Cooperative Driving of Connected Autonomous Vehicles Using Responsibility Sensitive Safety Rules

by

Harshith Allamsetti

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved June 2020 by the
Graduate Supervisory Committee:

Aviral Shrivastava, Chair
Arunabha Sen
Fengbo Ren

ARIZONA STATE UNIVERSITY

August 2020

ABSTRACT

In the recent times, traffic congestion and motor accidents have been a major problem for transportation in major cities. Intelligent Transportation Systems has the potential to be an effective solution in order to tackle this issue. Connected Autonomous Vehicles can cooperate at intersections, ramp merging, lane change and other conflicting scenarios in order to resolve the conflicts and avoid collisions with other vehicles. A lot of works has been proposed for specific scenarios such as intersections, ramp merging or lane change which partially solve the conflict resolution problem. Also, one of the major issues in autonomous decision making - deadlocks have not been considered in some of the works. The existing works either do not consider deadlocks or lack a safety proof. This thesis proposes a cooperative driving solution that provides a complete navigation, conflict resolution and deadlock resolution for connected autonomous vehicles. A graph-based model is used to resolve the deadlocks between vehicles and the responsibility sensitive safety (RSS) rules have been used in order to ensure safety of the autonomous vehicles during conflict detection and resolution. This algorithm provides a complete navigation solution for an autonomous vehicle from its source to destination. The algorithm ensures that accidents do not occur even in the worst-case scenario and the decision making is deadlock free.

ACKNOWLEDGMENTS

I would like to thank my Thesis chair, Dr. Aviral Shrivastava who has been with me through this entire journey, helping me at every moment with everything he could. I would like to also thank my committee members, Dr. Arunabha Sen and Dr. Fengbo Ren for their support.

I would also thank the Compiler Microarchitecture Lab, especially Mohammad Khayatian and Mohammadreza Mehrabian, who had helped me through my research with their valuable feedback.

Lastly, I would like to thank my parents, sister and my friends who have been with me during this complete journey providing me the constant moral support I required during the hard times.

TABLE OF CONTENTS

CHAPTER Page

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

With the increase in motor fatalities, the need for safety has become utmost important for people to travel around. According to the NHTSA report on Motor Vehicle Crashes in 2018, 36,560 fatalities have been reported during the year 2018. Many of these fatalities have been attributed to human error. In order to tackle this problem, Autonomous Vehicles could be a potential soluation. Autonomous Vehicles can not only provide safe journeys but also increase efficiency and provide more comfort to people. According to the 2019 Urban Mobility report, congestion in traffic caused Americans to travel an extra 8.8 billion hours and use an extra 3.3 billion gallons of fuel leading to an estimated cost of approximately $166 billion. They can provide efficient and reliable solutions to the problem of accidents and traffic optimization. This potential comes not only from the safety that the autonomous vehicles can ensure, but also because they perform innovative traffic control schemes. With the development of reliable and innovative technologies, a lot of progress has been made towards a zero-accident future. With the advent of communication technologies such as 5G, the vehicles will be able to communicate with each other or with road-side units with reduced delays and with more efficiency. Also, with the advancements in machine learning and Artificial Intelligence (AI), simulation environments can be created, where a complete city can be made virtual and the autonomous vehicles can be tested in the virtual environment to test rare scenarios or hardware tweaks.

Autonomy in vehicles is either done solely by the vehicle itself or by interacting with the environment around it. When the vehicles perform maneuvers solely, they make use of hardware sensors installed on them such as cameras, lidars and radars. With the help of these, they perceive the environment around them and make the corresponding decisions to drive the vehicle around. Computer Vision has had a great impact in this field of research for autonomous vehicles. Computer Vision makes use of cameras that are present on the autonomous vehicle to create 3D maps and using these maps, the autonomous vehicles can understand the surrounding better while spotting obstacles on the map and create alternate routes. This can also be used in conjunction with the LIDAR sensors in order to determine the distance of the obstacles to predict accidents and deploy safety measures appropriately. Machine Learning and Artificial Intelligence have also proven to be cutting-edge technologies for this field of autonomous driving. While this dimension of autonomous driving involves no other entities and the complete driving is performed by the vehicle itself, from the sensing till the actuation, the other dimension involves vehicles interacting with the environment around them. This can further be classified as Vehicle-to-Infrastructure(V2I), Vehicle-to-Vehicle(V2V) and Vehicle-to-Everything(V2X). A lot of research has been done on centralized traffic management(V2I), where there is a single controller and the cars submit their request and the controller responds accordingly back to the cars. While various approaches have been proposed to manage intersections, merging and round-about, these approaches are not scalable as these are operated in isolation leading to congestion in the areas not controlled by the infrastructural units and also these are not applicable to scenarios where the vehicle makes maneuvers such as overtaking or lane changing. In order to tackle such issues, we propose a method for traffic-management of connected autonomous vehicles by using Vehicle-to-Vehicle communication and incorporating safety-rules when the vehicles make appropriate decisions.

Prominent contributions related to this field include works by Dresner and Stone (2008), Hausknetch and Stone (2011, September), Rios-Torres and Malikopoulus (2016). However, all these approaches have been based on a centralized controller and also each of the methods are applicable to either an intersection scenario, or a ramp-merging scenario. Another work by Aoki and Rajkumar (2017, April) proposed a Vehicle-to-Vehicle based communication protocol where conflicts between vehicles are modelled as dynamic intersections, and an artificial traffic light is created in order to schedule the crossing of the vehicles. This results in a very low throughput as the vehicles have to stop before they have to cross the conflicts and also, it cannot be scalable as this cannot be applied where conflicts occur between multiple vehicles and also there is no deadlock resolution that occurs often in an intersection scenario.

This thesis introduces a city-wide traffic management approach for Connected Autonomous Vehicles (CAV) which uses Vehicle-to-Vehicle(V2V) communication and is applicable to various scenarios while driving and enables interactive driving maneuvers. This approach involves complete routing of a vehicle from the source to its destination, conflict detection between other vehicles on its path, conflict resolution and motion control. In this, each of the vehicles broadcast their future paths and receive the same from the other vehicles in the vicinity of the vehicle. The vehicles determine if there are any conflicts between them and if there are any, the vehicle determines which vehicle has the disadvantage. The vehicle with the disadvantage performs conflict resolution such that there would be no collision in the conflict zone. This method also incorporates safety rules such that the conflict resolution algorithm is safe even in the worst-case scenario.

In order to test the different aspects of the algorithm, MATLAB® was used to simulate the algorithm for CAVs. A driving scenario was created in the simulator with two loops of road connected by an intersection. The safety evaluation of the algorithm is done by considering an intersection and a merge scenario, where two CAVs would have a conflict in their future trajectories, and how the algorithm would provide safe trajectories for the CAVs when one of the CAVs decelerates such that there would be no collision between them. The deadlock resolution aspect provided by the algorithm is tested by considering four CAVs present at the intersection and how the CAVs detect and resolve the conflicts along with how the velocity profile of the CAVs change. A scaled RC car experiment is done in order to study the effects of communication delay and execution delay. The experiment consisted of 2 RC scaled cars that are 1/10th the size of actual cars and the algorithm is tested as to how the cars would maneuver at an intersection.

This thesis will discuss the current research that has been done in this field, the methodology proposed for city-wide traffic management, the evaluation of the algorithm and concluding with the future iterations that can be done with this work.

CHAPTER 2

RELATED WORKS

In the literature, a lot of research has been proposed related to Intelligent Transportation Systems (ITS). Prominent works have been done on managing autonomous vehicles during specific scenarios such as intersections, ramp merging, etc. Some research has been done on formulating safe trajectories for autonomous vehicles and cooperation of autonomous vehicles on a large scale.

One of the most prominent work in the field of autonomous vehicles for intersections management was proposed by Dresner and Stone (2008). In this work, each CAV sends a request to a central Intersection Manger (IM) which serves the CAVs based on a First-Come-First-Serve (FCFS) basis for crossing the intersection. Khayatian, Lou, Mehrabian and Shrivastava (2019) proposed Crossroads+ which makes the CAV's behavior deterministic despite the existence of unknown round-trip delay using timestamping and synchronization between the CAV and the IM. Khayatian, Mehrabian & Shrivastava (2018, December) proposed RIM which could handle external disturbances by allowing the CAV's to follow a feasible trajectory in order to cross the intersection. Lin et al. (2019) proposed a graph-based modelling for intersection management in order to resolve deadlocks at the intersections between connected autonomous vehicles. A lot of research had been done for ramp-merging and lane-changing scenarios. Rios-Torres and Malikopoulos (2016) formulated an optimization problem for a ramp-merging scenario Nie, Zhang, Ding, Wan, Chen, & Ran (2016) proposed a decentralized cooperative lane-changing framework for CAVs based on prediction. Researchers have also worked on management and conflict resolution of connected autonomous vehicles. Liu, Lin, Shirashi and Tomizuka (2017) proposed a distributed conflict resolution technique where an intersection is divided into conflict zones and each CAV then determines if it has a conflict with another CAV. In order to resolve the conflict, the CAV with the disadvantage changes its velocity such that there would be no collision at the conflict zone. This approach considers deadlocks between two vehicles while deadlocks may occur between more than two vehicles which is not considered in this work.

There has been considerable research towards developing safe trajectories for autonomous vehicles which is an essential part for navigation of autonomous vehicles. McNaughton, Urmson, Dolan, & Lee (2011, May) propose a spatio temporal lattice method for trajectory generation where the nodes which represent the state of the vehicle such as the position, velocity and acceleration and edges are computed using an optimization algorithm. The optimization algorithm finds the edges which denote paths connecting nodes that satisfy the dynamic constraints of the vehicle that connect the nodes and solve for a cost function for each trajectory. The graph search algorithms create trajectories based on optimization which are constrained by a cost function - the cost function does not consider more complex behaviors such as passing through vehicle's blind spots, merging traffic. Also, more better acceleration profiles need to be generated as constant acceleration can lead to execution errors.

Sampling based techniques randomly sample the free space around the vehicle in order to find a connection between the vehicle's current position and the goal position. The most commonly used Sampling based technique is RRT (Rapidly-exploring Random Tree). RRT methods for trajectory planning incrementally build a tree from the vehicle's current position to the goal position by sampling random samples from the free space around the vehicle. Each vertex of the tree represents a state and an edge represents a feasible connection between two states. Many variations of RRT have been proposed such as RRT* by LaValle & Kuffner Jr (2001), RRT-X by Otte and Frazolli (2015) to apply the RRT algorithm to robotic systems in dynamic environments. However, RRT based planners always generate jerky continuous trajectories leading to trajectories that may not be feasible to steer and it worsens as the road becomes curvy resulting in the car leaving the roadway. Also, the trajectories generated by RRT do not consider curve continuity resulting in unsafe maneuvers.

Dolgov, Thrun, Montemerlo & Diebel (2010) proposed an optimization method based on A* planning algorithm. The algorithm consists of two phases - in the first phase, a feasible trajectory is created using the A* algorithm. The second phase improves the solution by applying nonlinear optimization. Ferguson, Howard & Likhachev (2008) combined a model predictive trajectory generator algorithm with two high level planners to generate trajectories. The motion planner finds a path from the current position of the car to the final position. A set of candidate trajectory functions that follow the path are generated, then the initial parameters of the trajectory are optimized, and the best trajectory is selected based on the smoothness, end point error, velocity error. The numerical optimization methods suffer from high computation time as at every instant, the car has to optimize the trajectories, and this may not be safe for autonomous vehicles which are real time systems requiring very fast computation. Also, the cost function for the optimization formulated are not applicable for complex scenarios that autonomous vehicles are generally involved in such as merge scenarios.

Therefore, all these methods which can compute trajectories lack a safety proof and are applicable to particular scenarios. In order to account for a complete safe trajectory creation, Shalev-Shwartz, Shammah and Shashua (2017) proposed Responsibility-Sensitive-Safety model. These rules provide the minimum-safe distance that an autonomous vehicle must maintain from another autonomous vehicle in order to prevent collision among them even in the worst-case scenario. The RSS model considers the worst-case behavior of an autonomous vehicle including the reaction time for longitudinal and lateral scenarios and then provides a minimum safe distance to be maintained.

All of the related works have been either limited to particular scenarios such as intersections, lane change, etc. or do not consider deadlock scenarios or do not prove safety of the algorithm. In this thesis, we propose an algorithm to tackle all these issues which provides conflict resolution at various scenarios, deadlock resolution and does not result in collisions between autonomous vehicles even in the worst-case scenarios, thus proving safety of the algorithm.

CHAPTER 3
COOPERATIVE TRAFFIC MANAGEMENT

This chapter explains the complete working of the city-wide traffic management algorithm for connected autonomous vehicles. This algorithm consists of eight different components/modules that each autonomous vehicle should execute: 1)Global Planner 2)Broadcast of future trajectory 3)Conflict detection 4)Deadlock detection and resolution 5)Safe distance calculation 6)Conflict resolution 7)Local Planner 8)Motion Controller. The autonomous car uses these modules in each iteration of the algorithm and makes maneuvers safely while driving and avoids collisions with other vehicles. The next subsection explains an overview of the complete algorithm and subsequent subsections explain each of the above-mentioned modules in detail.

