Motion Planning for Autonomous Vehicles in RoadRunner

AER1516 Robot Motion Planning
Project Report
University of Toronto Institute for Aerospace Studies

Ryo Teraoka
(ryo.teraoka@mail.utoronto.ca)

Bhavishey Thapar
(bhavishey.thapar@mail.utoronto.ca)

Alexis Bruneau
(alexis.bruneau@mail.utoronto.ca)

Kevin Mano
(kevin.mano@mail.utoronto.ca)

June 19, 2023

Contents

List of Figures

List of Tables

1 Introduction

Path planning algorithms are an essential component of motion planning in the fields of robotics and automated driving. These algorithms aim to find a feasible path from a starting point to a target point while avoiding obstacles in the environment. Various approaches have been proposed to achieve this goal, such as the randomized approach using RRT[7] and the graph search approach using A* [5]and others. However, these basic approaches do not consider real-world constraints such as non-holonomic constraints and traffic rules, making them insufficient for path planning in self-driving applications. To address this limitation, our project focuses on incorporating these constraints into the path planning algorithm to make it applicable to automated driving scenarios.

The interest in automated vehicles has been on the increase due to their potential to revolutionize the transportation industry by increasing safety, reducing traffic congestion, and improving overall efficiency. Based on [14], the market for autonomous vehicles is expected to have a compound annual growth rate of 53.6% from 2022 to 2030.

As autonomous vehicles become more advanced and widespread, the demand for robust and reliable path-planning algorithms capable of navigating complex and unpredictable environments has grown exponentially [21]. Consequently, this project's focus on the development of a path-planning algorithm that accounts for real-world constraints not only addresses a pressing need in the field but also contributes to the ongoing pursuit of creating truly autonomous driving systems.

Automated driving and path planning research is a rapidly evolving field with major players spanning both academia and industry. Notable institutions include MIT [6], the University of Shanghai [2] the University of Toronto (UofT) with their aUToronto project and many others. Additionally, leading technology companies such as Tesla, and Google's Waymo are helping with research and development [10]. These organizations are actively working on advanced algorithms and solutions to address the challenges associated with path planning and other aspects of autonomous driving. Collaborations between academic institutions, research labs, and industry partners have accelerated the pace of innovation in this domain, making it an exciting and rapidly evolving field with immense potential for global impact.

In this project, we work on the development of a path planning algorithm for an autonomous vehicle as part of aUToronto's MathWorks simulation challenge. For this project, our team plans to assist the aUToronto's simulation team by designing a planner for them to compete in this year's MathWorks Simulation Challenge. The MathWorks simulation challenge requires that we use their Roadrunner simulation package to design and test our algorithm against different scenarios.

RoadRunner is an interactive editor that lets you design 3D scenes for simulating and testing automated driving systems. We plan to customize roadway scenes by creating region-specific road signs and simulating static/dynamic actors to create a test bench we can use to fulfill all the requirements of the challenge. In the past, aUToronto has used MATLAB's Driving Scenario Designer toolbox which is a 2D simulator for testing previously developed planning and control algorithms. We primarily focus on the development of the local planning algorithm with the consideration of constraints, and scene generation in Roadrunner.

We chose the A* algorithm for the global planner, which generates a path from point A to point B without considering object avoidance. For the local planner, a hybrid A* algorithm was selected that considers vehicle dynamics and can handle static obstacles using a space occupancy grid.

2 Background and Relevance

2.1 Path Planning Algorithm

Automated driving technology relies heavily on the development of safe and reliable path-planning algorithms, especially in complex and dynamic environments such as busy urban areas and highways. These algorithms must ensure that automated vehicles can travel smoothly and safely, adapting to changing conditions in real-time while complying with applicable regulations.

To achieve this, the path-planning algorithms must be capable of avoiding obstacles, staying within defined lanes, and navigating through complex scenarios. Additionally, these algorithms must be designed to be computationally efficient so that they can respond quickly to rapidly changing environments.

Efficiency is critical, as the algorithm's ability to respond quickly to changes in the environment depends on its computational speed. Thus, the algorithm must be able to move efficiently through complex environments while considering a variety of factors, such as static/dynamic obstacle avoidance and traffic rules. To achieve higher safety and reliability in complex environments, several approaches are proposed.

Over the years, researchers in the field of autonomous driving have derived various approaches for path-planning algorithms through a combination of theoretical developments, empirical experimentation, and real-world testing. Initially, the focus was on applying classical search algorithms like A* and Dijkstra's algorithm, which provided a solid foundation for understanding the basic principles of path planning. As the complexity of the driving environment increased, and the need for more efficient algorithms became apparent, researchers began to explore variants of A*, which offered a more adaptable and computationally efficient way of navigating complex search spaces [11]. Concurrently, the field also saw advances in geometric analysis approaches, like the trajectory generation in a Frenet frame, which leveraged advanced mathematical concepts to generate feasible paths that respected various constraints [18], [20]. The three main groups of algorithms presented in [11] are Randomized approaches, graph search approaches, and geometric methods.

Randomized approaches, such as the Rapidly-exploring Random Tree (RRT) [7] algorithm, generate random samples in the search space and connect them to existing structures to explore unexplored regions. These approaches are effective in finding feasible paths in complex, high-dimensional search spaces, but their performance depends on the quality of the random samples generated, and they are not always optimal In addition, the randomized approach cannot consider non-holonomic constraints as well as the other two approaches in the following.

Graph search approaches, such as A* [5], use graph structures to represent the search space and search for the optimal path from the start to the goal using heuristics. These approaches are effective in finding optimal paths in simple, low-dimensional search spaces, but they can be computationally expensive and memory-intensive in larger, more complex search spaces.

Geometric analysis approaches, such as trajectory generation in a Frenet frame [19], use mathematical models and tools to generate feasible paths that take into account constraints, such as traffic rules. These approaches are computationally efficient and easy to implement in simple, low-dimensional search spaces, but they may require advanced mathematical concepts and tools and are not suitable for complex, high-dimensional search spaces.

From the above three approaches, we picked several algorithms that we believe are effective for automated driving. We then scored each algorithm in terms of path optimality, computational efficiency, robustness, ease of implementation, and smoothness of the path as shown in the table. As a result, we found that hybrid A* and Trajectory generation in a Frenet frame were the most capable of achieving our objectives. Those two algorithms are discussed in the next section 3.

Criteria	Weight	A*	Hybrid A*	Lattice Planner	Elastic Bands	Model Predictive Control	Trajectory Generator Frenet
Path Optimality	0.15	9	8	7	5	9	8
Computational Speed	0.2	5	7	6	10	8	9
Robustness	0.1	8	7	5	10	9	7
Ease of Implementation	0.3	10	9	3	2	2	4
Path Smoothness	0.25	3	9	10	2	10	9
Total		6.9	8.25	6.15	4.85	6.95	7.15

Figure 1: Planning Algorithm Decision Matrix

2.2 Simulation

A simulation environment is important to verify the feasibility of the algorithms selected for automated driving. This is because testing these algorithms in a real car is problematic in terms of safety and cost. Therefore, it is necessary to select a simulator that can test the algorithms under conditions similar to the real environment. Although it is possible to verify algorithms using Python and MATLAB, it is extremely difficult to reproduce a situation close to reality using only Python and MATLAB, taking into account non-holonomic constraints and traffic information. Although several simulators exist that can take into account vehicle dynamics and traffic information, we used RoadRunner provided by MathWorks as our simulator. The reason for using Roadrunner is that it is one of the requirements for the MathWorks Challenge. RoadRunner is a 3D scene simulator for simulating and testing autonomous driving systems. The advantage of using RoadRunner is that it is integrated with other MathWorks tools such as MATLAB and Simulink, allowing for simulation and analysis of complex systems. In addition, map information including traffic rules and obstacle information can be obtained for route generation.

3 Literature Review

In their paper titled *Practical Search Techniques in Path Planning for Autonomous Driving* [4] Dmitri Dolgov et al. describes a practical path-planning algorithm that generates smooth paths for an autonomous vehicle operating in an unknown environment, where obstacles are detected online by the robot's sensors. Their work was motivated by and experimentally validated in the 2007 DARPA Urban Challenge. Their approach uses a variant of the well-known A* search algorithm, applied to the 3D kinematic state space of the vehicle, but with a modified state-update rule that captures the continuous state of the vehicle in the discrete nodes of A*. Unlike traditional A*, which only allows visiting centers of cells, their hybrid-state A* associates a continuous 3D state of the vehicle with each grid cell. This results in a drivable path that satisfies the non-holonomic constraints of the vehicle. The second step of their algorithm uses conjugate gradient (CG) descent to locally improve the quality of the solution. This step produces a path that is at least locally optimal but often attains the global optimum as well.

