5-2023

Analyzing the Influence of Stale Data on Autonomous Intelligent Transportation Systems

August St. Louis
amstlou@g.clemson.edu

Analyzing the Influence of Stale Data on Autonomous Intelligent Transportation Systems

---

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Masters of Computer Engineering
Computer Engineering

---

by
August Maxwell St. Louis
May 2023

Accepted by:
Dr. Jon C. Calhoun, Committee Chair
Dr. Jerome McClendon
Dr. Melissa C. Smith

Abstract

Intelligent transportation has been at the forefront of recent technological advancement. Individuals have developed a number of algorithms intended to automate and improve essential intelligent transportation functions. New developments include the incorporation of vehicle platooning and path planning algorithms within a number of use cases. Data perturbation can affect both algorithms significantly. We define data perturbation as any natural or unnatural phenomenon that causes the data to be skewed in any way. Perturbations within either system can cause its respective algorithm to operate with stale or incorrect data. This can significantly affect performance. This paper conducts a fault injection campaign to analyze the impact of data perturbations in platooning and path planning models. This campaign enters perturbed data into each model to simulate the several unknown occurrences that may arise. Our analysis provides an understanding of model parameter sensitivity for causing system failures. By understanding which parameters are most influential to the fidelity of the model, we gain the ability to make intelligent transportation algorithms safer.

Dedication

I dedicate this work to my family and close friends who have supported me throughout my graduate school education. I would like to thank Dr. Jon C. Calhoun for his support as my research advisor, and also Dr. Jerome McClendon and Dr. Melissa Smith for their assistance on my thesis committee.

Acknowledgments

This work was supported by the Simulation Based Reliability and Safety Program for modeling and simulation of military ground vehicle systems, under the technical services contract No. W56HZV-17-C-0095 with the U.S. Army DEVCOM Ground Vehicle Systems Center (GVSC). This material is based upon work supported by the National Science Foundation under Grant No. SHF-1910197 and SHF-1943114.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Intelligent Transportation Systems

The need for mobility has increased the importance of transportation around the world. However, are large amount of vehicles on the roads contribute to traffic congestion, along with unpredictable emergencies and accidents [10]. Additionally, transportation is a fundamental base for economic growth for many countries but problems with uncontrolled growth in traffic can cause a number of issues for society:

- delays
- traffic jams
- higher fuel prices
- increase of CO_2
- accidents and emergencies

These issues with transportation that can be solved with a number of methods. One of which is to provide these vehicles with intelligence. Intelligence can mean many things, but specifically, in the case of this thesis, we refer to intelligent transportation systems (ITS) as vehicles that use next generation technologies. ITS can usually run autonomously with robust perception algorithms and information processing systems [44]. These systems are also provided with actuators that perform actions based on the information it has perceived and the actions it has chosen. ITS are seeing increasing use in everyday purposes. They can be used to fix many current problems

by utilizing vehicle-to-vehicle and vehicle-to-infrastructure communication technology to improve traffic congestion, subsequently fixing many other problems [17]. ITS can also utilize algorithms to enable the vehicle to drive autonomously in several environments. Several ITS use cases depend on modern algorithms to operate effectively. For example, use cases can be internal for a vehicle, like how it will plan a path through an environment. Use cases can also incorporate several vehicles to achieve a goal, like a platooning algorithm.

Platooning vehicles move together as a unit, communicating with each other to navigate the changing environment safely, dodging obstacles. Path planning is a grid-based search algorithm that develops successful routes through an environment. While these technologies are growing increasingly helpful, there are certain problems associated with each. First, platooning models require a large dependency on data collection and communication. Issues with sensors or communication systems can cause significant problems for the system. There are several uncertainties that can affect system fidelity. Small errors in data accuracy can lead to system failure under certain circumstances. Second, there are a large amount of robust path planning algorithms currently available. However, a path planner is only as successful as its map data is accurate and current. The main danger to path planning algorithms is the fidelity of map data. Several problems can arise when map data is unknowingly incorrect or out-dated. Map data can change weekly or daily depending on weather conditions in the environment. Also remote sensing techniques can produce maps with small errors.