3.1 Algorithm Overview

The term "ego vehicle" is used to represent the CAV that is controlled by this algorithm. The autonomous vehicle gives the source and the destination of the current journey to the global planner. The global planner uses Dijkstra's algorithm in order to compute the shortest path between the given source and destination. The waypoints computed by the global planner is now given to the local planner which computes the reference velocity and the reference heading for the vehicle to follow in order to reach the next waypoint in its map. It then broadcasts the future trajectory that it is going to follow along with the velocity. Also, it receives the same information that is broadcasted by the other vehicles that are present in its vicinity. It then finds out if there are any conflicts between its own trajectory and the trajectory of the other vehicles. If any conflicts are present, first the vehicle determines if there is a possibility of a deadlock and if they are present, they are first resolved and then it checks who has the disadvantage over that conflict zone by using the arrival time at the conflict zone as well as safety rules described by the Responsibility-Sensitive-Safety model. If the vehicle has a disadvantage, it resolves the conflict by using the global planner to find if there are any alternate paths that the vehicle can use or lower its velocity such that there would be no conflict. This velocity is now given to the motion controller to perform the throttle and brake commands to the vehicles. This complete algorithm is done iteratively so that the vehicle can respond to dynamic conditions and it can resolve conflicts whenever there are new conflicts. The following flowchart depicts the algorithm

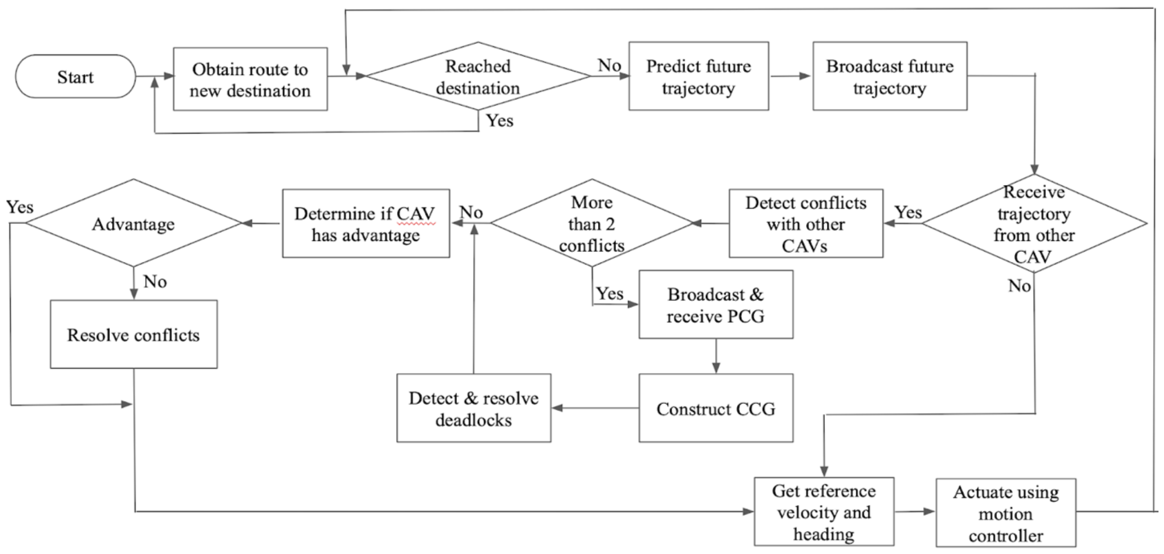
Start
Obtain route to new destination
Reached destination
No
Yes
Predict future trajectory
Broadcast future trajectory
Receive trajectory from other CAV
Yes
No
Detect conflicts with other CAVs
More than 2 conflicts
No
Yes
Broadcast & receive PCG
Construct CCG
Detect & resolve deadlocks
Determine if CAV has advantage
Advantage
Yes
No
Resolve conflicts
Get reference velocity and heading
Actuate using motion controller

Fig 1: Flowchart illustrating the algorithm

The following is the pseudocode of the algorithm –

```
[W, V] = gPlanner(map);
while has not reached the destination do
    future_traj = estimateFutureTraj();
    broadcast(future_traj);
    list_of_nearby CAVs = listen();
    if list != empty then
        tMap = Map;
        for all CAVs on the list do
            [Conflicts, PCG] = Conflict_detection();
        end
        if there are at least two conflicts then
            broadcast(PCG);
            others_PCG = listen();
            CCG = reconstruct(PCG, others_PCG);
            Conflicts = deadlock_d_&_r(CCG);
        end
        for all Conflicts do
            if ego CAV has disadvantage then
                tMap = conflict_resol(Conflicts, tMap);
            end
        end
        [W, V] = gPlanner(t_Map);
    end
    lPlanner(W, V);
end
```

Initially, the CAV uses the global planner (gPlanner) to determine the set of waypoints to follow in order to reach the desired destination. While the CAV has not reached its destination, the CAV estimates its future trajectory and broadcasts it, the CAV also listens for the trajectories of the other CAVs. If it receives any information, it first detects if it has any conflicts in the trajectories, if there are more than 2 conflicts, then it checks to see if there could be any deadlocks. So, it first constructs a Partial Conflict Graph (PCG), broadcasts it and listens for the PCG of other CAVs in order to construct the Complete Conflict Graph (CCG). It then detects and resolves deadlocks using the CCG. The CAV then determines who has the advantage over the conflict zone and if it does not have the advantage, it resolves conflicts and then actuates with the desired velocity and steering which is computed by the local planner (lPlanner).

3.2 Global Planner

The global planner is used by each of the cars in order to determine the shortest path between the source and the destination. The global planner is made up of nodes V and edges E - where the nodes V represent pre-defined waypoints on the roads and the edges E represent the route between the waypoints. Each of the nodes are given a unique ID by using the latitude and longitude of the corresponding waypoint and it is defined as <Latitude, Longitude>. An edge exists between two nodes if there is a possible route between the two waypoints. For any two nodes, there either exists only one edge if there is a possible route or there exists no edge if there is no possible route. Each of the edges are unidirectional where the direction represents the direction in which the vehicle should move in order to travel from source waypoint to the destination waypoint of that particular edge. The edges are accompanied with a weight which is based on the distance between the two nodes connected by the particular edge and the maximum velocity that the vehicle can travel in that route.

With these edges and nodes, a map is created with the waypoints as the nodes and the routes as the edges. The weight of each edge in the map represents the time it takes to travel that corresponding edge and is given by

$$w_i = \frac{d_i}{v_i} \tag{3.1}$$

where w_i represents the weight, d_i represents the length and v_i represents the maximum velocity that a vehicle can travel on that particular part of the route. Once the map is set up, the vehicle uses this map in order to find the shortest path between the source and the destination node. The global planner uses Dijkstra's algorithm in order to find the shortest path between the source and the destination considering the weight of the edges. The source node is the waypoint closest to the current position of the CAV and the destination node is the waypoint closest to the destination which the CAV intends to go to. The following is an example of a graph G - it is made of nodes V and edges E.

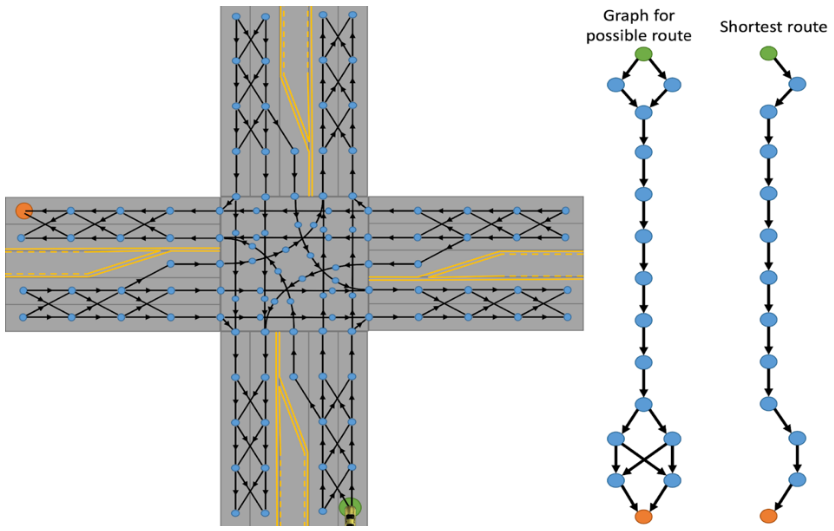

Fig 2: Graph of an intersection consisting of nodes and edges

In Fig. 1, the blue nodes represent the waypoints on the lanes of a road and the black line represent the edges with the direction of the edge representing the direction of travel. There are many possible ways with which a vehicle reaches its destination from its source like the two possible routes represented in the figure. However, the global planner uses Dijkstra's algorithm to find the shortest possible route by using the weight of the edges. The possible routes for the vehicle to move from the green waypoint to the orange waypoint is represented and the shortest route represented is calculated by the global planner.

3.3 Broadcast of future trajectory

Each CAV determines the set of waypoints to travel from the source to destination using the global planner. Then, the CAV determines the future trajectory by using the waypoints and the dynamics of the vehicle. The kinematic bicycle model is used to predict the future trajectory of the vehicle. The following equations describe the model:

$$\dot{x} = vcos(\phi) \tag{3.2}$$

$$\dot{y} = vsin(\phi) \tag{3.3}$$

$$\dot{\phi} = \frac{v}{L} tan(\psi) \tag{3.4}$$

$$\dot{v} = a \tag{3.5}$$

where x and y represent the cartesian coordinates, ϕ represents the CAV heading angle w.r.t. X-axis, v and a are the velocity and acceleration of the CAV respectively, L is the wheel-base distance and represents the steering angle of the front wheels. The acceleration and the steering angle are limited to a range of values given by: $\{a_{min}, a_{max}\}$ and $\{\phi_{min}, \phi_{max}\}$ respectively.

Each of the CAVs broadcasts its future trajectory so that the other CAV can receive this information. The length or the future horizon time for which each CAV broadcasts is dependent on the worst-case reaction time (ρ), broadcast time and worst-case stopping time of a CAV (t_b). The maximum distance until which the trajectory is broadcasted is given by the following equation:

$$d_{MAX} = v_{MAX}(\rho + 2T + t_b) \tag{3.6}$$

The worst-case braking time (t_b) is calculated as $\frac{v_{max}}{|a_{brake}|}$. Along with the future trajectory, each CAV also broadcasts the current velocity with which it is travelling and a unique ID that is provided for each one of them. This complete data is broadcasted by the CAV at each time step which is received by the other vehicles for further processing.

3.4 Conflict Detection

Once each CAV determines its own future trajectory, it receives the future trajectory of the other CAVs, it checks if there are any conflicts between the two trajectories. The conflict detection between two CAVs is done by checking if all the x-y coordinates of the future trajectories of the two CAV are less than a threshold distance d_{th}:

$$\sqrt{\left(x_i - x_j\right)^2 + \left(y_i - y_j\right)^2} < d_{th} \tag{3.7}$$

where <x_i, y_i>, *i=1,2....N* are the set of waypoints on the future path of this CAV and <x_j, y_j>, *j=1,2..M* are the set of waypoints on the future path of the other CAV. The size of the threshold distance d_{th} should be small enough such that two vehicles having waypoints on adjacent lanes should not be considered as conflicts. Each of the conflicts determined is represented by a unique ID based upon the IDs of the CAVs involved and their departure and arrival time.

3.5 Deadlock detection and resolution

Deadlock is a state in which no operations can progress. It occurs when at least two tasks wait for each other and cannot resume until the other task proceeds. A simplest example of a deadlock for autonomous vehicles can be given by an intersection scenario. Consider a situation where autonomous vehicles negotiate the right-of-way to cross an intersection by using a first-come first-served basis, when four vehicles on four different lanes reach the intersection at approximately the same time, then each of the vehicle waits for the other vehicle to cross the intersection and none proceed until determination of the priority, resulting in a deadlock. The following figure illustrates a deadlock situation at an intersection.

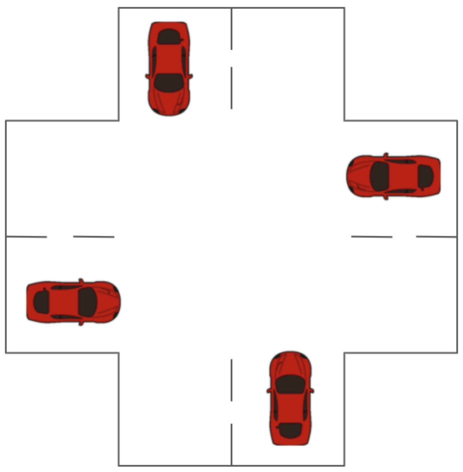

Fig 3: Deadlock scenario at an intersection based on first-come first-served principle

A more detailed example of a deadlock scenario in terms of conflict zones and the time of arrival has been illustrated below

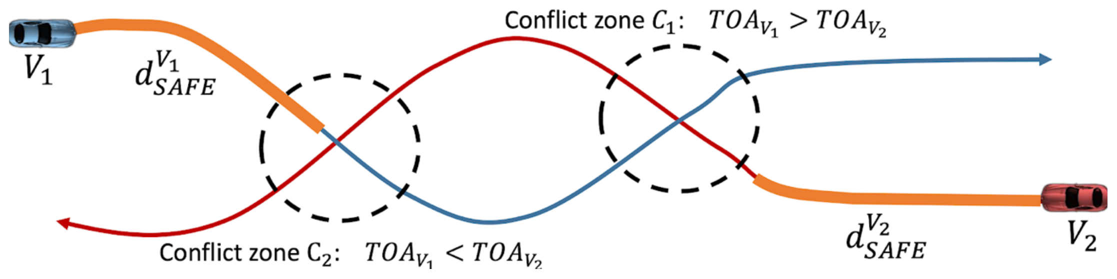
Conflict zone C_1: $TOA_{V_1} > TOA_{V_2}$
V_1
$d_{SAFE}^{V_1}$
Conflict zone C_2: $TOA_{V_1} < TOA_{V_2}$
$d_{SAFE}^{V_2}$
V_2

Fig 4: Deadlock illustration using conflict zones

In the above figure, the vehicle V_1 arrives earlier in the conflict zone C_2 and arrives later in the conflict zone C_1 compared to vehicle V_1. Likewise, the vehicle V_2 arrives earlier in the conflict zone C_1 and arrives later in the conflict zone C_2 compared to vehicle V_1. So, the vehicle V_1 has disadvantage in conflict zone C_1 and the vehicle V_2 has disadvantage in conflict zone C_2. So, each of the vehicles reduce their speed in order to maintain a safe distance from the conflict zone that they have disadvantage in, reducing each of their velocities and thus, resulting in a complete stop of both the vehicles. This represents a deadlock situation.