A* is a popular greedy sampling-based motion planning algorithm. It was first introduced by Hart et al [5] as part of the work on Shakey the robot. Nodes in a graph are stored in a priority-based queue. However, A* does not consider the dynamics of the vehicle, which could result in a path being infeasible due to infeasible changes in the configuration of the vehicle.

Hybrid A*, which was first proposed by Dolgov et al[3] considers the constraints of the vehicle when applying the A* algorithm[16]. It only considers nodes as successor nodes to be explored if the nodes satisfy the vehicle constraints.

The paper "Optimal trajectory generation for dynamic street scenarios in a Frenet frame" by Moritz et al.[19] proposed an algorithm for generating trajectories for autonomous vehicles in a two-dimensional Frenet Frame space. The algorithm is particularly effective in structured environments, such as highways, where pre-defined reference paths are available. The Frenet frame trajectory generation algorithm provides a localized, curvature-based approach to path planning, resulting in smoother and more natural trajectories. Additionally, the algorithm is invariant to rotation and translation, making it more robust and flexible. In other words, this approach is a potential solution for trajectory planning in autonomous vehicles, particularly in structured environments where reference paths are available.

Here, the definition of the Frenet Frame is explained. The Frenet frame is a coordinate system used in self-driving vehicles for path planning and control. Frenet coordinates use the variables s and d to represent the position of the vehicle on the road or reference path. s coordinates represent the distance along the road (longitudinal displacement) and d coordinates represent the left and right positions on the road relative to the reference path(lateral displacement).

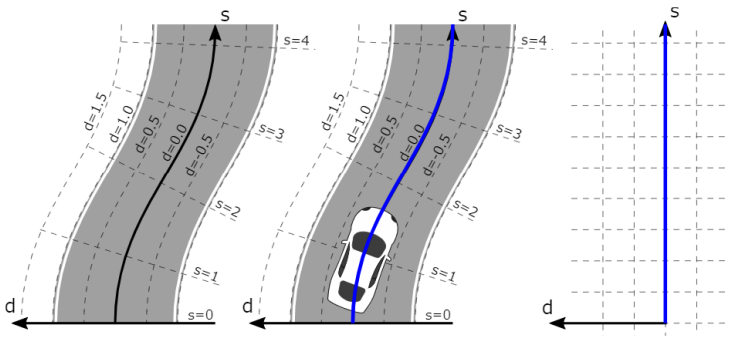


Figure 2: Frenet Coordinate System[12]

In the Frenet frame, the s coordinate represents the distance along the path starting from $s = 0$, which is the starting point of the path. The d coordinate represents the lateral position of the vehicle relative to a reference path, with positive values indicating the left side of the

reference path and negative values indicating the right side. The d coordinate conforms to local coordinates, meaning that it is perpendicular to the tangent vector at each point along the path.

As shown in the figure 2, when the car moves along the path, the d coordinate is zero, indicating that the vehicle is aligned with the reference path and represented as a straight line on the s-axis of the Frenet coordinates. However, if the vehicle deviates from the reference path, an offset is created in the d-axis direction, resulting in a non-linear movement in the Frenet coordinates.

In the paper "Optimal Trajectory Planning for Autonomous Driving Integrating Logical Constraints: An MIQP Perspective" by Bo Zhang et al, the authors introduce a novel trajectory planning approach for autonomous vehicles, considering dynamic environments, vehicle constraints, and traffic rules. Their method, based on Mixed Integer Quadratic Programming (MIQP), aims to generate optimal trajectories for autonomous vehicles. To achieve optimal trajectories, the authors incorporate a lane-centering penalty and an obstacle penalty in their formulation [13]. Although their approach relies on MIQP rather than the A* algorithm, we adapt the key concepts of applying penalties for obstacles and lane centering to enhance our path planning algorithm.

One of the key topics in path planning for autonomous vehicles is how smoothly they can avoid objects and change lanes. Several papers have proposed b-spline-based algorithms. B-splines are a type of mathematical function that can generate smooth curves by connecting a series of control points.

In their paper entitled "Obstacle Avoidance Path Planning Algorithm for Autonomous Vehicles Based on B-Spline Algorithm,"[17] the author's wang et al. proposed an algorithm for autonomous vehicles that address safety and real-time performance requirements in dynamic traffic scenarios. The proposed algorithm utilizes B-spline curves and incorporates obstacle avoidance measures to ensure the safety of the vehicle and its passengers. Furthermore, the algorithm is designed to operate in real-time, taking into account the dynamic traffic conditions that an autonomous vehicle may encounter on the road. in the paper"Planning Smooth and Obstacle-Avoiding B-Spline Paths for Autonomous Mining Vehicles", they proposed the path planning algorithm based on b-spline.In this paper, the authors report that b-spline can be used to generate smoother paths faster in complex environments such as mines, resulting in safe obstacle avoidance.

In addition, in the paper "Planning Smooth and Obstacle-Avoiding B-Spline Paths for Autonomous Mining Vehicles"[1], the authors proposed a path planning algorithm based on B-splines for autonomous mining vehicles. The authors showed that using B-splines for path planning can lead to smoother paths that can be generated faster in complex environments, such as mines. The smoother paths can result in safer obstacle avoidance and better overall vehicle control. Additionally, the authors demonstrated that their algorithm can handle constraints, such as vehicle dynamics, and can generate paths that are optimal in terms of energy consumption. In short, the use of B-splines in path planning can lead to more efficient and effective autonomous mining vehicles.

4 RoadRunner Overview

RoadRunner is an interactive editor for creating and editing driving scenarios, road networks, and 3D environments for simulating and testing automated driving systems. It integrated with the Automated Driving Toolbox in MATLAB and Simulink.

As part of the Mathworks challenge with aUToronto, we were required to use RoadRunner along with MATLAB and Simulink to build and test our motion planning algorithms for autonomous driving scenarios. In this section, a detailed overview is given of the setting up of the RoadRunner software and integrating it with MATLAB. With RoadRunner we can:

- Design complex road networks and driving scenarios using built-in road elements.
- Visualize and analyze the performance of your automated driving system in various scenarios. RoadRunner, along with the Automated Driving Toolbox, provides a comprehensive environment for designing, simulating, and testing autonomous driving algorithms in MATLAB and Simulink.
- Generate synthetic sensor data, such as camera, lidar, and radar signals, for testing our autonomous driving algorithms. Create custom 3D scenes using the built-in library of road and roadside objects, as well as import custom 3D models to create realistic environments.

RoadRunner is divided into two separate working windows, a scene, and a scenario[9]. A scene in RoadRunner refers to a virtual 3D environment that represents a driving environment, including roads, intersections, traffic signals, road markings, vehicles, pedestrians, and other objects typically found in real-world driving situations. The scene is created and edited using the RoadRunner interactive editor. The following can be done in a RoadRunner scene:

- Design road networks using built-in road elements like straight roads, curved roads, intersections, and roundabouts
- Add roadside objects, such as buildings, trees, traffic signs, and streetlights, to create a realistic environment
- Adjust lighting conditions, weather, and other environmental factors to test our algorithms in a wide range of conditions

An example of a RoadRunner scene is shown in the figure below 3.

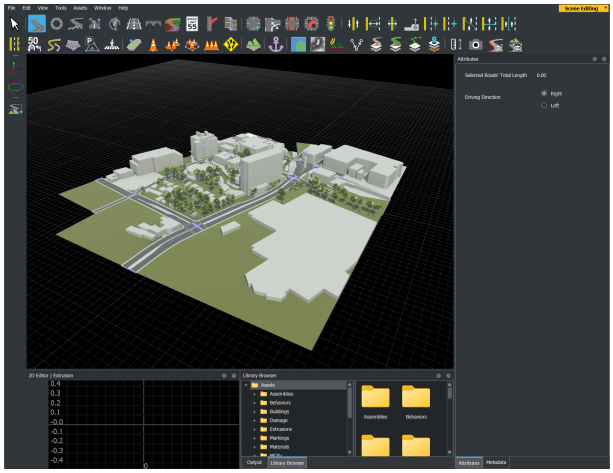


Figure 3: RoadRunner environment for generating scenes

A RoadRunner scenario is a specific driving situation or sequence of events designed to test autonomous driving algorithms and systems within a virtual 3D environment. It is created within a scene. A scenario contains a series of actions, interactions, and behaviors involving vehicles, pedestrians, and other objects in the environment. The following can be done in a RoadRunner scenario:

- Define the starting positions, trajectories, and behaviors of vehicles and pedestrians to create specific traffic situations, such as merging, lane changes, or crossings.
- Program vehicle-to-vehicle or vehicle-to-infrastructure interactions, like communication between vehicles or a vehicle responding to a traffic signal.
- Set triggers, conditions, and events that control the flow of the scenario, such as vehicle activation, pedestrian crossing, or changes in traffic light states.