Platooning and path planning are significant algorithms that require a large amount of accuracy to ensure safe and reliable functionality of corresponding systems. The effectiveness of ITS depend largely on their methods of accessing, collecting, and processing accurate data [17]. The main motivation of this paper is to understand how effective those algorithms will be in the presence of wrong or old data. To test the fidelity of these models under stress, we purposefully inject errors into data. Fault injection techniques that perturb data introduce stale data into models. Stale data refers to a system with data that is not updating regularly. Systems with stale data commonly experience errors due to inaccurate assumptions based on incorrect data. Stale data injection is a useful tool to test the ability of an algorithm to operate with false and/or old data. Understanding the effects of data perturbance in ITS algorithms allows for better understanding of the algorithms and provides us with a chance to possibly mitigate the weaknesses present within them.

Chapter 2

Stale Data Analysis in Intelligent Vehicle Platooning Models

2.1 Importance of study

This study investigates the effects of data perturbance and stale data on a platooning model. Our research focuses on determining which model variables that are most sensitive to the perturbations included. As these variables are highlighted, it allows individuals to introduce mitigatory guidelines within the algorithm to increase its robustness. More robust algorithms means safer and more reliable systems.

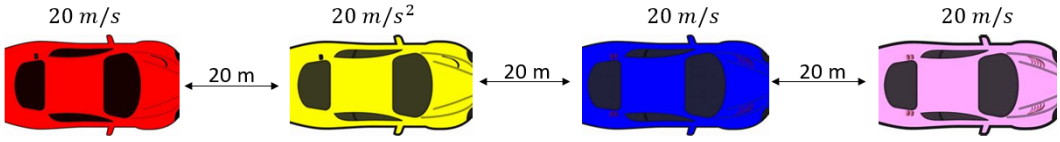
2.2 Introduction

In recent years, intelligent transportation systems have seen several advancements in collision avoidance and safety improvements [39]. The increased robustness of perception and communication technologies have allowed for a better situational awareness between vehicles [43]. Better situational awareness has allowed for the implementation of autonomous vehicle platooning. The vehicles in the platoon can not only understand their own surroundings, but are able to understand the states of the other vehicles in the platoon. While this provides each vehicle with a more holistic view of its environment, it also requires communication between each vehicle. While navigating the

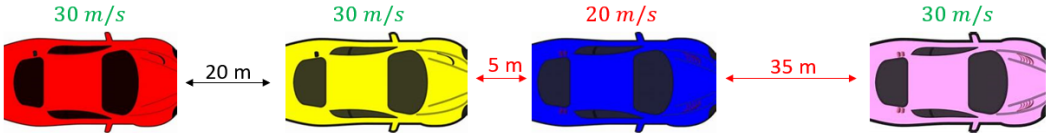
environment, vehicles communicate certain parameters such as speed, acceleration, and position to following vehicles. Following vehicles calculate their next step by considering other vehicles' current parameter measurements and adjusting its own to follow behind at a safe distance and speed. This update needs to occur every second, as changes in conditions can occur in an instant. This introduces a significant problem, what if something interrupts that flow of information? A connection issue can cause a vehicle to go blind to the other vehicles in the network. The longer the communication interference, the greater the chance the vehicle inevitably crashes. Connection issues arise in many ways, including: denial of service (DoS), false data injection, and modification attacks [16]. These systems depend greatly on communication between vehicles to account for obstacles in the environment. A perturbation of as little as a couple seconds can be enough to cause a crash.

