EVALUATION OF LOCAL KINEMATIC MOTION PLANNING ALGORITHMS FOR A TRUCK AND TRAILER SYSTEM

A Thesis

by

GRAYSON LANDIS WOODS

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Srikanth Saripalli
Committee Members,	Swaminathan Gopalswamy
	Dylan Shell
Head of Department,	Andreas A. Polycarpou

May 2020

Major Subject: Mechanical Engineering

ABSTRACT

Over the past few decades, researchers have worked towards developing autonomous systems that can be used in everyday transportation, and with the emergence of new sensor, hardware, and software technologies, the goal of self-driving vehicles is now on the brink of becoming a reality. In order for these systems to properly plan and react to their complex environments, they need to be equipped with the proper tools and algorithms to ensure safe deployment for all stakeholders. Navigating tight spaces with truck and trailer systems in dynamic environments can be a difficult task due to their their nonlinear dynamics, delayed actuation, and large physical dimensions. This thesis presents a kinematic approach to local motion planning for truck and trailer vehicles in the forward motion. This approach was applied to the sample based planning algorithms RRT* and RRT^X in order to adapt and replan in the presence of dynamic obstacles. A combined motion planning and control framework was then developed and deployed in both simulation, using American Truck Simulator, and on an International ProStar 122+ truck. After the feedback controllers were iteratively tuned, the motion planners were evaluated alongside a deterministic Hybrid A* approach using a lane change and seaport scenario with simulated static and dynamic obstacles. In both cases, the approach demonstrated the ability for the sample based planner approach to provide real-time and feasible plans for the controller to execute at low speeds while maintaining a safe distance away from nearby obstacles.

DEDICATION

To my loving wife Lindsey.

ACKNOWLEDGMENTS

I would like to sincerely thank my advisor, Dr. Srikanth Saripalli, for fostering such a great research environment over the past couple of years that I have been with the Unmanned Systems Lab. His support and guidance through my various academic endeavors has made my graduate career that much more fulfilling and enjoyable. I would also like to express my gratitude to the rest of my research advisory committee, Dr. Swaminathan Gopalswamy and Dr. Dylan Shell, for their enthusiastic discussions and feedback that helped shape my research. On a similar note, I cannot thank the entire Unmanned Systems Lab enough for being such great colleagues and sources of inspiration. I want to especially thank the open source community and more specifically Michael Otte and Atsushi Sakai for making their motion planning code easily readable and accessible. I couldn't have done this without my wife, Lindsey, who supported me through the countless hours of simulation and experimentation needed to finish this thesis. Last but not least, I will always be thankful to my parents who have shown me unconditional love and encouragement throughout my life.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This graduate work was supported by a thesis committee consisting of Dr. Saripalli and Dr. Gopalswamy of the Department of Mechanical Engineering, and Dr. Shell of the Department of Computer Science & Engineering.

None of this work could have been possible without the Unmanned System Lab team that initially developed testing and the simulation platform for the truck and trailer system. This includes Amir Darwesh, Tim Overbye, and Garrison Neel. Drew Hubbard acted as the primary safety driver for all experimentation.

All other work conducted for the thesis was completed by the student independently.

Funding Sources

This graduate study was supported through Texas A&M's Department of Mechanical Engineering funding for the Unmanned Systems Lab.

NOMENCLATURE

OCP	Optimal Control Problem
GPS	Global positioning system
GNSS	Global Navigation Satellite System
IMU	Internal measurement unit
LiDAR	Light detection and ranging
ROS	Robot Operating System
INS	Inertial Navigation System
EKF	Extended Kalman filter
PID	Proportional, Integral, Derivative
ECU	Engine Control Unit
RRT	Rapidly Exploring Random Tree
CAN	Controller Area Network
SDK	Software Development Kit
TCP	Transmission Control Protocol
RPM	Revolutions Per Minute

TABLE OF CONTENTS

Page

ABSTRACT ii

DEDICATION iii

ACKNOWLEDGMENTS iv

CONTRIBUTORS AND FUNDING SOURCES v

NOMENCLATURE vi

TABLE OF CONTENTS vii

LIST OF FIGURES ix

LIST OF TABLES xi

1. INTRODUCTION AND OVERVIEW 1

    1.1 Introduction 1
    1.2 Background and Motivation 2
    1.3 Autonomous System Architecture 3
    1.4 Literature Review 5
    1.5 Contributions 8
    1.6 Assumptions 8
    1.7 Overview 9

2. KINEMATIC VEHICLE MODELS 10

    2.1 Nonholonomic System 10
    2.2 Kinematic Bicycle Model 11
    2.3 Dubins Car Model 12
    2.4 The Truck and Trailer Model 14
    2.5 Velocity Profile 15

3. MOTION PLANNING 17

    3.1 Problem Formulation and Approach 17
    3.2 Steering Function 19
    3.3 Heuristic and Cost Function 21
    3.4 Collision Detection 23

3.5 Search-based Methods 24
3.5.1 Hybrid A* 25
3.5.2 RRT* 26
3.5.3 RRT^X 28
3.5.4 Example Scenario 30

4. RESULTS AND DISCUSSION 32

4.1 Evaluation on ProStar 122+ Truck 32
4.1.1 Powertrain Specifications 32
4.1.2 By-wire Kit 33
4.1.3 Novatel SPAN System 35
4.1.3.1 Antenna Placement & Configuration 35
4.1.3.2 Accuracy 35
4.2 Simulation 36
4.2.1 Simulation Interface 37
4.2.2 System Response Comparison 38
4.3 Control 40
4.3.1 Lateral Control 41
4.3.2 Longitudinal Control 41
4.4 Static Scenarios 43
4.4.1 Lane Change with single obstacle 43
4.4.2 Lane Change with Moving Obstacles 46
4.4.3 Seaport 50
4.4.3.1 Curvature 55
4.5 Dynamic Scenarios 57
4.5.1 Lane Change with Car that Unpredictably Moves 57
4.5.2 Seaport with Limited Sensor Range 58
4.6 Overview of the Experimentation 63

5. SUMMARY AND CONCLUSION 65

5.1 Conclusions 65
5.2 Further Study 67

REFERENCES 68

LIST OF FIGURES

FIGURE Page

1.1 General control architecture for automated vehicles based on [1], where red modules indicate the focus of the thesis. 4

2.1 Kinematic bicycle model for car-like system 12

2.2 Dubins RSR path for car-like system with minimum turning radius (r_{min}) 13

2.3 Kinematic model for truck and trailer system 15

3.1 Hierarchical approach with local plan (black), controller execution (red), and global path (dashed) 19

3.2 Euclidean heuristic underestimates the path cost for nonholonomic vehicle 22

3.3 Two approaches to encapsulating truck and trailer: circle (blue), oriented rectangles (orange) 24

3.4 Illustration of how Hybrid A* enforces differential constraints during expansion. 26

3.5 Truck and trailer system executing RRT^X plan in dynamic scenario 31

3.6 Evolution of RRT^X graph for dynamic scenario with vertices (red), edges (dark blue), and obstacles (green), planned path (light blue), and executed path (black) 31

4.1 2013 International ProStar 122+ with sleeper cabin 33

4.2 Brake pedal pulley diagram with (A) brake pedal, (B) disengagement safety switch, and (C) throttle 34

4.3 Sensor and actuator system communication diagram 34

4.4 Novatel SPAN system diagram 36

4.5 American Truck Simulator interface 37

4.6 ATS communication flow diagram using ROS wrapper and virtual joystick 38

4.7 Comparison of open loop longitudinal response of ProStar 122+ and ATS truck 39

4.8 Comparison of acceleration delay of ProStar 122+ and ATS Truck 40

4.9 Longitudinal control flow chart 42

4.10 Longitudinal tracking performance of ProStar 122+ and ATS truck 44

4.11 RRT^X paths of the rear axle of the truck with single obstacle shown in red 45

4.12 Execution of lane change with single static obstacle in simulation and real life 46

4.13 Aerial shot of the inspiration for the lane change scenario from Google Maps [2] 47

4.14 ProStar executing lane change using RRT^X with obstacles moving at constant velocity, where (a) is the initial planning stage, (b) is the start of lane change, (c) is maintaining lane near a obstacle, (d) is the final configuration..... 49

4.15 Initial and final RRT^X Graph for Lane Change with obstacles moving at constant velocity with vertices (red), edges (dark blue), and obstacles (green) 50

4.16 Aerial shot of the inspiration for the seaport scenario from Google Maps [3] 51

4.17 ProStar navigating seaport using RRT^X with static obstacles 53

4.18 Execution of seaport with static obstacles in simulation and real life 54

4.19 Evolution of search graph for seaport with static obstacles 54

4.20 System struggles to execute Hybrid A* with discontinuous curvature at high speeds. 55

4.21 Curvature and sharpness of Hybrid A* path for seaport maneuver 56

4.22 ProStar executing lane change using RRT^X with obstacle that suddenly changes lane, where (a) is the initial planning stage, (b) is adjusting plan due to unforeseen lane change, (c) is start of new lane change obstacles, (d) is the final configuration... 59

4.23 Evolution of search graph for lane change with vehicle that unexpectedly changes lanes..... 60

4.24 ProStar navigating seaport using RRT^X with limited sensor range..... 62

4.25 Evolution of search graph for seaport with limited sensor range 63

LIST OF TABLES

TABLE Page

1.1 Comparison of search-based motion planners. Adapted from [1] 5

4.1 2013 International Prostar 122+ vehicle specifications 33

4.2 Pose uncertainty from SPAN system with and without wheel speed information 36

4.3 Performance of motion planners for lane change with single static obstacle 44

4.4 Motion planner performance for lane change with obstacles moving at constant velocity 48

4.5 Motion planner performance for seaport with static obstacles 52

4.6 Motion planner performance for lane change with car that unpredictably moves 58

4.7 Motion planner performance for seaport with limited sensor range 61

1. INTRODUCTION AND OVERVIEW

1.1 Introduction

Over the past few decades, researchers have worked towards developing autonomous systems that can be used in everyday transportation. With the emergence of new sensor, hardware, and software technologies, the goal of self-driving vehicles is now on the brink of becoming a reality. Following the success of the DARPA Grand and Urban Challenges in the late 2000s [4, 5], the world had a resurgence of interest in a future involving self-driving cars. Both automotive and technology companies are racing to see who can be the first to develop a safe and reliable self-driving fleet for the market.

The move to self-driving cars is in the interest of the general population. It is predicted that widespread autonomous vehicle adoption will reduce the approximate 1.35 million deaths that occur annually primarily due to human error when operating a vehicle [6]. With that said, autonomous driving in urban environments with dynamic elements like vehicles, pedestrians, and cyclists is still an ongoing field of research with many difficult challenges that do not yet have a clear solution.

This thesis discusses a potential solution to a subset of the self-driving vehicle task. More specifically, this thesis presents an approach to motion planning for truck and trailer systems that can generate local kinematic paths in dynamic environments with little a priori information. This approach is evaluated in both simulation and on an actual truck system to verify its feasibility and real-time performance.

This rest of this chapter details the motivation for this research into motion planning for self-driving vehicles, and more specifically, the need for truck and trailer motion planning. Next, a review of literature regarding recent efforts into this research area will be presented to outline alternative approaches and inspirations for this thesis's contributions to the field. Lastly, this chapter ends with a brief overview of the rest of the thesis.

1.2 Background and Motivation

According to the United States Department of Transportation, truck and trailer systems accounted for about 62.7% of all freight between North American countries in 2018 [7]. Due to the extreme driving hours and potential cost savings, automotive companies are looking for technology that can automate these vehicles as they complete their transportation routes. In order to properly navigate, plan, and react to their dynamic environments, autonomous systems need to be equipped with the proper tools and algorithms to ensure safe deployment for all stakeholders.

Since the degree to which a vehicle can be automated varies, the SAE J3016 standard has been widely adopted to describe the level of automation on a scale from 0 to 5 [8]. This thesis will focus on Level 2 through Level 4, which spans from advanced driver assistance for situations like a lane-change maneuver, to situations where the vehicle is driving fully autonomously with the assistance of a human safety operator.

Automated trucking has seen both academic and industry sponsored research over the past two decades. Primarily, this research has focused on Level 1 and 2 truck platooning through projects like SARTRE, GCDC, PATH and SCANIA [9, 10], There has be relatively less research into truck motion planning in urban and unstructured environments. This has mainly been due to the complex nature of the planning problem for the truck and trailer system, along with a lack of hardware and software to implement the solutions in real-time. With that said, due to the technological advancements over the past few years, there has been a significant push from industrial players like Embark, Tesla, and Waymo to focus on Level 4 automation for trucking [11]. These companies see a growing incentive to reduce human labor costs by developing algorithms that can handle most of the transportation process from loading and unloading to highway driving. Furthermore, these companies have seen promising initial results with Embark operating over 124,000 miles between 2017 and 2018 with only 1 disengagement per 1,392 miles in Q4 of 2018 [12]. On a more public note, Plus.ai's Level 4 truck and trailer system was able to haul butter 2,800 miles across the United States while only stopping at federally mandated increments [13]. With the slowed market adoption of Level 1 and 2 platooning due to overestimation of potential fuel savings [14, 9, 15],

there is a growing need for more research into robust motion planning algorithms for truck and trailer systems to assist in Level 4 deployment.

Research on local motion planning for unmanned systems is being conducted throughout the world to further advance our ability to push autonomous vehicles into increasingly unknown and sophisticated areas. Because of their nonlinear dynamics, delayed actuation, and large physical dimensions, navigating tight spaces with truck and trailer systems in dynamic environments can be quite a difficult task. Motion planning algorithms attempt to solve this optimization problem in the presence of unknown obstacles by finding a feasible sequence of system configurations to achieve a desired goal condition while typically minimizing a defined cost metric [16].

1.3 Autonomous System Architecture

Since self-driving vehicles need to perform such complex tasks, one approach to designing the overarching system architecture is to modularize the system into subsystems that each solve a specific self-driving objective. This modularization assists with development by allowing engineers to isolate issues and design metrics for each solution. Although there are many approaches to the control architecture of autonomous vehicles, a general architecture is shown in Figure 1.1. Sensors measure and obtain information about the vehicle's internal state as well as the environmental actors such as pedestrians, vehicles, and other dynamic objects. Common sensors for this are GPS, IMU, LiDAR, radar, wheel/motor encoders, and cameras. That data is then abstracted for obstacle detection, vehicle localization, and mapping. From there, the vehicle takes the desired mission from the user and develops a global plan to solve it. A mission could be anything from the desire to get one vehicle from one city to another, to utilize multiple trucks and trailers to deliver a payload, or even drop off a freight container at a seaport.

That global plan is then broken into several sub-objectives which is then fed into a local planner to generate a feasible trajectory for the vehicle to execute. This trajectory relies on the perception layer of the vehicle to not only ascertain where the vehicle is relative to its environment, but also to inform the planner about the intentions of other actors. In addition, the perception layer can identify, track, and translate the raw inputs from the sensors into synthesized objects and obstacles

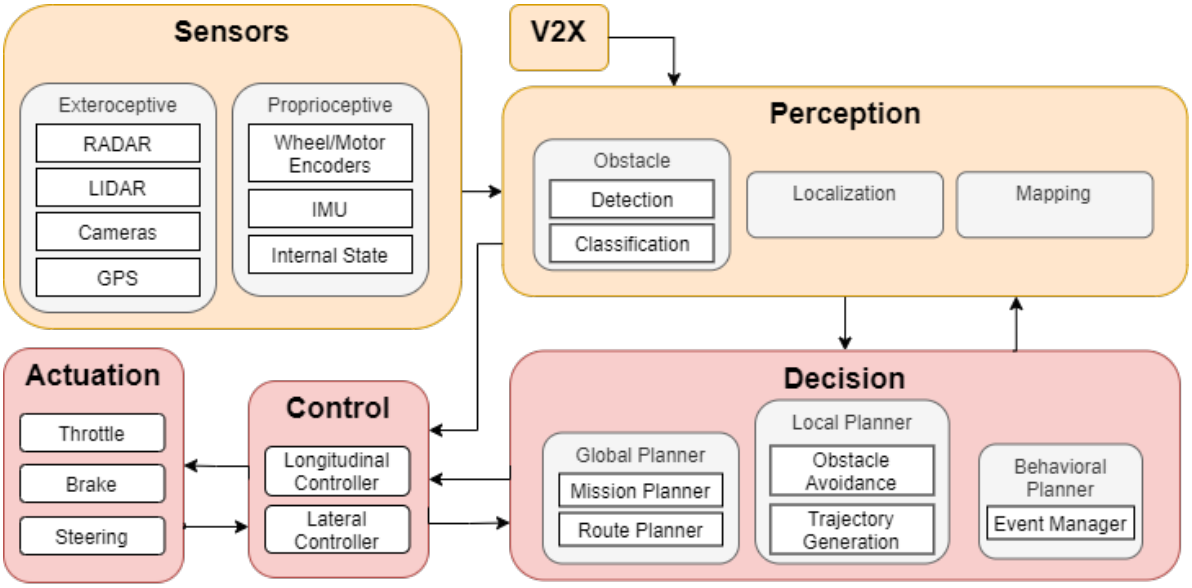


Figure 1.1: General control architecture for automated vehicles based on [1], where red modules indicate the focus of the thesis.

to avoid.

The current vehicle state and global mission is sent to the local planner which then outputs a set of optimal vehicle configurations that will navigate the system to intermediate objectives while avoiding collision with other obstacles and itself. That local plan is then fed into the lower level controllers that determines what type of actuation will most optimally follow the nominal local plan while minimizing a defined error. The controllers are designed to reject disturbances like wind and road characteristics, all while accounting for any modeling error from the higher level planners. In addition they can be designed to improve elements such as passenger comfort and fuel efficiency.

Since the sensor range of the vehicle is limited, dynamic obstacles can appear at any point during the planning process. It is for this reason that the local motion planner must be quick enough to adapt and re-plan a safe alternative path around any obstacles that obstruct the previous plan. This process is constantly repeating during the overall motion planning sequence as the vehicle traverses the workspace. This thesis explores solutions for the issue of continually planning in

unstructured environments for truck and trailer systems by modifying computationally efficient motion planning algorithms for real-time execution in simulation and in real life.

1.4 Literature Review

This section provides a look at current research in the field of autonomous technology as it pertains to vehicle navigation and control. Additionally, resources are presented with increased specificity pertaining to truck and trailer systems.

A. Survey of Motion Planning Algorithms [1]

Paden et. al discussed the state-of-the-art developments for motion planning algorithms and outlines a general approach on executing functions from route planning to vehicle actuation. In this survey, elements of graph based motion planning and control were investigated, focusing on the performance and results of each approach. In terms of motion planning algorithms, when comparing sampling based motion planners to an approach like Hybrid A*, the main differences lie in the completeness and optimality they provide, which is outlined in Table 1.1, as both approaches have seen successful implementations. In addition, anytime planners, like RRT and RRT*, present an ability to quickly identify an initial, suboptimal path, and then incrementally improve the path with additional time. In terms of lateral controllers, while the front wheel based feedback approach is limited to forward-only driving, it performs better in highway driving and parking maneuvers when compared to an approach like pure pursuit.

Table 1.1: Comparison of search-based motion planners. Adapted from [1]

	Completeness	**Optimality**	**Differential Constraints**	**Anytime**
A*	Resolution Complete	Resolution Optimal	No	No
Hybrid A*	No	No	Yes	No
RRT	Probabilistically Complete	Suboptimal	Yes	Yes
RRT*	Probabilistically Complete	Asymptotically Optimal	Yes	Yes

B. Sampling-Based Algorithms for Optimal Motion Planning [17]

Karaman and Frazzoli expanded upon the works of LaValle [16] by developing RRT*, an asymptotically optimal sampling based motion planner that guarantees probabilistic completeness. Similar to its predecessor RRT, which finds a feasible path for the system, RRT* incrementally builds a tree of nodes from the root towards the goal configuration of the system. Unlike RRT, RRT* maintains a hyberball around each generated node in order to extend and rewire tree connections based on their cost, all while avoiding collisions with static obstacles. One of the benefits of both of these planners is that unlike the A* algorithm [18], RRT and RRT* can plan in the state space of the vehicle and can therefore impose kinematic constraints on the system. In addition, these sampled based planners are inherently biased to explore larger Voronoi regions to sample all potential states.