An example of a RoadRunner scenario is shown in the figure below 4.

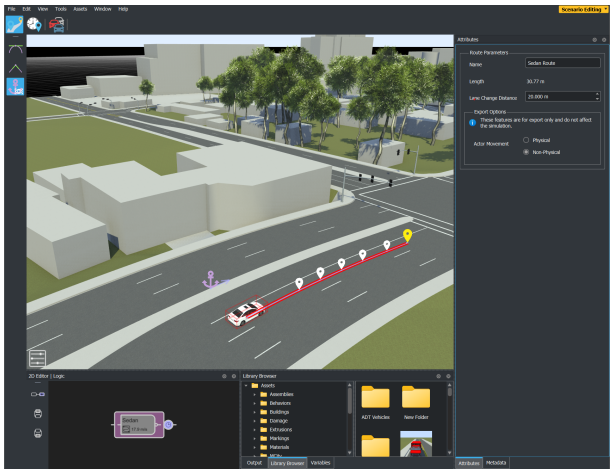


Figure 4: RoadRunner environment for generating scenarios

Vehicle behavior is modified in RoadRunner using behavior files. These files can be generated from within RoadRunner in which case the vehicle behavior is programmed using the GUI or they can be connected to a live Simulink model in which case the behavior is coded using a custom algorithm. In RoadRunner behavior refers to a set of rules, algorithms, or logic that define how an actor (such as a vehicle, pedestrian, or obstacle) interacts with the environment and other actors in a simulation. Behaviors dictate the movement, decisions, and reactions of actors in response to various conditions and scenarios during the simulation. For example, the below code snippet sets a course for the truck by reading the target path structure.

```
path_action = obj.mActorSimulationHdl.getAction('PathAction');
if(isNotEmpty(path_action))
    obj.mActor.path = path_action.PathTarget.Path;
    obj.mActor.numPts = path_action.PathTarget.NumPoints;
    obj.mActor.currPt = 1;
end
```

A lane-following behavior might ensure that a vehicle actor stays within its lane while maintaining a constant speed, whereas a path-following behavior could guide an actor group, like a truck with a trailer, to follow a specific path while keeping a constant distance between the truck and the trailer. By implementing custom behaviors in MATLAB or other programming

environments, users can create more realistic and complex scenarios in RoadRunner simulations, facilitating the development and testing of advanced driver assistance systems (ADAS) and autonomous vehicle systems.

The scene that was used to test the algorithm was provided was MathWorks as part of the challenge. Additionally, they provided a *Highway Lane Change* example [8] with Simulink blocks and a functioning planner and controller which we replaced with our planner. The Simulink block took inputs from the RoadRunner scenario to get information about the map, lanes, ego vehicle pose, and the pose data for other objects in the scenario. During the simulation, from the example, the ego vehicle changes lanes and navigates through the scenario. The Simulink model was set up as shown below.

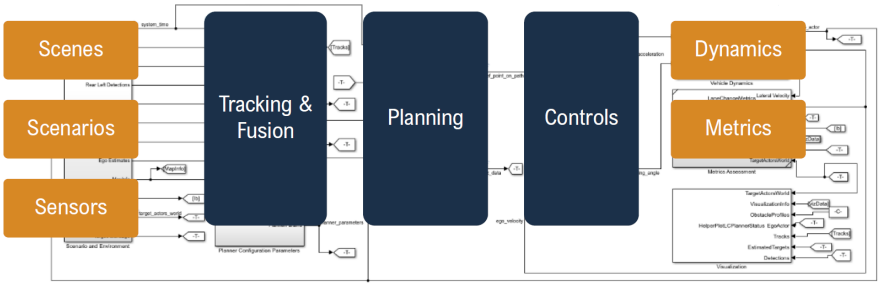


Figure 5: Simulink model of highway lane change example provided by MathWorks

As seen in the Simulink model above, for our planner, there were minor changes made to the sensors, tracking, and control blocks. Mostly, the planner was modified and also the data going into the planner block was changed to match the input accepted by the planner we worked on. To navigate the ego vehicle, the RoadRunner model requires a predefined global reference path. One of the requirements of the challenge was to determine this global path. In addition, to avoid collision with dynamic objects it was also required to develop a local planner. Additionally, the MathWorks challenge required that the planner be implemented in a RoadRunner scene created by them, this scene emulates the MCity which is a real-life testing place created by the University of Michigan to test the potential of connected and automated vehicle technologies. The real MCity and the RoadRunner scene are shown in the image below.

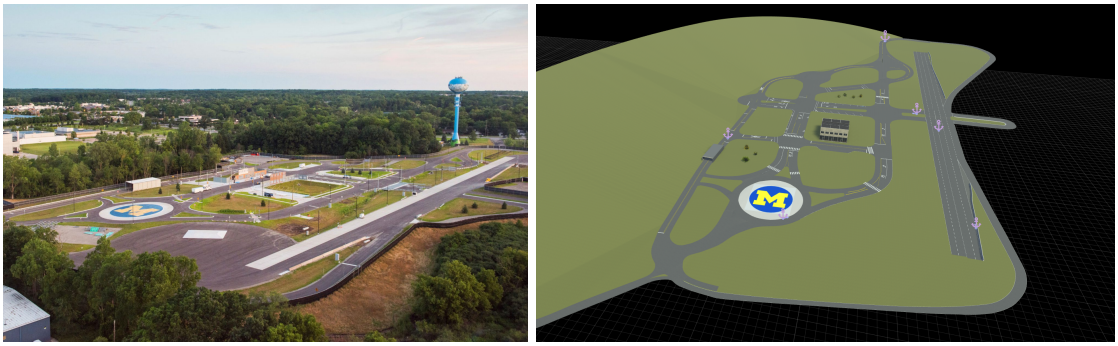


Figure 6: Real-life MCity and MCity scene in RoadRunner

5 Implementation

We present a comprehensive overview of our implementation methodology, delineating the architecture and sequential procedures involved in guiding a vehicle from point A to B.

Our approach accounts for obstacles and real-world driving constraints, such as maintaining lane centering and circumventing sharp angles. The code featured in this section has been developed using MATLAB Simulink and is tailored to interface with the MathWorks RoadRunner simulator, facilitating safe and efficient navigation in intricate environments.

For our implementation, we established a series of diverse scenarios within the MathWorks RoadRunner simulator to evaluate our model's performance. Each scenario necessitates defining a start node and goal for the ego vehicle while configuring obstacles in the environment.

Leveraging this information, our model tries to compute a collision-free path for the vehicle that adheres to real-world driving constraints, ensuring safe navigation through the designated environment.

5.1 Global Path - A*

The primary objective of our global path planning algorithm is to find the shortest route between points A and B, taking into account lane environment constraints but not considering obstacles. To accomplish this, we employ the A* algorithm, which is a widely-used graph search algorithm that determines the optimal path by evaluating neighboring nodes and selecting the node with the lowest cost, $g(n)$. The cost of reaching the goal node from the neighboring node, denoted as $h(n)$, is subsequently computed. Provided that the heuristic function does not overestimate the cost of reaching the goal, the A* algorithm returns the optimal path. The total cost of the A* algorithm is given by the function $f(n) = g(n) + h(n)$, where $g(n)$ is the cost of reaching node n from the start node, and $h(n)$ is the estimated cost of reaching the goal node from node n. It is crucial to note that the global planner was provided to us for use in a competition and the focus was mainly on the local planner.

5.2 Local Path - Hybrid A* (High-Level View)

When encountering an obstacle, the local path planner is utilized to deviate from the global path. In our implementation, we employ Hybrid A* as the local planner, a variation of A* that modifies the standard cost function, $f(n) = g(n)+h(n)$, by incorporating additional constraints and cost penalties. This modified planner aims to generate a path that adheres to the holonomic constraints of the car and follows a safe trajectory. Once the obstacles are safely avoided, the calculated path merges with the global path.

Further details on the steps involved in the local planner are discussed in subsequent sections, which include generating the occupancy grid map, addressing the non-holonomic constraints of the car, adhering to road safety measures, and merging the path with the global planner. A high-level view pseudo-code of our implementation for Hybrid A* can be found underneath.