2.2.1 Stale Data

There are several vulnerabilities that, if exploited, negatively influence the autonomous vehicle platooning system. Perception and communication devices are integral to a platooning system. If there is interference within one of those devices, the system has problems updating. significant vulnerabilities include interfering with a vehicle's electric control unit (ECU). The ECU controls data processing and connection between the vehicle and other entities [16]. Disrupting the ECU causes errors in both perception and communication. Jamming attacks prevent sensor information from being translated to the ECU, false data injection attacks send spoofed information to the ECU, and a DoS attack bombards the ECU with too much information, making the ECU incapable of collecting data from vehicle sensors [16]. In response to an interference, the system commonly returns the last known values for perturbed data; this is called stale data. Stale data refers to a system with data that is not updating regularly. Systems with stale data commonly experience errors due to inaccurate assumptions based on incorrect data. Figure 2.1 displays the effects of stale data on a 1D platooning model. The effects of stale data on a 1D platooning model are displayed in the blue car. Originally, all the vehicles are moving at a constant 20 m/s, 20 meters apart. As the vehicles increase their velocities to 30 m/s, the blue car's velocity fails to update and remains unchanged. This causes the car to have uneven spacing and in the worst case will lead to an eventual crash.



(a) Platoon before stale data.



(b) Platoon after stale data.

Figure 2.1: 1D platooning model before and after stale data.

2.2.2 Contributions

The goal of this chapter is to analyze the effects of introducing stale data into a 1D platooning model. We inject perturbations into the model to test which model variables are most sensitive to a perturbation; this is which variable, if perturbed, will most likely result in a model failure. Observing aggregate simulation results allows us to discover which variables and variable settings are most sensitive. Determining the sensitive part of the model allows enables development of measures to make the model more fault-tolerant. This chapter makes the following contributions:

- Presents a portable methodology for injecting stale data into a MATLAB/Simulink model.
- Analyzes impact of stale data in a 1D platooning model.
- Results show our 1D platooning model is sensitive to stale data, with 33% of simulation instances ending in collision.
- Conveys that sensitivity varies based on which model variable is observed.

The following sections of this chapter are as follows. Section 2.3 presents a background on vehicle platooning, stale data, and fault injection. Section 2.4 introduces our injection methodology and evaluation metrics. Section 2.5 provides a detailed analysis of the effect of stale data on a 1D platooning model. Section 2.6 discusses study trends. Section 2.8 concludes the chapter and provides future work considerations.

2.3 Background

2.3.1 Vehicle Platooning

Current implementations of intelligent transportation systems are becoming increasingly robust, but are not perfect. Several accidents relating to self-driving vehicles occur because of a lack of knowledge about that vehicle's surroundings [13]. In this work, researchers have explored several methods of mitigating the dangers associated with self-driving vehicles. One of which is cooperative driving. A common cooperative driving application is called Cooperative Adaptive Cruise Control (CACC). CACC and cooperative driving are common implementations of vehicle platooning. Applications of CACC use vehicle-to-vehicle communication (V2V). This methodology allows a vehicle to obtain information from a preceding vehicle in order to inform its own next step. This allows a system of vehicles to better anticipate problems and react quickly to those problems. Possible problems that may arise are influenced by several unknowns. Significant unknowns include adverse weather conditions, physical obstacles, and foreign entities to the network (e.g. other vehicles, animals, or humans). CACC has a significant positive effect on traffic safety and efficiency [46].

The vehicles within an Autonomous Vehicle Platoon (AVP) are split up into two groups, platoon leader (PL) and platoon followers (PF) [49]. Newer implementations of AVPs use an effective Reputation-based Leader Election scheme that observes the trustworthiness of each vehicle in the platoon based on past actions and trips. This framework decides which vehicle becomes the PL. The PL has the greatest responsibility and has a direct influence on the actions of the platoon. The PL is tasked with dynamically monitoring road conditions, collecting and processing information, and issuing driving instructions to PFs [49]. Utilizing a vehicle platoon increases fuel efficiency by greatly decreasing wind resistance to the PFs [49]. To offset the increased drag on the PL, the PFs share some of the fuel they have saved from reduced wind resistance. While several aspects of AVPs will not be directly implemented in our 1D platooning model, the topics discussed in this section greatly influence real life implementations for vehicle platooning.