After a CAV detects any conflicts with another CAV, if the number of conflicts is more than two, deadlock detection and resolution is done. We use a graph-based model in order to detect and resolve conflicts similar to the method proposed by Lin et al. (2019). A dependency graph is created representing the conflicts that each vehicle has with the other vehicles. The nodes of the dependency graph are represented as $<C_i, V_j>$ where C_i represents the conflict zone and V_i represents the ID of the vehicle. There are two types of edges in the dependency graph:

1. **Type-1 edge**: A Type-1 edge represents geometrical edges and exists between two nodes, from <C_i, V_j> to <$C_{i'}, V_j$> if the CAV V_j travels from the conflict zone C_i to conflict zone $C_{i'}$.In other words, the CAV V_j reaches the conflict C_i before it reaches conflict zone $C_{i'}$.
2. **Type-2 edge**: A Type-2 edge represents advantage edges and exists between two nodes, from <C_i, V_j> to <$C_i, V_{j'}$> if CAVs V_j and $V_{j'}$ have a conflict at zone C_i. This means that the CAV V_j has to yield to $V_{j'}$ at the conflict zone C_i.

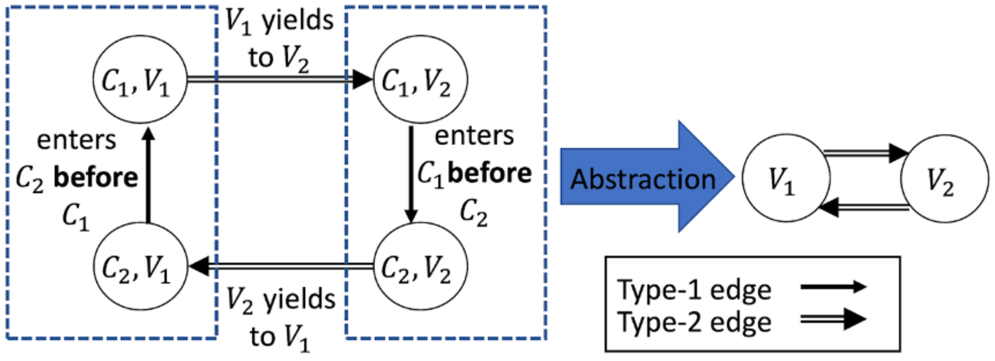

Fig 5: Dependency graph for scenario shown in Fig and the abstraction

In order to detect deadlocks, the dependency graph is abstracted to represent the dependency among the CAVs. The dependency graph is abstracted such that all the nodes of the graph having the same CAV ID are merged together into a single node. As shown in Fig 5, the Type-1 edges are geometrical edges which represent the proposed route for each CAV to travel and the Type-2 edges are advantage edges that represent which vehicle has the advantage over that conflict zone. The resulting graph is called the Partial Conflict Graph (PCG). Then, the ego CAV broadcasts its PCG so that the other vehicles can know about the dependency with each other and the ego CAV also receives the PCG from the other vehicles. Using these, the ego CAV reconstructs the Complete Conflict Graph (CCG). The following is an example which illustrates how the PCG and the CCG are constructed.

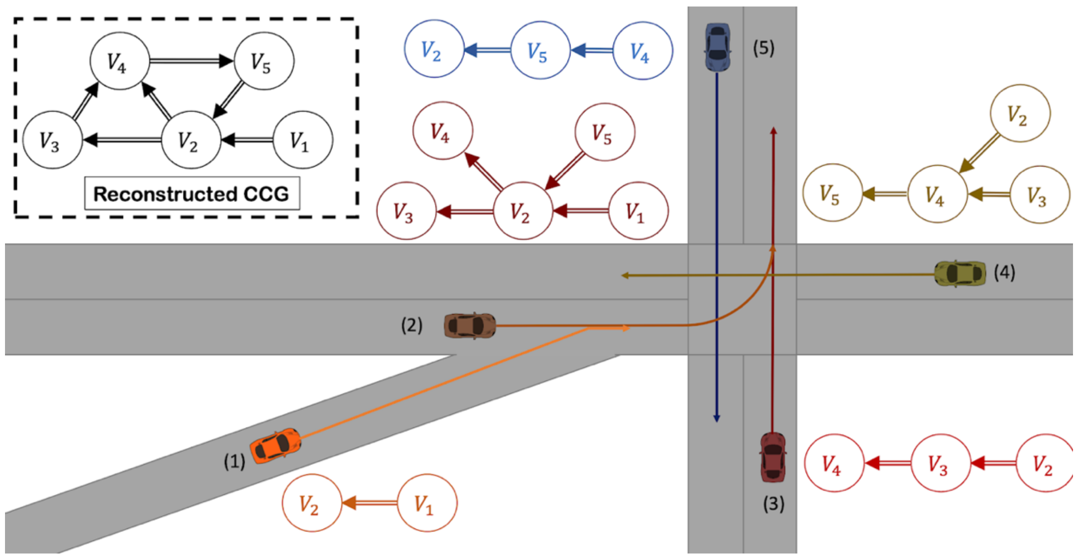

Fig 6: Illustration of PCG and CCG construction for deadlock resolution

In the above figure, the colored graphs represent the Partial conflict graph that is constructed by each of the vehicles. These are then broadcasted and received by the other vehicles. The Complete Conflict Graph (CCG) is constructed by the ego vehicle and it is represented in grayscale in the above figure. After the CCG is reconstructed, the ego vehicle checks if there is a cycle in the graph. A cycle in the graph implies that a deadlock exists and needs to be resolved. Depth-First Search (DFS) is used to check whether a cycle exists in the graph. If a cycle exists, a score is computed for each of the vehicles by using the time of arrival in the conflict zones as follows:

$$S = \frac{\sum_{i=1}^{m} TOA_i}{m} \tag{3.8}$$

where S_i represents the score for each vehicle, TOA_i represents the time of arrival of the CAV in each of the conflict zones and m represents the total number of conflict zones.

Then, all the outgoing edges for the node that represents the CAV with the least score is eliminated. The process is repeated till all the cycles have been eliminated. When all the cycles have been eliminated, it means that it is deadlock free and the order in which the vehicles should enter the conflict zone is determined from the acyclic graph.

3.6 Safe distance calculation

Once the CAV determines the set of conflicts with another CAV, the CAV determines who has the advantage. First, the CAV determines who has the earlier arrival time at the conflict zone. This is calculated using the velocity of the two vehicles. If both the vehicles have the same arrival time, the CAV with the lower ID is given advantage.

Then, the CAV with the disadvantage is at a safe distance from the CAV with the advantage. For this, we use the worst-case behavior of the vehicles similar to the Responsibility-Sensitive-Safety rules. This algorithm uses a modified version of the RSS model which projects 2-dimensional paths and scenarios into a single dimension trajectory. The RSS model has different rules for the calculation of safe distance under different scenarios such as when the two vehicles are in the same lane, when the vehicles are in adjacent lanes, for intersections and merging scenarios. So, a single general rule has been defined that can be applied to all these scenarios without losing the notion of safety. The general rule is defined first and how the general rule is applicable for all other scenarios is explained later.

3.6.1 General rule for safe distance

Consider two CAV's that have conflicts on their paths, without loss of generality, it is assumed that one of the CAV's has an advantage and the other CAV has a disadvantage. The stopping distance of the vehicle with the advantage and disadvantage is given by d_{start}^{A} and d_{stop}^{D} respectively. The stopping distance of the CAV with the advantage is defined as:

$$d_{stop}^{A} = \frac{v_A^2}{2|a_{brake}|} \tag{3.9}$$

where v_A is the velocity of the vehicle with the advantage and a_{brake} is the deceleration of the vehicle. The stopping distance of the CAV with disadvantage is defined as:

$$d_{stop}^{D} = v_D\rho + \frac{1}{2}a_{ACC}\rho^2 + \frac{(v_D + a_{ACC}\rho)^2}{|2a_{brake}|} \tag{3.10}$$

where v_D is the velocity of the vehicle with disadvantage, ρ is the communication delay, a_{ACC} is the acceleration of the vehicle and a_{brake} is the deceleration of the vehicle. The distance of the vehicle with the advantage from the start of the conflict zone is denoted as d_{start}^{A} and the distance of the vehicle with the advantage from the end of the conflict zone is defined as d_{end}^{A}. If the distance of the CAV from the end of the conflict zone is greater than the stopping distance of the CAV with the disadvantage, it may stop in the conflict zone. So, the CAV with the disadvantage must maintain a safe distance from the conflict zone given by:

$$d_{SAFE} = d_{stop}^{D} - \left(d_{stop}^{A} - d_{begin}^{A}\right) + \frac{VL_A}{2} + \frac{VL_D}{2} \tag{3.11}$$

Even in the worst case, the vehicle with the disadvantage travels a distance equal to $d_{stop}^{A} - d_{begin}^{A}$ inside the conflict zone, then the vehicle will have enough distance to travel inside the conflict zone before coming to a full stop. The two terms $\frac{VL_A}{2} + \frac{VL_D}{2}$ have been added to the equation since all the distances are assumed to be measured from the center of the vehicle and these two terms compensate for the length of the vehicles. If the distance of the CAV with the advantage from the end of the conflict zone is less than the stopping distance, the vehicle with disadvantage need not maintain any safe distance because even in the worst-case scenario, the vehicle with the advantage will stop outside the conflict zone. The safe distance for the vehicle with the disadvantage is given by:

$$d_{begin}^{D} \geq 0 \tag{3.12}$$

This is because even when the CAV with advantage decelerates, it will stop outside the conflict zone and so, does not cause a conflict with the CAV having disadvantage. Now, the general rule is applied to various scenarios and explained how it is applicable.

3.6.2 Same lane scenario

In the scenarios where the two vehicles are in the same lane, the vehicle which is in front will have the advantage and the vehicle that is behind will have the disadvantage. The conflict zone starts from the current position of the vehicle with the advantage and so d_{begin}^{F} is zero. So, the safe distance for the vehicle with the disadvantage is given by the equation:

$$d_{begin}^{R} \geq d_{stop}^{R} - d_{stop}^{F} + \frac{VL_F}{2} + \frac{VL_R}{2} \tag{3.13}$$

where d_{stop}^{R} is the stopping distance of the rear vehicle, d_{stop}^{F} is the stopping distance of the front vehicle VL_F and VL_R are the length of the front and the rear CAV respectively.

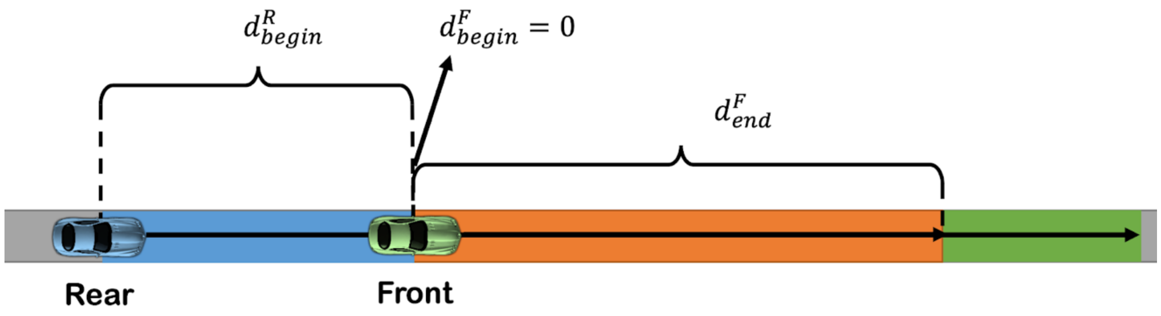

Fig 7: Safe distance illustration for two vehicles on a same lane

3.6.3 Intersection scenario

Consider an intersection scenario where two vehicles are approaching the intersection. Since vehicle 1 is closer to the intersection, it has an advantage over vehicle 2. If the vehicle 1 stops outside the conflict zone, the vehicle 2 need not maintain any safe distance because even in the worst-case scenario, the vehicle 1 stops outside the conflict zone. But, if vehicle 1 stops inside the conflict zone, the vehicle 2 must maintain a safe distance given by the equation:

$$d^2_{begin} \geq d^2_{stop} - (d^1_{stop} - d^1_{begin}) + \frac{VL_1}{2} + \frac{VL_2}{2} \tag{3.14}$$

In the worst case if vehicle 1 stops at the end of the conflict zone, vehicle 2 will have enough distance to stop safely without hitting vehicle 1.

