088/1742-6596/1566/1/012061.

[35] R. Kala and K. Warwick, "Multi-Level Planning for Semi-autonomous Vehicles in Traffic Scenarios Based on Separation Maximization", Journal of Intelligent & Robotic Systems, vol. 72, no. 3–4, pp. 559–590, 2013, doi: 10.1007/s10846-013-9817-7.

[36] J. Yu, J. Houand G. Chen, "Improved Safety-First A-Star Algorithm for Autonomous Vehicles", 2020. doi: 10.1109/icarm49381.2020.9195318.

[37] D. Harabor and A. Grastien, 'Online Graph Pruning for Pathfinding on Grid Maps', in Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, 2011, pp. 1114–1119.

[38] T. Siméon, J.P. Laumond, C. Nissoux (2000) Visibility-based probabilistic roadmaps for motion planning, Advanced Robotics, 14:6, 477-493, DOI: 10.1163/156855300741960

[39] S. M. LaValle, "Rapidly-exploring random trees: a new tool for path planning," The annual research report, 1998.

[40] L. E. Kavraki, P. Svestka, J.-C. Latombeand M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces", IEEE Transactions on Robotics and Automation, vol. 12, no. 4, pp. 566–580, 1996, doi: 10.1109/70.508439.

[41] Y. Kuwata, G. A. Fiore, J. Teo, E. Frazzoliand J. P. How, "Motion planning for urban driving using RRT", 2008. doi: 10.1109/iros.2008.4651075.

[42] S. Karaman, E. Frazzoli, "Sampling-based Algorithms for Optimal Motion Planning", arXiv:1105.1186

[43] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli and S. Teller, "Anytime Motion Planning using the RRT*," 2011 IEEE International Conference on Robotics and Automation, 2011, pp. 1478-1483, doi: 10.1109/ICRA.2011.5980479.

[44] A. Perez, R. Platt, G. Konidaris, L. Kaelblingand T. Lozano-Perez, "LQR-RRT*: Optimal sampling-based motion planning with automatically derived extension heuristics", 2012. doi: 10.1109/icra.2012.6225177.

[45] Q. Li, J. Wang, H. Li, B. Wangand C. Feng, "Fast-RRT *: An Improved Motion Planner for Mobile Robot in Two-Dimensional Space", IEEJ Transactions on Electrical and Electronic Engineering, vol. 17, no. 2, pp. 200–208, 2022, doi: 10.1002/tee.23502.

[46] M. Elbanhawi and M. Simic, "Sampling-Based Robot Motion Planning: A Review", IEEE Access, vol. 2, pp. 56–77, 2014, doi: 10.1109/access.2014.2302442.

[47] J. J. Kuffner and S. M. Lavalle, "RRT-connect: An efficient approach to single-query path planning". doi: 10.1109/robot.2000.844730.

[48] J. Xu, Z. Tian, W. Heand Y. Huang, "A Fast Path Planning Algorithm Fusing PRM and P-Bi-RRT", 2020. doi: 10.1109/phm-jinan48558.2020.00098.

[49] Y. Kuwata, J. Teo, S. Karaman, G. Fiore, E. Frazzoliand J. How, "Motion Planning in Complex Environments Using Closed-loop Prediction", 2008. doi: 10.2514/6.2008-7166.

[50] D. Zivojevic and J. Velagic, "Path Planning for Mobile Robot using Dubins-curve based RRT Algorithm with Differential Constraints", 2019. doi: 10.1109/elmar.2019.8918671.

[51] J. D. Gammell, S. S. Srinivasa and T. D. Barfoot, "Informed RRT: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic,"

2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2014, pp. 2997-3004, doi: 10.1109/IROS.2014.6942976.

[52] A. Yershova, L. Jaillet, T. Simeonand S. M. Lavalle, "Dynamic-Domain RRTs: Efficient Exploration by Controlling the Sampling Domain". doi: 10.1109/robot.2005.1570709.

[53] H. Liu, Q. Sun and T. Zhang, "Hierarchical RRT for humanoid robot footstep planning with multiple constraints in complex environments," 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, Vilamoura-Algarve, Portugal, 2012, pp. 3187-3194, doi: 10.1109/IROS.2012.6385836.

[54] S. R. Lindemann and S. M. Lavalle, "Current Issues in Sampling-Based Motion Planning", in Springer Tracts in Advanced Robotics, Springer Tracts in Advanced Robotics, 2005, pp. 36–54. doi: 10.1007/11008941_5.

[55] G. Mehr, P. Ghorai, C. Zhang, A. Nayak, D. Patel, S. Sivashangaran, and A. Eskandarian, "X-CAR: An Experimental Vehicle Platform for Connected Autonomy Research", IEEE Intelligent Transportation Systems Magazine, pp. 2–19, 2022.

[56] P. Polack, F. Altché, B. d'Andréa-Novel and A. de La Fortelle, "The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles?," 2017 IEEE Intelligent Vehicles Symposium (IV), Los Angeles, CA, USA, 2017, pp. 812-818, doi: 10.1109/IVS.2017.7995816.

[57] S. A. Goli, B. H. Far and A. O. Fapojuwo, "Vehicle Trajectory Prediction with Gaussian Process Regression in Connected Vehicle Environment⋆," 2018 IEEE Intelligent Vehicles Symposium (IV), 2018, pp. 550-555, doi: 10.1109/IVS.2018.8500614.

[58] C. Fulgenzi, C. Tay, A. Spalanzaniand C. Laugier, "Probabilistic navigation in dynamic

environment using Rapidly-exploring Random Trees and Gaussian processes", 2008. doi: 10.1109/iros.2008.4650959.

[59] F. Grothe, V. N. Hartmann, A. Orthey and M. Toussaint, "ST-RRT*: Asymptotically-Optimal Bidirectional Motion Planning through Space-Time," 2022 International Conference on Robotics and Automation (ICRA), Philadelphia, PA, USA, 2022, pp. 3314-3320, doi: 10.1109/ICRA46639.2022.9811814.

[60] E. Koyuncu and G. Inalhan, "A probabilistic B-spline motion planning algorithm for unmanned helicopters flying in dense 3D environments", 2008. doi: 10.1109/iros.2008.4651122.

[61] L. Palmieri, S. Koenig and K. O. Arras, "RRT-based nonholonomic motion planning using any-angle path biasing," 2016 IEEE International Conference on Robotics and Automation (ICRA), Stockholm, Sweden, 2016, pp. 2775-2781, doi: 10.1109/ICRA.2016.7487439.

[62] AASHTO (2011) A Policy on Geometric Design of Highways and Streets. The American Association of State Highway and Transportation Officials, AASHTO Green Book, Washington DC.

[63] S. Sivashangaran and A. Eskandarian, "XTENTH-CAR: A proportion-ally scaled experimental vehicle platform for connected autonomy andall-terrain research,"arXiv preprint arXiv:2212.01691, 2022

[64] D. Patel and A. Eskandarian, "Probabilistic RRT Connect with intermediate goal selection for online planning of autonomous vehicles" axRiv preprint arXiv:2305.08080, 2023

[65] A. Nayak, A. Eskandarian and Z. Doerzaph, "Uncertainty Estimation of Pedestrian

Future Trajectory Using Bayesian Approximation," in IEEE Open Journal of Intelligent Transportation Systems, vol. 3, pp. 617-630, 2022, doi: 10.1109/OJITS.2022.3205504.