C. RRT^X: Motion Planning in Environments with Unpredictable Obstacles [19]

Otte and Frazzoli further expanded upon RRT* by developing RRT^X. This motion planner accounts for unpredictable, dynamic obstacles, thus allowing for real-time replanning and navigation, similar to D* [20]. The algorithm is able to achieve asymptotic optimality and the same amortized per iteration runtime as RRT* by selectively cascading its rewire operation to the affected children branches when there are observed changes in the graph. This is done by ensuring the graph is ϵ-consistent and by maintaining running sets of incoming and outgoing neighbors for each node . Unlike D*, which works in the discretized workspace of the system, RRT^X can plan in the state space of the vehicle. Due to the limited turning radius of the truck and trailer system, these kinematic constraints become critical in generating a feasible path for the vehicle to execute [19]. Furthermore, the algorithm is rooted at the end configuration which allows for the use of the same graph as the vehicle's state evolves throughout navigation.

D. Closed loop RRT for Car with a General 2-trailer Configuration [21]

Holmer implemented a closed loop RRT approach to motion planning on a 2-trailer system to execute parking maneuvers. Since the system is unstable in the reverse motion, a closed loop

controller locally stabilizes the trailer during the motion planner process based on the system's kinematics. Once locally stabilized, the RRT algorithm can then simulate the motion of the closed loop system as it samples potential configurations throughout the search space. This approach is an adaptation of [22], which placed in the 2007 DARPA Grand Challenge. The path generated has no guarantees of optimality, but adopts many of the benefits of other sampling based approaches while delivering practical results in small scale testing with an average success rate of 98.7%.

E. A Path Planning and Following Framework for a General 2-trailer Configuration [23]

Ljungqvist, et al. proposed a lattice based approach to solving the local motion planning problem for a general 2-trailer system. Utilizing both forward and reverse trailer motion, they were able to demonstrate that their lattice based planner could effectively generate kinematically feasible paths for real-world parking maneuvers that avoided static obstacles. By discretizing the motion planning problem, they were able to efficiently solve for the system's optimal control problem with precomputed motion primitives that were generated offline. Additionally, by using the back axle of the trailer as the reference point, their team demonstrated the ability to accurately control the position of the system in reverse.

F. A Path Planning Algorithm Based on Hybrid A* for Trailer Truck [24]

Sakai proposed a Hybrid A* approach to solving the local path planning problem for a truck and trailer system. His work was heavily influenced by that of Dolgov et al. [25] who had initially proposed the underlying Hybrid A* algorithm. Hybrid A* planning is an adaption of A* that imposes kinematic constraints on the system by discretely sampling inputs of the vehicle. As the planner extends to a new undiscovered node, the steering function translates the vehicle to a kinematically feasible location inside the grid cell. That location is then referenced for all future expansions. Sakai utilized this framework to solve complex parking maneuvers with static obstacles for a truck and trailer system by adding the trailer angle to the state space and its associated jack-knife cost.

1.5 Contributions

Throughout the process of this research, and the development of local motion planning for truck and trailer systems, several methods originated through personal contributions. Outlined below are individual contributions made during the research of this thesis.

- Developed sample-based motion planning approach for truck and trailer system based on propagating trailer angle along Dubins path
- Integrated approach with existing sample-based motion planning algorithms, RRT* and RRT^X, for real-time replanning in the presence of obstacles that unexpectedly appear, disappear, and change position
- Implemented path planning and following architecture for real-time execution in realistic scenarios
- Evaluated motion planning algorithms against Hybrid A* in both simulation using American Truck Simulator and on an International ProStar truck
- Assessed sensor uncertainty and tracking error to tune motion planning and controller parameters that further ensure safe navigation in structured and unstructured environments
- Created, tested, and iterated on proof of concept testing platform for local motion planning on truck and trailer systems

1.6 Assumptions

The following assumptions are made by this thesis regarding the development of the truck and trailer motion planner.

- There are sensors and perception to provide estimates of nearby obstacles' pose and velocity.
- The vehicle is operated at low enough speeds such that the inertial effects are not dominant.
- The vehicle's motion is predominantly planar.

- The vehicle is operated in conditions such that there is negligible tire slipping or skipping.
- The vehicle is constrained to forward only motion.
- There is a global mission being provided from a human operator or a geometric planner.
- There is a feedback controller that can account for the neglected vehicle dynamics during planning and generate a bounded lateral error.
- There is a bounded uncertainty associated with the vehicle's current pose and orientation.

1.7 Overview

The remainder of this thesis is broken into chapters that establish the fundamental basis used to develop the motion planning algorithms. First, the wheeled system modeling equations and subsequent assumptions will be explained as they pertain to two axle vehicles, and more importantly, how they compare to a truck and trailer system. Additionally, methods to simplify the system model for computational efficiency will be discussed. Then, the overarching motion planning problem will be introduced and defined for the scope of this research. Promising search-based motion planning methods will be evaluated as well as their fundamental subroutines will be established. Then, a simulated scenario will combine the vehicle model and the proposed motion planning approach to provide an intuitive understanding of the overall objective the planner is trying to achieve.

The Results and Discussion chapter initially focuses on how the algorithms were executed by first detailing the simulation and real life platforms utilized for evaluation. This includes sensor uncertainty quantification as well as how the feedback controllers were chosen and iterated upon. From there, the testing process and metrics will be detailed as the algorithms are evaluated in multiple static and dynamic scenarios. Finally, the conclusions will describe the significance of the findings and how they could impact future motion planning development for truck and trailer systems.

2. KINEMATIC VEHICLE MODELS

In order to define the motion planning problem that the algorithm is trying to solve, one must first understand how to model the system they are trying to control. For that reason, this section will establish models typically used for a four-wheeled vehicle. This includes the kinematic bicycle model as well as the Dubins car model. From there, the kinematic model will be extended for a truck-trailer system. It should be noted that this section is only meant to be a brief overview of the approach taken for this thesis and a more detailed explanation can be found in [16] [26].

2.1 Nonholonomic System

Since wheels are not intended to move sideways, a common assumption made when deriving a model of a wheeled system is that it rolls without slipping. This assumption is typically only valid on dry roads and at low speeds with low lateral forces [16]. This velocity constraint restricts the motion of the system in lateral directions which means that the system is underactuated and nonholonmic.

Considering the configuration space, $\mathcal{C}$, of the system as the smooth manifold of all possible system configurations, $q \in \mathcal{C}$, one can express the implicit constraints imposed on the system with Equation (2.1) [16]. A specific class of nonholonomic constraints where the linear velocity constraint cannot be integrated explicitly is expressed in Equation (2.2), where k is the amount of constraints and is less than the dimensions of the manifold, n. These are known as *Pfaffian* constraints and are linearly independent [16].

$$g(q, \dot{q}) = 0 \tag{2.1}$$

$$\sum_{i=1}^{k} g_i(q)\dot{q}_i = 0 \tag{2.2}$$

To observe how the input, *u*, affects the system, the parametric representation can be generated utilizing the constraints expressed in Equation (2.2) to form Equation (2.3) [16]. This representa-

tion is used by the following vehicle models to derive the equations of motion.

$$\dot{q} = f(q, u) \tag{2.3}$$

2.2 Kinematic Bicycle Model

Kinematic models are typically used at the planning level because their low amount of state-space dimensions provide quick real-time performance when compared to the more complex dynamic models with relatively large dimensional state spaces. In addition, they typically require less a priori information regarding the inertial and friction components of the system. Since the model only considers kinematic constraints, it is assumed that the lower level controller can account for the neglected dynamic components. This assumption is only appropriate at low speeds, on dry roads, where the inertial effects are typically negligible [16].

The bicycle model is one of the most pervasive models used for the planning and control of wheeled systems. In terms of kinematics, the simple bicycle model is constrained by its limited steering actuation of the front axle, which makes it impossible for the vehicle to make any instantaneous turns that are smaller than the minimum turning radius of the vehicle. Consider the vehicle denoted in Figure 2.1. Constraining the system to planar motion, the configuration space is $\mathcal{C} = \mathbb{R}^2 \times \mathbb{S}$, where $q = (x, y, \theta_0)$.

In order to derive the equations of motion, one must look back to Equations (2.2) and (2.3). Given a small time increment, the *Pfaffian* constraint shown in Equation (2.4) must be satisfied, where x and y are the Cartesian coordinates of the rear axle of the car [16].

$$-\dot{x}\sin\theta_0 + \dot{y}\cos\theta_0 = 0 \tag{2.4}$$

In addition, if one assumes Ackerman steering is obeyed by the front axle, then a steering angle, δ, can represent the average of the inner and outer front wheel angle [27]. From there, one is able to derive a relation between the angle of the car, θ_0, and δ to form Equation (2.5), where d_0 is the

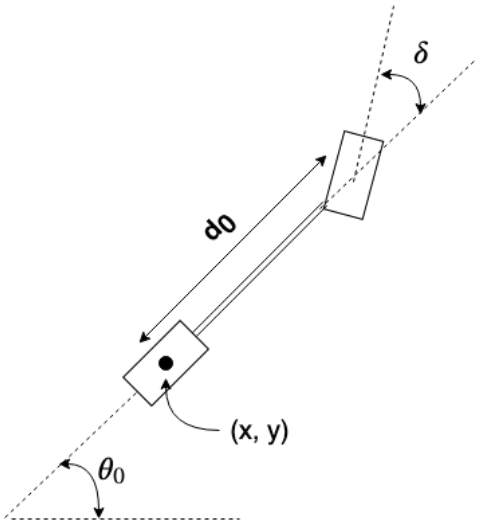


Figure 2.1: Kinematic bicycle model for car-like system

wheelbase of the car and v is the longitudinal velocity of the vehicle.

$$\dot{\theta}_0 = \frac{v}{d_0} \tan \delta \tag{2.5}$$

Combining this information, the full state of the system can be defined using the 3 state equations below.

$$\begin{aligned} \dot{x} &= v \cos \theta_0 \\ \dot{y} &= v \sin \theta_0 \\ \dot{\theta}_0 &= \frac{v}{d_0} \tan \delta \end{aligned} \tag{2.6}$$

2.3 Dubins Car Model

The Dubins car model simplifies the kinematic bicycle by limiting the motion to only the forward direction with a constant speed and minimum turning radius. The Dubins path generated is proven by Pontryagin's maximum principle to be the shortest curve between two vehicle configurations with prescribed tangents [28]. The optimal path is a choice of 6 possible combinations of a right turn, straight path, and left turn (RSR, LSL, RSL, LSR, LRL, RLR). Each turn is performed at

the car's maximum steering angle (usually 25-40°) which is related to the minimum turning radius by Equation (2.7). An example of a RSR turn is shown in Figure 2.2. Due to its computational efficiency, the path is pervasive for planning for wheeled systems [1].

$$r_{min} = \frac{d_0}{\tan(\delta_{max})} \tag{2.7}$$

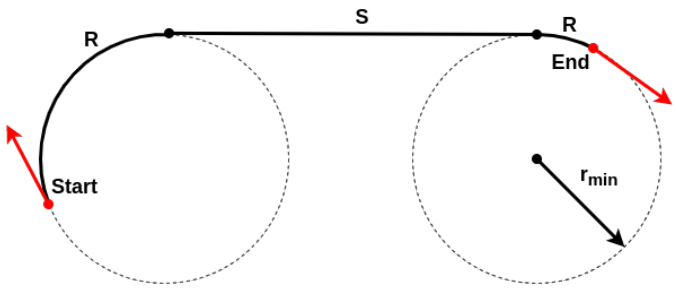


Figure 2.2: Dubins RSR path for car-like system with minimum turning radius (r_{min})

This approach does have some drawbacks in terms of actual feasibility. In all cases, since the path generated has discontinuous curvature, the steering actuation is assumed to be instantaneous, i.e. the steering rate has no limitations. This affects how systems, like the truck and trailer, with limited steering rates execute the path. There are several approaches to resolve this issue. One option is to generate clothoids that sacrifice some optimality but result in paths with continuous curvature and its derivative, as seen in [29]. Another approach is to smooth the Dubins path with a cubic spline. The spline path has continuous curvature required for execution with a limited steering rate but also releases the limited steering angle constraint imposed by the Dubins curve. Additionally, one needs to verify that the spline does not overfit the Dubins path or that the new spline path does not collide with any obstacles. This effect of instantaneous curvature change will be further investigated empirically in Section 4.4.3.1.

Another inherent drawback to the Dubins approach falls with the constraint on the vehicle to

only move forward. With the Dubins approach, there can be cases where the vehicle might get stuck in a local corner with no feasible way to maneuver away [30]. By also considering reverse motion, the Reeds-Shepp model can efficiently navigate more potential trajectories through the search space [31]. With all of that being said, it is assumed for the purposes of this thesis that the scenarios can be executed using only forward motion, and that the sensors on the vehicle have enough range to detect obstacles and prevent the vehicle from getting stuck in a corner.

2.4 The Truck and Trailer Model

The truck and trailer model is an expansion of the kinematic model for a simple car with an added rigid body connected to the center of the truck's rear axle, making $\mathcal{C} = \mathbb{R}^2 \times (\mathbb{S}^1)^2$. The same nonholonomic constraints regarding steering actuation are imposed from the simple car model, but unlike two axle cars, the angle between the truck and trailer is also limited in order to prevent self-collision, known as jack-knifing. Unlike the general n-trailer case presented in [32], it is assumed that the trailer pivots from the center of the rear axle of the vehicle with negligible off-axis hitching. This assumption is typically valid for truck and trailer transportation vehicles and simplifies the equations of motion for the system. In addition, there is no active trailer steering, i.e. the trailer's rear axle is rigid to the body. Just like the kinematic model for the car, the truck and trailer model is only valid for low speed maneuvers due to the inertial effects. Furthermore, the model has been empirically observed to start diverging from the actual response of the system when the truck and trailer angle have a difference that is more than 7° [33]. Keeping these constraints in mind, the full state of the system can be defined using the 4 state equations below in Equation (2.8), where d_1 is the wheelbase of the trailer and θ_1 describes the angle of trailer [16]. These kinematic equations must be obeyed by any motion planning algorithms in order to generate feasible plans.

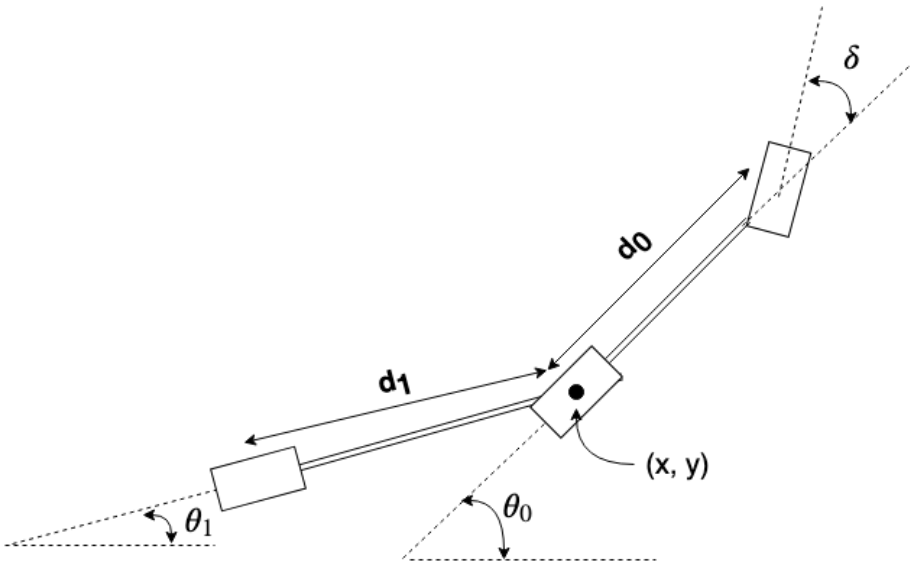


Figure 2.3: Kinematic model for truck and trailer system

$$\begin{aligned} \dot{x} &= v\cos\theta_0 \\ \dot{y} &= v\sin\theta_0 \\ \dot{\theta}_0 &= \frac{v}{d_0}\tan\delta \\ \dot{\theta}_1 &= \frac{v}{d_1}\sin\left(\theta_0 - \theta_1\right) \end{aligned} \tag{2.8}$$

2.5 Velocity Profile

Since the velocity of the system is linearly proportional to all state variables in Equation (2.8), in order to numerically solve for the optimal control problem between a starting and final vehicle state, a common practice is to scale the time, thus separating the velocity from the motion planning problem [34]. Using this property, one can separate the motion planning problem into a path planner that generates a set of feasible vehicle configurations and an associated velocity profile, which is generated based on the acceleration limitations of the vehicle. It should be noted that this approach is only valid when the kinematic model is valid, i.e. low speeds with no tire slipping.

For this thesis, the velocity profile takes into consideration the maximum and minimum longitudinal acceleration achievable through throttle and braking, respectively. In addition, it considers

the maximum desired lateral acceleration to generate a series of desired velocities for the associated path. Although the longitudinal acceleration bounds are determined empirically using open loop actuation, a lateral acceleration limit of 2.0 m/s^2 is imposed in order to ensure passenger comfort and minimize any rolling effects on the system [35]. The lateral acceleration of the vehicle, a_y, can be related to the path using Equation (2.9). By inserting Equation (2.7) into Equation (2.9), the velocity constraint in Equation (2.10) can be imposed in addition to the longitudinal acceleration requirements which assists in the generation of the a motion plan for the vehicle.

$$a_y = \frac{v^2}{r} \tag{2.9}$$

$$v_{cmd} \leq \sqrt{\frac{d_0 * a_{y,max}}{\tan \delta}} \tag{2.10}$$

The following chapter will detail some approaches to the general motion planning problem and provide the algorithms that were chosen to integrate with the truck and trailer model for this thesis.

3. MOTION PLANNING

Now that the system models have been established, this section will initially focus on defining what problem the motion planner is trying to solve. Then, it will introduce a general framework that is followed by most search-based motion planners, as well as the fundamental functions they all share. At the end of the section, a graph-based approach called Hybrid A* and sampling-based approaches RRT* and RRT^X will be briefly introduced along with an explanation regarding how they were modified to handle the trailer condition.

3.1 Problem Formulation and Approach

By modeling the wheeled vehicle as a nonlinear, time-invariant system, one can utilize Equation (3.1) to express the vehicle's motion, where the state, $x(t)$, lies on an n-dimensional smooth manifold described in Section 2.1. For wheeled systems in the real world that are physically constrained due to limits in actuation and configuration, both the state, $x(t)$, and the input, $u(t)$, are physically limited to the state space, $\mathcal{X}$, and the input space, $\mathcal{U}$, respectively.

$$\dot{x}(t) = f(x(t), u(t)), \quad x(t_0) = x_0, \quad x(t) \in \mathcal{X} \subseteq \mathbb{R}^n, \quad u(t) \in \mathcal{U} \subseteq \mathbb{R}^m \tag{3.1}$$

Given an initial vehicle state, x_0, and goal state x_g, the motion planner attempts to find a feasible motion plan that navigates through the free space, $\mathcal{X}_{free}$, where $\mathcal{X}_{free} = \mathcal{X} \setminus \mathcal{X}_{obs}$ and is typically non-convex. The free space is usually determined by discovering the possible states that do not collide with any obstacles or itself in the search space. These obstacles can be static or dynamic, meaning that as the vehicle traverses the space, previously valid plans can be invalidated due to dynamic obstacles. Based on this information, one can formally define the optimization problem as an OCP in Equation (3.2), where L is the *running* or *Lagrangian cost*. It should be noted that

this OCP assumes that both the initial and final state are collision free.

$$\begin{aligned} \underset{u(\cdot),\, t_f}{\text{minimize}} \quad & \int_{t_0}^{t_f} L(x(t), u(t), t)\, dt \\ \text{subject to} \quad & \dot{x}(t) = f(x(t), u(t)), \\ & x(t_0) = x_0, \;\; x(t_f) = x_g, \\ & x(t) \in \mathcal{X}_{free}, \;\; u(t) \in \mathcal{U} \end{aligned} \tag{3.2}$$

Although the OCP in Equation (3.2) can be directly solved numerically with a optimal control solver like ACADO [34], it generally takes too long to execute at the rate required for mobile robotic navigation. Even with no obstacles, the solver must still evaluate the nonlinear dynamics of the system to get to its final goal state. In the presence of obstacles, the problem becomes even more difficult since $\mathcal{X}_{free}$ is usually non-convex, which makes the initial guess of the solution imperative to finding a feasible plan. Furthermore, representing $\mathcal{X}_{free}$ analytically using the vehicle's sensors is not a trivial task even with the assistance of precomputed maps [1].