In the Hybrid A* algorithm, the search process explores the vehicle's continuous configuration space (x, y, and orientation angle theta) while considering motion constraints. The algorithm maintains an open list of nodes to be visited and a closed list of visited nodes. At each step, the node with the lowest total estimated cost (g-value + h-value) is selected and expanded by generating its successors.

The g-value is the cost of the path taken to reach the current node from the start node. The h-value is an estimate of the cost from the current to the goal node estimated with Euclidean distance.

Successor nodes are generated by applying the vehicle's motion model and checking for collisions. If a generated successor is collision-free and satisfies any other constraints, it is added to the open list with its g-value and h-value. The g-value of a successor node is calculated by

Algorithm 1 Hybrid-A* (High-Level View)

```
function HYBRID_A*(start, goal, configuration_space, motion_model)
    open_list ← PriorityQueue()
    open_list.push(start, 0)
    closed_list ← []
    path ← []
    while not open_list.is_empty() do
        current ← open_list.pop()
        if IS_GOAL(current, goal) then
            path ← RECONSTRUCT_PATH(current)
            break
        end if
        closed_list.add(current)
        for successor in GET_SUCCESSORS(current, configuration_space, motion_model) do
            if successor not in closed_list then
                cost ← current.cost + MOTION_COST(current, successor)
                total_cost ← cost+ HEURISTIC(successor, goal)
                open_list.push(successor, total_cost)
            end if
        end for
    end while
    return path
end function
```

adding the cost of the motion taken to reach the node from the current node to the g-value of the current node. The h-value of a successor node is calculated using the same heuristic function as the h-value of the current node.

This process continues until the goal node is reached or the open list is empty. When the goal node is reached, the algorithm reconstructs the path taken to reach the node by tracing back through the parent nodes stored in each node. The resulting path is collision-free and satisfies any other constraints that were checked during the expansion of nodes.

5.3 High-Level View Process

Figure 7 offers a high-level overview of the ego vehicle's navigation process, from the start node to the goal node, as well as the interaction between Simulink and RoadRunner. The process begins with the development of a scenario in RoadRunner. After the scenario is created, the simulation starts. At the beginning of the simulation, essential information for path planning is obtained from RoadRunner.

Should an obstacle be detected on the global path, further actions are required. For each time step, the model extracts relevant information from RoadRunner, including the relative distance of obstacles concerning the ego vehicle. If there are no obstacles nearby, the ego vehicle continues along the global path until encountering an obstacle. When approaching an obstacle, a local path gets generated. The ego vehicle follows the local path until it safely avoids the obstacle.

After successfully navigating around the obstacle, the model determines if any additional obstacles remain on the global path. If no other obstacles are found, the ego vehicle can proceed along the global path to reach its goal. However, if more obstacles are encountered, the previously described loop is repeated as needed. This is achieved by merging the local path with the global path which will be discussed in section 5.6.1.

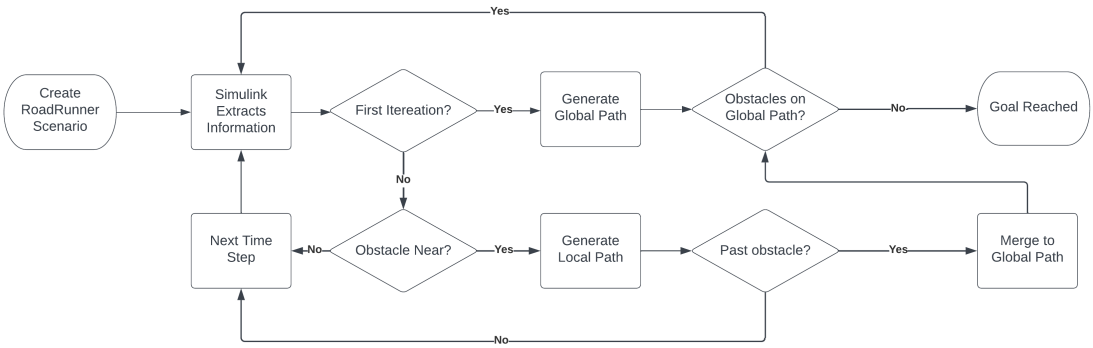


Figure 7: High-Level View Path Planning

5.4 Hybrid A* Generating Grid Map

5.4.1 GridSize

When navigating around obstacles, a local map is employed to determine an appropriate path for the ego vehicle. The initial step involves generating a grid map for computing the Hybrid-A* algorithm. Selecting the optimal size for the generated map is crucial for several reasons.

First, to closely resemble real-life scenarios, the sensors that detect obstacles have a limited range. Consequently, an excessively large grid map would not be practical in real-world applications. From a time complexity perspective, an overly large grid would be inefficient. The time complexity of the Hybrid A* algorithm is polynomial when there is a single goal state, and the heuristic satisfies the following condition: $|h(x) - h^*(x)| = O(\log h(x))$. Here, h^* represents the optimal heuristic (the exact cost from x to the goal). When employing the Euclidean distance as the heuristic, the time complexity would be $O((\log n)^2)$. This shows that increasing the number of grid cells ($n \times n$) for the occupancy matrix would significantly elevate the time complexity.[15]

On the other hand, selecting a grid size that is too small may result in inaccurate heuristic estimates and necessitate additional preprocessing to generate a feasible, non-holonomic path. Determining the optimal grid size is a vital step, and was achieved through a trial-and-error process.

5.4.2 Coordinate Transformation

For local planning, all the global points that are in the field of view of the vehicle have to convert from the global frame of reference to the local frame of reference. For this, we use a rotation matrix and only consider the 2-D pose of the vehicle. For simulation, it is safe to ignore roll and pitch since for a road vehicle they are negligible. To convert global coordinates to local coordinates for an autonomous vehicle using a rotation matrix, we need to follow these steps:

- **Define the global coordinate system:** The global coordinate system is the reference frame that we want to transform our coordinates from. In the case of an autonomous vehicle, the global coordinate system can be defined as the Earth's coordinate system (e.g., latitude, longitude).
- **Define the local coordinate system:** The local coordinate system is the reference frame that we want to transform our coordinates to. In the case of an autonomous vehicle, the local coordinate system can be defined as the vehicle's coordinate system (e.g., x, y).

- **Calculate the rotation matrix:** The rotation matrix is a 2x2 matrix that describes how the global coordinate system is oriented with respect to the local coordinate system. This matrix is calculated using the vehicle's yaw angle ϕ which represents rotation about the z-axis.
- **Use the local coordinates for navigation:** Once we have converted the global coordinates to local coordinates, we can use them for navigation purposes. For example, we can use the local coordinates to calculate the distance and direction to a target location relative to the vehicle's current position and heading

The local axes and the rotation about each of the axis are shown in the image below.

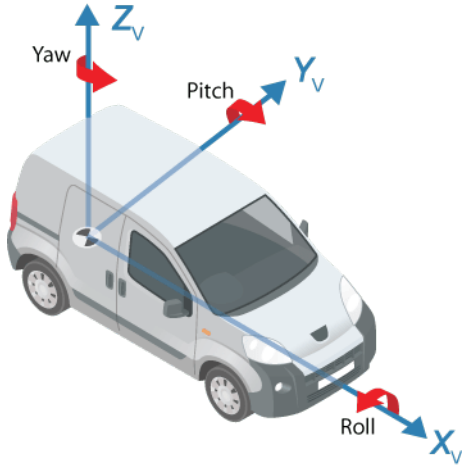


Figure 8: Autonomous vehicle's rotational axes

The rotation matrix used is shown below.

$$\begin{bmatrix} x_{local} \\ y_{local} \end{bmatrix} = \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} x_{global} - x \\ y_{global} - y \end{bmatrix} \tag{1}$$

Where (x_{local}, y_{local}) is local coordinate systems relative to the car, ϕ indicates Yaw angles of the car, (x_{global}, y_{global}) is global coordinate systems, and (x, y) is the car position in global coordinate.

5.4.3 Extracting Goal

A challenge emerges when the global goal is outside the local planner's grid search scope. To tackle this issue, we devised a technique to estimate the vehicle's desired position within the local planner's search domain. Our approach extrapolates the local target by analyzing the Frenet path, which represents the global path in Frenet coordinates. Subsequently, we pinpoint the intersection between the local grid search and the global path, which is denoted as the desired local path by a star symbol in 9 .