2.3.2 Stale Data

Self-driving operating mechanisms are controlled and monitored by computer-based algorithms [42]. Data and information fidelity are nontrivial aspects of intelligent transportation systems. Attacks on intelligent systems require insight into the failure conditions of the equipment, control

principles, process behavior, signal processing, etc. [32]. In this work, Krotofil et al. discuss how attackers introduce stale data into systems. They describe how that interference can propagate in other areas of the system. Programmable logic controllers (PLC) are entities that are used as automation controllers. PLCs operate in a scan cycle architecture, meaning their control logic uses the last saved input values to relay commands to actuators [32]. An attacker interferes with a PLC by jamming its sensor input readings, forcing the system to continuously read in the same values. Attackers also jam the connection between the PLC and an actuator, allowing the state of the controller to update but blocking the system from actually acting on the update [32]. Several of these principles are common to stale data attacks. The way in which we inject stale data into our model is largely similar by the methodologies described in Krotofil et al [32].

2.3.3 Fault Injection

Faults can be categorized as either hard or soft and introduce errors into a system [1]. A hard fault is systemically reproducible. An example of a hard fault is the inability to communicate to a vehicle that is offline. A soft fault is a fault where activation is not systematically reproducible. These errors are often transient, such as dropped messages and data corruption via cosmic radiation [38].

As integrated circuit designers and manufacturers explore more robust technologies in circuit design, sensitivity in these circuits become a non-trivial issue [34]. In this work, the authors describe the necessity of dependability analysis in combating several natural and deliberate perturbations. These perturbations are examples of system faults; faults can formulate in a number of different ways. Particle strikes and electromagnetic interference are examples of natural system perturbations. The presence of natural phenomena will result in faulty logical behavior and possibly application failures. A deliberate fault-based attack can include lasers that are utilized to hack critical data stored in circuits, such as cryptographic keys and other security features. The presence of a fault can result in application failure either from an erroneous value induced on a circuit output, or from an erroneous sequential behavior due to one or more incorrect bits in internal registers [34]. These internal errors are defined as soft errors.

In order to inject faults into a computing system, software based fault injectors represent low-cost and flexible methods by corrupting values in the executing binary [35], values at the register level using a compiler [7], or perturbing communication [14]. In this work, we inject stale data into our simulation at a software level using a Simulink module.