It is for this reason that motion planning algorithms are typically relied upon to find a solution to Equation (3.2). Motion planning, in the general sense, is utilized by various fields of research and can subsequently have various definitions. For the purposes of this thesis, the focus will be on local motion planning which typically considers the system dynamics and constraints to find a feasible path that minimizes a determined cost metric to reach its goal while avoiding obstacles. This path is then sent along with a desired velocity profile to a lower level controller to execute and track the desired motion plan. Typically, a higher level geometric planner that neglects the system dynamics will feed the local planner with intermediate waypoints to navigate its overall mission. This process is constantly repeating as the vehicle traverses the environment, adjusting to account for changes in the estimated position of dynamic obstacles, system modeling inaccuracies, and changes in the overall mission. A diagram describing this segmentation is shown Figure 3.1.

By modularizing the planning problem, the generated path can become suboptimal in terms of the global mission. With that said, the approach has been proven quite effective in actual de-

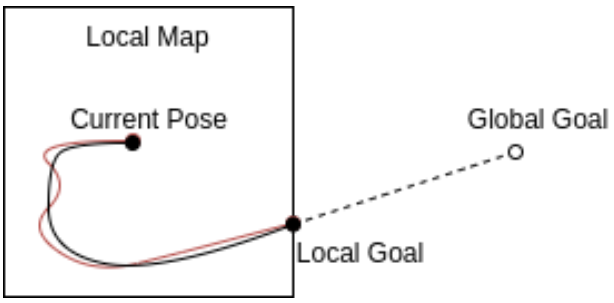


Figure 3.1: Hierarchical approach with local plan (black), controller execution (red), and global path (dashed)

ployments [1]. One of the major benefits of this approach is due to the computational time and complexity of solving the non-convex, optimal control problem for large areas. By planning in the local area, the motion planning algorithm is able to generate feasible paths at the rate needed for mobile platforms to react to changes in the environment, which tends to be 0.1-50Hz [1]. Attempting to solve the global mission without segmentation is still an active area of research but cannot typically provide a plan in the time required for this thesis.

Before introducing the motion planning algorithms, it is important to describe the fundamental functions that each algorithm will utilize to develop its solution. These functions are applicable to both holonomic and nonholonomic systems and assists the planner in finding the time optimal solution in unstructured environments.

3.2 Steering Function

The motion planners discussed in this thesis rely on a steering function that can navigate the planner between two intermediate system states. It is imperative that the steering algorithm obeys the system dynamics in order to create a feasible trajectory. In most cases, this steering function does not consider obstacles and a separate function later checks for collision [16]. Depending on the planner, the steering function will either generate trajectories in real-time or rely on precomputed motion primitives that were generated offline. Since the graph-based algorithm Hybrid A* uses a discretized representation of the state space, it can utilize the offline trajectories from any intermediate point by exploiting the system's position invariance. On the other hand, since the

sampling-based planners are not discretized, the amount of storage required for all of the possible motion primitives makes it difficult for online use. Since this thesis is focused on relying on as little a priori knowledge of the system dynamics as possible, only real-time steering functions are considered.

In order to solve the optimal solution to the boundary value problem between two intermediate states, the optimal control problem in Equation (3.3) must be solved. The primary differences between this equation and Equation (3.2) is that the steering function does not consider obstacles, i.e, $x(t) \in \mathcal{X}$, and that while L_{str} is usually equivalent to global running cost function, L, some steering functions optimize based on different criteria than the global function.

$$\begin{aligned} \underset{u(\cdot),\, t_{str}}{\text{minimize}} \quad & \int_0^{t_{str}} L_{str}(x(t), u(t))\, dt \\ \text{subject to} \quad & \dot{x}(t) = f(x(t), u(t)), \\ & x(0) = x_0, \quad x(t_{str}) = x_f \\ & x(t) \in \mathcal{X}, \quad u(t) \in \mathcal{U} \end{aligned} \tag{3.3}$$

In an attempt to solve the 4D OCP stated in Equation (3.3) with the equations of motion outlined in Equation (2.8), Casadi [36], a modern numerical solver, was utilized. With some initial testing, the solver was able to find a solution within 20-300ms for randomized truck-trailer poses within a 10 meter radius. Since this steering operation is performed thousands of times during the planning stage, a significantly quicker steering function is needed for real-time planning.

For this thesis, a version of a Dubins curve [37] was used to connect intermediate states. Constraining the vehicle to only forward motion, the Dubins curve is proven via Pontryagin's minimum principle to generate a time optimal solution for the kinematic bicycle model [28] operating at a constant speed. Although this steering function is more computationally efficient than using a online numerical solver, it does have some drawbacks. Since the Dubins curve does not consider the trailer angle during its trajectory generation, looser constraints must be imposed on the intermediate states of the motion planner.

More specifically, the approach taken for this thesis is to propagate the trailer angle through the intermediate states using the kinematic model outlined in Equation (2.8) as the system traverses the Dubins path. Then, in order for the final solution to be valid, the final trailer angle must be within a set margin of the desired trailer angle. In addition, while the system is traversing each path, the difference between the truck and trailer angles must not exceed a set threshold in order to prevent self-collision. This approach is inspired by how humans typically prioritize the truck orientation and location when deciding their forward driving maneuvers, like during lane changes, as the trailer lags behind the truck.

3.3 Heuristic and Cost Function

In order to to guide the planning algorithm when navigating between intermediate states, a heuristic is utilized. There are many different approaches to developing an optimal heuristic that can significantly depend on the system dynamics and the environment. Accordingly, in order for a heuristic function to be admissible, it must never overestimate the cost associated with traversing between states. This constraint is stated by Equation (3.4), where $c(x, y)$ represents the cost associated with traversing between states and $h(x, y)$ represents the heuristic, i.e. the estimated cost between states. Furthermore, in order for a heuristic to be consistent, it must obey the triangle inequality expressed in Equation (3.5) with all of its successors, x_{i+1}, and the goal, x_g. As long as these two conditions are met, then the graph-based motion planners will eventually generate an optimal solution.

$$h(x, y) \leq c(x, y) \tag{3.4}$$

$$h(x_i, x_g) \leq c(x_i, x_{i+1}) + h(x_{i+1}, x_g) \tag{3.5}$$

One approach to generate a heuristic is to precompute a look-up table offline by simulating the motion required to get to nearby states [38]. This method relies on a priori information about the vehicle dynamics but has shown promise with discrete planners due to it's quick performance with nonlinear and complex systems. For this thesis, since the objective is to minimize the path length generated by the planner using as little a priori information about the system as possible, the

Euclidean distance between state variables will be used for the heuristic function. This is expressed in Equation (3.6). Subsequently, the cost, $c(x, y)$, is the distance traversed by the vehicle along the Dubins curve from x to y. Although the heuristic does not always equal the cost function, the heuristic never overestimates the Dubins path length and obeys the triangle inequality. Thus, the metric is admissible and consistent.

$$h(x, y) = \sqrt{(x_\theta - y_\theta)^2 + \sum_{n=1}^{2}(x_i - y_i)^2} \tag{3.6}$$

It should be noted that there are some inherent issues with using the Euclidean distance as a heuristic. Due to the nonholonomic nature of the truck-trailer system with a large turning radius, the metric can significantly underestimate the Dubins path length in scenarios where nearby nodes are within the turning radius of the car. An example is shown in Figure 3.2 where the Euclidean heuristic estimates a much lower cost to navigate from the start to end goal than the actual distance traversed by the Dubins curve. With that said, the heuristic is still valid, and future efforts can be done in order to derive a more appropriate heuristic for these nonholonomic vehicles.

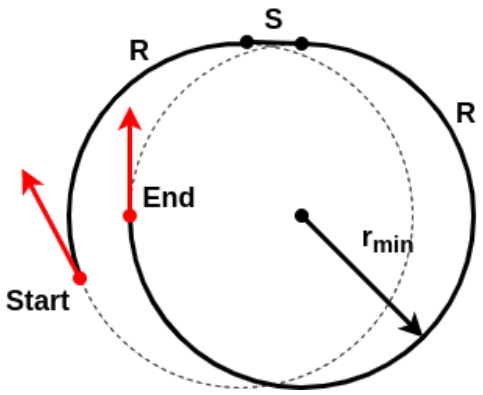


Figure 3.2: Euclidean heuristic underestimates the path cost for nonholonomic vehicle

In the case where time is added to the state-space of the vehicle to account for obstacles moving at a constant velocity, the heuristic function can be rewritten as Equation (3.7) [19]. If one sets velocity limits on the vehicle, and allows for instantaneous changes in the planned velocity of the vehicle, the distance and time associated with traversing the Dubins path can be utilized for the cost function. This is done by allowing each Dubins curve between nodes of the graph to have different constant velocities as long as they do not violate the velocity constraints, otherwise the path cost is ∞. In practice, the velocity limits are set close enough to the nominal desired constant speed in order to ensure that the vehicle's acceleration limits are not violated over the path.

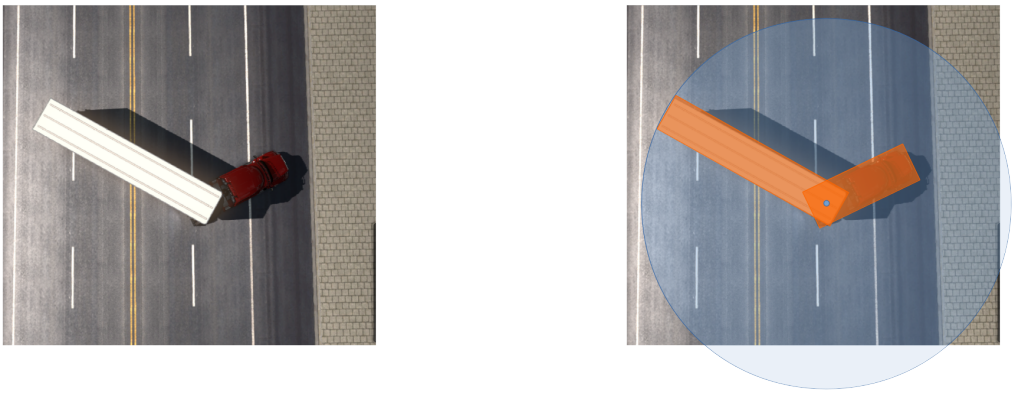
$$h(x, y) = \sqrt{(x_t - y_t)^2 + (x_\theta - y_\theta)^2 + \sum_{n=1}^{2}(x_i - y_i)^2} \tag{3.7}$$

3.4 Collision Detection

Avoiding obstacles is paramount for any motion planner to prevent collisions. It is the role of the collision detection layer to ensure that the configuration being evaluated is within the free space, i.e. $x \notin \mathcal{X}_{obs}$. Since the detection procedure is performed thousands of times during the planning state, an efficient algorithm must be used. Furthermore, there are a few common approaches used for wheeled systems that do not rely on a specific planner.

One approach is to encapsulate the entire system inside of a circle. This approach is quite computationally inexpensive as the truck and trailer headings are not required, but it tends to be over-conservative when trying to drive through narrow passages, especially in the case of the truck and trailer system, where the length is much larger than the width. Another approach is to treat the truck and trailer as two rectangles. This requires consistent knowledge of the truck and trailer heading and is thus more computationally expensive. This being the case, a hierarchical approach is typically used to capitalize off of both methods. This is done by only checking the rectangular oriented boxes if the enclosed circle check fails [1].

In order to implement the collision checking functions in software, the continuous trajectories are typically discretized into a set of states that are then sampled to ensure there is no collision.



(a) Example configuration of vehicle

(b) Bounding regions

Figure 3.3: Two approaches to encapsulating truck and trailer: circle (blue), oriented rectangles (orange)

This step can be avoided in obstacle-sparse environments when using the geometry of Dubins curves. This is because as long as there are no obstacles within the radius of the vehicle and the minimum turning diameter, then the two configurations are collision free. If not, then further steps must be taken in order to ensure that the intermediate set of system configurations do not collide with obstacles.

When considering dynamic obstacles, their trajectories are usually estimated by the perception layer using sensor data from exterior LiDAR, radar, or cameras, and as such, their trajectories are subject to sudden changes. While the static planners assume constant motion relative to the state space of the vehicle, i.e. $\Delta\mathcal{X}_{obs} = \emptyset$, dynamic planners like [19, 20] account for the sudden obstacle changes as the motion plan is traversed by the vehicle, i.e. $\Delta\mathcal{X}_{obs} \neq \emptyset$.

3.5 Search-based Methods

Now that the problem has been defined and the general functions have been established, this section will elaborate on how search-based planners go about solving the motion problem. First, some notation will be established.

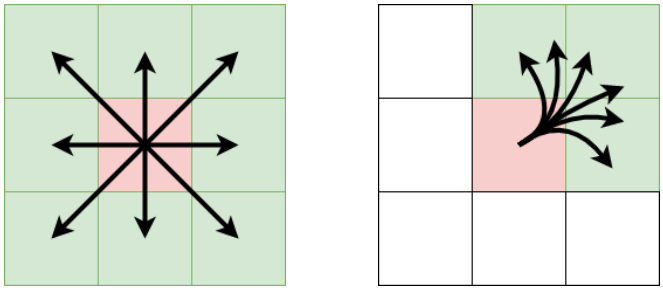
In general, search-based motion planners construct a graph, $\mathcal{G}$, made up of vertices, $\mathcal{V}$, con-

nected with edges, $\mathcal{E}$, such that $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the node set of states and $\mathcal{E}$ is the edge set of motions. For the scenarios in this thesis, the motion planner is attempting to compute a trajectory that connects the initial state, x_{start}, to the goal state, x_{goal}.

Generally speaking, the planners below are initialized at an initial state, x_{init}, that is either x_{start} or x_{goal} with $\mathcal{E} = \emptyset$. As the planners attempt to connect to a new state, x_{i+1}, a candidate for expansion, x_i, is picked based on the cost $c()$ and/or the estimated cost heuristic $h()$. The planners then attempt to steer toward x_{i+1} from x_i using a predefined steering function. If the steering function is able to generate a trajectory to x_{i+1} without collision, the state is added to the graph and the associated cost is updated, i.e. $c(x_{init}, x_{i+1}) = c(x_{init}, x_i) + c(x_i, x_{i+1})$. If $x_{i+1} \in \mathcal{V}$, then the current cost is evaluated against the previous cost in order to determine if it is advantageous for insertion. This process is then repeated until a termination condition is met. Then, all valid solutions are compared based on the defined cost function and the minimum solution is returned if it exists, otherwise no solution is returned.

3.5.1 Hybrid A*

The Hybrid A* algorithm builds off of A* by discretizing the input space in order to deterministically generate kinematically feasible paths for the system to follow. Just like A*, the algorithm discretizes the work space into a grid-like search graph with each cell representing potential configurations. Hybrid A* then assigns each cell with a continuous state of the vehicle that is generated during its initial expansion. This expansion is detailed in Figure 3.4. As more nodes are queued from the open list generated by A*, each of the discretized input commands are simulated using a kinematic model of the system for a predefined time step. If a future node generates a path with a lower cost that ends up in the same cell as the previous trajectory, then the state is updated with the configuration. Because the algorithm operates within the input space, there are no guarantees of optimality or completeness. With that said, this method has proven quite effective at generating trajectories real-time on actual systems [25]. Furthermore, it was successfully implemented on a truck and trailer system in [24].



(a) A* Expansion (b) Hybrid A* Expansion

Figure 3.4: Illustration of how Hybrid A* enforces differential constraints during expansion.

3.5.2 RRT*

Unlike the deterministic approach of Hybrid A*, RRT* randomly samples node configurations to probabilistically determine a feasible path for the system to navigate. The algorithm terminates once sufficient stopping criteria have been met. This is usually after either a set amount of time or nodes are generated. It has been proven effective for holonomic and nonholonomic systems, as it can operate within the state-space of the system to impose nonholonomic constraints. In addition, it is asymptotically optimal. Thus, it will converge to an optimal solution in terms of the cost function as the amount of nodes increases.

The RRT* algorithm, shown in Algorithm 1, incrementally samples nodes to build a directed search tree from the root configuration. Once a random node is sampled, an $extend()$ function is then used to extend from the nearest neighbor node towards the random node and form a new node. This expansion is limited by a set distance in order to assist in navigation in tight spaces. If the trajectory avoids collision, then a vertex representing the new node and an edge representing the path is added into the search graph. A parent node is selected using the $findBestParent()$ function which finds the minimum cost node among its neighbors. The $steer()$ function is then used to generate a path from the parent node towards the new node. During each expansion, a hyperball with a radius defined by Equation (3.8) is utilized by $rewireNeighbors()$ to rewire previously generated tree edges in which the cost to travel from the new node is lower, where n is the number of nodes in the search tree, D is the dimensions of the configuration space, γ is a user

defined hyperball constant, and *r* is the current hyperball radius.

$$r = \gamma(\frac{\log n}{n})^{1/D} \tag{3.8}$$

This rewiring operation replaces the nearby node's previous parental edge with an edge from the new node. Since this thesis will be utilizing the Dubins path, which is an exact and efficient steering function, a goal region is not needed as a criteria for a valid final path. Instead the goal node can be exactly reached. In addition, a sampling bias of 1% towards the goal configuration will be used to bias propagation towards the goal. The trailer's angular configuration is propagated to all future configurations using the $rewireTrailer()$ function along the Dubins path along with the kinematic equations in Equation (2.8) to approximate the trailer angle over the path towards the new node.

Algorithm 1: RRT* Pseudocode

input : Initial Configuration x_{init}
Stopping Time t_{stop}
Expansion Distance Δq
Hyperball Constant γ
output: Search Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$

$\mathcal{V} \leftarrow \{x_{init}\}$;
$\mathcal{E} \leftarrow \emptyset$;
while $t < t_{stop}$ **do**
  $x_{rand} \leftarrow randomNode()$;
  $x_{nearest} \leftarrow nearest(x_{rand})$;
  $x_{new} \leftarrow extend(x_{rand}, x_{nearest}, \Delta q)$;
  if $x_{new} \notin \mathcal{X}_{obs}$ **then**
    $\mathcal{V} \leftarrow \mathcal{V} \cup \{x_{new}\}$;
    $r = \gamma(\frac{\log n}{n})^{1/D}$;
    $\mathcal{X}_{near} \leftarrow \{x \in \mathcal{V} \setminus \{x_{new}\} : c(x, x_{new}) < r\}$;
    $x_{par} = findBestParent(x_{new}, \mathcal{X}_{near})$;
    $\mathcal{E} = steer(x_{new}, x_{par}, \mathcal{E})$;
    $\mathcal{E} = rewireNeighbors(x_{new}, \mathcal{X}_{near}, \mathcal{E})$;
    $rewireTrailer(x_{new})$;
end
return $\mathcal{G}$

3.5.3 RRT^{X}

RRT^{X} is an extension of RRT* that can quickly replan in dynamic environments in the same amortized time as RRT* [19]. Although this thesis will briefly overview the general RRT^{X} algorithm and how it was adapted for the trailer condition, the reader is encouraged to see [19] for a more in-depth explanation. Due to the algorithm's ability to utilize the same expanding tree during replanning, it is able to save computation time by not constantly destroying and rebuilding the same tree as obstacles appear/disappear. By rooting the tree at the end configuration, the algorithm is also able to quickly branch out to the current system configuration as it moves across the workspace.

RRT^{X} evolves sets of neighbors that are directed in (-) and out (+) of respective nodes. While the initial neighbor sets are always remembered in order to ensure that the RRT* solution is always realizable, the running neighbor sets are culled as the hyberball r shrinks. This culling operation is why RRT^{X} is able to maintain $\mathcal{O}(\log n)$ edges for each node. In addition, the algorithm performs rewiring cascades that propagate cost-to-goal information to nodes that are ϵ-inconsistent, where ϵ is user defined. This constraint is expressed in Equation (3.9), where $c(x_{goal}, x)$ cost to reach x_{goal} from x and $lmc(x)$ is the look-ahead estimate of the cost to reach x_{goal} and is defined in Equation (3.10), where $\mathcal{X}^{+}_{near}$ are the outbound neighbors of x.