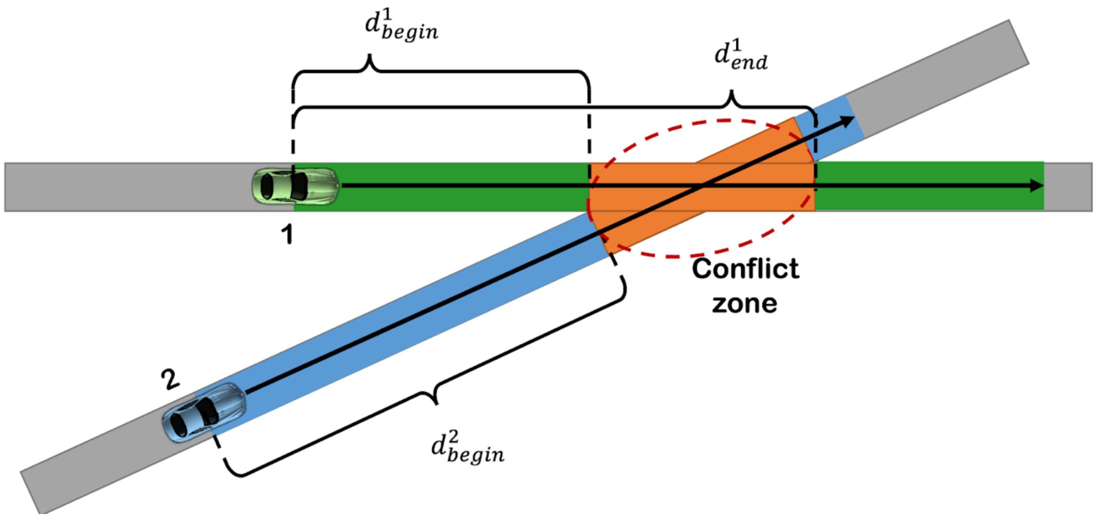

Fig 8: Safe distance illustration for two vehicles at an intersection

3.6.4 Merging scenario

Consider a road merging scenario where two vehicles are approaching the merge point. Since green vehicle is closer to the conflict zone, it has an advantage over the blue vehicle. If the green vehicle stops inside the conflict zone, then the safe distance that should be maintained by the blue vehicle is given by the equation:

$$d^D_{begin} \geq d^D_{stop} - (d^A_{stop} - d^A_{begin}) + \frac{VL_A}{2} + \frac{VL_D}{2} \tag{3.15}$$

After the vehicle merge, the two vehicles will still have conflicts but the vehicle with the disadvantage will now determine the safe distance based on the rules for the same lane scenario.

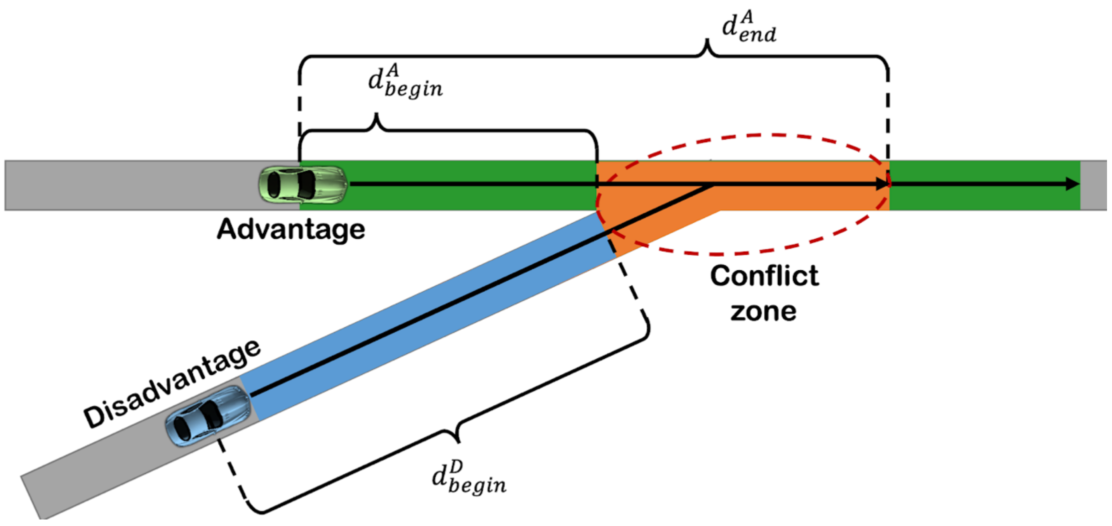

Fig 9: Safe distance illustration for a merge scenario

3.7 Conflict resolution

Once a conflict has been detected and the CAV finds that it has the disadvantage, it resolves the conflict by one of the following three ways:

1. If an alternate path exists and there is no conflict with any other CAV on that path exists.
2. If no alternate path exists, then the vehicle with the disadvantage will lower its velocity such that the new velocity satisfies the safe distance conditions.

The CAV with the disadvantage determines the safe velocity. The safe velocity is determined by using the safe distance as well as the 3 second rule as determined by NHTSA. The safe velocity based on safe distance calculated is given as follows:

$$v_{SAFE} = \frac{d^D_{begin} - d_{SAFE}}{\frac{d^A_{begin}}{v_A}} \tag{3.16}$$

This ensures that the vehicle with the disadvantage reaches the conflict zone such that it maintains a safe distance from the vehicle with advantage. This velocity provides an upper bound for the velocity assignment. However, there may be situations where the vehicle with the advantage is too close to the conflict zone, which makes the denominator in the above equation to be very low, making the safe velocity pretty high, in order to deal with these situations, the CAV with the disadvantage also determines a safe velocity based on the 3 second rule. This is given as:

$$v_{3s} = \frac{d_{begin}^{D}}{\frac{d_{begin}^{A}}{v_A} + 3} \tag{3.17}$$

This ensures that the vehicle with the disadvantage reaches the conflict zone 3 seconds after the vehicle with the advantage has reached. The new velocity to be assigned is the minimum of the two safe velocities computed above.

$$v_{assigned} = \min(v_{SAFE}, v_{3s}) \tag{3.18}$$

In order to accomplish the conflict resolution, the ego CAV uses a copy of the map and updates the weight of the edges as follows:

1. For all the conflicts that the ego CAV has, let P_{others} denote the set of nodes that are present on the paths of the other vehicle, the edges connecting these nodes are updated with a weight given by:

   $$w_i = \max\left(\frac{d_i}{v_{assigned}}, w_i\right) \tag{3.19}$$

   where d_i is the distance of the edge and $v_{assigned}$ is the velocity assigned to the CAV according to equation 3.18. The weight of the edges is only updated using the newly assigned velocity only if the assigned velocity is less than the reference velocity, else the CAV can move with the reference velocity as this would not lead to a collision at the conflict zone.

2. For all the conflicts that the ego CAV has, it determines the set of nodes that are present on its path and those that are within the safe distance from the conflict zone, then the weights of these edges are updated according to the equation 3.19

The weight updating in Part 1 accounts for the presence of other CAVs and the weight updating in Part 2 ensures that the ego CAV maintains a safe distance from the conflict zone. The following figure shows how the weights are updated after a CAV determines it has the disadvantage.

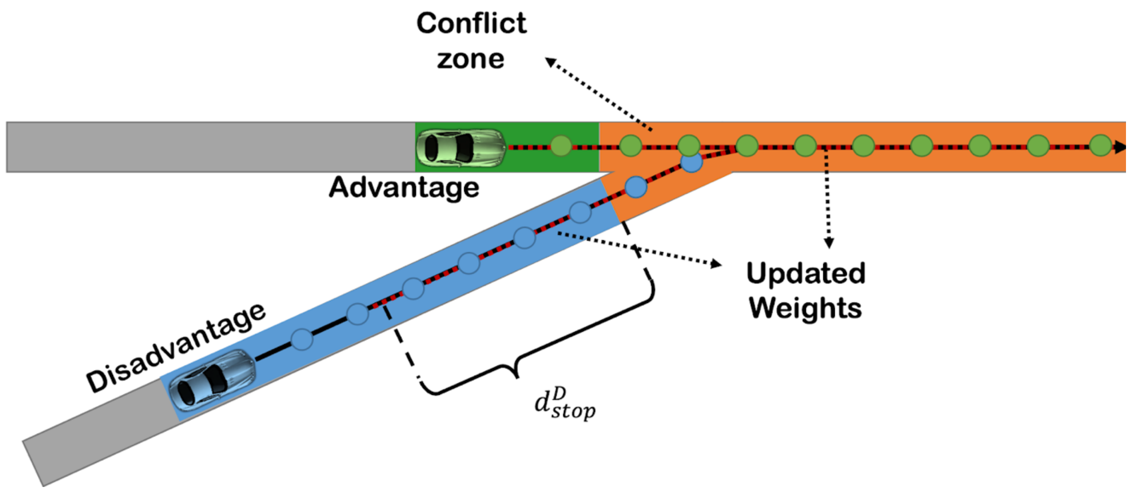

Fig 10: Weight update of the edges of the graph

The blue CAV has the disadvantage while the green CAV has the advantage. The blue path represents the future trajectory of the blue CAV while the green path represents the future trajectory of the green CAV. The conflict zone between the two vehicles is given by the orange path. Since, the blue CAV has the disadvantage, the nodes that are present on the path of the green CAV are updated which accounts for Part 1 of the weight updating and the nodes that are present within the safe distance of the conflict zone on the path of the ego CAV are also updated which accounts for Part 2 of the weight updating as explained above.

3.8 Local Planner

The set of waypoints given by the global planner determine the complete route that the CAV has to follow in order to reach the destination of the current journey. The local planner uses the future waypoints in order to determine the current heading and current velocity for the CAV to follow at the current timestep. At each timestep, the local planner is called upon by the CAV to determine the reference heading and a reference velocity. In order to determine the reference heading and reference velocity, a look-ahead point is first determined similar to the pure-pursuit algorithm.

The pure-pursuit algorithm is a tracking algorithm that works by calculating a curvature between the current position of the vehicle to some goal position. The algorithm chooses a look-ahead point that is at a fixed distance from the vehicle along the current path. The look-ahead point changes as the CAV travels along the path. At each time step, the reference heading is calculated using the coordinates of the lookahead point. If the coordinates of the look ahead point is denoted by (x_i, y_j) and the vehicle's current position by (x, y), then the reference heading for the vehicle is given by:

$$\theta_{ref} = \arctan 2\,(x - x_l, y - y_l) \tag{3.20}$$

This reference heading is used by the vehicle to steer towards the next waypoints that lie on its path. For finding the desired velocity, the local planner selects the next m waypoints that are within the safe distance d_{SAFE} where the distance to each waypoint is calculated as:

$$d_i = d_{i-1} + \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2} \tag{3.21}$$

where, d_i is the distance from the previous waypoint and $d_1 = 0$. Once the set of waypoints are determined, the local planner checks if the velocity at all the future edges is equal to v_{max}, else it will set the velocity using the weight of the edges $\frac{d_i}{w_i}$. So, the local planner takes in the set of waypoints that the CAV has to follow and the current position of the CAV. Then, it determines what should be the current reference heading and current reference velocity that the CAV has to follow and sends it to the motion controller to apply the appropriate throttle and brake commands to the motion controller.

3.9 Motion Controller

Once the desired velocity and steering are computed by the local planner, they are given to the Proportional-Integral-Derivative controller to determine the steering angle and acceleration for the vehicle which are given by the following equations:

$$\psi = k_P e_\theta + k_I \int e_\theta + k_D \dot{e_\theta} \tag{3.22}$$

$$a = k'_P e_v + k'_I \int e_v + k'_D \dot{e_v} \tag{3.23}$$

where the constants k_P, k_I, k_D and k'_P, k'_I, k'_D are tuned such that a smooth trajectory is obtained for the vehicle. The acceleration can be converted to throttle and brake signals for the vehicle to actuate on.

CHAPTER 4

SAFETY PROOF

This chapter provides a proof that shows that the algorithm produces safe trajectories even in the worst-case scenario. The algorithm is shown to produce safe trajectories by proving that

1. When the two CAV's first detect a conflict, the CAV with the disadvantage has enough distance to react and slow, without causing a collision.
2. After detecting and resolving a conflict, the CAV with disadvantage produces proper response without causing an accident.

The following assumptions are applied while explaining the safety proof of the algorithm:

1. There is no packet drop while communication information with other CAVs.
2. The information that is broadcasted by the CAVs is not compromised and is correct.
3. The broadcast delay is bounded and less than a threshold T.
4. The tracking error of the waypoints by the CAVs is always less than a threshold.