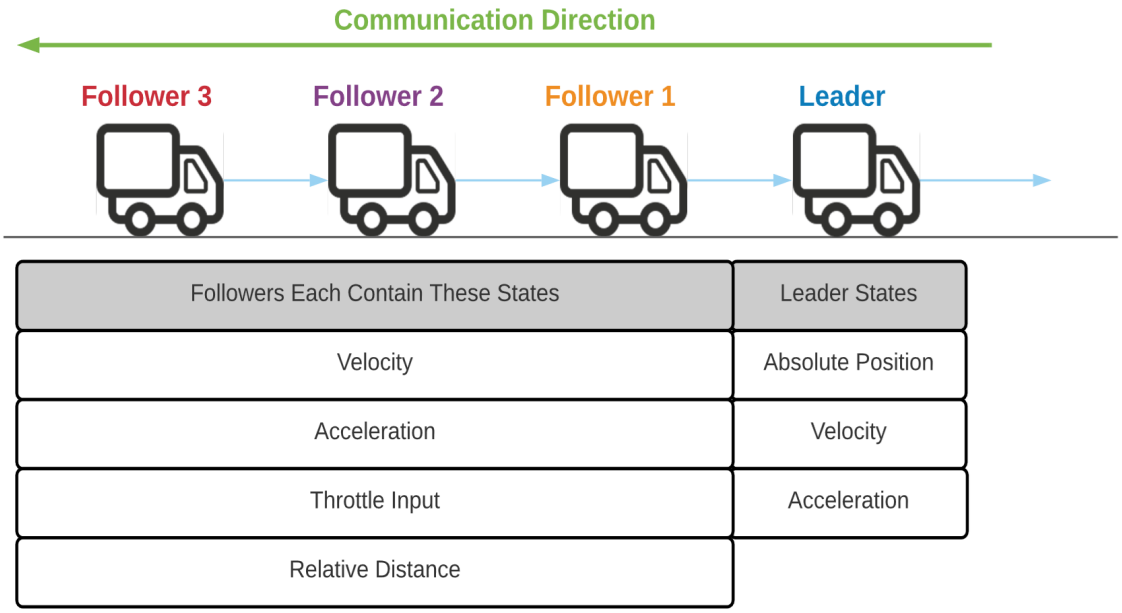


Figure 2.2: Vehicle features of our 1D platooning model.

2.4 Methodology

2.4.1 Model Introduction

Xuan and Naghnaeian [47] present a mathematical formulation of a 1D platooning model that includes a lead vehicle and a variable number of follower vehicles. The model updates continuously, relaying information from each vehicle throughout the system. Figure 2.2 displays the vehicle features for the follower and lead vehicles: relative distance, position, velocity, throttle input, and acceleration are collected and stored within the model's database. These values are then used in mathematical equations to calculate the current states of the other vehicles in the model. The updated information is integral in allowing each vehicle to *see* the vehicle to its anterior. The results of these equations control the simulation's trajectory.

2.4.2 Stale Data Injection

The model simulates vehicles traveling down a road in 1D with a fixed distance between each vehicle. Each vehicle communicates with the vehicle to its immediate anterior. As discussed before, it is possible that data will be communicated late or not at all. At any juncture, and

for any duration, the model can experience a disruption. In the occurrence of a lapse in data, the model relays stale data. In order to inject stale data into the model, we must implement a custom Simulink block written in MATLAB. This Simulink block is placed on the input signals of our platooning model [21]. This allows us to simulate stale data injections by essentially turning off the model's input for a specified duration. This method is portable within the MATLAB and Simulink environments and can be applied to other simulations. Observing the perturbed model output allows us to quantify the effect of specific stale data injections that occur at any time-step and for any duration.

Understanding the error propagation associated with different stale data injections allows us to understand not only sensitive model variables, but also important injection junctures and durations. Error propagation impacts the future states of the vehicles. For example, if the acceleration of follower vehicle two is perturbed, and stale data is introduced, the model calculates incorrect values for connected model features. The model sustains the same acceleration value instead of the model calculated next step acceleration value. The relative distance, velocity, and throttle input displays values complimentary to the incorrect, repeated acceleration value. Errors are likely to propagate within the other features of the same vehicle. In most cases, error is likely to be passed on to following vehicles as they also attempt to adjust to perturbed feature values.

2.4.3 Evaluation Metrics

In order to properly analyze the output of a simulation instance, we must define what constitutes a simulation's success or failure. To classify a simulation as a success or failure, we determine whether there has been a crash between any vehicle. A successful simulation features no crashes, while a failed simulation features one or more crashes. In order to determine the presence of a crash, we identify relative distance between vehicles as our evaluation metric. Relative distance is a variable unique to each follower vehicle that represents the distance in meters a vehicle trails the vehicle to its immediate anterior. If at any point in a simulation, the relative distance of a vehicle falls is zero or less, that vehicle has crashed into the vehicle in front of it.

The goal of injecting stale data into the model is to determine what variables are most sensitive. This means which variables, if perturbed, result in the most model failure. To determine which model variables are most sensitive, we observe the frequency of crashes for each variable. This allows us to quantify which variables have a significant effect on the success or failure of the

simulation. Relative distance is considered over other metrics because it is an all-encompassing metric for our purposes. Variations in other metric results in a zero relative distance if the changes are significant enough. For example, if vehicle acceleration is perturbed drastically but does not result in a vehicle crash, the model eventually returns to a steady state.