Algorithm 2 outlines the pseudocode for RRT^{X}. Other than being rooted at the goal state, the main differences when compared to RRT* lie in the $updateObstacles()$, $rewireNeighbors()$, and $reduceInconsistency()$ functions. The $updateObstacles()$ function is called upon to update $\mathcal{G}$ to reflect any changes in the obstacles. If any nodes are affected, then $reduceInconsistency()$ cascades a rewiring operation to all subsequent nodes that break ϵ-consistency. Similarly during the initial $rewireNeighbors()$ operation, any incoming edge cost reductions that break ϵ-consistency are queued for the rewiring cascade.

$$\epsilon > c(x_{goal}, x) - lmc(x) \tag{3.9}$$

$$lmc(x) = \min_{x_{near} \in \mathcal{X}^{+}_{near}} c(x, x_{near}) + lmc(x_{near}) \tag{3.10}$$

Algorithm 2: RRT^{X} Psuedocode

input : Goal Configuration x_{goal}
Stopping Time t_{stop}
Expansion Distance Δq
Hyperball Constant γ
ϵ
output: Search Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$

$\mathcal{V} \leftarrow \{x_{goal}\}$;
$\mathcal{E} \leftarrow \emptyset$;
while $t < t_{stop}$ **do**
  $updateObstacles()$;
  $x_{rand} \leftarrow randomNode()$;
  $x_{nearest} \leftarrow nearest(x_{rand})$;
  $x_{new} \leftarrow extend(x_{rand}, x_{nearest}, \Delta q)$;
  if $x_{new} \notin \mathcal{X}_{obs}$ **then**
    $\mathcal{V} \leftarrow \mathcal{V} \cup \{x_{new}\}$;
    $r = \gamma(\frac{\log n}{n})^{1/D}$;
    $\mathcal{X}_{near} \leftarrow \{x \in \mathcal{V} \setminus \{x_{new}\} : c(x, x_{new}) < r\}$;
    $x_{par} = findBestParent(x_{new}, \mathcal{X}_{near})$;
    $\mathcal{E} = steer(x_{new}, x_{par}, \mathcal{E})$;
    $\mathcal{E} = rewireNeighbors(x_{new}, \mathcal{X}_{near}, \mathcal{E})$;
    $reduceInconsistency()$;
    $rewireTrailer(x_{new})$;
end
return $\mathcal{G}$

It should be noted that since there are dynamic obstacles, there is a chance that a sudden obstacle appearance can prevent the vehicle from finding a feasible solution to the end configuration, especially since the system is constrained to forward motion. In that case, the vehicle will apply its brakes until motion has stopped, all while looking for other potential paths to reach the end goal. Since this is a single threaded planner, one alternative approach would be to delegate another instance to be constantly looking for trajectories that maximize distance from nearby obstacles to

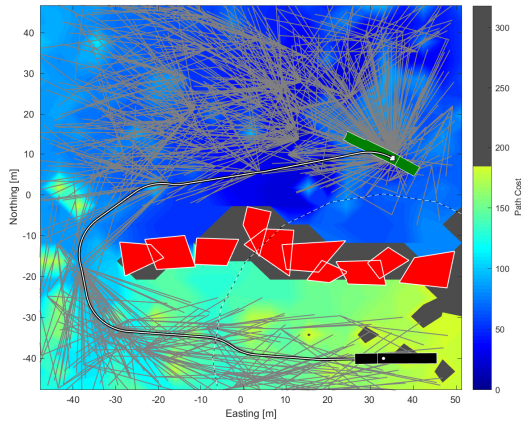
provide a safe escape. Thus, when the main planner fails to find a feasible path, the safe trajectory can be followed.

3.5.4 Example Scenario

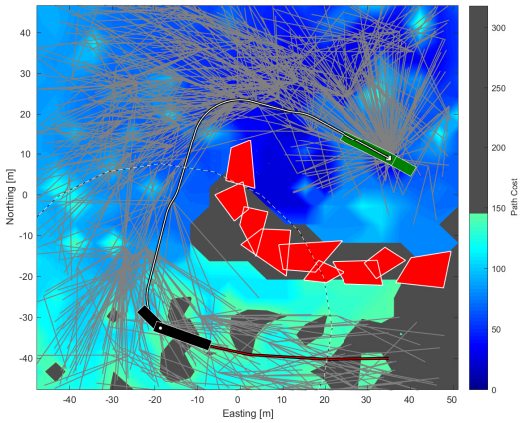
In order to illustrate how the replanning algorithm handles dynamic obstacles, the following scenario is shown in Figure 3.5. During the initial planning stage, the sensors can only see what is inside the radius denoted by the black and white circle hovering around the car. The contour illustrates the path length from the goal that is averaged over each section of the map. The starting pose of the truck is denoted by the two black rectangles while the goal configuration is represented in green. The white path denotes the current optimal subtree being outputted from RRT^X. The straight edges illustrate the connections between nodes of the tree. These are shown in light gray and do not represent the trajectory between the points.

Obstacles both disappear and appear while the vehicle is in motion, and the planner is able to account for the changes by propagating the path cost changes when the ϵ-consistency is broken. The planner is able to continually generate kinematically feasible paths utilizing the Dubins curve with trailer propagation. As seen in Figure 3.5d, the final trailer heading is not exactly the same as the desired goal heading. This goes back to the bounds set by the planner for an acceptable goal region for the trailer.

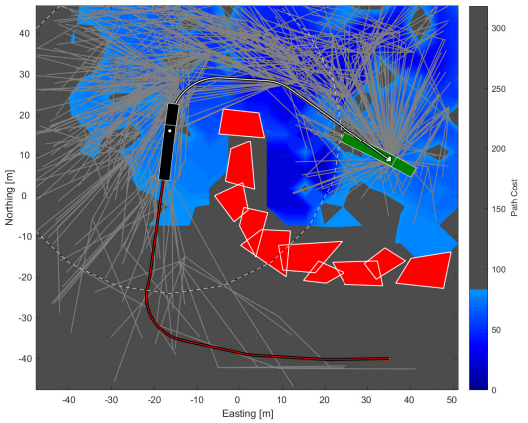
Although Figure 3.5 provides some intuitive understanding of the vehicle's execution of the path in the Northing/Easting plane, one can also look at the other state dimension, θ_0, to see how the search graph expands throughout all of the collision-free configuration space. Figure 3.6 illustrates the initial and final RRT^X graph. Comparing the narrow corridor of the initial vehicle configuration and the final configuration, the graph is significantly denser. This phenomenon is not only due to the geometry of the environment but is also attributed to tunable parameters of the planner like the hyperball constant, the expansion distance, and the heuristic, all of which affect how the algorithm searches throughout the configuration space. There are also some invalidated nodes shown in the middle of Figure 3.6b which reflect the discovery of the vertical obstacles as the vehicle progressed through the space.



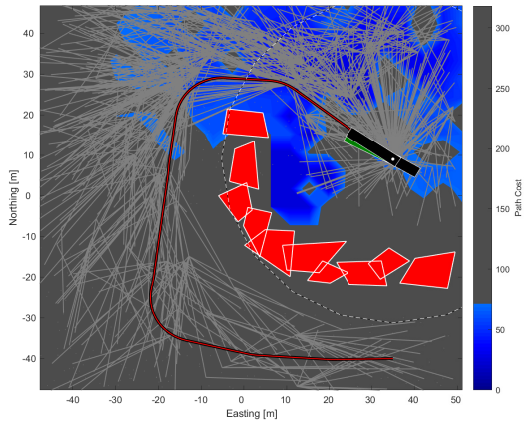
(a) Initial planning stage



(b) Shorter path due to obstacle disappearance

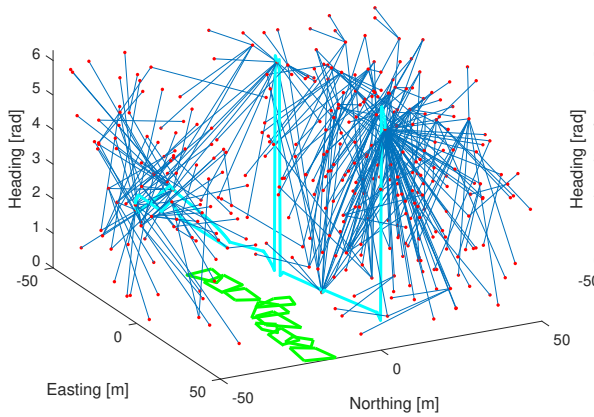


(c) Alternative path due to added obstacles

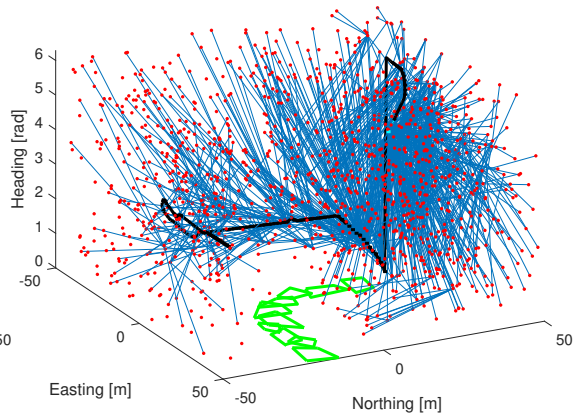


(d) Final configuration

Figure 3.5: Truck and trailer system executing RRT^X plan in dynamic scenario



(a) Initial Planning Stage



(b) Final Configuration

Figure 3.6: Evolution of RRT^X graph for dynamic scenario with vertices (red), edges (dark blue), and obstacles (green), planned path (light blue), and executed path (black)

4. RESULTS AND DISCUSSION

In order to evaluate the motion plan on the actual truck and trailer system, this thesis first establishes the physical system characteristics, the simulation, and the controllers used to execute the planned path. From there, the test case scenarios will be established and the subsequent performance and tracking results will be discussed.

4.1 Evaluation on ProStar 122+ Truck

The hardware platform used for evaluation utilizes a 2013 International ProStar 122+ truck modified with by-wire capabilities as seen in Figure 4.1 in combination with a Novatel GNSS+INS combined system for the truck's current position, velocity, and attitude. The by-wire functionality is provided by a PACMod kit developed by AutonomouStuff which has seen pervasive use within the research community for autonomous vehicles [39]. Other sensors on the truck included a MobilEye detection system, Delphi ESR 2.5 Radar, and Velodyne VLP-32c, but were not used for any of the experimentation presented in this thesis. Instead the perception provided from these sensors was simulated to create virtual obstacles. For computing, an off-the-shelf desktop tower is used, along with several KVaser Leaf CAN-to-USB adapters to support CAN communications. The operating system runs Ubuntu 16.04 and uses the Robotic Operating System (ROS) [40] as the communications framework for the automation software.

4.1.1 Powertrain Specifications

The Prostar 122+ Powertrain information is shown in 4.1. A primary difference in the powertrain as compared to most trucks is the presence of an automatic Eaton transmission. Furthermore, the automation software and automatic gearing are independent systems, which has certain advantages and disadvantages. Benefits include system robustness, and a manufacturer tuned engine-gear map. The drawbacks are increased difficulty in obtaining transmission gear information, and one less control input method for the longitudinal dynamics.

Figure 4.1: 2013 International ProStar 122+ with sleeper cabin

Table 4.1: 2013 International Prostar 122+ vehicle specifications

Specifcation	**Description**
Engine	International MAXXForce 11
Transmission	Eaton UltraShift+ PLUS Automated Manual
Cab Dimensions	9.0 x 2.6 x 1.9 m
Towing Capacity	32,000 - 60,000 lbs.
Braking System	Air Drum and Disc with ABS
Gross Vehicle Weight	52,350 lb
Fuel Capacity	200 U.S. Gallons

4.1.2 By-wire Kit

The PACMod kit allows for the control over the throttle, braking, and steering of the vehicle at 30Hz. An EPAS Actuator by Allied Motion is utilized to manipulate the braking and steering actuation for vehicle control. The braking system is actuated via a pulley cable system attached to the EPAS motor, as illustrated by Figure 4.2. For steering, the EPAS motor is directly connected to the steering column. The throttle is digitally controlled from PACMod by utilizing the ProStar's J1939 CAN bus. A diagram of the overarching communication flow for the system can be seen in Figure 4.3.

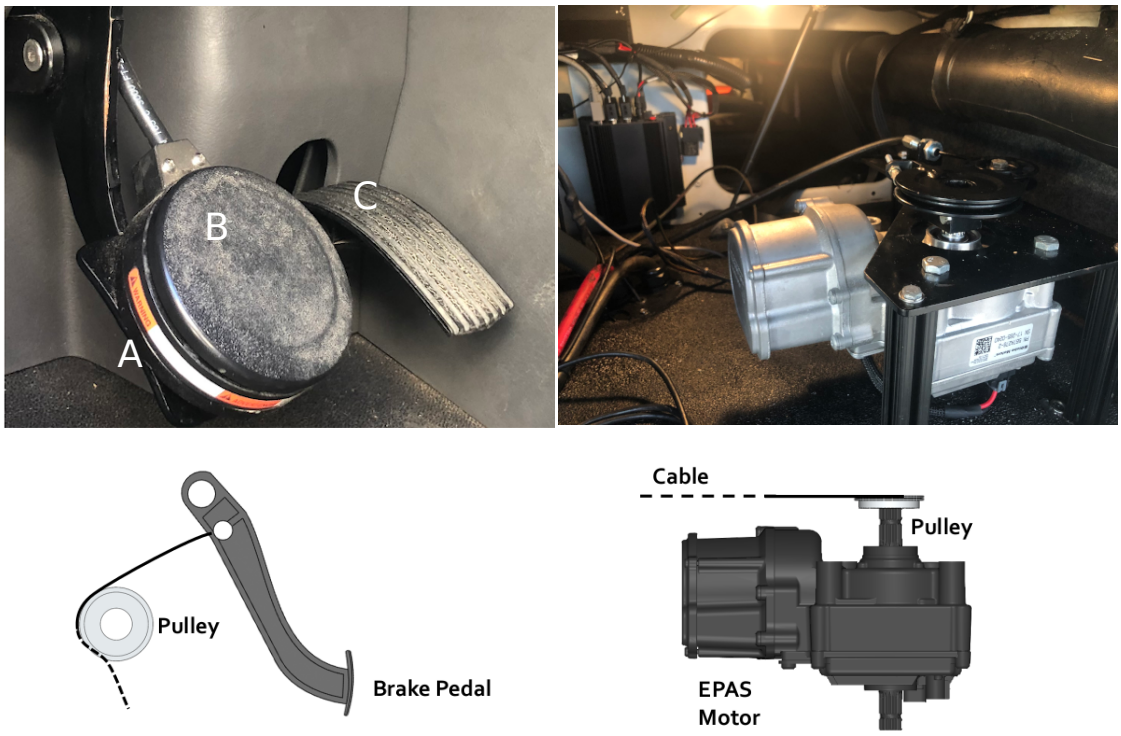


Figure 4.2: Brake pedal pulley diagram with (A) brake pedal, (B) disengagement safety switch, and (C) throttle

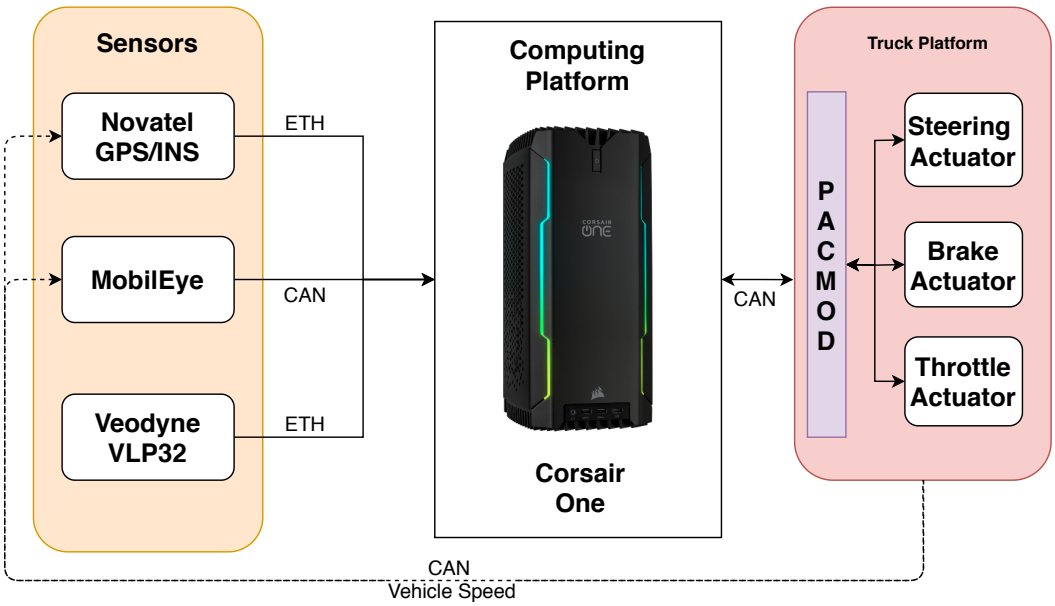


Figure 4.3: Sensor and actuator system communication diagram

Though the input design for the steering and throttle are sufficient, it should be mentioned that the braking input design is not desirable. Firstly, the brake pulley system actuates the brake pedal directly, introducing an additional mechanical failure mode. Secondly, the position of the brake pedal is not a preferred control input, as the same actuation of the pedal may not always yield the same braking torque due to temperature, air supply, and slack developed in the pulley cable. Even though more precise and repeatable control could be achieved by controlling brake pressure, the actuation proved adequate for experimentation of this thesis.

4.1.3 Novatel SPAN System

For odometry, the Novatel SPAN system is utilized, which includes a ProPak 6 GNSS receiver, two VEXXIS GNSS-500 antennas, and an IMU-IGM-S1 module. GPS/INS information is provided over Ethernet to the computing platform, as shown in Figure 4.4. GPS/INS information is logged at 50 Hz, while IMU data is published at 125 Hz.

4.1.3.1 Antenna Placement & Configuration

The two antennas are mounted on either side of the truck, located on top of the side view mirrors. The IMU module is mounted centrally inside the cab. To setup the Propak 6, offset measurements from the antennas and IMU are needed. Due to the large size of the vehicle, it is difficult to obtain high measurement accuracy. Offsets were taken using a laser distance tool, but uncertainties were around 10 cm, which is then propagated into the reported uncertainty in the Novatel INS solution. This uncertainty was acceptable for the test case scenarios, but if less uncertainty is desired, more precise offsets could be found with the use of the Lever Arm Calibration Routine within the Novatel SPAN software.

4.1.3.2 Accuracy

To increase the accuracy from the GNSS+INS solution, wheel speed information from PACMod is provided to the Novatel SPAN system by means of a ROS driver. Repeated tests were performed with and without wheel speed information supplied, and results are shown in Table 4.2. The most notable decrease was in the heading uncertainty, with a reduction of standard deviation

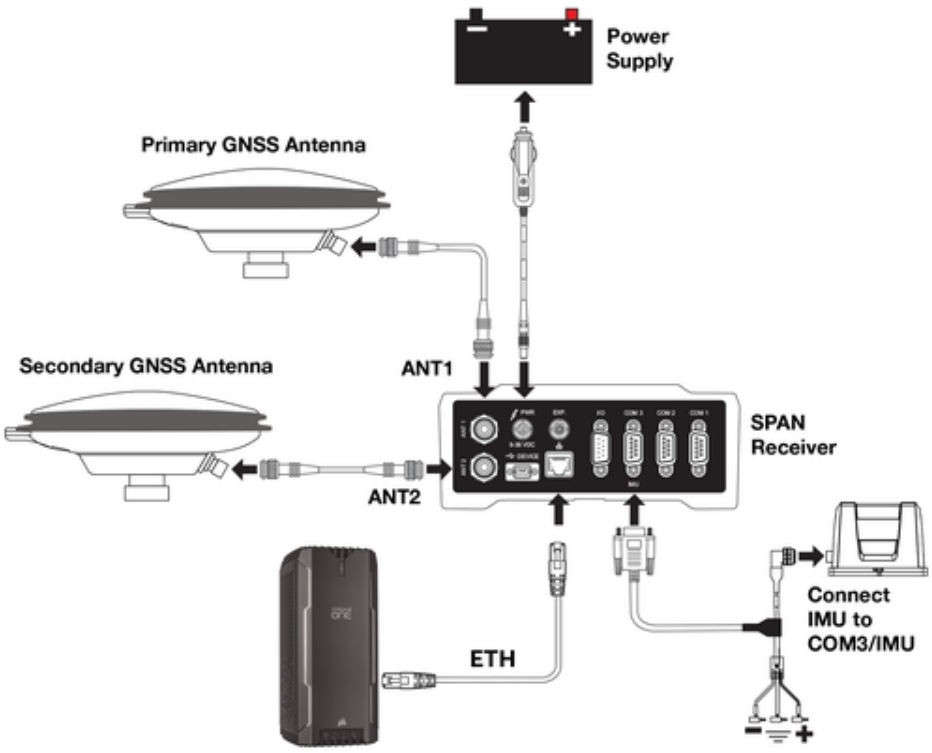


Figure 4.4: Novatel SPAN system diagram

by over 50%. Other methods to increase GNSS+INS solution accuracy include a GPS correction subscription service such as TerraStar; however, with consistent uncertainties of around 24cm in open sky conditions, the corrections subscription did not justify the high costs for this testing.