Although the extracted information is in Frenet coordinates, the local path calculation takes place in a coordinate system relative to the vehicle. To obtain a goal within this system, we interpolate a series of points along the local grid search's perimeter line and identify the point closest to the desired goal by calculating the Frenet distance d. The nearest point is then converted into x, y coordinates and its position relative to the ego vehicle is returned. This point serves as the goal for the local path.

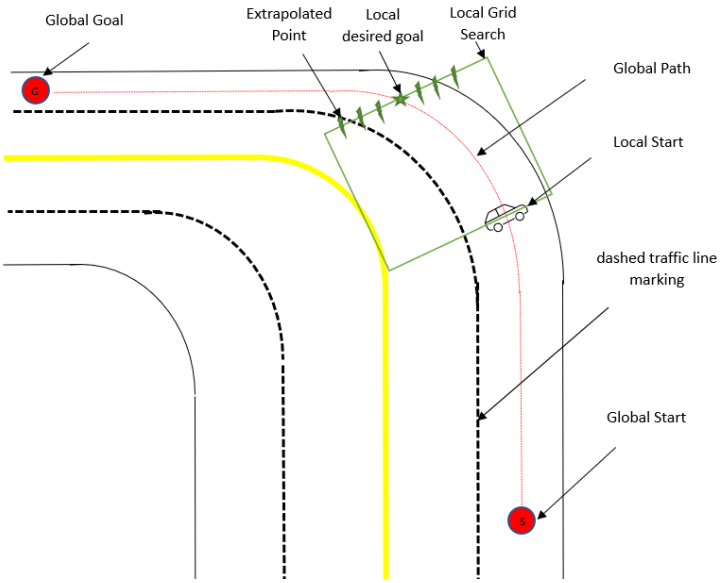


Figure 9: Extrapolating for Local Goal Node

An alternative approach involved examining a point at a specific distance s relative to the car from the global Frenet path and converting the point into x and y coordinates. However, this method presented a drawback. During sharp turns, the local path might not align with the edge of the local map due to the arc length of the Frenet path, as demonstrated in 10. The first approach ensures the local goal is situated near the edge of the local grid perimeter, thus preventing unnecessary computational power. This can be observed in the figure below, where the s value at the local grid search perimeter is larger for the road with a turn.

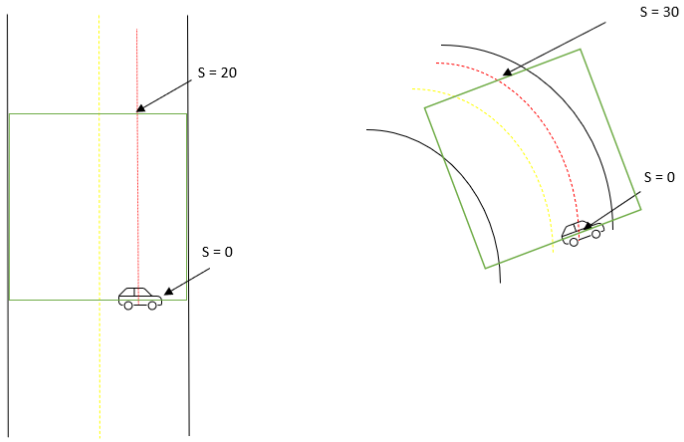


Figure 10: S value straight road vs curved road

5.5 Extracting Obstacles Information

The subsequent information needed for the grid map involves identifying obstacles to avoid. At the start of the simulation, each obstacle is assigned a unique ID. As the simulation progresses, the local map checks whether the ID exists within the local grid map. If the ID is present,

the obstacle's relative position to the car is returned, along with its width and length. This information is extracted from RoadRunner and does not need any conversion. It is important to note that all obstacles are simplified to rectangular shapes.

5.6 Hybrid A* non-holonomic constraints

5.6.1 Modified Neighbour Search

To generate a feasible motion planning path using the A* algorithm, it is essential to take into account the non-holonomic constraints of the ego vehicle. For our study, we are specifically interested in examining forward motion only. As a result, we have adapted the neighboring search process in the hybrid-A* algorithm to ensure compliance with this constraint.

In a standard A* search, the cost of traveling to all neighboring nodes is assessed for each node. However, in our modified approach, the search is limited to visiting only neighboring nodes 1 to 3 (green arrows) in 11. This restriction is imposed because nodes 3 and 5 represent translations, which are not relevant to our focus on forward motion. Additionally, nodes 6 to 8 are excluded from consideration as they correspond to the ego vehicle moving in reverse (red arrows), which is beyond the scope of our investigation. This adaptation ensures the resulting motion plan is both feasible and reflective of the real-world constraints experienced by the vehicle.

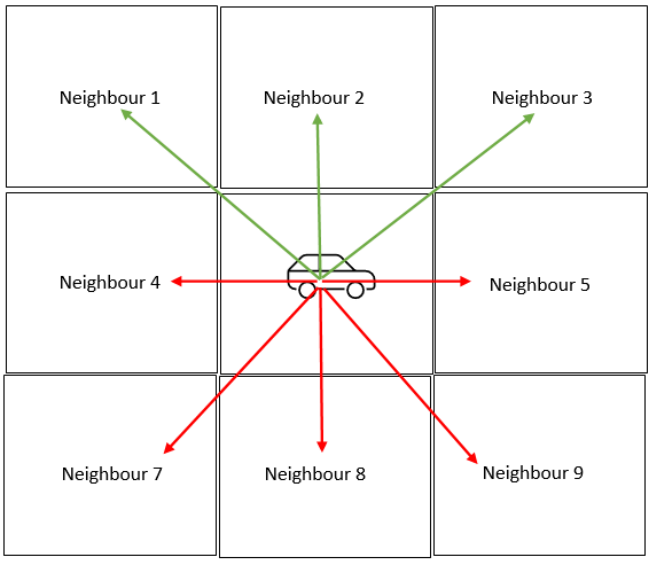


Figure 11: Modified Neigbour Search

5.6.2 Smoothing of Path

When using the A* algorithm to plan a path for a car, it produces a discrete set of waypoints that the ego vehicle should follow to reach its goal. Even after excluding neighboring search to avoid translation and reversing the generated path may include sharp angles or turns that are not feasible for the car to execute. The resulting path may still have corners or cusps that could cause jerky or discontinuous motion for the car as shown in 12.

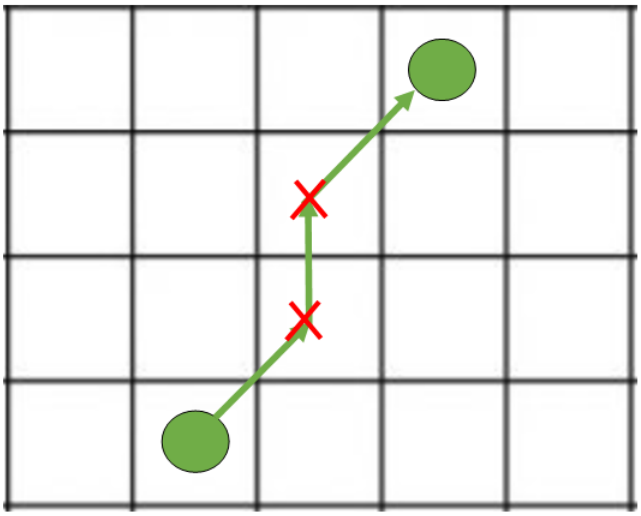


Figure 12: Sharp Angle Turns

To create a continuous and smooth path that respects the car's non-holonomic constraints, we use spline interpolation to fit a curve to the discrete set of waypoints generated by A*. Spline interpolation is a method for constructing a curve that passes through a set of points while minimizing sharp bends or changes in direction. By using spline interpolation, we can generate a smoother and more continuous path that avoids sharp angles or turns and is more feasible for the car to execute.

5.7 Safety Measures

5.7.1 Obstacles Penalty

In the given motion planning problem, the obstacle penalty term $P_{obstacle}$ is employed to penalize paths that are too close to obstacles. This penalty is determined by the distance between each node of the path and the nearest obstacle. A quadratic penalty function, together with a weight term, is applied to nodes that are within a specified threshold distance. The obstacle penalty function is given by:

$$P_{obstacle} = w_{obs} \sum_{i=1}^{N} \sigma_{obs}(|x_i - obs_i| - d_{max,obs})^2 \tag{2}$$

Here, x_i represents the ith node of the path to be penalized, obs_i denotes the location of the nearest obstacle to node x_i, $d_{max,obs}$ is the maximum distance threshold for an obstacle to influence the path cost, σ_{obs} is a quadratic penalty function based on the distance to the obstacle, and w_{obs} is a weight term adjusting the impact of the obstacle penalty.