As explained above, the most important evaluation metric we consider is relative distance. As we discover sensitive model variables, we are essentially defining which model variables have the most effect on relative distance. Another method of quantifying a variable's effect on relative distance is to observe the magnitude of error introduced into the relative distance after a perturbation. We calculate the difference in relative distance values in a perturbed simulation to the baseline fault-free simulation.

2.4.4 Defining Perturbations

We perturb simulations of the model to understand which model parameters are most sensitive. In order to simulate perturbations, we inject stale data into the model. There are three perturbation parameters we use to determine the location and duration of the perturbation: Perturbation Juncture (PJ), Perturbation Duration (PD), and Perturbation Variable (PV).

- **Perturbation Juncture (PJ):** represents a time-step where the perturbation begins.
- **Perturbation Duration (PD):** determines how long the perturbation lasts.
- **Perturbation Variable (PV):** signifies the model feature that is perturbed.

At a random time (PJ) in the model, the values for a feature (PV) will not update for an arbitrary duration (PD). Throughout that duration, the value of that variable remains constant until the injection is complete.

Figure 2.2 displays significant vehicle features that are recorded and updated continuously. The lead vehicle has its absolute position, velocity, and acceleration recorded. The following vehicles have their velocity, acceleration, throttle input, and relative distance to its preceding vehicle recorded. This data is stored in a table that depicts the changes in each feature for each time-step throughout the simulation. Each perturbation propagates error differently throughout its own vehicle and to others. Figure 2.3 displays an example of how a perturbation introduces stale data into the model (i.e. visible in near 40 seconds in the velocity graphic where the velocity stays constant