Table 4.2: Pose uncertainty from SPAN system with and without wheel speed information

	Position StD [m]	**Velocity StD [m/s]**	**Heading StD [rad]**
GPS Only	0.24	0.15	0.11
GPS with Wheel Odom	0.23	0.14	0.05

4.2 Simulation

Unlike car simulators, there are fewer options available for heavy-truck vehicle simulation. Typical software used in research and academia include TruckSim [41], ASM Truck/Trailer by

dSpace [42], and Truckmaker [43]. Though these simulators offer advanced and configurable dynamic simulations, they are less accessible as they usually require a large investment cost. In the Autonomous Driving community, Grand Theft Auto V has been popular as a low-cost car simulator [44] [45], as well as CARLA [46] and AirSim [47]. Similarly, another video game, American Truck Simulator [48], is utilized as a low cost simulation platform for heavy-truck vehicles. ATS simulates an 18-wheeler truck, where players can emulate a truck-driver and choose delivery routes to haul cargo across the Western United States. Although the map is scaled down, to allow for reasonable game play times, the game simulates the truck engine, transmission, brakes, suspension, and even road traction. Although these simulation parameters are mostly not configurable, and the dynamic models used for simulation are not publicly available, ATS is advantageous in that it can run on most desktop computers, and is significantly less expensive than other truck simulation software. It is for this reason that the ATS simulation platform was chosen to evaluate the motion planning algorithms.

Figure 4.5: American Truck Simulator interface

4.2.1 Simulation Interface

Interface to the simulation is made possible through a Telemetry SDK plugin developed for ATS that is installed in the game. The plugin has been configured to publish TCP packets of the

vehicle and truck and trailer state information, which include:

- Position and Orientation
- Linear and Angular Velocities
- Linear and Angular Accelerations
- Engine Gear and RPM
- Effective Braking, Throttle, and Steering

The packets are parsed by a custom made ROS wrapper, detailed in Figure 4.6, and are converted into standardized ROS messages. It is important to note that the simulator does not provide any world information about lane positions or other vehicles on the road. Thus, for the evaluation of the motion planners, virtual obstacles were simulated, similar to experiments on the ProStar.

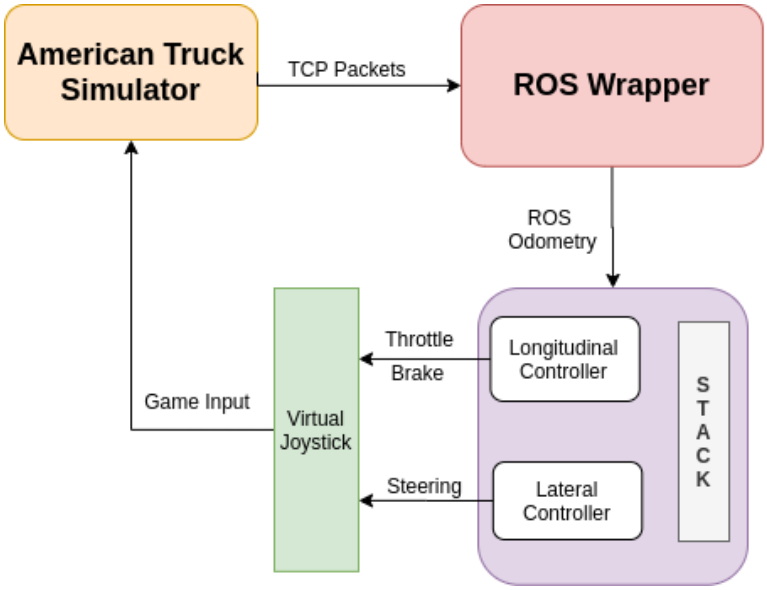


Figure 4.6: ATS communication flow diagram using ROS wrapper and virtual joystick

4.2.2 System Response Comparison

With the same input sequence, both the ProStar and ATS truck have similar first order lag responses to the pedal command, shown in Figure 4.7. Similarly to the ProStar truck, throttle

and brake pedal inputs are given over the range of 0-1, which corresponds to the percent of pedal deflection. Differences between the two responses can be attributed to the simplified dynamic model used by ATS, and a different engine/transmission used in game.

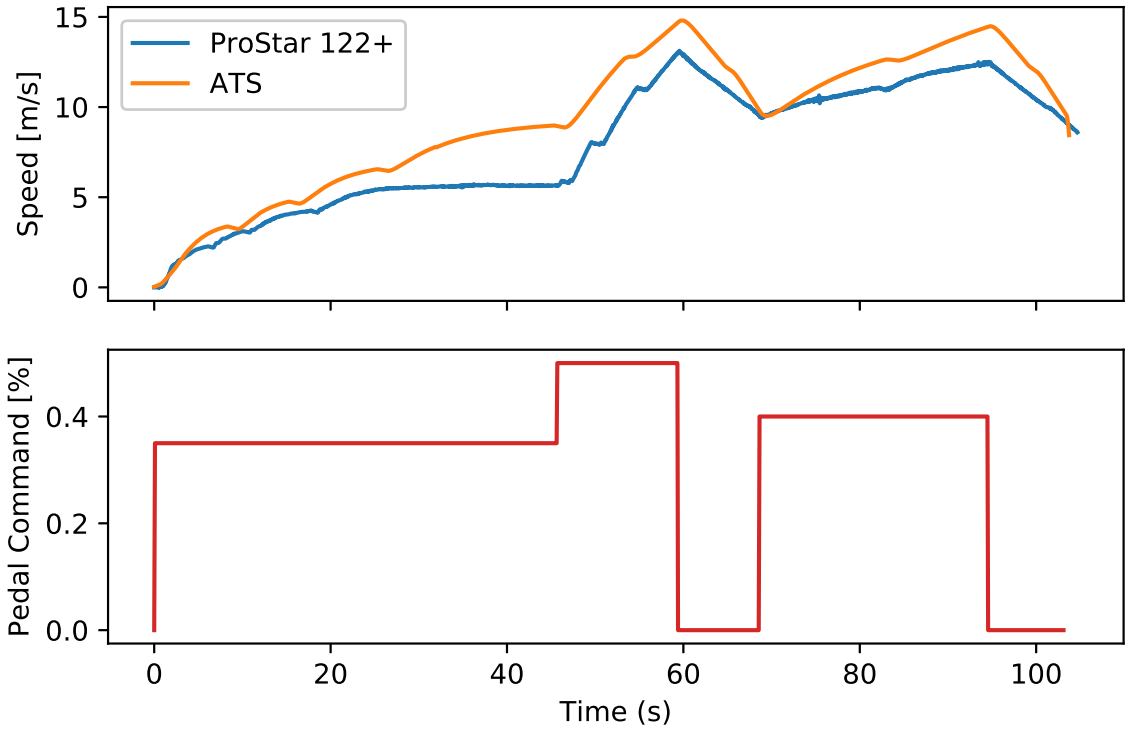


Figure 4.7: Comparison of open loop longitudinal response of ProStar 122+ and ATS truck

Another difference in simulation is the actuation delay between the pedal and subsequent acceleration response, as shown in Figure 4.8. These average delays represent the time between the depression of the throttle/brake pedal and any subsequent acceleration from the system. These actuation delays affect the controllability of the system, and can be seen in the longitudinal control performance elaborated in Section 4.3. Though there are several differences in simulation, ATS still offers a valuable platform to develop autonomous driving functions due to its consistency while testing and its relative ease of use. Furthermore, the ROS wrapper developed for ATS utilizes the same input/output topics as the PACMod module on the ProStar. This provides easy transition

from simulation to experimentation.

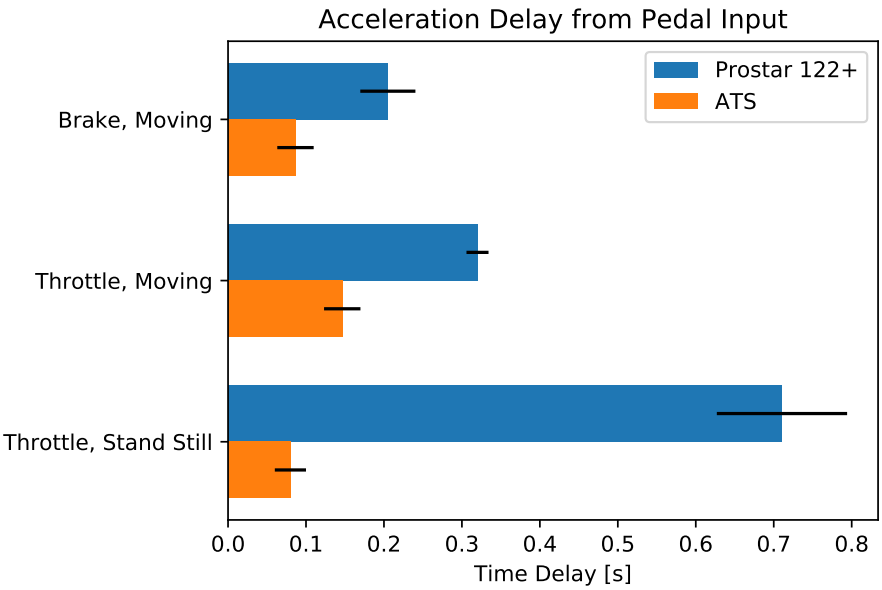


Figure 4.8: Comparison of acceleration delay of ProStar 122+ and ATS Truck

4.3 Control

The truck's low level feedback controller utilizes the motion plan and the current state of the truck to thus determine the input needed to correct the error in the presence of disturbances and modeling errors. Given that the vehicle is locally controllable around the planner's generated trajectory, the feedback controller attempts to track and stabilize the system as it traverses the planned path.

For a truck and trailer system, the controller determines the longitudinal speed and steering angle required to follow the motion plan. This thesis will follow the approach taken by many other self-driving systems by separating the controller to its longitudinal and lateral components [1], where the lateral controller follows the path provided from the planner while minimizing orientation and lateral error. The longitudinal controller ensures that the speed profile of the planner is followed. The controller operates at a higher frequency than the motion planner and follows the

planned path until termination or until a new one is generated.

4.3.1 Lateral Control

For lateral control, the Stanley Controller [49] was implemented, as shown in Equation (4.1). The steering control law is readily programmable, as the path planner provides a desired heading and position to calculate the subsequent heading error, $\theta_e(t)$, and lateral error from the path to the front axle of the truck, $e_y(t)$. A k_{soft} gain is implemented to avoid over steering at velocities less than 1 m/s.

$$\delta(t) = \theta_e(t) + \arctan\left(\frac{k_p e_y(t)}{v_x(t) + k_{\text{soft}}}\right) \tag{4.1}$$

In experimentation on the ProStar, a lane change maneuver at 7 m/s yielded less than 30 cm in lateral error. While testing 90 degree turns at lower speeds, the controller yielded less than 60 cm. This additional error can be attributed to the limited steering rate of the truck. Furthermore, this lateral controller was designed for a single vehicle configuration. Although in forward motion, the trailer does not affect the local stability of the system, it is important to note that a lateral controller that controls based on the trailer angle like that seen in [50] might result in less tracking error.

4.3.2 Longitudinal Control

Creating a dynamic model of the powertrain is often a difficult task, requiring either manufacture information on the ECU, or collection of large datasets of the transmission gear, RPM, and wheel velocity to estimate drive-train parameters [51]. Because the information of transmission gear and RPM is not known in experimentation, an approach similar to [52] was taken where throttle and braking pedal deflection percentages were mapped to both the current vehicle velocity and measured acceleration. Those mappings were then used as feed forward terms, shown in Equations (4.2) and (4.3).

$$brake_{pred} = 0.41 + 0.0022 * v_{cur} + 0.076 * a_{cmd} \tag{4.2}$$

$$thr_{pred} = 0.29 + 0.0072 * v_{cur} \tag{4.3}$$

To reduce chatter between the brake and throttle actuation and prevent simultaneous actuation, a throttle-brake-coast switching function was generated based on the desired acceleration outputted from the Speed PID shown in Figure 4.9. The threshold for switching was determined by observing a coasting acceleration of approximately -0.6 m/s^2 at various velocities ranging from 0-15 m/s. Thus, braking would only be actuated if the desired acceleration was less than -0.6 m/s^2.

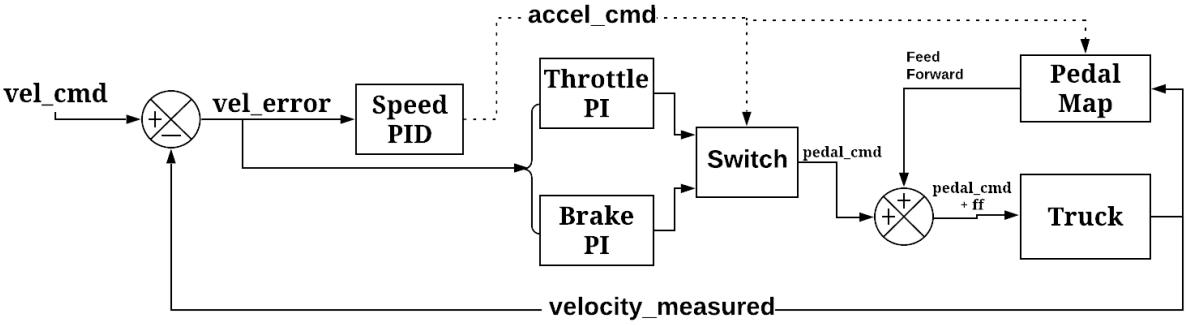


Figure 4.9: Longitudinal control flow chart

Both of these components were integrated into a control architecture illustrated in Figure 4.9. First, a PID controller based on the error from the desired and current velocity generates a desired acceleration. Depending on the desired acceleration, two separate PI controllers with feed-forward terms are used for the throttle and braking with velocity and acceleration error as the input, respectively. The PI gains were tuned by approximating a first order lag response between the throttle and brake pedal and velocity and acceleration, respectively. The longitudinal control algorithm, detailed in Algorithm 3, continuously outputs the pedal commands (ranging from 0-1) that are sent to PACMod via ROS for actuation.

The longitudinal controller was successful at tracking the desired speed profile in Figure 4.10, which shows both the ProStar 122+ and ATS tracking over a series of speed commands ranging from 0-10 m/s. The throttle and braking feed-forward terms were advantageous in supplementing the PI pedal controllers; however, further tuning and implementation of a gain scheduler is

Algorithm 3: Longitudinal Control Algorithm

input : Desired Speed v_{cmd}
Current Speed v_{act}
Current Acceleration a_{act}
output: Pedal Commands u_{thr}, u_{br}

a_{cmd} = speedPID(v_{cmd}, v_{act});
$case$ = switch(a_{cmd}, v_{act});
if $case$ == *throttle* **then**
    u_{thr} = thrPI(v_{cmd}, v_{act}) + thrMap(v_{cmd});
    u_{br} = 0;
else if $case$ == *coast* **then**
    u_{br} = 0;
    u_{thr} = 0;
else
    u_{thr} = 0;
    u_{br} = brakePI(a_{cmd}, a_{act}) + brakeMap(a_{cmd}, v_{act});
end
return u_{thr}, u_{br}

recommended in order to improve tracking performance and mitigate overshoot.

4.4 Static Scenarios

Static scenarios in the context of this thesis are in reference to obstacles that do not change relative to the configuration space of the truck and trailer. For a configuration space without time, these obstacles could be a road median, stopped car, etc. In order to evaluate the performance of each of the planners, this thesis compared execution times, the number of valid nodes generated, the path length, and the minimum obstacle clearance in ATS and on the ProStar truck.

4.4.1 Lane Change with single obstacle

To get an initial understanding of the execution and performance of the algorithms, each motion planner was tested three times in both ATS and on the ProStar system with a static lane change scenario using a single obstacle inflated by 1 meter. The total area of the workspace is about 2000 m^2 and took the Hybrid A* algorithm approximately 1.4 seconds to find a solution, with its state space discretized to intervals of 1 meter and 15 degrees. The position discretization was deter-

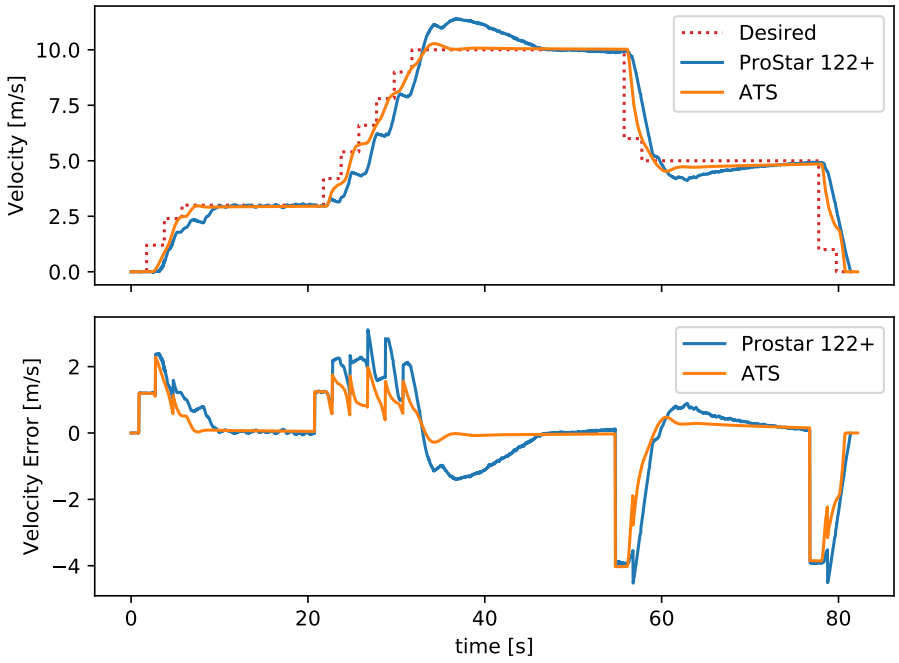


Figure 4.10: Longitudinal tracking performance of ProStar 122+ and ATS truck

mined based off of the physical dimensions of the system shown in Table 4.1 and the goal region margin of 2 meters, while the angular discretization was determined based off of initial testing and prior approaches conducted by [24, 25]. Increasing the resolution might decrease the path length but would increase the amount of storage and computations required to deterministically find the solution. One of the benefits of the sample-based algorithms is that they can be terminated at anytime [1], so in order to ensure comparable results, the other sample-based planners were then evaluated at the same execution time.

Table 4.3: Performance of motion planners for lane change with single static obstacle

	Hybrid A*	**RRT**	**RRT***	**RRT^X**
Valid Nodes	247	6901	1078	580
Path Length [m]	164.4	N/A	165.2	165.4
Planners Min Clearance [m]	1.5	N/A	2.6	1.4
ATS Min Clearance [m]	1.0	N/A	2.1	1.9
ProStar Min Clearance [m]	3.2	N/A	4.3	3.5

An example path from RRT^X is shown in Figure 4.11 comparing the planned path, the ATS path, and the path actualized on the ProStar truck. Due to some underactuation of the ProStar truck by the lower-level lateral controller, the ProStar path diverges slightly from the planned and ATS path. With that said, the planned path had enough clearance to ensure that the ProStar's path did not collide with the obstacle. This showcases that even though the planned path might be collision-free, without a properly tuned controller, the system may not be able to actualize the path and could lead to collision.