For the first part of the proof, consider a case where two CAVs are going to have a conflict in their future trajectory. The CAV that will have disadvantage at the conflict zone is at a distance $d_{max} + \epsilon$ and it is travelling with a velocity v_{max}. This is taken in order to account for the worst-case scenario. Initially, the CAV would not detect a conflict as the conflict zone is at a distance more than d_{max}. In the next iteration of the algorithm, the CAV would have travelled a distance $v_{max}(\rho + 2T)$, so the remaining distance would be

$d_{max} + \epsilon - v_{max}(\rho + 2T)$, which would be equal to $v_{max}\left(\frac{v_{max}}{|a_{min}|}\right) + \epsilon$, determined from equation 3.6. This is more than the safe distance to stop, which is $v_{max}\left(\frac{v_{max}}{2|a_{min}|}\right)$. So, the CAV with the disadvantage has enough distance to stop even in the worst-case scenario. If the initial distance is less than d_{max}, the CAV would have detected the conflict in the first iteration itself and still have enough distance to stop.

In the second part of the proof, consider the case where two CAVs detect a conflict and the CAV with the disadvantage has to produce a response. According to the conflict resolution algorithm, if the CAV finds an alternate path to resolve the conflict, this would automatically lead to a safe trajectory as the two vehicles would no longer have a conflict. If the CAV does not find a path, then the CAV's velocity is updated according to Eq. Since, the velocity is always less than v_{SAFE}, the distance to the conflict zone is always be greater than d_{SAFE} and hence, the CAV with the disadvantage has enough distance to stop.

CHAPTER 5

EXPERIMENTAL RESULTS

The algorithm is evaluated by simulating a driving environment on the MATLAB® simulator. The driving scenario consists of two loops where the larger loop consists of two lanes per road and the smaller loop consists of single lane per road. The loops are connected by an intersection and so a merge and a fork point are created. The following figure shows the driving scenario:

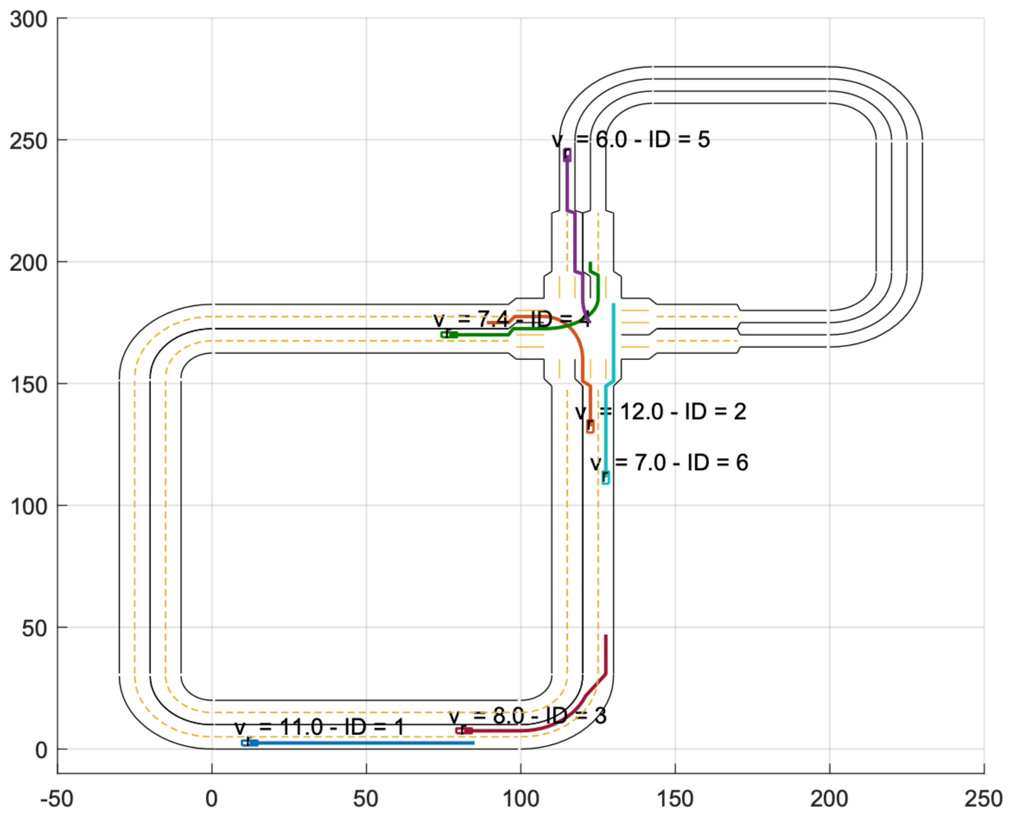

Fig 11: A snapshot of the driving scenario from MATLAB® simulator

5.1 Safety Evaluation

In order to test the safety of the algorithm, a situation where two CAVs would have a conflict in their future trajectory is considered. The CAV with the advantage decelerates and stops and so, different times at which the car would decelerate is considered. It is shown that there would be no collision at an intersection and a merge scenario when the CAV with the advantage would decelerate and come to a stop. The reaction time and broadcast delay are modelled by picking a random delay sampled uniformly between [0, ρ] where ρ is set to 0.1s and the broadcast delay *T* is set to 0.1s

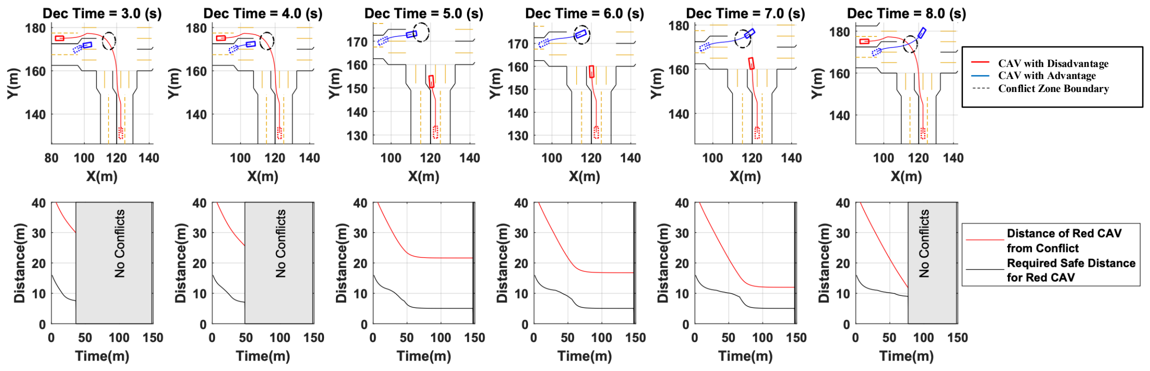

Fig 12: An intersection scenario where two CAVs have conflict

The top row of Fig 12 shows how the trajectories of the two CAVs change when the deceleration time of the blue CAV changes. The dotted box represents the initial position of the CAVs and the completely drawn box represents the final position. The bottom row of Figure 12 represents the distance of the red CAV from the conflict zone and the required safe distance to be maintained. For the cases where CAV with advantage (blue CAV) stops outside the conflict zone, the CAV with the disadvantage need not maintain any safe distance which is given by the "No conflicts" greyed out area. A similar experiment is done where the two CAVs have a conflict at a merge scenario is shown in Fig 13.

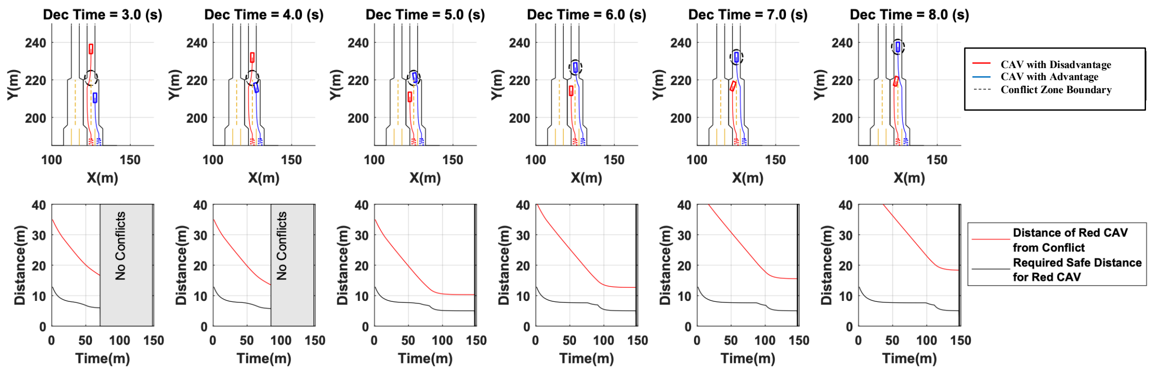

Fig 13: A merge scenario where two CAVs conflict

Similar to Fig 12, the top row of Fig 13 represents the trajectory of the two CAVs when the deceleration time of the blue CAV changes. The bottom row represents the distance of the red CAV from the conflict zone and the required safe distance from the conflict zone. It can be seen that the distance of the conflict zone from the CAV is always greater than the required safe distance, proving the safety of the algorithm.

5.2 Deadlock Resolution

In order to test the deadlock resolution implemented in the algorithm, a deadlock situation is created at an intersection. All the CAVs are at the same distance from the intersection. This is compared with no deadlock resolution mechanism. The figure below represents the deadlock scenario:

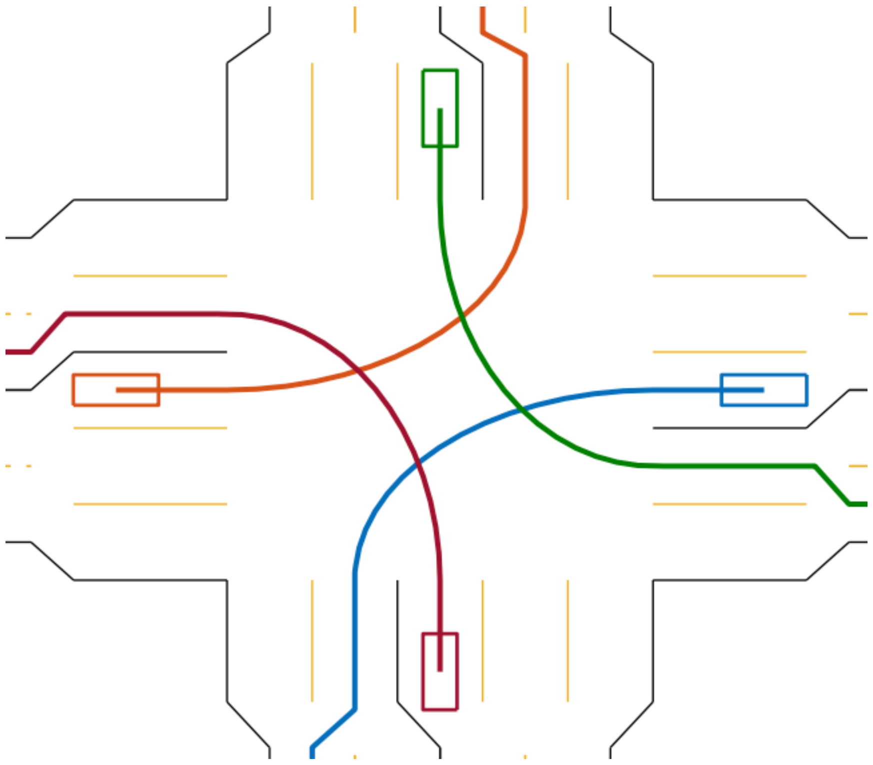
Fig 14: A deadlock scenario at an intersection

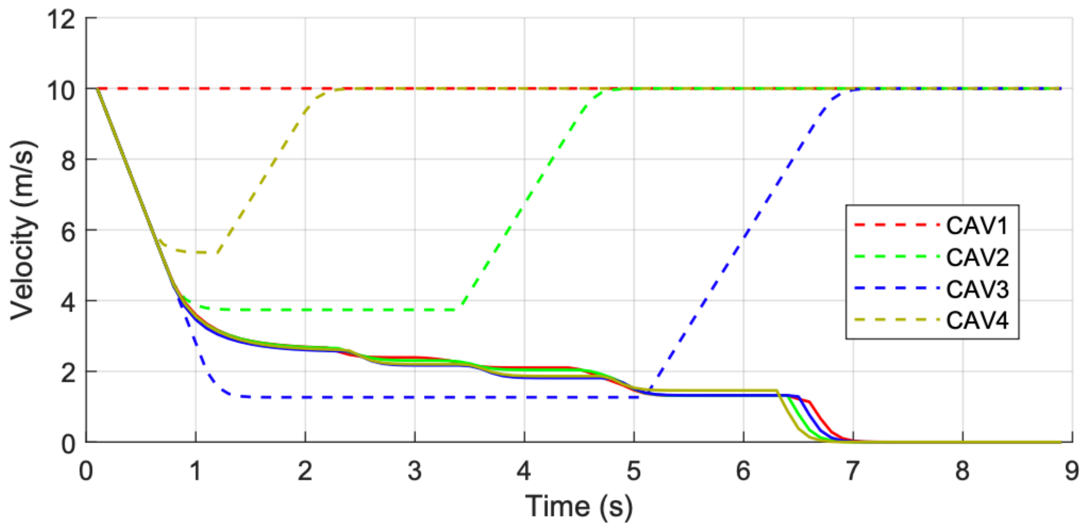

Fig 15: Velocity profile of all CAVs with and without deadlock resolution

Figure 15 represents how the velocity of the CAVs reach the desired speed by resolving the deadlocks and also represents how the velocity of all the CAVs reach zero when deadlock resolution is not used. The velocity profile of the CAVs when deadlock resolution is used is represented by the block lines whereas the velocity profile of the CAVs when deadlock resolution is not used is represented by the dotted lines. When deadlock resolution is used, the CAVs decrease their velocities, resolve the conflicts and then reach the desired velocity after crossing the conflict zone. When deadlock resolution is not used, the CAVs velocity reach zero in 7 seconds.