The quadratic penalty function, σ_{obs}, is represented by:

$$\sigma_{obs}(x) = \begin{cases} (1 - \frac{x^2}{d_{max,obs}^2})^2 & \text{if } x < d_{max,obs} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Here, x stands for the distance to the nearest obstacle, and $d_{max,obs}$ is the maximum distance threshold for an obstacle to influence the path cost. The penalty function is quadratic within the range $0 \leq x < d_{max,obs}$, and zero outside of it. At $x = 0$, it reaches its maximum value of 1. The quadratic function is often used due to its smooth nature and increasing penalties as the distance to the obstacle decreases. The specific form of this penalty function can be adjusted.

5.7.2 Switching Lanes

The local planner may generate a path that involves changing lanes to avoid an obstacle, but ensuring that the ego vehicle remains centered within the lane after the lane change is crucial. Without additional steps, the planned path may position the vehicle between lanes, compromising its stability and safety. Figure 13 (a) displays a planned path that is not in the center of the lane and is too close to obstacles. This can be the case when only the obstacle penalty is implemented.

A penalty cost can be added for deviating from the center of the lane. This penalty cost encourages the planner to generate paths that keep the vehicle close to the center of the lane while still avoiding any obstacles in the path.

The lane-centering penalty cost is defined as:

$$P_{lane} = \sum_{i=1}^{N} f_{lane}(i)\sigma_{lane}(|x_i - lane_{c_{lane}}| - d_{max,lane})^2 \tag{4}$$

where $f_{lane}(i)$ is the function that varies depending on the vehicle's current lane position, $lane_{c_{lane}}$ is the center position of the lane closest to the vehicle's current position, x_i is the position of the vehicle at index i, $d_{max,lane}$ is the maximum allowed deviation from the center of the lane, and σ_{lane} is a weighting function that determines the strength of the penalty.

The purpose of the function $f_{lane}(i)$ is to adjust the penalty term based on the vehicle's current position within the lane, ensuring a balance between the obstacle penalty and the lane-centering penalty. $f_{lane}(i)$ is defined as:

$$f_{lane}(i) = e^{-k|i-c_{lane}|} \tag{5}$$

When the vehicle is in the center of the lane, the value of $f_{lane}(i)$ is close to 1, meaning that the lane deviation penalty term has full effect. As the vehicle moves away from the centerline of the lane towards the lane boundaries, the value of $f_{lane}(i)$ decreases, reducing the impact of the lane deviation penalty term. This allows the planner to generate paths that involve changing lanes if necessary while still encouraging the vehicle to stay close to the centerline of the lane. The constant k controls how quickly the penalty increases as the vehicle moves away from the lane centerline, with a higher value of k meaning that the penalty increases more quickly as the vehicle moves away from the centerline. The figure 13 (b) shows that without the term $f_{lane}(i)$, the penalty might be too strong and the path might force the car to back in the middle of the lane. Having $f_{lane}(i)$ allows the planned path to first avoid the obstacle and switch lanes, and then keep the car centered in the lane as shown in 13 (c).

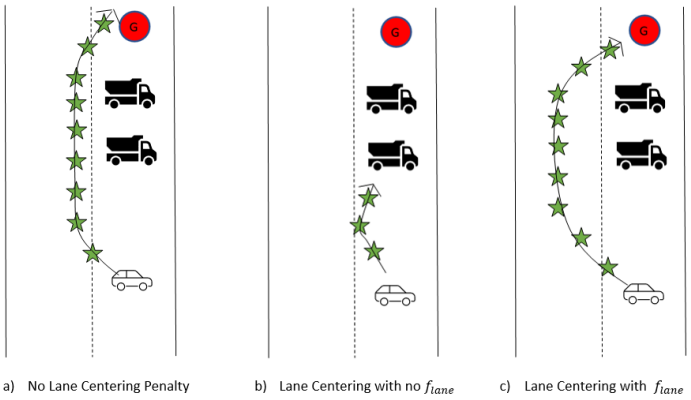


Figure 13: Effect of different penalties

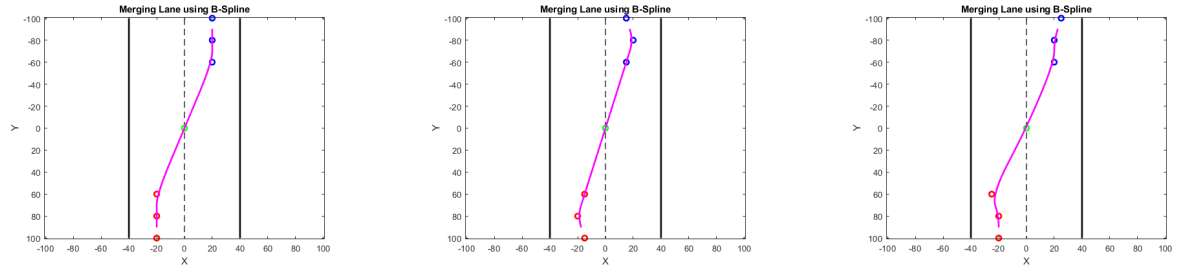
5.7.3 Total Penalty Cost

The total penalty cost serves a dual purpose: obstacle avoidance and lane centering. The obstacle penalty term $P_{obstacle}$ primarily focuses on penalizing paths that come too close to obstacles, ensuring that the vehicle safely avoids any potential hazards. Once the obstacles are successfully avoided, the lane centering penalty term P_{lane} comes into play, guiding the ego vehicle to remain close to the center of the lane. By combining these two penalty terms, the vehicle will first prioritize obstacle avoidance, and then emphasize the importance of maintaining an optimal position within the lane boundaries for a safe and stable trajectory.

5.8 Merging to Global Path

To optimize computational resources, the ego vehicle primarily adheres to the global path, generating a local path only when an obstacle is in close proximity. Once obstacles are cleared, the ego vehicle smoothly transitions back to the global planner. A merging strategy is employed to ensure a seamless and continuous path between the local and global planners, even when they occupy different lanes.

Initially, three waypoints beyond the last obstacle are chosen from the local planner (red circles in Figure 14). Next, three corresponding waypoints at specific distances from the local points are identified within the global planner (blue circles in Figure 14). The average of all selected waypoints is then calculated to guide the path (green circle in Figure 14). These points are subsequently fitted using a B-spline curve, with spline points generated through the MATLAB function 'spcrv'. The order of the B-spline curve was set to 3. The methodology described successfully created a smooth path for each scenario presented in Figure 14.



((a)) Spline Fit on Ideal Path ((b)) Spline Fit Scenario 2 ((c)) Spline Fit Scenario 3

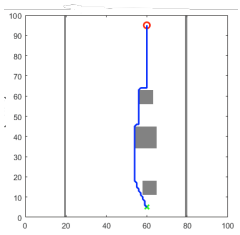
Figure 14: Visualizing B-Spline on different way points

5.9 Results

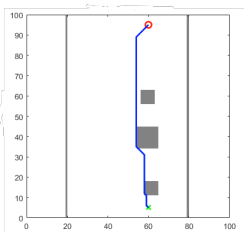
To assess the performance of the proposed algorithm, we carried out a series of experiments using a 100 x 100 grid size in MATLAB. The preliminary implementation did not include any restrictions on the neighbor search or any penalties. Nonetheless, the path, as depicted in 15(a), features sharp turns and necessitates vehicle translation, which is unrealistic for real-world scenarios. Additionally, the path's proximity to obstacles compromises safety margins, underscoring the importance of incorporating further constraints and penalties into the algorithm.

In the subsequent stage, we implemented restrictions on the visited neighbor nodes, as outlined in 5.6.1. By allowing only neighbor nodes 1, 2, and 3 to be visited, the constraint eliminates the possibility of vehicle translation or reverse movement. Figure 15(b) displays the results of this implementation, demonstrating that the translation issue is effectively addressed. Nevertheless, the generated path remains too close to obstacles, and there are instances of sharp angles that may challenge the ego vehicle's maneuverability.

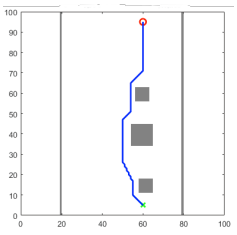
Subsequently, we incorporated the penalty term discussed in Section 5.7.1. Figure 15(c) reveals that the algorithm generates a more reasonable path that maintains a safety margin around obstacles. However, it is also evident from the figure that the path consists of numerous small turns, which could make it difficult for the vehicle to maintain its lane position.