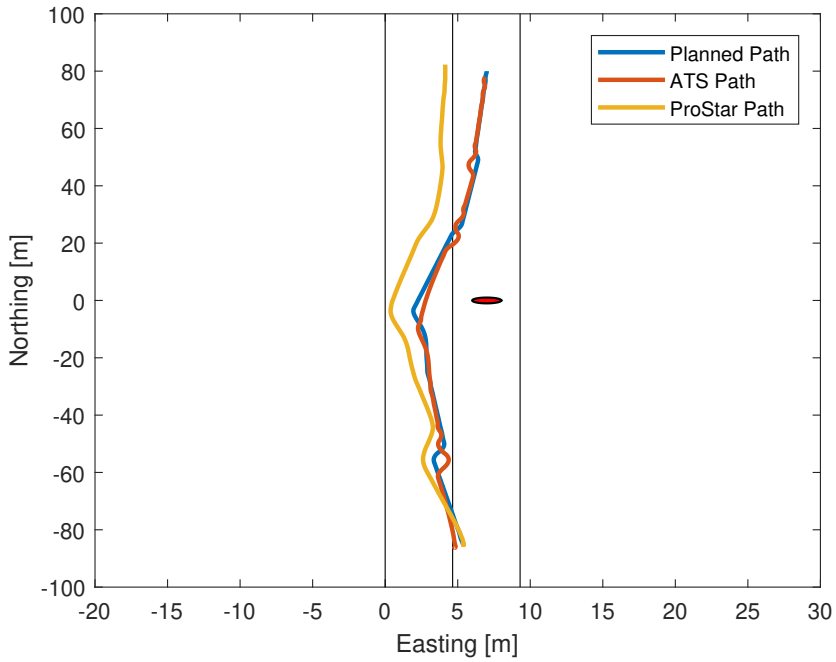


Figure 4.11: RRT^X paths of the rear axle of the truck with single obstacle shown in red

Furthermore, when comparing the planners that found a feasible path within the allotted time in Table 1, their overall path lengths are within 0.7 meters of each other, which is less than 1% of the optimal path length of approximately 164 meters. The RRT algorithm was not able to find a feasible solution within the allotted time and therefore does not have data in the table. The data also suggests that the quantity of nodes generated does not solely determine the path generated,

(a) ATS (b) ProStar

Figure 4.12: Execution of lane change with single static obstacle in simulation and real life

as RRT, with an average of 6901 nodes, could not find a solution. When comparing the planned, simulated, and actual minimum distance between the truck and the obstacle along the path, the discretization used for the Hybrid A* algorithm, along with the planners optimizing for path length and not proximity to obstacles, led to a smaller margin for cross-track error compared to RRT* and RRT^X. Accordingly, while the low level controllers used for the simulated and actual system do not perfectly follow the planned path, all algorithms were able to generate paths that maintained a safe distance from the obstacle when executed by the actual truck. Images of both the ATS and ProStar truck executing the path can be seen in Figure 4.12.

4.4.2 Lane Change with Moving Obstacles

Building upon the initial test, moving obstacles were added to the lane change scenario. The moving obstacles are intended to simulate sensor readings and perception of nearby vehicles, like on a city road seen in Figure 4.13. By including time in the state space of the vehicle, the obstacles can be treated as static with regards to the motion planner. For this scenario, the 1 meter radius obstacles move at constant speeds ranging from 2-5 m/s while the truck and trailer had a desired speed of 5-7 m/s. Both the RRT* and RRT^X algorithm were evaluated after 5 seconds of planning, with the obstacles inflated by 50 centimeters to account for the sensor uncertainty and expected

lateral error from the Stanley controller.

Figure 4.13: Aerial shot of the inspiration for the lane change scenario from Google Maps [2]

Furthermore, an additional constraint imposed on the motion planners as compared to the previous static example was that the paths generated between the nodes are not allowed to exceed the lane markings. This alteration drastically reduced the amount of valid nodes generated and can be seen in Table 4.4. Because the lanes are comparatively narrow compared the turning radius of the truck and trailer, the Dubins steering function tends to generate paths outside of the lanes as opposed to inside of the lanes. Because of the structured environment of the lanes, a planner like the one discussed in [53] might be more time efficient since it uses vehicle models derived relative to the road.

Comparing the average performance of the sample-based planners, once again the RRT^X algorithm is able to generate optimal paths in around the same time as the RRT* algorithm. Both algorithms were able to avoid obstacle collision for all of their tests in ATS and on the ProStar, but only with a clearance of 40 cm and 60 cm. This can mainly be attributed to the lateral error performance as the ATS simulation had larger clearances, but additional measures can be implemented to disincentivize the planner from navigating too close to nearby obstacles. One solution is to use a

Table 4.4: Motion planner performance for lane change with obstacles moving at constant velocity

	RRT*	**RRT^X**
Initial Solution Time [s]	2.8	2.4
Valid Nodes	175	144
Path Length [m]	166.3	164.4
ATS Min Clearance [m]	1.4	1.5
ProStar Min Clearance [m]	0.4	0.6
ATS Final Trailer Heading Error [°]	1.9	2.1
ProStar Final Trailer Heading Error [°]	2.1	1.0

cost associated with Voronoi fields between obstacles to incentivize path generation towards open areas [54].

Even with the narrow corridor, the planners were both able to generate feasible trajectories that avoided collision for the truck and trailer to execute. An example of the ProStar executing the RRT^X solution is presented below in Figure 4.14. The local planner provides updated paths at 5Hz to correct for the lateral error caused by a combination of kinematic modeling errors and controller performance. The color of the path denotes the time required for the vehicle to reach the goal configuration.

One observation from the evolution of the truck's pose is that the generated path does not adhere strictly to the lanes. This deviation is in part because there is no cost associated with staying within the lane in addition to the constraints imposed by the Dubins curve. One approach to correct this issue would be to integrate a logic function that shifts the path into the appropriate lane after generation. Although this would alleviate the sudden turns due to the Dubins path, switching lanes could be problematic when determining the sufficient conditions needed in order to initiate, cancel, or revert a lane change maneuver.

Although Figure 4.14 provides some intuitive understanding of the vehicle's execution of the path in the Northing/Easting plane, one can also look at the other two state dimensions, θ_0 and t, to see how the search graph expands throughout all of the collision-free configuration space. Figure 4.15 illustrates the initial and final RRT^X graph. Since the vehicle is limited to only forward

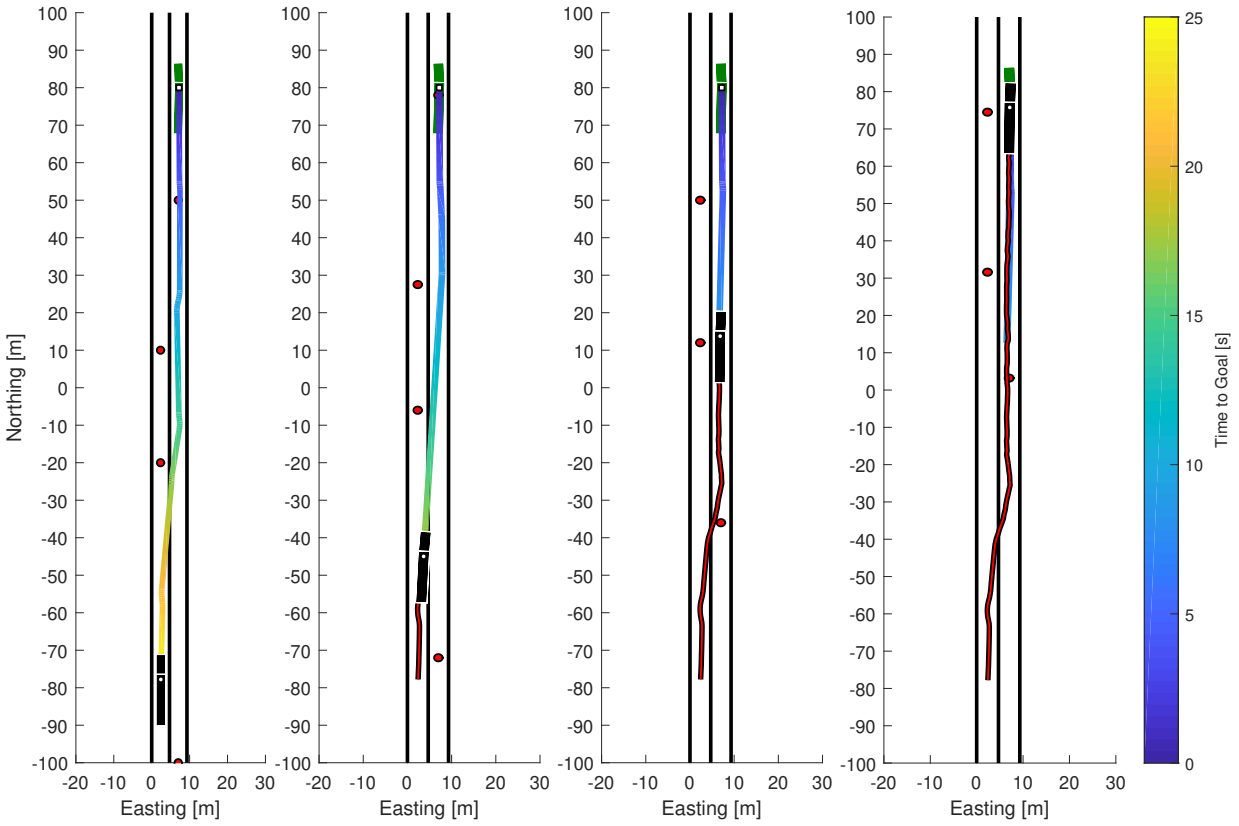
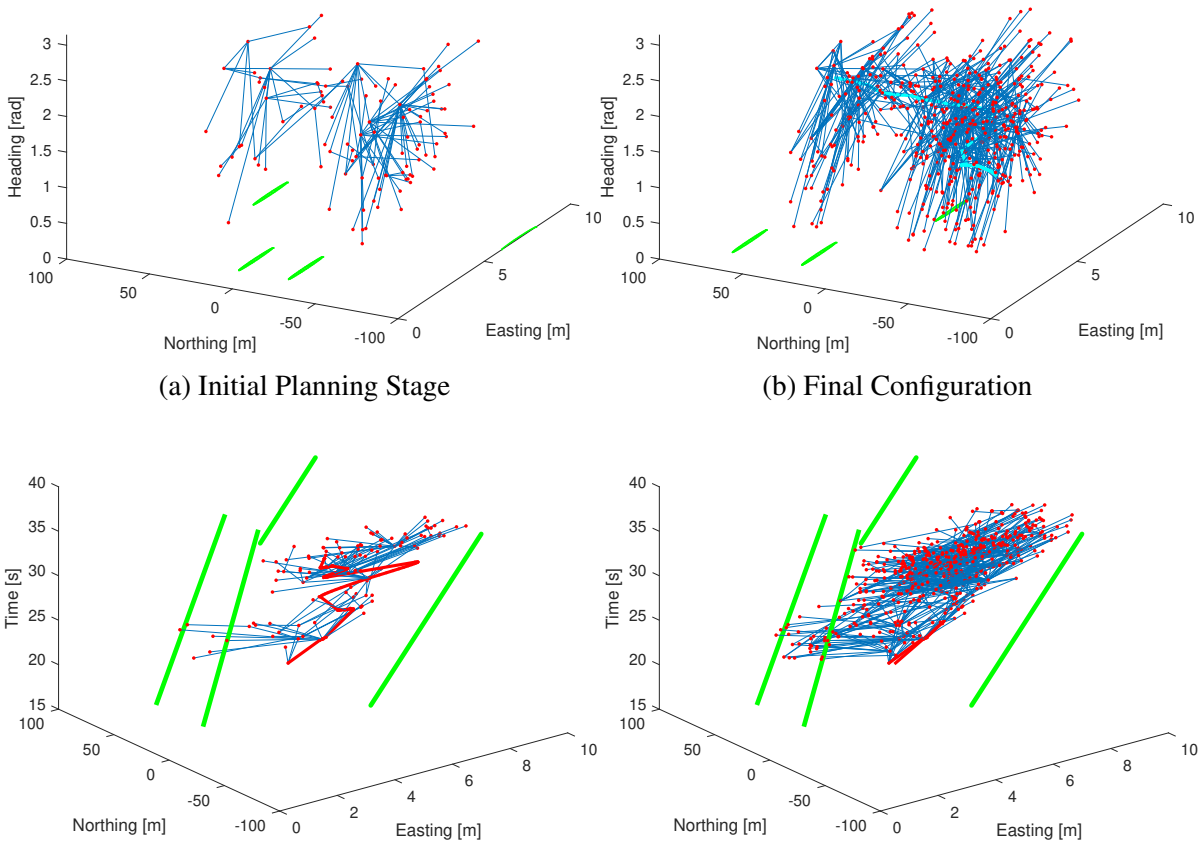


Figure 4.14: ProStar executing lane change using RRT^X with obstacles moving at constant velocity, where (a) is the initial planning stage, (b) is the start of lane change, (c) is maintaining lane near a obstacle, (d) is the final configuration.

motion and the turning radius is larger than the lane width, the motion planner was not able to find any valid configuration with the heading greater than π, i.e. facing south.



(a) Initial Planning Stage

(b) Final Configuration

(c) Initial Planning Stage

(d) Final Configuration

Figure 4.15: Initial and final RRT^X Graph for Lane Change with obstacles moving at constant velocity with vertices (red), edges (dark blue), and obstacles (green)

4.4.3 Seaport

For the other scenario, the truck is tasked with delivering a shipping container at a seaport. This scenario provides a more unstructured environment for the planner to search the potential configurations for the vehicle to traverse. The inspiration for this scenario comes from the growing economical need for automated freight transportation. An example seaport can be seen in

Figure 4.16. In addition, due to the legislative requirements in place to drive autonomously on public roads, a closed area like a seaport makes a great candidate for initial testing of self-driving technology [55].

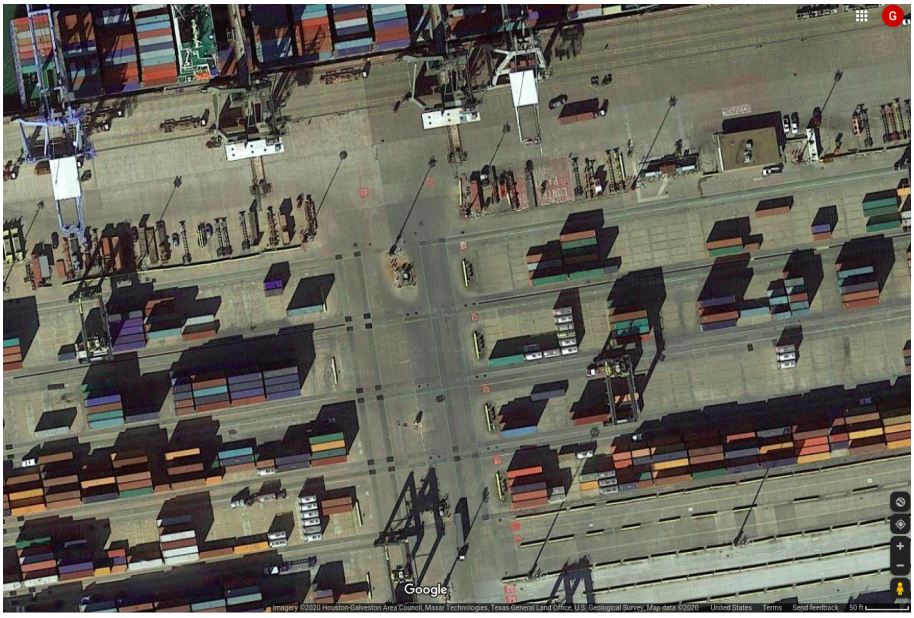


Figure 4.16: Aerial shot of the inspiration for the seaport scenario from Google Maps [3]

For this scenario, the same methodology was used as the experiments above. In this test, obstacles were inflated by 1 meter to account for the increased estimated lateral error while executing sharper turns. The system was initially given a desired nominal speed of 8 m/s, but was reduced to 4 m/s after some initial testing due to an instability issue, which is elaborated on in Section 4.4.3.1. For the static scenario, Hybrid A*, RRT*, and RRT^X were evaluated with the assumption of perfect perception of all obstacles. The seaport scenario provides a realistic and obstacle dense environment to evaluate the planners in Table 4.5.

In terms of the planners' performance, it can be seen that all three planners on average find a feasible path within the same amount of initial time of approximately 1.7 seconds. In addition, due to the increased inflation of the obstacles, the minimum obstacle clearance in ATS is increased compared to the lane change scenario. The paths generated by the sampling based planners are on

Table 4.5: Motion planner performance for seaport with static obstacles

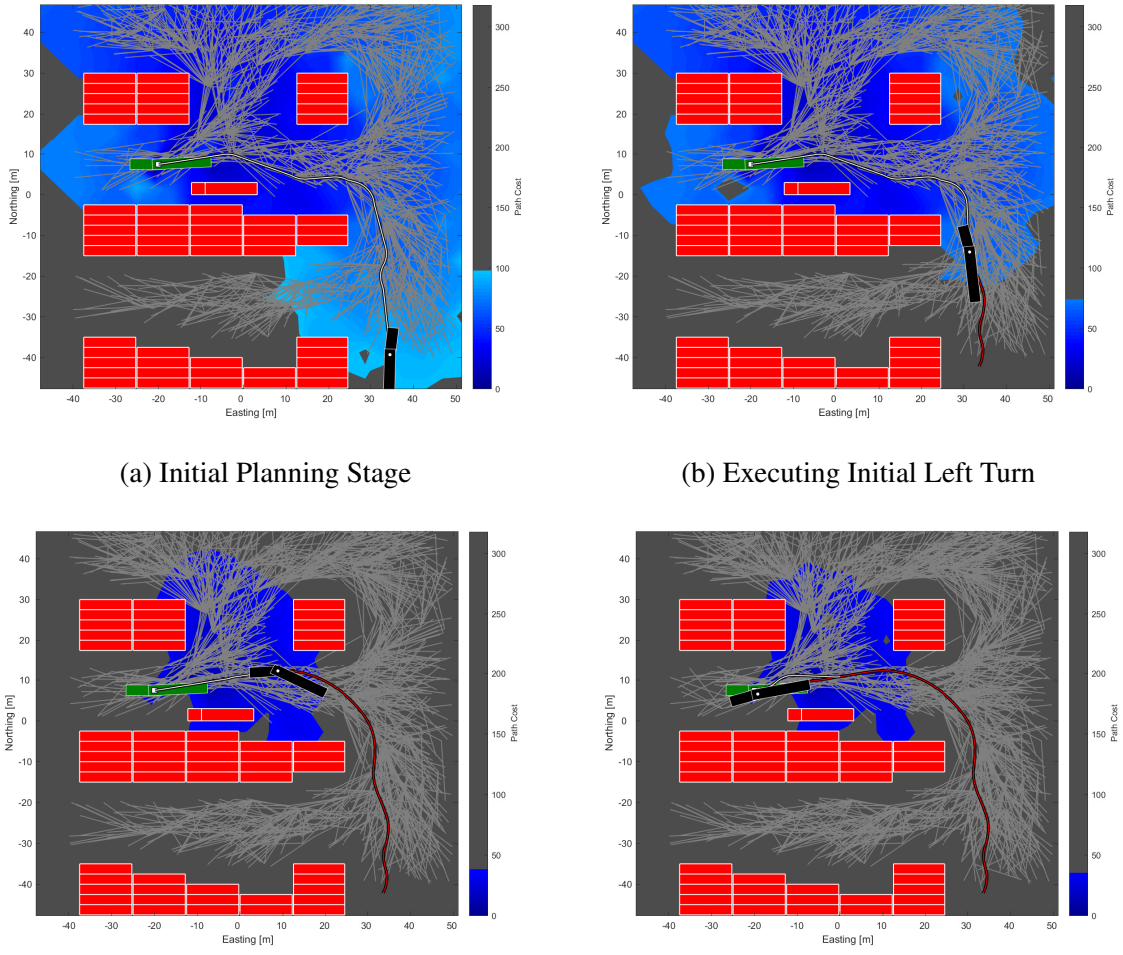
	Hybrid A*	**RRT***	**RRT^X**
Initial Solution Time [s]	1.8	1.6	1.7
Valid Nodes	105	282	159
Path Length [m]	94.7	93.5	91.2
ATS Min Clearance [m]	2.7	2.6	1.9
ProStar Min Clearance [m]	1.9	0.8	0.9
ATS Final Trailer Heading Error [°]	3.7	3.4	3.6
ProStar Final Trailer Heading Error [°]	6.3	9.8	2.1

average shorter than that from the Hybrid A* algorithm. This can be attributed to the discretized resolution and the lack of guaranteed optimality with Hybrid A*.

While all planners on average have a trailer angle within the desired margin for ATS, more irregularities were observed on the ProStar truck. This is in part due to the Stanley controller not considering the trailer angle in its error term, but also illustrates a potential issue with the Dubins curve. Since the Dubins curve always ends in a turn, the final curve connecting to the goal configuration has a tendency to be a small, but abrupt turn. In the experiments, there was a goal region for the lower level controller of the truck set with a 2 meter radius encompassing the desired goal configuration. Since the planners usually waited until the end to make their final Dubins turn, the ProStar truck did not always align the truck and trailer as much as planned before reaching the goal region. One solution to this issue would be to narrow the goal region by also considering the angular configurations of the vehicle. Another, potentially more effective, solution would be to negatively weight turns closer towards the goal. This would incentivize the planner to make turns earlier in the planning process, thus making this issue less likely to occur.