5.3 Scaled Cars Experiment

In order to demonstrate the proposed algorithm and to study the effect of communication delay and computation delay that is inherent in real time systems, it was experimented on RC cars that are approximately 1/10th the size of the actual cars. The experimental bed was set up in the Drone Studio at ASU. The experimental setup consists of Opti-Track cameras that are used to find the position of the vehicles inside the test bed. The scaled cars are TRAXASS remote controlled cars. The cars are mounted with an ESP8266 Wi-Fi module in order to send and receive data. The positional data from the Opti-track system is sent to a host PC through mocap module using Robot Operating System. This information is used by the cars in order to know the current position and this is then used to determine the future trajectory which is broadcasted to other vehicles. An intersection scenario is considered with two RC cars travelling around the intersection.

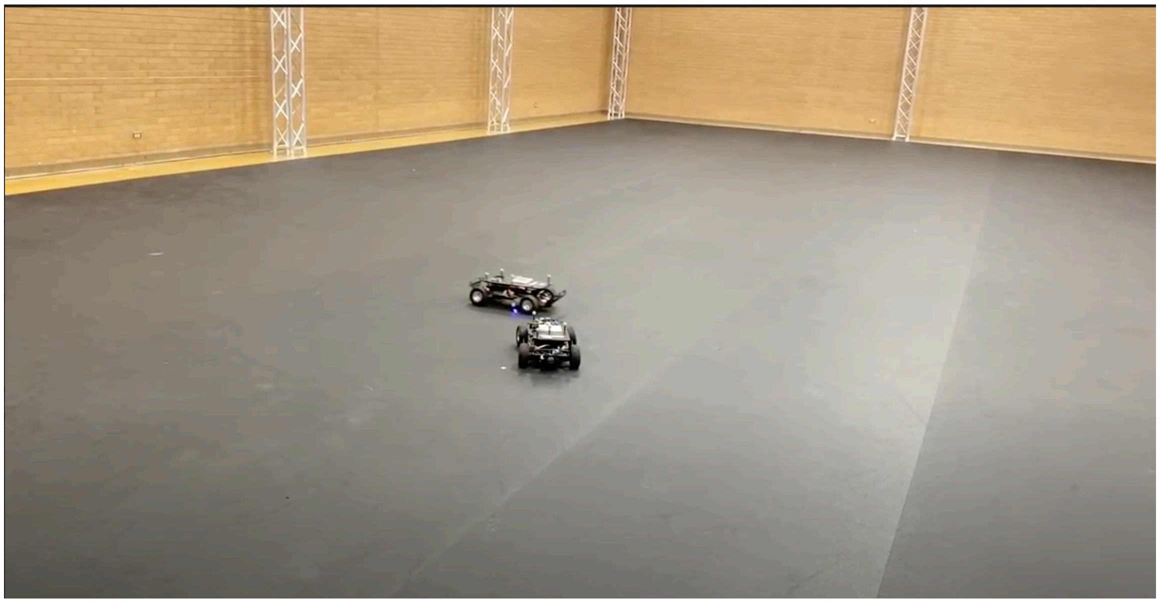

Fig 16: A snapshot of the testbed for the demo using RC scaled cars

Fig 16 shows a snapshot of the testbed that was used for performing the demo of the algorithm using RC scaled cars. The cameras are mounted above the experimental zone which detect the position of the vehicle. The map for the cars is similar to the one shown in Fig 2 except that a single lane is present in each road. The cars are allowed to go to random destinations on the map and the maximum permissible velocity is set to be 1.5m/s. The demo is performed such that the cars would continuously move around the map going to different destinations. Since, the cars have to receive the information from the other vehicle, there would some communication delay and also since, the cars have to process the received information and actuate accordingly, there would be some computation delay. With this, the effect of communication and computation delay is tested.

The video for the demo - https://youtu.be/pqKAq3Bms_I shows the demo of the algorithm. With the communication delay and computation delay present in the system, it can be observed that it does not cause a collision and the vehicle with the disadvantage reacts well in advance such that it would not cause a collision. This is because of the inclusion of communication delay and reaction time in the maximum future trajectory broadcast distance and also in the safe distance calculation. Therefore, the even with the presence of communication and computation delay, the algorithm proves to be providing safe trajectories avoiding collision between the vehicles.

CHAPTER 6

MY CONTRIBUTIONS IN THIS WORK

This work is a joint work between me, Harshith Allamsetti, Mohammad Khayatian (Arizona State University), Dr. Aviral Shrivastava (Arizona State University), Mohammadreza Mehrabian (Arizona State University) and Dr. Chung-Wei Lin (National Taiwan University).

The following are my contributions towards this work:

- I had worked on the background literature of the work as to what are the current techniques that are used for the management of autonomous vehicles especially the decentralized methods.
- I performed a simulation of the most closely related work – "Distributed conflict resolution for connected autonomous vehicles" proposed by Liu et al.
- The limitations of the work were studied and then this technique was further developed for our work.
- The initial algorithm for our work was developed by me which included resolving conflicts and formulating a trajectory for resolving the conflict.
- A demo of the algorithm on two RC scaled cars was performed by me in the Drone Studio at Arizona State University in order to study the effect of communication delay and computation time.

CHAPTER 7

SUMMARY

This thesis presents an approach for the management of Connected Autonomous Vehicles (CAV) across all the driving scenarios such as intersections, ramp merging, lane change while also ensuring that the trajectory created is safe and the decision making is deadlock free. The CAVs broadcast their future trajectories and check for any conflicts with the other CAVs. Then, they check for any deadlocks, resolve these deadlocks and resolve the conflicts such that there would be no collision and the CAVs maneuver safely. The graph-based deadlock resolution mechanism ensures that the vehicles when making a decision in determining who has the advantage does not result in a deadlock which is a common problem with the decentralized management of autonomous vehicles. The incorporation of the Responsibility-Sensitive-Safety rules ensures that the trajectories created by the vehicles do not result in a collision, thereby avoiding accidents even in the worst-case scenario. The experimental results show that the algorithm can provide safe trajectories even in the worst-case scenarios when one of the CAVs decelerates, it is able to detect and resolve deadlocks in finite time and the algorithm is also capable of handling communication and computation delays that are inherent in autonomous vehicles.

7.1 Future Work

This algorithm considers all the vehicles to be connected such that each and every vehicle would send and receive information with each other. The algorithm can be further developed by considering some of the vehicles to be not connected by which the vehicle's information can be received by the road infrastructure. Further considerations can be made by taking into account various faults that could occur such as when a CAV breaks down, the information such as the position and trajectory sent by a CAV is compromised or when the CAV is not able to communicate, etc. The algorithm can be made more resilient and robust by considering these faults such that it will be able to handle all these fault models and scenarios.

REFERENCES

Aoki, S., & Rajkumar, R. (2017, April). A merging protocol for self-driving vehicles. In *2017 ACM/IEEE 8th International Conference on Cyber-Physical Systems (ICCPS)* (pp. 219-228). IEEE.

Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., ... & Zhang, X. (2016). End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*.

Chen, C., Seff, A., Kornhauser, A., & Xiao, J. (2015). Deepdriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the IEEE International Conference on Computer Vision* (pp. 2722-2730).

Chen, Y., Lu, C., & Chu, W. (2020). A Cooperative Driving Strategy Based on Velocity Prediction for Connected Vehicles with Robust Path-following Control. *IEEE Internet of Things Journal*.

Dolgov, D., Thrun, S., Montemerlo, M., & Diebel, J. (2010). Path planning for autonomous vehicles in unknown semi-structured environments. *The International Journal of Robotics Research*, *29*(5), 485-501.

Dresner, K., & Stone, P. (2008). A multiagent approach to autonomous intersection management. *Journal of artificial intelligence research*, *31*, 591-656.

Duering, M., & Pascheka, P. (2014, June). Cooperative decentralized decision making for conflict resolution among autonomous agents. In *2014 IEEE International Symposium on Innovations in Intelligent Systems and Applications (INISTA) Proceedings* (pp. 154-161). IEEE.

Fassbender, D., Heinrich, B. C., & Wuensche, H. J. (2016, October). Motion planning for autonomous vehicles in highly constrained urban environments. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (pp. 4708-4713). IEEE.

Ferguson, D., Howard, T. M., & Likhachev, M. (2008). Motion planning in urban environments. *Journal of Field Robotics*, *25*(11-12), 939-960.

Ferguson, D., & Stentz, A. (2006). Using interpolation to improve path planning: The Field D* algorithm. *Journal of Field Robotics*, *23*(2), 79-101.

Gao, W., Hsu, D., Lee, W. S., Shen, S., & Subramanian, K. (2017). Intention-net: Integrating planning and deep learning for goal-directed autonomous navigation. *arXiv preprint arXiv:1710.05627*.

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, *4*(2), 100-107.

Hausknecht, M., Au, T. C., & Stone, P. (2011, September). Autonomous intersection management: Multi-intersection optimization. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 4581-4586). IEEE.

Janson, L., Schmerling, E., Clark, A., & Pavone, M. (2015). Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions. *The International journal of robotics research*, *34*(7), 883-921.

Kavraki, L. E., Svestka, P., Latombe, J. C., & Overmars, M. H. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, *12*(4), 566-580.
Karaman, S., & Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, *30*(7), 846-894.

Khayatian, M., Lou, Y., Mehrabian, M., & Shirvastava, A. (2019). Crossroads+ A Time-aware Approach for Intersection Management of Connected Autonomous Vehicles. *ACM Transactions on Cyber-Physical Systems*, *4*(2), 1-28.

Khayatian, M., Mehrabian, M., & Shrivastava, A. (2018, December). RIM: Robust intersection management for connected autonomous vehicles. In *2018 IEEE Real-Time Systems Symposium (RTSS)* (pp. 35-44). IEEE.

LaValle, S. M., & Kuffner Jr, J. J. (2001). Randomized kinodynamic planning. *The international journal of robotics research*, *20*(5), 378-400.

Li, B., Zhang, Y., Zhang, Y., & Jia, N. (2018, June). Cooperative lane change motion planning of connected and automated vehicles: A stepwise computational framework. In *2018 IEEE Intelligent Vehicles Symposium (IV)* (pp. 334-338). IEEE.

Li, X., Sun, Z., Cao, D., He, Z., & Zhu, Q. (2015). Real-time trajectory planning for autonomous urban driving: Framework, algorithms, and verifications. *IEEE/ASME Transactions on Mechatronics*, *21*(2), 740-753.

Li, X., Sun, Z., Cao, D., Liu, D., & He, H. (2017). Development of a new integrated local trajectory planning and tracking control framework for autonomous ground vehicles. *Mechanical Systems and Signal Processing*, *87*, 118-137.

Likhachev, M., Ferguson, D. I., Gordon, G. J., Stentz, A., & Thrun, S. (2005, June). Anytime Dynamic A*: An Anytime, Replanning Algorithm. In *ICAPS* (Vol. 5, pp. 262-271).

Lin, Y. T., Hsu, H., Lin, S. C., Lin, C. W., Jiang, I. H. R., & Liu, C. (2019). Graph-Based Modeling, Scheduling, and Verification for Intersection Management of Intelligent Vehicles. *ACM Transactions on Embedded Computing Systems (TECS)*, *18*(5s), 1-21.

Liu, C., Lin, C. W., Shiraishi, S., & Tomizuka, M. (2017). Distributed conflict resolution for connected autonomous vehicles. *IEEE Transactions on Intelligent Vehicles*, *3*(1), 18-29.

Nie, J., Zhang, J., Ding, W., Wan, X., Chen, X., & Ran, B. (2016). Decentralized cooperative lane-changing decision-making for connected autonomous vehicles. *IEEE Access*, *4*, 9413-9420.

Luo, Y., Xiang, Y., Cao, K., & Li, K. (2016). A dynamic automated lane change maneuver based on vehicle-to-vehicle communication. *Transportation Research Part C: Emerging Technologies*, *62*, 87-102.

McNaughton, M., Urmson, C., Dolan, J. M., & Lee, J. W. (2011, May). Motion planning for autonomous driving with a conformal spatiotemporal lattice. In *2011 IEEE International Conference on Robotics and Automation* (pp. 4889-4895). IEEE.

Otte, M., & Frazzoli, E. (2015). $${\mathrm {RRT^{X}}} $$: Real-Time Motion Planning/Replanning for Environments with Unpredictable Obstacles. In *Algorithmic Foundations of Robotics XI* (pp. 461-478). Springer, Cham.

Pokle, A., Martín-Martín, R., Goebel, P., Chow, V., Ewald, H. M., Yang, J., ... & Vázquez, M. (2019, May). Deep Local Trajectory Replanning and Control for Robot Navigation. In *2019 International Conference on Robotics and Automation (ICRA)* (pp. 5815-5822). IEEE.

Ratliff, N., Zucker, M., Bagnell, J. A., & Srinivasa, S. (2009, May). CHOMP: Gradient optimization techniques for efficient motion planning. In *2009 IEEE International Conference on Robotics and Automation* (pp. 489-494). IEEE.