((a)) Hybrid A Star No Restriction



((b)) Hybrid A Star Neighbour Restriction



((c)) Hybrid A Star Obstacle Penalty

Figure 15: Hybrid A* Different Versions

After incorporating the obstacle penalty, the final step involved implementing the lane penalty discussed in Section 5.7.2 and applying smoothing techniques to eliminate sharp angles. In this scenario, the grid was modified to indicate the center of each lane. Two lanes were created, each with a width of 30 grid points, with their respective centers at X = 40 and X = 60. The scenario's key parameters can be found in the table below 1.

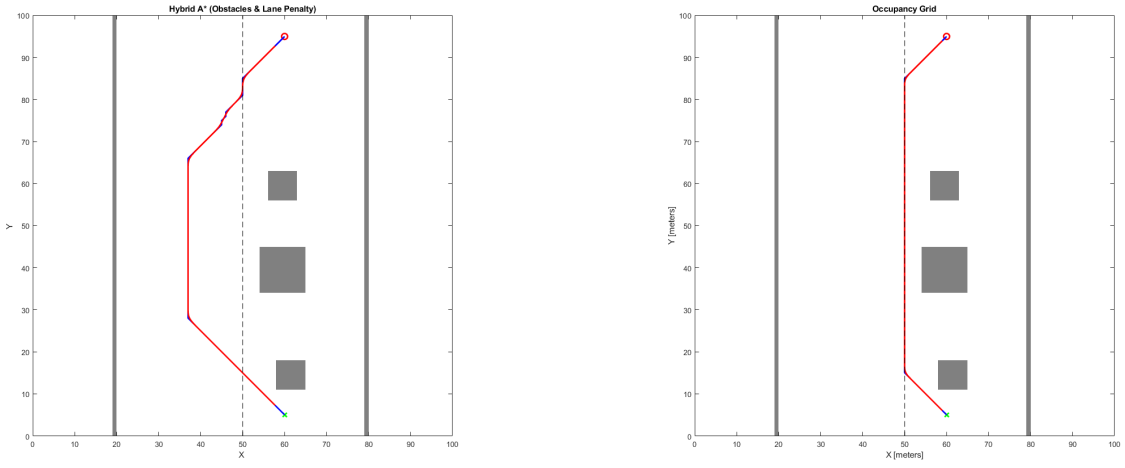
Table 1: Setup Information

Parameter	Value
Start Node	[60, 95]
Goal Node	[60, 5]
Grid Size	100
Obstacle Center	[60, 60], [62, 15], [60, 40]
Obstacle Dimension	[2, 6], [4, 6], [10, 10]
Lane Center	2 Lanes Center [40, 60]

The subsequent figure demonstrates the ego vehicle successfully avoiding obstacles and maintaining its position near the center of the lane it switches to. Once the path is generated, it is smoothed (shown in red) using the technique mentioned in Section 5.8. It is worth noting that although not depicted in Figure 16(a), the local path merges with the global path beyond the last obstacle using the B-Spline smoothing technique discussed in Section 5.8. It is important to note that multiple values for $d_{max,obs}$, w_{obs}, $d_{max,lane}$, σ_{lane}, and k were tested. A balance between the coefficient was found to have an algorithm that follows a desired path. The following table 2 showcases some of the values that were tested. The path generated for each different test can be seen in the figure underneath.

Table 2: Penalty Values by Test

Parameter	Test 1	Test 2	Test 3	Test 4
$d_{max,obs}$	27.5	5	35	15
w_{obs}	10^5	5	10^7	10^7
$d_{max,lane}$	30	10	30	35
σ_{lane}	10	10	10	10^3
k	0.05	0.02	0.025	0.5



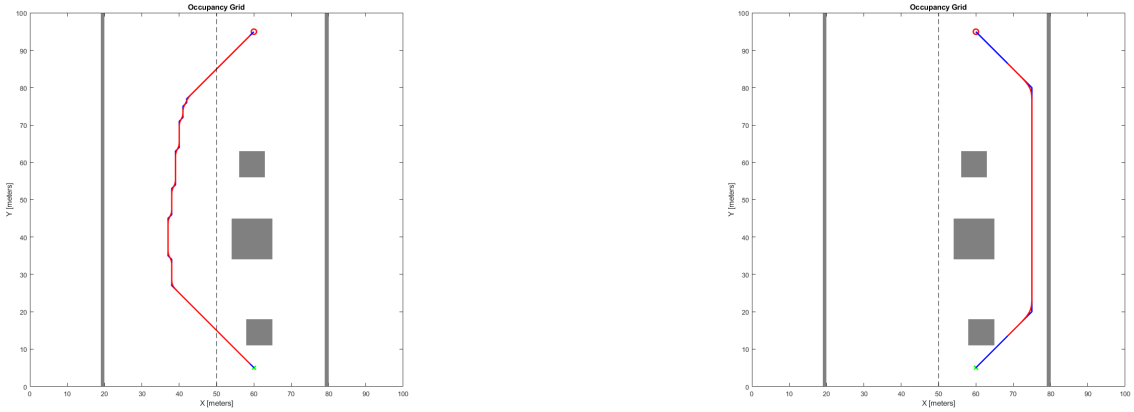
((a)) Final Result 1

((b)) Final Result 2

Figure 16: Hybrid A* for test 1 and 2

The selection of values for $d_{max,obs}$, w_{obs}, $d_{max,lane}$, and σ_{lane} can significantly impact the path generated by the Hybrid A* algorithm. However, choosing an appropriate set of values will produce a path that successfully avoids obstacles while keeping the ego vehicle centered in its lane, as demonstrated in Figure 16(a). Figure 16(b) illustrates the consequences of having

$d_{max,obs}$ and $d_{max,lane}$ set too small: the chosen path lies between the two lanes, as the algorithm finds a balance point where neither penalty affects the path. Therefore, it is crucial to set a minimum distance that penalizes the ego vehicle if it is positioned between two lanes.



((a)) Final Result 3

((b)) Final Result 4

Figure 17: Hybrid A* for test 3 and 4

In Figure 17(a), the penalty generated by the obstacles outweighs the lane penalty. The resulting path gradually deviates from the center, and subtle changes in the path occur as obstacles are altered. Lastly, Figure 17(b) shows the generated path veering to the right. In this case, the lane penalty is set too high, causing the obstacle penalty to be overpowered and prompting the ego vehicle to remain in the same lane. This behavior would be undesirable in real-world situations. The presented figures highlight the influence of each coefficient and emphasize the importance of selecting coefficients that ensure a safe and smooth path.

6 Open Problems

We worked on the implementation of a highly accurate and computationally efficient path planning algorithm for safe motion planning in complex real-world situations. However, it was challenging to fully demonstrate the feasibility of this algorithm in a real-world environment in this project. To address this, we propose several possible next steps to demonstrate the practicality of our algorithm.

6.1 Connecting MATLAB/Simulink and RoadRunner

In this project, we extracted the information needed for our planner from RoadRunner: obstacle information (location and dimensions), road lane information, and start and goal locations. We then converted this data using MATLAB/Simulink into the appropriate format and coordinate system required by our planner. However, we have encountered difficulties in connecting the transformed data in Simulink to our planner and returning the generated paths to RoadRunner. To move forward, our next step is to gain a better understanding of the Simulink data system format, including BUS, and integrate our planner into Simulink. We will also need to establish a connection between our planner and RoadRunner, which will enable us to validate our algorithm in an environment that is closer to real-world scenarios.

6.2 Determing Appropriate Grid Size

Determining the appropriate grid map size is a critical aspect of developing an efficient and safe algorithm. The size of the grid map can significantly impact the computational efficiency of the algorithm, and it plays a crucial role in safe path planning in complex and rapidly changing environments. However, setting the map size too small to increase computational efficiency may result in the algorithm being unable to accurately interpret the environment and avoid obstacles.

Therefore, it is necessary to test various grid sizes in multiple scenarios to identify the optimal map size. This process can help strike a balance between computational efficiency and safety, and ensure that the algorithm performs optimally in real-world situations. By carefully selecting the appropriate grid map size, we can achieve both computational efficiency and safety in our algorithm, thus enabling it to operate effectively in a range of environments.

6.3 Penalizing Differently Based on the Relative Obstacles Positions

Modifying the penalty term to account for the relative position of obstacles is beneficial in safety motion planning. Since vehicles may pass obstacles or other vehicles in adjacent lanes, penalizing obstacles above a node more heavily than obstacles to the side can lead to a more practical path.

The penalty function can be modified by introducing an angle term to distinguish between obstacles above and obstacles to the side.