Looking into the evolution of the truck and trailer system's planned RRT^X path in Figure 4.17, one can observe that though the controller does not perfectly track the initial path generated in Figure 4.17a as it leads into Figure 4.17b, the system is able to stay clear of the wall of shipping containers to its left. In addition, in order to ensure there is at least a 1 meter gap between the system and nearby obstacles, the planner generates a wide turn around the initial corner that pre-

vents the trailer from clipping the container around the turn in Figure 4.17c. Finally, the system navigates around the extended container to reach the goal configuration, as seen in Figure 4.17d and in Figure 4.18. As mentioned previously, since the controllers termination condition is based purely off of the position of the vehicle, the final orientation of the system does not exactly align with the goal configuration.



(a) Initial Planning Stage

(b) Executing Initial Left Turn

(c) Avoiding extended shipping container

(d) Final Configuration

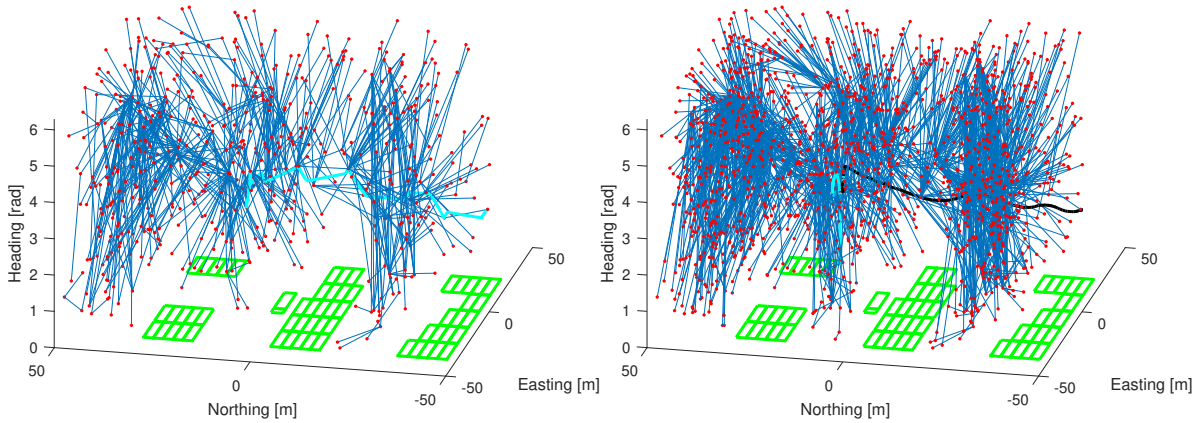
Figure 4.17: ProStar navigating seaport using RRT^X with static obstacles

Figure 4.19 illustrates the initial and final RRT^X search graph including the truck heading. As opposed to the lane change scenario, the planner now is able to sample and discover valid nodes

(a) ATS (b) ProStar

Figure 4.18: Execution of seaport with static obstacles in simulation and real life

distributed across the entire 0-2π range. In addition, it should be noted that the obstacles are positioned at 0 rads in order to visualize the rest of the search graph. Although the obstacles do not have a heading per se, they extend through all the 0-2π range as any vehicle configuration inside this range would result in collision.

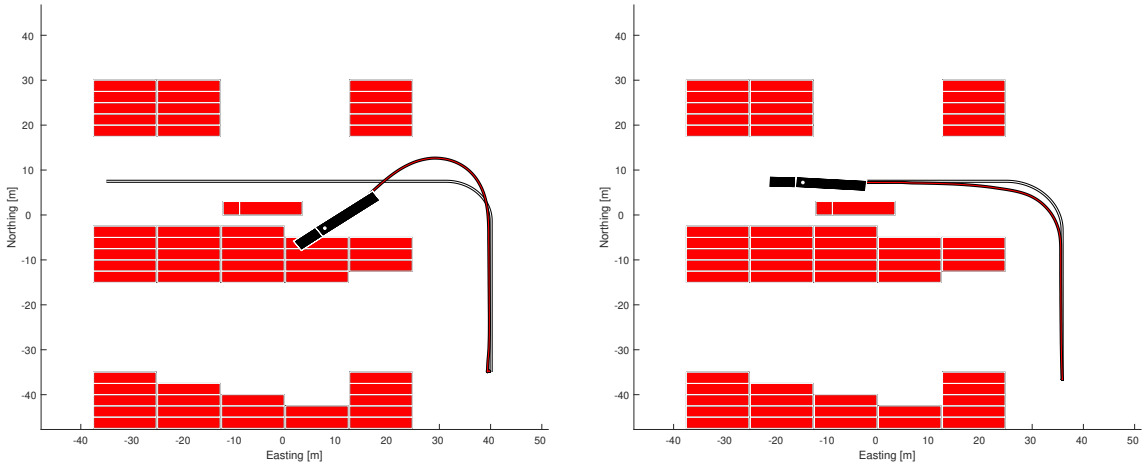


(a) Initial Planning Stage (b) Final Configuration

Figure 4.19: Evolution of search graph for seaport with static obstacles

4.4.3.1 Curvature

As mentioned in the previous section, the desired speed of the ProStar vehicle had to be lowered from 8 m/s to 4 m/s due to observed instability of the lateral controller during execution. Since the Dubins path has instantaneous changes in curvature, κ, the truck and trailer system, with a limited steering rate, has difficulty actualizing the plan at higher speeds. This can be seen when comparing the paths in Figure 4.20a and Figure 4.20b. Because the Hybrid A* algorithm generates a sharp left turn for this scenario, the instability is more observable when compared to the sample based algorithms that generate a path with more intermediate turns that result in a less disjointed trajectory when discretized. This is in part due to the expansion distance imposed on the sample-based algorithms along with the tendency for those algorithms to hug the obstacles in order to minimize the path length. With that being said, all planners experienced this issue to some degree based on the Dubins steering function.



(a) Execution at 8 m/s (b) Execution at 4 m/s

Figure 4.20: System struggles to execute Hybrid A* with discontinuous curvature at high speeds

In order to mitigate this curvature issue, the Stanley controller was further tuned and a cubic spline was fitted to the motion planners path at 1 meter increments. Since cubic splines provide

the added benefit of continuous curvature, they are used in many cases to provide a smoothed path for the vehicle to follow. This increment was chosen in order to ensure the smoothed path did not diverge from the planned path, thus discarding the collision-free and optimality guarantees that the planner provides as well as its consideration of differential constraints imposed by the minimum turning radius of the vehicle. Even with the spline fitting, the instantaneous change in curvature was too steep for the truck to actualize as seen in Figure 4.21. Although the curvature is below the maximum curvature of the vehicle, the steep jumps are not actualizable given that the steering rate of the vehicle is approximately 0.13 rads/s. At 8 m/s and at the maximum steering angle, the maximum sharpness, $\alpha = \partial\kappa/\partial s$, of the path is approximately 0.02 m^{-2} based on the maximum steering rate [56]. One could further smooth the path at sparser increments, but then more checks would be needed to guarantee collision avoidance as the fitted path could drastically diverge from the planner's path. It is for this reason that the speed was reduced for testing and would be further reason to look into an alternative steering function like [29, 56, 23].

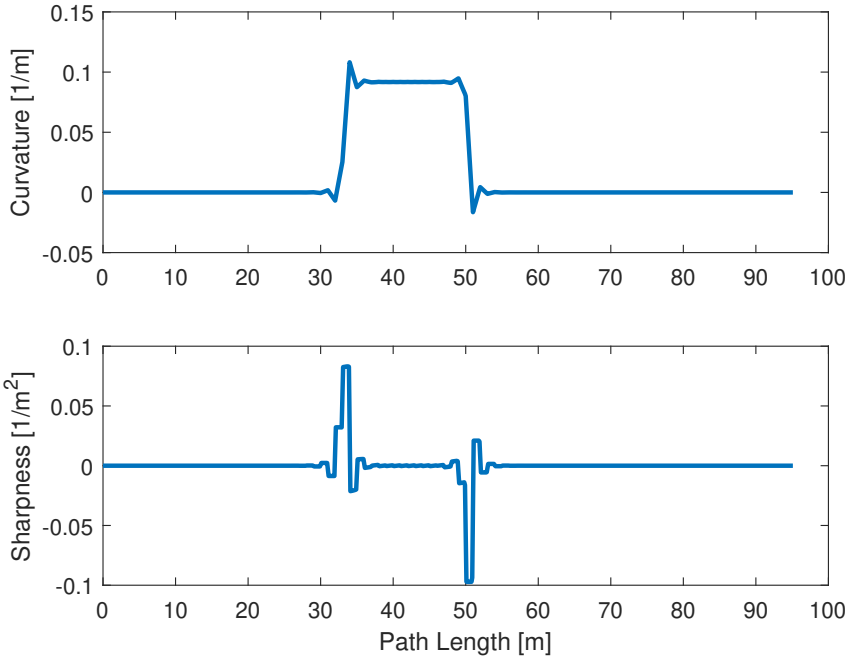


Figure 4.21: Curvature and sharpness of Hybrid A* path for seaport maneuver

4.5 Dynamic Scenarios

To simulate more realistic sensor input and perception, dynamic obstacles were implemented on the aforementioned scenarios. Dynamic obstacles can suddenly appear, disappear, or move during the execution of the plan. In reality, most obstacles are dynamic to the vehicle since the obstacles states are uncertain and constantly fluctuating. Static scenarios can be somewhat unrealistic in practical scenarios due to the unlikeliness of having perfect perception of all external actors. Most static planners mitigate this issue with high enough update rates, such that dynamically moving obstacles are approximately static for the small time step of planning. Although this has been a valid and successful approach [1, 25], this thesis investigates maintaining the same search graph over the course of completing the entire mission, and thus cannot make that assumption.

4.5.1 Lane Change with Car that Unpredictably Moves

Although the perception layer might use constant velocity to predict the velocity of nearby obstacles, the intentions of nearby vehicles can change at a moment's notice. To simulate this occurrence, one of the simulated vehicles in the previous lane change scenario was programmed to suddenly change lanes in front of the truck and trailer. This prevents the trailer from following the previously planned path and the motion planning algorithm must plan in real time in order to avoid collision. The planner only has knowledge of the vehicle's current trajectory, and thus does not know future intentions of the obstacle. The RRT^X algorithm was evaluated in ATS and on the ProStar truck and the results can be viewed in Table 4.6 with the same planning constraints as were previously imposed on the planner.

In terms of the planner's performance in ATS and on the ProStar truck seen in Table 4.6, the initial solution time, the number of valid nodes, and the path length are similar to the static scenario. This is due to how the dynamic scenario is identical to the static scenario during the initial planning stage. Since the truck and trailer was forced to change lanes closer towards the desired goal, the ProStar's final trailer angle exceeds the 5° difference from the desired angle. Since the ATS trailer did not share similar results, one explanation for this could be tied back to

Table 4.6: Motion planner performance for lane change with car that unpredictably moves

	RRT^X
Initial Solution Time [s]	2.7
Valid Nodes	121
Path Length [m]	163.1
ATS Min Clearance [m]	1.1
ProStar Min Clearance [m]	0.9
ATS Final Trailer Heading Error [°]	1.1
ProStar Final Trailer Heading Error [°]	5.3

the instantaneous curvature of the Dubins curve. Since the steering rate of the ProStar truck is significantly lower than the ATS truck, the system has a more difficult time following the desired path.

Even though the final trailer angle error on the ProStar exceeds the planned limit of 5° on average, one way to mitigate this issue would be to have the planned margin be a fraction of a larger actual trailer error margin. This approach is similar to how the obstacles are inflated during planning in part to account for modeling uncertainties and assumptions on the actual truck.

By looking into the other two state dimensions, θ_0 and t, one can see how the search graph expands throughout all of the collision-free configuration space. Figure 4.23 illustrates the initial and final RRT^X graph. Once again, the planner is unable to find valid configurations with truck headings opposite that of the lane. In addition, due to the unpredicted lane change of the car in front of the truck, there are a significant portion of invalidated nodes seen in Figure 4.23b and Figure 4.23d.

4.5.2 Seaport with Limited Sensor Range

In a similar fashion, the seaport scenario was augmented to simulate a limited sensor range by making the obstacles appear and disappear dynamically. More specifically for this scenario, obstacles around the corner of the shipping containers are unknown to the system during the initial planning phase, so the planner must react quick enough once it rounds the corner.

Once again, RRT^X was evaluated in ATS and on the ProStar truck system with successful

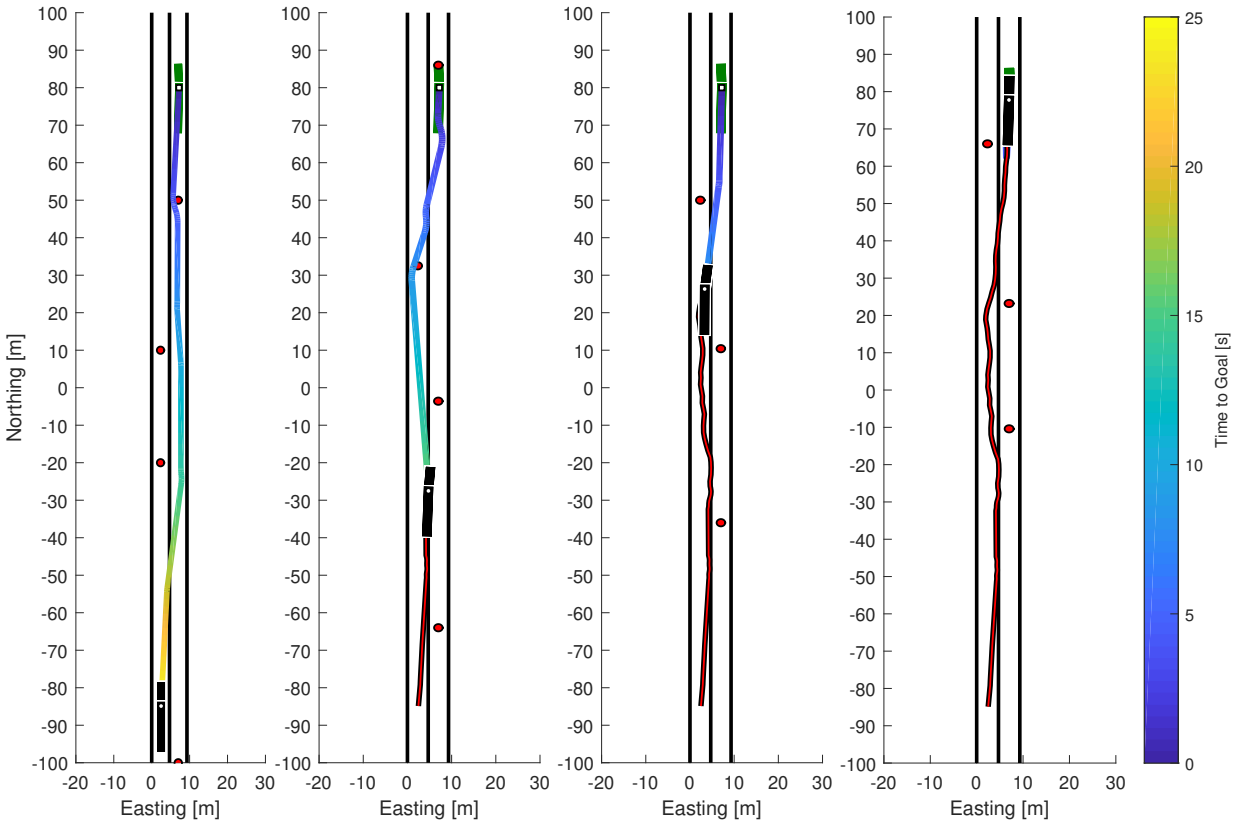
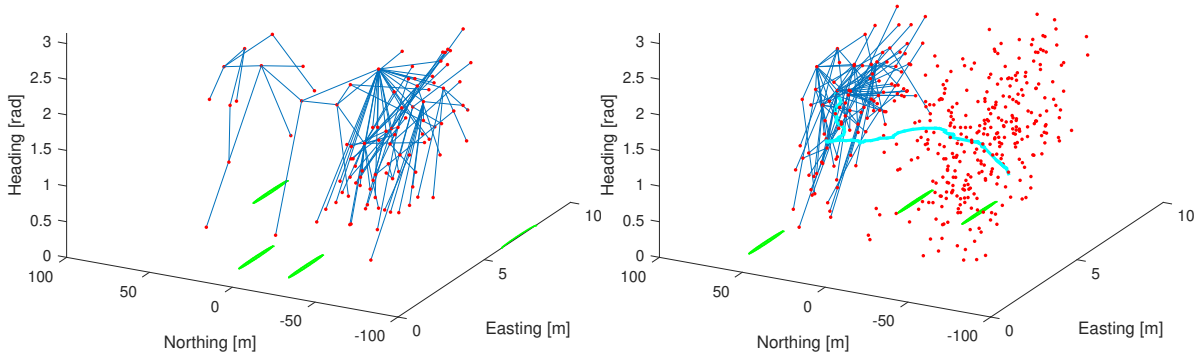
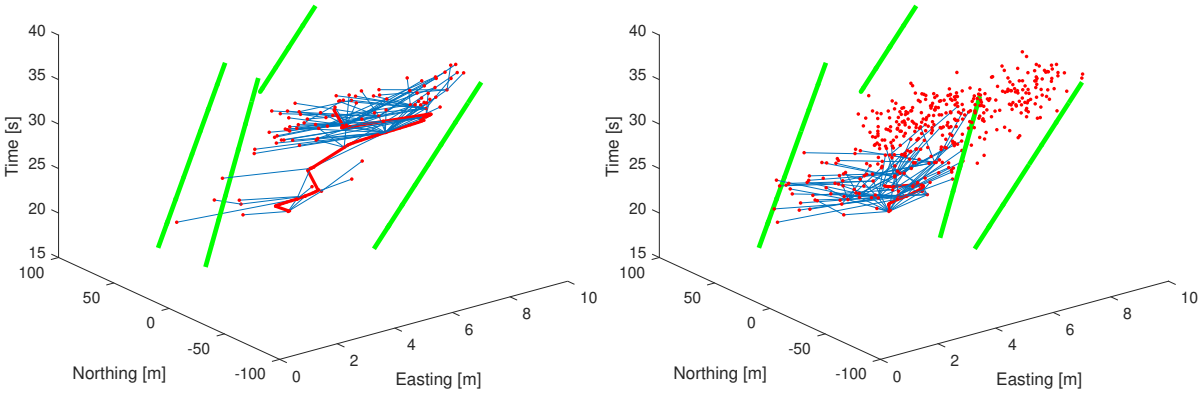


Figure 4.22: ProStar executing lane change using RRT^X with obstacle that suddenly changes lane, where (a) is the initial planning stage, (b) is adjusting plan due to unforeseen lane change, (c) is start of new lane change obstacles, (d) is the final configuration.



(a) Initial Planning Stage

(b) Final Configuration



(c) Initial Planning Stage

(d) Final Configuration

Figure 4.23: Evolution of search graph for lane change with vehicle that unexpectedly changes lanes

results. In terms of the planner's performance in ATS and on the ProStar, similar results were observed compared to the static scenario and can be seen in Table 4.7. The average time for an initial solution is slightly less and the amount of valid nodes generated is slightly more. This is due to the perceived lack of obstacles around the corner of the shipping containers. With such an open area, the RRT^X algorithm expands without as many limitations and collision checks, thus it is able to converge to a feasible solution in less time.