Rios-Torres, J., & Malikopoulos, A. A. (2016). Automated and cooperative vehicle merging at highway on-ramps. *IEEE Transactions on Intelligent Transportation Systems*, *18*(4), 780-789.

Schmerling, E., Janson, L., & Pavone, M. (2015, May). Optimal sampling-based motion planning under differential constraints: the driftless case. In *2015 IEEE International Conference on Robotics and Automation (ICRA)* (pp. 2368-2375). IEEE.

Shalev-Shwartz, S., Shammah, S., & Shashua, A. (2017). On a formal model of safe and scalable self-driving cars. *arXiv preprint arXiv:1708.06374*.

Stentz, A. (1997). Optimal and efficient path planning for partially known environments. In *Intelligent Unmanned Ground Vehicles* (pp. 203-220). Springer, Boston, MA.

Zucker, M., Ratliff, N., Dragan, A. D., Pivtoraiko, M., Klingensmith, M., Dellin, C. M., ... & Srinivasa, S. S. (2013). Chomp: Covariant hamiltonian optimization for motion planning. *The International Journal of Robotics Research*, *32*(9-10), 1164-1193.

Kalakrishnan, M., Chitta, S., Theodorou, E., Pastor, P., & Schaal, S. (2011, May). STOMP: Stochastic trajectory optimization for motion planning. In *2011 IEEE international conference on robotics and automation* (pp. 4569-4574). IEEE.

APPENDIX A

MOTION PLANNING TECHNIQUES

The methods for motion planning can be broadly categorized as:

1. Sampling based methods
2. Graph search methods
3. Optimization based methods

Sampling based methods

The Sampling Based methods are the most popular and widely used methods for the motion planning problem. This involves discretization of the space that a robot can move into points and then connecting these points using straight lines taking into account the presence of obstacles. The points produced by the discretization are the various positions that a robot can be in and the straight-line segments represent a feasible path between the two positions.

Kavraki, Svestka, Latombe & Overmars (1996) proposed a method to compute collision-free paths among stationary obstacles. It consists of two phases - learning phase and query phase. In the learning phase, the complete configuration space is discretized by choosing random configurations which constitute the nodes and then sampling random points around the chosen node which are within a fixed distance. Then it checks if a straight-line path exists between the chosen node and each of the sampled nodes and also if it is collision free, then the edge along with the nodes are added. This results in a graph of the complete configuration space with a set of nodes and edges connecting these nodes. Also, the graph is expanded in order to improve the connectivity of the graph. In the query phase, given the initial and final configuration, the path is computed by selecting the nodes that are nearest in the set of nodes of the graph. This is called the PRM (Probabilistic Road Map) algorithm. The following is the algorithm for the learning phase –

```
Algorithm 1: PRM (preprocessing phase)
1 $V \leftarrow \emptyset$; $E \leftarrow \emptyset$;
2 for $i = 0, \ldots, n$ do
3     $x_{\text{rand}} \leftarrow \texttt{SampleFree}_i$;
4     $U \leftarrow \texttt{Near}(G = (V, E), x_{\text{rand}}, r)$ ;
5     $V \leftarrow V \cup \{x_{\text{rand}}\}$;
6     foreach $u \in U$, in order of increasing $\|u - x_{\text{rand}}\|$, do
7         if $x_{\text{rand}}$ and $u$ are not in the same connected component of $G = (V, E)$ then
8             if $\texttt{CollisionFree}(x_{\text{rand}}, u)$ then $E \leftarrow E \cup \{(x_{\text{rand}}, u), (u, x_{\text{rand}})\}$;
9 return $G = (V, E)$;
```

Source: https://arxiv.org/pdf/1105.1186.pdf

Here, initially the set of vertices and edges are empty. Then x_{rand} is sampled randomly from the free space, a set of points U represents the nodes that are present in the graph within a fixed distance r. For each of the point in U, it is checked if x_{rand} is already not included in the graph, and the path between the selected point and x_{rand} is collision free, that edge is added to the graph. The following figure shows an example of the PRM algorithm –

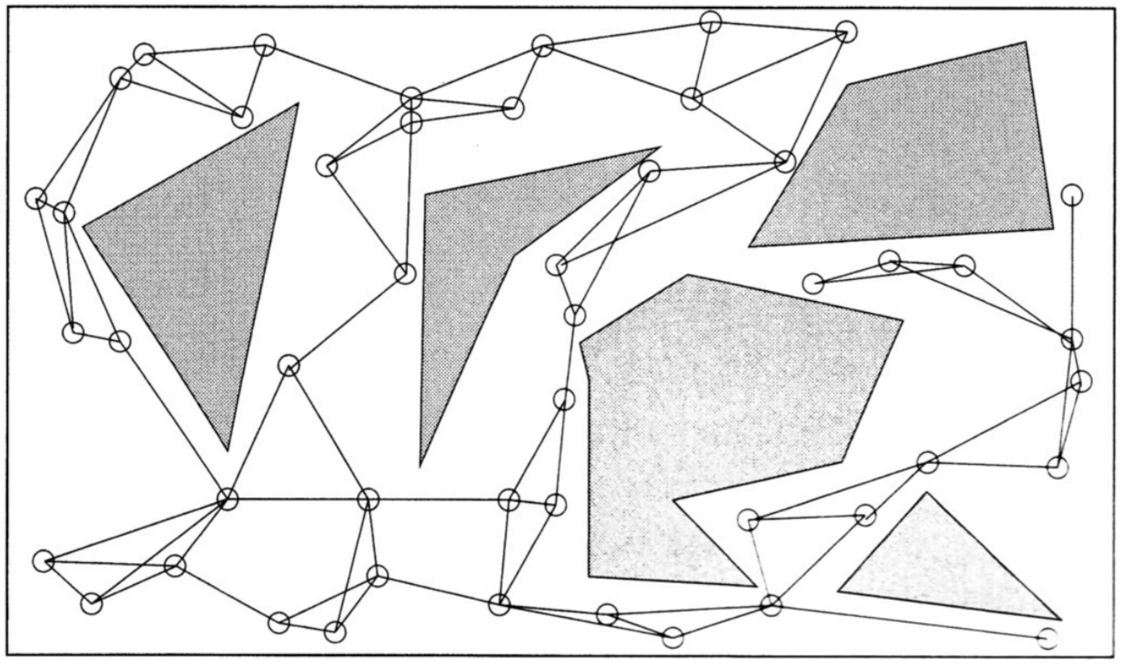

Fig 17: Illustration of learning phase for PRM
Source: http://www.cs.columbia.edu/~allen/F15/NOTES/Probabilisticpath.pdf

Here, the rectangle box represents the complete configuration space and the grey polygons represent the obstacles, the PRM algorithm initially builds a graph of the complete configuration space considering the obstacles – the empty circles represent the nodes of the graph and the edges represent a feasible path between these nodes. This is the learning phase.

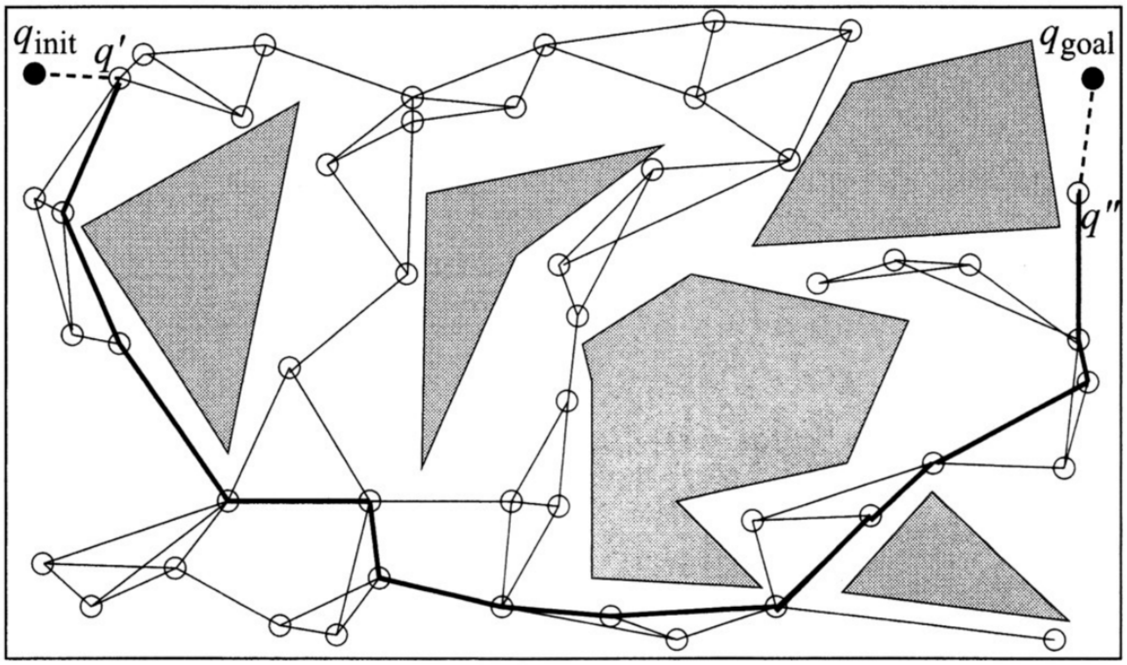

Fig 18: Illustration of query phase for PRM
Source: http://www.cs.columbia.edu/~allen/F15/NOTES/Probabilisticpath.pdf

This figure represents how the query is done for the PRM algorithm. The configurations q_{init} and q_{goal} are first connected to the roadmap through q’ and q”. Then a graph search algorithm is used to find a path between the two nodes. However, the path computed by this algorithm does not lead to an asymptotic stable solution i.e. the cost of the solution returned by the algorithm does not lead to a global optimum. Karaman and Frazolli (2011) proposed an extension to the PRM algorithm - PRM*. The main modification made was that in the PRM algorithm, the connections between road vertices are made which are within a finite fixed distance from one another whereas in the PRM* algorithm, the connection distance is a variable which is dependent on n - the total number of possible random configurations in the configuration space.

LaValle and Kuffner (2001) introduced the first motion planner that could handle kinodynamic problems. This algorithm tackled the problem where path planning is only done considering just the kinematics such as the presence of the obstacles but do not consider the dynamics of the robot. The planner uses a rapidly exploring random tree and the tree is constructed incrementally from samples drawn randomly from the search space and grows towards large unsearched areas in the search space. The initial state is taken as a vertex. Then, a point is selected randomly and, in each iteration, a new vertex is added which is biased by a random-selected state. Also, the algorithm selects the nearest vertex already in the RRT to the given sample state. The tree is extended by applying an input for some time increment. Then, it is checked whether the new node and the sampled node is obstacle-free, if it is, then it is added to the tree. The RRT can be used as a motion planner as it is randomly explored, and it can be extended until it reaches the goal region.

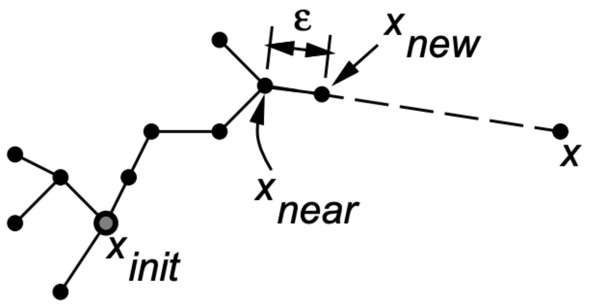

Fig 19: Illustration of the construction of RRT
Source: http://msl.cs.uiuc.edu/~lavalle/papers/LavKuf01b.pdf

The Fig shows how the RRT is constructed. Here, x is the randomly selected point and x_{near} is the nearest vertex of the RRT, when the input is applied for some time increment, x_{new} is the newly added node to the RRT. So, in this way, the RRT explores the complete configuration space. The following is the algorithm for RRT –

Algorithm 3: RRT

```
1 $V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$;
2 for $i = 1, \ldots, n$ do
3     $x_{\text{rand}} \leftarrow \texttt{SampleFree}_i$;
4     $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}})$;
5     $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})$ ;
6     if $\texttt{ObtacleFree}(x_{\text{nearest}}, x_{\text{new}})$ then
7         $V \leftarrow V \cup \{x_{\text{new}}\}$; $E \leftarrow E \cup \{(x_{\text{nearest}}, x_{\text{new}})\}$ ;
8 return $G = (V, E)$;
```

Initially, the set of vertices comprises the initial position and the set of edges is empty. Then x_{rand} is sampled randomly from the free space, and $x_{nearest}$ represents the nearest vertex of the graph. x_{new} is a point obtained by applying the control input for a finite time. Now, if the path between the points x_{new} and $x_{nearest}$ is collision free, then x_{new} is added to the set of vertices and the edge between x_{new} and $x_{nearest}$ is added to the set of edges.

Karaman and Frazolli (2011) introduced two new algorithms based on RRT - RRG and RRT*. The main difference between RRT and RRG is that whenever a new point is added, a connection is attempted between all other vertices in the tree that are within a specified volume which is dependent on the number of vertices already present in the tree. This results in an incrementally better solution as the tree grows. The RRT* is another variant of the RRT algorithm which leads to an optimum solution. The key difference is that in RRT*, an edge is created between the sampled point and the closest vertex in the graph that results in a minimum cost. Also, rewiring is done where after a vertex has been connected to the nearest neighbor, the neighbors are again examined if being rewired to the newly added vertex will make their cost decrease. If the cost does indeed decrease, the neighbor is rewired to the newly added vertex. This leads to an asymptotic optimal solution which is not achieved by the original RRT algorithm.