First, the angle between the obstacle's position relative to the node and the direction of the path at that node is calculated. To find this angle, the arc-tangent function (atan2) can be used:

$$\theta_i = \texttt{atan2}(obs_i - x_i, path_{dir,i}) \tag{6}$$

Next, a new weighting term that depends on the angle $theta_i$ is introduced. A simple weighting function can be used that gives a heavier penalty when obstacles are above the node:

$$w_{angle}(\theta_i) = \begin{cases} 1 & \text{if } |\theta_i| \leq \frac{\pi}{2}\ k \\ \text{otherwise} \end{cases} \tag{7}$$

Here, k is a constant factor between 0 and 1 that controls the relative penalty for side obstacles. Lower values of k will penalize side obstacles less compared to obstacles above the node.

Finally, the penalty term $P_{obstacle}$ is modified to include a new weight term w_{angle}:

$$P_{obstacle} = w_{obs} \sum_{i=1}^{N} w_{angle}(\theta_i)\sigma_{obs}(|x_i - obs_i| - d_{max,obs})^2 \tag{8}$$

By using this modified penalty term, the hybrid A* algorithm can consider the relative position of lateral obstacles while more heavily penalizing obstacles above the node. Therefore, the next step is to implement this approach and verify whether a safer and more practical route can be obtained when overtaking obstacles and vehicles in adjacent lanes are expected.

6.4 Determining the threshold for passing obstacles

In the current algorithm, when an object is detected on the global path, the local path planner generates a path that avoids the object. After passing the object, the local path merges smoothly with the global path. The problem with this process is the determination of the threshold at which an object is judged to have been passed from the vehicle's position. If this decision is made too early, there is a risk of colliding with the object. The next step is to test multiple scenarios to determine the appropriate threshold that the autonomous vehicle can safely avoid obstacles.

6.5 Considering Traffic Light and Sign

To be effective in the real world, the path planning process for autonomous vehicles must incorporate consideration of traffic signs and signals. The next step is to develop a path planning process that takes into account traffic signals, signs, and obstacles so that autonomous vehicles can safely travel in compliance with traffic laws. This requires the development of advanced algorithms that can interpret and respond to information provided by signals and signs and integrate it with real-time data from the vehicle's surroundings to determine its path.

7 Conclusion

In conclusion, path-planning algorithms are critical for the development of safe and reliable automated driving technology. The algorithms must be efficient, and adaptable, and consider a variety of factors such as obstacle avoidance, traffic rules, and non-holonomic constraints. Various approaches such as randomized, graph search, and geometric analysis have been proposed, and hybrid A* and trajectory generation in a Frenet frame was found to be effective for automated driving. Simulation environments are essential for testing algorithms, and RoadRunner provided by MathWorks is an integrated tool that can simulate and analyze complex systems, making it suitable for testing automated driving systems. It can be integrated with the Automated Driving Toolbox in MATLAB and Simulink, making it a comprehensive environment for designing, simulating, and testing autonomous driving algorithms. RoadRunner allows for the design of complex road networks, the creation of realistic environments, the adjustment of lighting conditions and weather, and the generation of synthetic sensor data for testing autonomous driving algorithms.

For implementation, the approach chosen includes a global path planning algorithm that uses the A* algorithm to find the shortest route between two points, and a local path planning algorithm that employs Hybrid A*. The local planner generates a path that adheres to the holonomic constraints of the car and follows a safe trajectory. The overall planning process includes developing a scenario in RoadRunner, obtaining essential information for path planning, generating a local path when an obstacle is detected, and continuing along the global path after successfully navigating around the obstacle.

There were challenges faced in demonstrating the feasibility of the algorithm in the test environment. To address this, the proposed steps include integrating the algorithm into Simulink by using advanced Simulink techniques, determining and testing for the appropriate grid size, modifying the penalty term to account for the relative position of obstacles, determining the threshold for passing obstacles, and considering traffic lights and signs. These steps will enable the algorithm to perform optimally in real-world situations, ensuring computational efficiency and safety.

References

[1] Tomas Berglund et al. "Planning Smooth and Obstacle-Avoiding B-Spline Paths for Autonomous Mining Vehicles". In: *IEEE Transactions on Automation Science and Engineering* 7 (2010), pp. 167–172.

[2] Chen Chai et al. "Evaluation and Optimization of Responsibility-Sensitive Safety Models on Autonomous Car-Following Maneuvers". In: *Transportation Research Record Journal of the Transportation Research Board* 2674 (Sept. 2020). DOI: 10.1177/0361198120948507.

[3] Dmitri Dolgov et al. "Path Planning for Autonomous Vehicles in Unknown Semi-structured Environments." In: *I. J. Robotics Res.* 29.5 (2010), pp. 485–501.

[4] Dmitri Dolgov et al. "Practical Search Techniques in Path Planning for Autonomous Driving". In: *AAAI Workshop - Technical Report* (Jan. 2008).

[5] Peter Hart, Neils Nilsson, and Bertram Raphael. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107.

[6] Oguzhan Kose. "Real-world application of various trajectory planning algorithms on MIT RACECAR". In: *arXiv preprint arXiv:2109.00890* (2021).

[7] Steven M. LaValle. "Rapidly-exploring random trees : a new tool for path planning". In: *The annual research report* (1998).

[8] MathWorks. *Highway Lane Change Planner with RoadRunner Scenario*. https://www.mathworks.com/help/driving/ug/highway-lane-change-planner-with-roadrunner-scenario.html. [Accessed 18-Apr-2023].

[9] MathWorks. *RoadRunner: User's Guide*. https://www.mathworks.com/help/pdf_doc/roadrunner/roadrunner_ug.pdf. [Accessed 18-Apr-2023].

[10] Abhijit Ogale Mayank Bansal Alex Krizhevsky. "ChauffeurNet: Learning to Drive by Imitating the Best and Synthesizing the Worst". In: *arXiv:1812.03079* (2018).

[11] Brian Paden et al. "A survey of motion planning and control techniques for self-driving urban vehicles". In: *IEEE Transactions on intelligent vehicles* 1.1 (2016), pp. 33–55.

[12] Franz Pucher. *Trajectory Planning in the Frenet Space — fjp.at*. https://fjp.at/posts/optimal-frenet/. [Accessed 18-Apr-2023].

[13] Xiangjun Qian et al. "Optimal trajectory planning for autonomous driving integrating logical constraints: An MIQP perspective". In: *2016 IEEE 19th international conference on intelligent transportation systems (ITSC)*. IEEE. 2016, pp. 205–210.

[14] Grand View Research. *GVR Report cover Autonomous Vehicles Market Size, Share, & Trends Analysis Report By Application (Transportation, Defense), By Region (North America, Europe, Asia Pacific, South America, MEA) And Segment Forecasts, 2022 - 2030*. 2022. URL: https://www.grandviewresearch.com/industry-analysis/autonomous-vehicles-market (visited on 04/01/2023).

[15] Stuart J Russell. *Artificial intelligence a modern approach*. Pearson Education, Inc., 2010.

[16] Kangbin Tu et al. "Hybrid A* Based Motion Planning for Autonomous Vehicles in Unstructured Environment". In: *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2019, pp. 1–4. DOI: 10.1109/ISCAS.2019.8702779.

[17] Pengwei Wang et al. "Obstacle-Avoidance Path-Planning Algorithm for Autonomous Vehicles Based on B-Spline Algorithm". In: *World Electric Vehicle Journal* 13.12 (2022). ISSN: 2032-6653. DOI: 10.3390/wevj13120233. URL: https://www.mdpi.com/2032-6653/13/12/233.

[18] Moritz Werling et al. “Optimal trajectories for time-critical street scenarios using discretized terminal manifolds”. In: *The International Journal of Robotics Research* 31.3 (2012), pp. 346–359.

[19] Moritz Werling et al. “Optimal trajectory generation for dynamic street scenarios in a Frenét Frame”. In: *2010 IEEE International Conference on Robotics and Automation*. 2010, pp. 987–993. DOI: 10.1109/ROBOT.2010.5509799.

[20] Xuetao Xing et al. “Vehicle Motion Planning With Joint Cartesian-Frenét MPC”. In: *IEEE Robotics and Automation Letters* 7.4 (2022), pp. 10738–10745.

[21] Ibrar Yaqoob et al. “Autonomous Driving Cars in Smart Cities: Recent Advances, Requirements, and Challenges”. In: *IEEE Network* 34.1 (2020), pp. 174–181. DOI: 10.1109/MNET.2019.1900120.