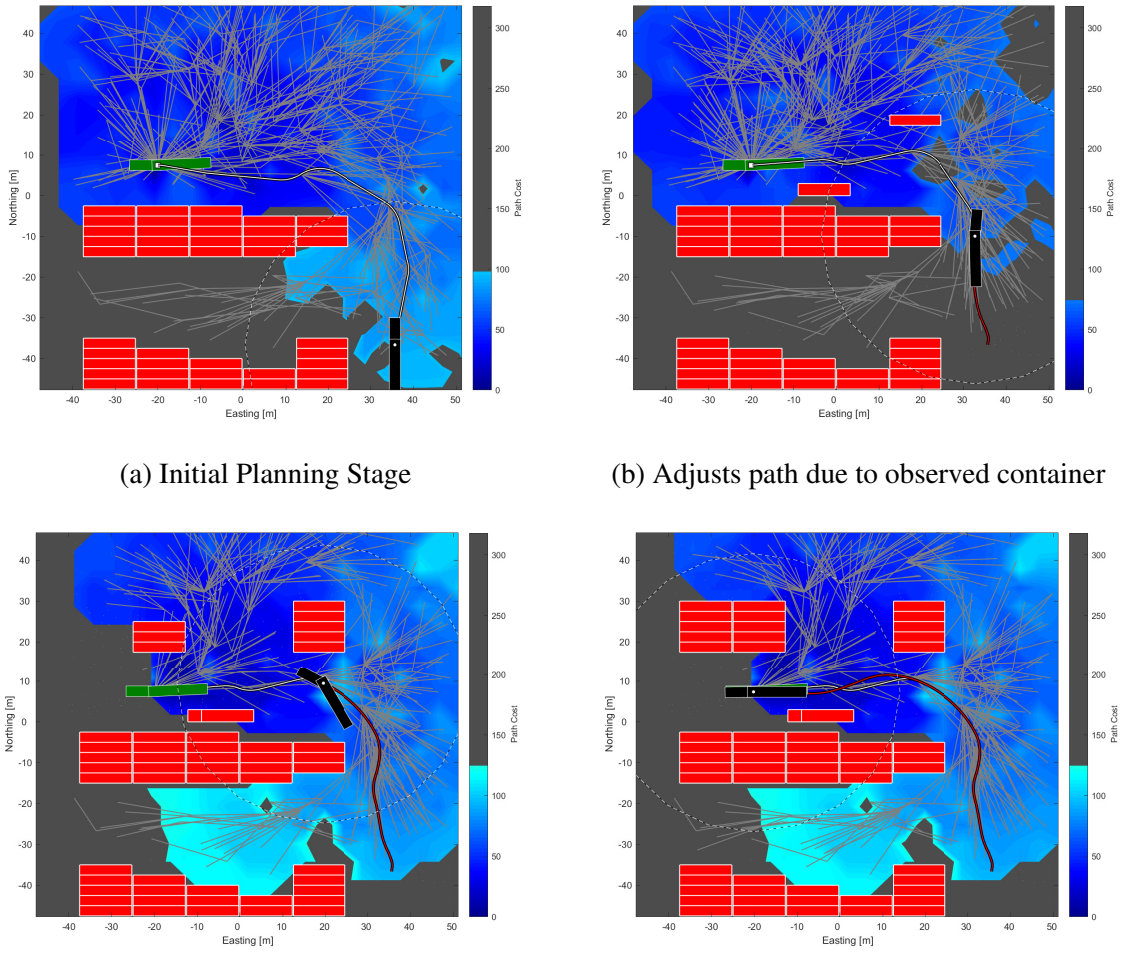
On average, the ProStar had a final trailer heading that was outside the goal criteria of 5°. As mentioned previously, this is partly due to how the lower level controller does not account for the trailer angle in its design. However, the error can also be attributed to how the Dubins curve generates paths that exceed the steering rate of the ProStar vehicle.

Table 4.7: Motion planner performance for seaport with limited sensor range

	RRT^X
Initial Solution Time [s]	1.5
Valid Nodes	198
Path Length [m]	92.5
ATS Minimum Obstacle Clearance [m]	1.9
ProStar Minimum Obstacle Clearance [m]	1.3
ATS Final Trailer Heading Error [°]	4.8
ProStar Final Trailer Heading Error [°]	6.2

In Figure 4.24, the evolution of the tree is shown, this time without the knowledge of the obstacles around the corner. With the unstructured and open environment, the RRT^X algorithm finds an initial feasible path, shown in Figure 4.24a in an average of 1.5 seconds. As the system navigates the environment, the previous path is invalidated due to the unexpected container and a new path is generated in Figure 4.24b to traverse until the system reaches the end configuration. This path is then followed for the rest of the experiment as the other containers above the goal configuration do not obscure the path. Figure 4.24d shows the controller's tracking performance towards the final configuration. This is because the planner does not update the desired path once

the vehicle is within the expansion distance from the goal configuration.



(a) Initial Planning Stage

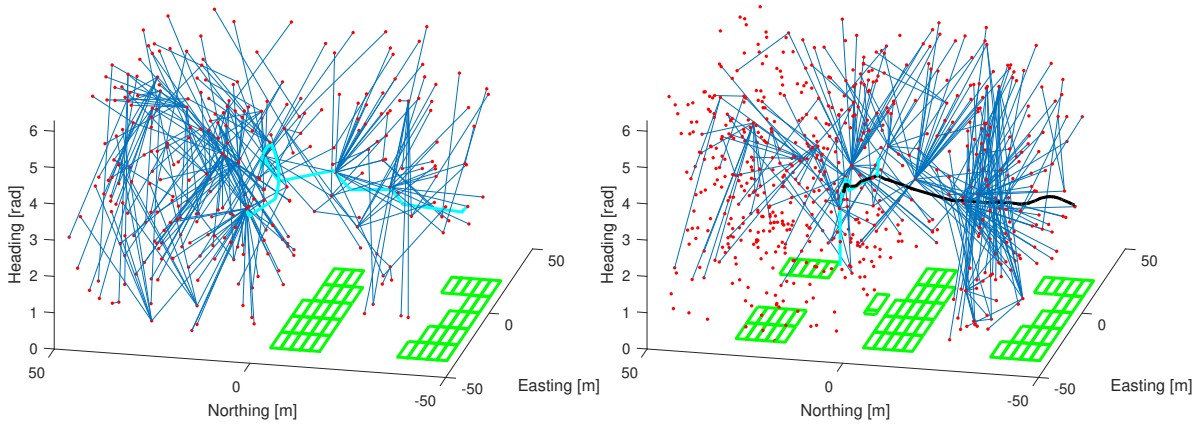
(b) Adjusts path due to observed container

(c) Avoiding Extended Container

(d) Final Configuration

Figure 4.24: ProStar navigating seaport using RRT^X with limited sensor range

Figure 4.25 illustrates how the appearance of the discovered shipping containers affected the potential heading configurations of the search graph. The vertices that were invalidated due to the sudden appearance of the containers can be seen in the northern section of Figure 4.25b, represented by the unconnected red points.



(a) Initial Planning Stage (b) Final Configuration

Figure 4.25: Evolution of search graph for seaport with limited sensor range

4.6 Overview of the Experimentation

The initial experiments were primarily focused on identifying the system's longitudinal response characteristics in addition to quantifying the sensor uncertainty from GNSS+INS solution. From there, a longitudinal controller framework was developed and evaluated alongside a Stanley lateral controller to provide nominal results for a prototype testing platform. The motion planners were then evaluated in real-time on ATS and the ProStar truck using criteria like the amount valid nodes generated, the path length, and the minimum obstacle clearance in ATS and on the ProStar truck averaged over 3 trials.

For the lane change scenario, three variations were tested. In the first variation, a single static obstacle was introduced to evaluate Hybrid A*, RRT, RRT*, and RRT^X. All planners except RRT were able to generate collision-free paths with similar, near-optimal lengths in approximately 1.4 seconds. Next, statically moving obstacles were used to evaluate RRT* and RRT^X. Both planners demonstrated the ability to find feasible motion plans but only had minimum clearances of 40 cm and 60 cm on the ProStar, respectively. Then, one of the vehicles was modified to unexpectedly change lanes in front of the system. The RRT^X approach was able to react and replan without collision and result in a trailer angle within the 5° margin.

For the seaport scenario, two variations were tested. In the first variation, static shipping containers were navigated by Hybrid A*, RRT*, and RRT^X. After addressing an issue with the ProStar's limited steering rate, all planners demonstrated the ability to generate initial solutions within approximately 1.7 seconds, with the sample-based planners generating more optimal paths in terms of length. Lastly, in order to simulate actual sensors, RRT^X was provided only a limited radius of nearby obstacles. Even with the sudden appearance of containers, the planner was able to navigate the trailer through the narrow corridor towards the goal configuration without collision.

5. SUMMARY AND CONCLUSION

5.1 Conclusions

This thesis presented an approach to motion planning for truck and trailer systems that can be applied in structured and unstructured environments. The proposed method is for a sample-based motion planning algorithm that can adapt and replan for a truck and trailer system constrained to forward motion in the presence of dynamic obstacles.

In order to evaluate the motion planners, an existing truck simulator, ATS, was modified to allow for autonomous control using a ROS framework. In addition, a longitudinal and lateral feedback controller framework was developed and evaluated for both ATS and a ProStar 122+ truck retrofitted with by-wire capabilities. In order to control longitudinal speed of the vehicle, a combination of PI controllers with feed forward terms were used in conjunction with a switching algorithm to control throttle and braking actuation. Since the controllers had limited information over transmission gear and engine RPM, the PI controllers were advantageous in that they required significantly less time and programming complexity compared to developing dynamic models of the powertrain. However, the longitudinal controller could be further improved as it had overshoots over 2 m/s. For the lateral controller, the Stanley controller introduced by [49] was adapted for the truck due to its versatility and ease of implementation, which saw lateral error bounded to approximately 30 cm in lane change scenarios.

Once the controllers and self-driving architecture were evaluated and established, the performance of the search-based motion planners of Hybrid A*, RRT*, and RRT^X were investigated on both testing platforms in a structured lane change scenario and an less structured seaport scenario. Each of the planners were implemented as Julia software with similar data structures in order to ensure comparable results. For the initial experimentation in a low speed, lane change scenario with a single static obstacle, the algorithms were able to produce similar paths in approximately 1.4 seconds that avoided collision in simulation and on the ProStar vehicle. RRT* and RRT^X were

then evaluated with a similar lane change scenario with multiple obstacles moving at various static velocities. By incorporating time into the configuration space of the system, the sample-based planners were able to consistently update paths at 5Hz that resulted in no obstacle collisions.

The same methodology was repeated for a seaport scenario made up of an obstacle dense area of shipping containers for Hybrid A*, RRT*, and RRT^X. After some observed instability at higher speeds, the scenario was tested at 4 m/s. This issue shed light on one of the drawbacks of utilizing a Dubins curve for the steering function, as it assumes the system has instantaneous steering actuation. Finally, both scenarios were augmented to see how well the RRT^X could handle dynamically changing obstacles. For the previous lane change scenario, a car unpredictably changes lanes, thus interfering with the motion planners initial path. Similarly, a limited sensor range was simulated for the seaport scenario, such that many shipping containers were discovered as the vehicle was traversing the workspace. In both cases, the sample-based planner approach was able provide real-time and feasible plans for the controller to execute at low speeds while maintaining a safe distance away from nearby obstacles.

With the ever-increasing interest in self-driving transportation, this research into motion planning has the potential to push truck and trailer systems into increasing complex areas. Since the research has been demonstrated to operate real-time on a physical prototype system, there is an opportunity to apply the motion planning framework on other systems that routinely navigate unstructured environments. With further optimization of the motion planning code, the planning time can be consistent enough for implementation on practical applications like freight delivery and transport within parking lots and seaports.

Even though the implementation of this research was on a heavy truck and trailer system, the methods developed via this research can be applied to any car-like system with a trailer. This include warehouse carts which offer an exciting potential area of further research into collaborative planning within a locally contained environment. By modifying the mission and cost function, the motion planner could be augmented in order to optimize for the various needs of a fast paced warehouse environment.

5.2 Further Study

While the current testing conditions allowed for a proof of concept testing platform for the motion planning algorithms, the integration of actual sensors onto the system would provide some interesting insight into the feasibility of scaling this sample-based approach for truck and trailer systems. Although the RRT^X planner can account for unpredictable obstacles, the added noise and constant fluctuation in the estimated pose of nearby obstacles could reduce how quickly the planner can update its graph and thus reduce the performance.

Since sample-based algorithms are only as effective as their heuristic and cost function, a more informed cost metric that considers elements like the trailer angle, steering actuation, distance to nearby obstacles, and other desired characteristics could drastically improve the convergence of the motion planner, especially for desired trailer angles that are drastically different from the truck heading.

Additionally, alternative steering functions could be considered to improve the overall path generation from the motion planner. For instance, adding the reverse capability by using a Reeds-Shepp curve would open another set of possible use cases. Another option is to generate clothoids that sacrifice some optimality but generate paths with continuous curvature and its derivative, thus allowing for paths that are feasible even with a limited steering rate.

Last but not least, even though the modified ProStar truck provided a general understanding of the truck and trailer system, attaching an actual trailer and subsequent encoder to the experimental setup would create a more realistic simulation to the conditions faced by truck and trailer systems in the real world.

REFERENCES

[1] B. Paden, M. Cap, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," 2016.

[2] Google Maps, "Houston-Galveston Are Council, Maxar Technologies, USDA Farm Service Agency," 2020.

[3] Google Maps, "Houston-Galveston Are Council, Maxar Technologies, Texas General Land Office, U.S. Geological Survey," 2020.

[4] G. Seetharaman, A. Lakhotia, and E. P. Blasch, "Unmanned vehicles come of age: The darpa grand challenge," *Computer*, vol. 39, no. 12, pp. 26–29, 2006.

[5] M. Buehler, K. Iagnemma, and S. Singh, *The DARPA urban challenge: autonomous vehicles in city traffic*, vol. 56. springer, 2009.

[6] World Health Organisation, "Global status report on road safety 2018," 2018.

[7] Bureau of Transportation Statistics, "June 2018 north american freight numbers," Aug 2018.

[8] S. Standard, "J3016," *Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems*, vol. 4, pp. 593–598, 2014.

[9] S. Tsugawa, S. Jeschke, and S. E. Shladover, "A review of truck platooning projects for energy savings," *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 1, pp. 68–77, 2016.

[10] C. Bergenhem, S. Shladover, E. Coelingh, C. Englund, and S. Tsugawa, "Overview of platooning systems," in *Proceedings of the 19th ITS World Congress, Oct 22-26, Vienna, Austria (2012)*, 2012.

[11] A. Charlton, "These 7 companies are leading the autonomous truck race," May 2019.

[12] Embark Trucks Inc., "Embark trucks," *Embark Trucks*, 2018.

[13] S. Blanco, "An autonomous semi-truck just drove across america to deliver butter," Dec 2019.

[14] J. Park, "Daimler nixes platooning, focuses on automation," Jan 2019.

[15] S. Kang, H. Ozer, and I. L. Al-Qadi, "Benefit cost analysis (bca) of autonomous and connected truck (act) technology and platooning," in *International Airfield and Highway Pavements Conference 2019: Innovation and Sustainability in Highway and Airfield Pavement Technology*, pp. 174–182, American Society of Civil Engineers (ASCE), 2019.

[16] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[17] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *CoRR*, vol. abs/1105.1186, 2011.

[18] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[19] M. W. Otte and E. Frazzoli, "Rrtx: Real-time motion planning/replanning for environments with unpredictable obstacles," in *WAFR*, 2014.

[20] A. Stentz *et al.*, "The focussed dˆ* algorithm for real-time replanning," in *International Joint Conference on Artificial Intelligence*, 1995.

[21] O. Holmer, "Motion planning for a reversing full-scale truck and trailer system," 2016.

[22] Y. Kuwata, J. Teo, S. Karaman, G. Fiore, E. Frazzoli, and J. How, "Motion planning in complex environments using closed-loop prediction," in *AIAA Guidance, Navigation and Control Conference and Exhibit*, p. 7166, 2008.

[23] O. Ljungqvist, N. Evestedt, D. Axehill, M. Cirillo, and H. Pettersson, "A path planning and path-following control framework for a general 2-trailer with a car-like tractor," 2019.

[24] A. Sakai, "Hybrid AStar Trailer: A path planning algorithm based on Hybrid A* for trailer truck." `https://github.com/AtsushiSakai/HybridAStarTrailer`, 2018.

[25] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Path planning for autonomous vehicles in unknown semi-structured environments," *I. J. Robotic Res.*, vol. 29, pp. 485–501, 04 2010.

[26] C. Chen and M. Tomizuka, "Dynamic modeling of tractor-semitrailer vehicles in automated highway systems," *Institute of Transportation Studies, UC Berkeley, Institute of Transportation Studies, Research Reports, Working Papers, Proceedings*, 01 1995.

[27] A. J. Weinstein and K. L. Moore, "Pose estimation of ackerman steering vehicles for outdoors autonomous navigation," in *2010 IEEE International Conference on Industrial Technology*, pp. 579–584, IEEE, 2010.

[28] H. Johnson, "An application of the maximum principle to the geometry of plane curves," *Proceedings of The American Mathematical Society - PROC AMER MATH SOC*, vol. 44, 02 1974.

[29] E. Bakolas and P. Tsiotras, "On the generation of nearly optimal, planar paths of bounded curvature and bounded curvature gradient," in *2009 American Control Conference*, pp. 385–390, IEEE, 2009.

[30] E. Bergstein, *Dirty Dancing*. Great American Films Limited Partnership, 1987.

[31] J. Reeds and L. Shepp, "Optimal paths for a car that goes both forwards and backwards," *Pacific journal of mathematics*, vol. 145, no. 2, pp. 367–393, 1990.

[32] C. Altafini, "Some properties of the general n-trailer," *International Journal of Control*, vol. 74, pp. 409–424, 03 2001.

[33] J. Salmon, *Guidance of an Off-Road Tractor-Trailer System Using Model Predictive Control*. PhD thesis, Auburn University, 2013.

[34] B. Houska, H. Ferreau, and M. Diehl, "ACADO Toolkit – An Open Source Framework for Automatic Control and Dynamic Optimization," *Optimal Control Applications and Methods*, vol. 32, no. 3, pp. 298–312, 2011.

[35] I. Bae, J. Moon, and J. Seo, "Toward a comfortable driving experience for a self-driving shuttle bus," *Electronics*, vol. 8, no. 9, p. 943, 2019.

[36] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – A software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, In Press, 2018.

[37] L. E. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," *American Journal of Mathematics*, vol. 79, no. 3, pp. 497–516, 1957.

[38] R. A. Knepper and A. Kelly, "High performance state lattice planning using heuristic look-up tables," in *Proceedings of 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3375 – 3380, October 2006.

[39] AutonomouStuff, "Products in use: Customer applications," Feb 2020.

[40] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, p. 5, Kobe, Japan, 2009.

[41] R. Grace, A. Guzman, J. Staszewski, B. Peters, M. Mallis, and D. Dinges, "The carnegie mellon trucksim: A tool to improve driving safety," in *17th DASC. AIAA/IEEE/SAE. Digital Avionics Systems Conference. Proceedings (Cat. No. 98CH36267)*, vol. 2, pp. I35–1, IEEE, 1998.

[42] dSPACE, "Asm truck: Truck model for tractor and trailer simulation in real time," Feb 2020.

[43] IPG Automotive, "TruckMaker," Jan 2020.

[44] B. Wu, A. Wan, X. Yue, and K. Keutzer, "Squeezeseg: Convolutional neural nets with recurrent crf for real-time road-object segmentation from 3d lidar point cloud," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1887–1893, IEEE, 2018.

[45] S. R. Richter, V. Vineet, S. Roth, and V. Koltun, "Playing for data: Ground truth from computer games," in *European conference on computer vision*, pp. 102–118, Springer, 2016.

[46] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," *arXiv preprint arXiv:1711.03938*, 2017.

[47] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and service robotics*, pp. 621–635, Springer, 2018.

[48] SCS Software, "American Truck Simulator," Feb 2016.

[49] G. M. Hoffmann, C. J. Tomlin, M. Montemerlo, and S. Thrun, "Autonomous automobile trajectory tracking for off-road driving: Controller design, experimental validation and racing," in *2007 American Control Conference*, pp. 2296–2301, IEEE, 2007.

[50] X. xu, L. Zhang, Y. Jiang, and N. Chen, "Active control on path following and lateral stability for truck–trailer combinations," *Arabian Journal for Science and Engineering*, vol. 44, 09 2018.

[51] X.-Y. Lu and J. K. Hedrick, "Longitudinal control design and experiment for heavy-duty trucks," in *Proceedings of the 2003 American Control Conference, 2003.*, vol. 1, pp. 36–41, IEEE, 2003.

[52] F. Zhu, L. Ma, X. Xu, D. Guo, X. Cui, and Q. Kong, "Baidu apollo auto-calibration system-an industry-level data-driven and learning based vehicle longitude dynamic calibrating algorithm," *arXiv preprint arXiv:1808.10134*, 2018.

[53] R. Oliveira, P. Lima, G. Pereira, J. Mårtensson, and B. Wahlberg, "Path planning for autonomous bus driving in urban environments," 05 2019.

[54] E. Nordeus, "Explaining the hybrid a star pathfinding algorithm for selfdriving cars," Nov 2015.

[55] S. P. Wood, J. Chang, T. Healy, and J. Wood, "The potential regulatory challenges of increasingly autonomous motor vehicles," *Santa Clara L. Rev.*, vol. 52, p. 1423, 2012.

[56] R. Oliveira, P. F. Lima, M. Cirillo, J. Mårtensson, and B. Wahlberg, “Trajectory generation using sharpness continuous dubins-like paths with applications in control of heavy duty vehicles,” 2018.