An alternative to the PRM* and RRT* algorithm was proposed by Janson, Schmerling, Clark & Pavone (2015). It was termed FMT*(Fast Marching Tree) where the discretization and the computation of the shortest path is done at the same time. It uses dynamic recursion programming from a set of vertices in order to create a fast-growing tree of trajectories. It improves upon the asymptotic optimal algorithms PRM* and RRT* in terms of the convergence rate to the optimal solution which is interpreted in terms of the execution time. The FMT* concurrently performs graph construction and graph search and it lazily skips collision checks when making local optimal connections. This may lead to suboptimal solutions but FMT* ensures that such conditions vanish as the number of samples reaches infinity.

Schmerling, Janson & Pavone (2015, May) proposed extensions of PRM* and FMT* under differential constraints. These algorithms are specifically applicable for drift less control affine (DCA) systems. DPRM* algorithm works by sampling a set of states in the configuration space and then attempts to connect to the state with the lowest cost provided it is collision-free. Then, the lowest cost between the initial and final configuration is computed using a graph search algorithm such as Dijkstra's algorithm. DFMT* is a streamlined version of DPRM* by performing "lazy" dynamic recursion to grow a tree of trajectories. At each iteration, DFMT* considers the neighborhood of a state and only attempts locally optimal connections that are collision free, in contrast to attempting a connection with all the states. This drastically reduces the number of collision check computations. The main difference between DFMT* and DPRM* from their counterparts is that in this, the neighboring vertices which are within a coordinate box are checked rather than within a Euclidean ball and the edges connecting vertices are optimal trajectories rather than straight lines.

Otte and Frazzoli (2015) proposed RRT-X to extend the RRT algorithm for dynamic changing environments with unpredictable obstacles. RRT-X builds upon the original RRT algorithm of finding a shortest path between the initial and final configuration. It computes the shortest path by building the RRT and also accounts for any obstacles present in the configuration space. The graph is refined by adding new samples and connecting them to the graph if possible. If obstacles are sensed, it is also accounted for and the graph is updated. With the robot moving, the node nearest to the robot position is computed. If the node does not lie in the goal region, then the graph is extended by inserting new nodes onto the graph if the connection is possible. However, Sampling based planners always generate jerky continuous trajectories leading to trajectories that may not be feasible to steer and it worsens as the road becomes curvy resulting in the car leaving the roadway. Also, the computation time for RRT is very high which makes it infeasible for real time motion planning of autonomous vehicles and trajectories generated by RRT do not consider curve continuity resulting in unsafe maneuvers.

Graph Search methods

The sampling-based methods create either a graph or a tree of the configuration space and then find a feasible path or an optimal path from the initial to the final configuration. There have been techniques in order to find the shortest path between the initial and final configuration given a graph of the configurations space had been constructed. One of the most prominent methods of finding the shortest path given a graph is the A* algorithm proposed by Hart, Nilsson & Raphael (1968). In each of the A* algorithm, the node with the lowest cost which comprises the cost to move from starting point to the given point, considering the path followed and an estimate of the cost to move from the given point to the final point. The Dijkstra's algorithm to search a graph is a special case of A* algorithm where the estimated cost to move from the given point to the final point is not computed. Stentz (1997) built upon the A* algorithm to propose D* which stand for Dynamic A* which is capable of planning paths in unknown, partially known and changing environments. The algorithm consists of two main functions, one function is used to determine the optimal path from the initial position to the final position and the second function is called whenever the robot detects an obstacle to change the cost of the edge.

Likhachev, Ferguson, Gordon, Stentz & Thrun (2005, June) proposed Anytime Dynamic A* which could combine the benefits of incremental replanning and anytime planning. When the robot initially gets the information about its environment or surroundings, it is generally incomplete due to the presence of dynamic obstacles and dynamically changing surroundings. So, the solution generated by the initial information may invalid or sub-optimal requiring the motion planning algorithm to dynamically gather information and produce an appropriate solution. This is called incremental replanning where the robot replans its path with the changing environment around it considering new information. When the motion planning problem is complex and the computation time is limited, the solution to the motion planning problem may be sub-optimal and the agent must be satisfied with the best possible solution within the given computation time. The algorithms that generate such solutions are called Anytime planning algorithms. Given the initial and the final state, the algorithm initially computes a suboptimal path quickly. Unless there is a change in the cost of the edges of the graph, the algorithm improves the quality of the solution until the optimal solution is obtained. If there is a change in the edge cost, the cost of those edges is updated. Then, if there is a substantial change in the edge cost, the algorithm computes a suboptimal solution by building the solution from scratch. Ferguson & Stentz (2006) proposed Field D* that could tackle two problems posed by algorithms such as A*, D*, Anytime Dynamic A* - first all these algorithms consider a small, discrete set of possible transitions from one node to another in a graph resulting in longer path lengths and unnecessary turning. The next limitation is that these algorithms use uniform resolution grids in order to represent the environment and this may not be feasible to represent large and sparsely populated environments. In this algorithm, the cell nodes are considered to be situated at the corners of the cells rather than at the center of the cells. The algorithm first computes the shortest path from the source to the destination and then iteratively changes the cell traversal costs, updates the cost of the nodes and computes the shortest possible path again. This paper also presented Multi-resolution Field D* where the configuration space is given by non-uniform grid cells such that it contains both high resolution and low-resolution cells. The same interpolation method used in Field D* algorithm is used here too. The main difference is that in a uniform resolution case, each node has exactly eight neighboring nodes whereas in the non-uniform resolution case, there can be more than eight neighboring edges.

State lattice methods have been extensively used in order to define the trajectory given an initial and a final position. A state lattice can be defined as a search graph in which the nodes represent the state of the vehicle such as the position, velocity and acceleration and the edges denote paths connecting nodes that satisfy the dynamic constraints of the vehicle. McNaughton et al. propose a spatio temporal lattice method for trajectory generation where the nodes are defined at a lateral offset from the center and edges are computed using an optimization algorithm. The optimization algorithm finds the edges that connect the nodes and solve for a cost function for each trajectory Li, Sun, Cao, He & Zhu (2015) used a spatio temporal lattice to generate candidate curves using cubic polynomial curves. A velocity profile is also computed to be assigned to poses of the generated paths. The resulting trajectories are evaluated by a cost function and the optimal one is selected Fassbender, Heinrich & Wuensche (2016, October) proposed a planner that extended the A* algorithm using two new schemes. The first scheme finds a trajectory between the current node and goal node using numerical optimization. The second scheme uses a pure pursuit controller to guide that car along the reference path. These algorithms create trajectories based on optimization which are constrained by a cost function - the cost function does not consider more complex behaviors such as passing through vehicle’s blind spots, merging traffic. Also, more better acceleration profiles need to be generated as constant acceleration can lead to execution errors.

Optimization based methods

The Optimization methods formulate a trajectory as a function with constrained variables. The optimization methods find a trajectory by minimizing a cost function that involves trajectory constraints such as acceleration, velocity, position. These methods consider the motion planning in the framework of nonlinear continuous optimization. The problem is formulated as a trajectory optimization problem which can be abstracted to a path planning problem where trajectory optimization is done for a unit amount of time. Given an initial and final position, variational methods first formulate a trajectory and then optimize this trajectory with the constraints of a cost function.

Ratliff, Zucker, Bagnell & Srinivasa (2009, May) first worked on this method by proposing CHOMP which generates and optimizes trajectories for robotic systems. CHOMP transforms a naive initial guess into a trajectory suitable for execution for the system without the requirement for a separate motion planner. The covariant gradient ensures that the solution converges to a local optimum. The cost of the trajectory is modelled using two functions - one to represent the cost of being near an obstacle and another which measures the dynamic quantities of the robot such as the smoothness and acceleration. Both these costs are a function of which maps time to robot configurations. The smoothness functional is defined as a metric in the space of trajectories and is updated such that the trajectory is smooth after each iteration of the modification. It measures the acceleration across the trajectory. The obstacle functional penalizes the robot parts for being near the obstacles or already in collision. Then covariant gradient descent is used in order to refine the trajectory and optimize it. The gradient of the cost functional is included in the update rule for the trajectory until the gradient falls below a threshold.

However, CHOMP suffers from the problem of local optimum as the gradient descent may result in a local optimum which depends on the initial solution taken for the problem. So, a variant of CHOMP using Hamiltonian Monte Carlo was proposed by Zucker et al. (2013). Hamiltonian Monte Carlo is used to solve the problem of local minima. HMC adds a momentum turn to the update equation to perturb the solution out of the current local minima. It gives the solution a little momentum, pushing it out of the current minima and exploring in another direction. Another algorithm was proposed by Kalakrishnan, Chitta, Theodorou, Pastor & Schaal (2011, May). called the STOMP which overcomes the problem that CHOMP faces which is producing local optimum solutions. The algorithm works initially by creating a trajectory in order to explore the space which may not be feasible. A cost function which is dependent on the obstacles and smoothness of the trajectory is optimized in each iteration. The cost function used is similar to the one used in the CHOMP algorithm which includes smoothness of the trajectory and a cost term to include state-dependent costs such as obstacle costs, constraints and torques. The STOMP algorithm works by optimizing the trajectory cost until it reaches convergence. The algorithm first creates noisy trajectories. Then, the cost per time step for each of the noisy trajectory is computed, and then the probabilities of each trajectory are computed. The trajectories are updated using the probabilities computed as the probability-weighted convex combination of the noisy parameters for that time-step. The noisy update is smoothed, and the trajectory cost is computed. This process is repeated until the trajectory cost converges.

Dolgov et al. proposed an optimization method based on A* planning algorithm. The algorithm consists of two phases - in the first phase, a feasible trajectory is created using the A* algorithm. The second phase improves the solution by applying nonlinear optimization. The optimization is done in two steps where the trajectory is smoothened and then it is made collision free. Model Predictive Control (MPC) for trajectory planning produces dynamic feasible commands by using the model of the car in order to make predictions about the car's future state. It solves an optimization algorithm to find the appropriate action in order to match the predicted future state. Ferguson et al. combined a model predictive trajectory generator algorithm with two high level planners to generate trajectories. The motion planner given a final position, finds a path from the current position of the car to the final position. A set of candidate trajectory functions that follow the path are generated that satisfy a set of state constraints whose dynamics are given by a set of differential equations. Then, the initial parameters of the trajectory are optimized until the trajectory end point is within an error bound from the goal position. A velocity profile is created, and the best trajectory is selected based on the smoothness, end point error, velocity error. Li et al. proposed a sampling-based trajectory planner based on optimization such that goal states are sampled along the route. An MPC is used to produce a path to the goal state and a velocity profile is generated for the path generated. A cost function that considers the safety and velocity is used to select the best trajectory. The numerical optimization methods suffer from high computation time as at every instant, the car has to optimize the trajectories, and this may not be safe for autonomous vehicles which are real time systems requiring very fast computation. Also, the cost function for the optimization formulated are not applicable for complex scenarios that autonomous vehicles are generally involved in such as merge scenarios.

End to End Deep Learning methods

With the recent advancements in deep learning, a lot of methods have been proposed in order to tackle the motion planning problem using deep learning. End-to-end deep learning methods solve the problem of motion planning by taking high-level sensory input such as data from a camera, LIDAR uses deep neural networks and outputs low-level control commands such as braking, steering.

Bojaski et al. (2016) introduced end to end deep learning for self-driving cars which uses a convolutional neural network in order to map raw pixels obtained from a single center facing camera directly to steering commands. The training data for the CNN was acquired from a car mounted with three cameras on the left, center and right side of the car along with the steering wheel angle from the CAN bus in order to represent the steering command. The images sampled from the video are given as input to the CNN which computes a steering command. The computed command is compared with the desired command and the weights of the CNN are adjusted in order to bring the computed command closer to the computed command. The images are augmented with artificial shifts and rotations in order to teach the network how to recover from drifting off the road.

Pokle et al. (2019) proposed a navigation system which combines both hierarchical planning and machine learning. The system consists of a global planner which is used to compute the shortest path from the source to the destination. The system also consists of a local trajectory planner which is used to compute the velocity and steering commands for the vehicle to execute. There are three levels of hierarchy in this system - a Global Planner, which takes as input the 2D environment around the robot and outputs a path from the current position to the goal position. The Local Planner gives out the immediate actions to be taken given the surrounding conditions such as obstacles in the vicinity of the robot. The Velocity Control computes the commands for the navigation of the robot. The Local Planner and the Velocity Control are implemented using Neural Networks. Ma et al. employed conditional Generative Adversarial Networks (cGAN) for feature extraction. In this method, raw visionary data from a camera along with the navigational data from the GPS is combined to obtain navigational cost maps which are then used. However, the deep learning methods lack an inherent problem is that there are no hardcoded safety measures. The output commands from the system depends on the training of the neural network and so, the training data should be able to cover all the unexpected scenarios that an autonomous vehicle would go through. Also, the actuation commands that are given out by the system depends on the vehicle used during training, so the actuating commands may not work properly for another vehicle using the same neural network.