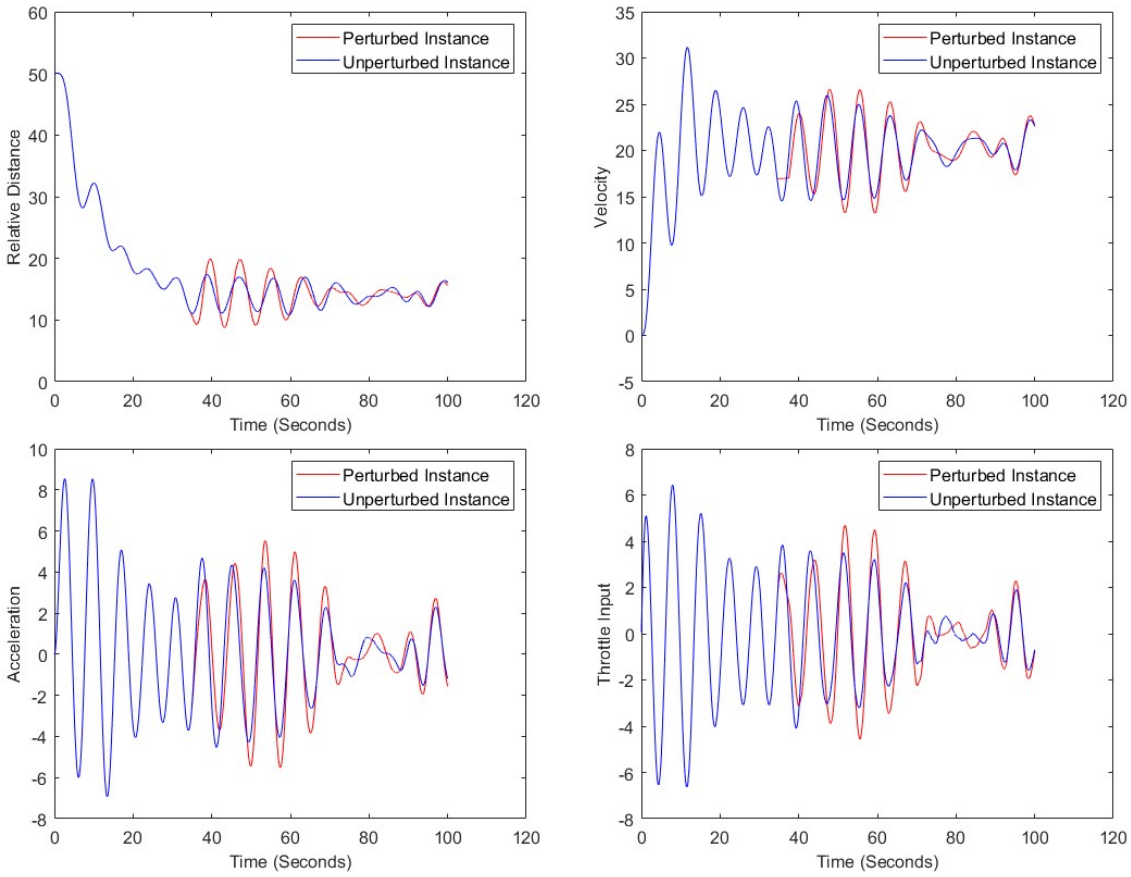


Figure 2.3: The propagation of an error in the vehicle features of follower vehicle 1 due to a perturbation.

for a short amount of time). The presence of stale data skews model values within multiple model features as the error propagates throughout the model.

2.5 Experimental Results

2.5.1 Model Details

The 1D platooning model is run in a MATLAB and Simulink environment. The model is a simulation of a group of four vehicles driving autonomously in a straight line on a road. The model simulates the vehicles through arbitrary start and end points; the vehicles accelerate to a designated speed, 20 meters per second, while sustaining 20 meters between each other. The four vehicles are categorized into two groups, a leader and its followers. A follower vehicle receives the

variable settings from the car directly in front of it. Then, considering those values and its own, updates its variables in the next time-step in order to sustain relative distance requirements.

In order to test for multiple perturbation scenarios, we run 10,000 instances of the model. The three perturbation parameters, selected at random, define the unique fault injection. The PJ is a random number from 1 to 102; this number represents the time-step during the simulation that the perturbation begins. The PD is a duration between 1 and 10 seconds, this is the number of time steps where stale data is entered into the model. The PV is a random number between 1 and 15, each number representing a different model feature to be perturbed. We run all experiments on a Windows 11 workstation with an Intel i9-12900K processor with 64.0 GB of RAM. The software leveraged MATLAB version 9.12.0 and Simulink version R2022a Update 1.

2.5.2 Model Results

At a high level, we look for results that highlight holistic model trends. 33.8% of runs result in a failure. This information is pertinent to understanding model trends, but lacks specificity. To understand what causes model failures, we look to attribute perturbation parameters to failures. To understand which parameter settings influence failures most, we observe four metrics: (1) Percentage of failures per instance, grouped by vehicle feature; (2) Average number of crashes per perturbation, grouped by vehicle feature; (3) Average model failure time, grouped by vehicle feature; and (4) Error propagation patterns for vehicle position in response to a perturbation.

2.5.2.1 Percentage of Failures

By summarizing our output data, we are able to depict the count of failures per each perturbation juncture. Figure 2.4 shows the percentage of runs where at least one vehicle crashes when a specific feature is perturbed. This information is important because it allows us to pinpoint the most significant model features that lead to failures. The common trend among the followers is that acceleration and throttle input are the most sensitive features, as stale data in them leads to crashes nearly 50% of the time, regardless of the vehicle perturbed. Follower 3 has very high fail rates due to its placement in the platoon, resulting in possible corruption when any vehicle suffers stale data. Knowing each model feature's failure percentage provides important information about the sensitivity of each feature. Understanding more model trends provides another level of analysis to base conclusions off. Each subsequent section includes results that make it easier to classify

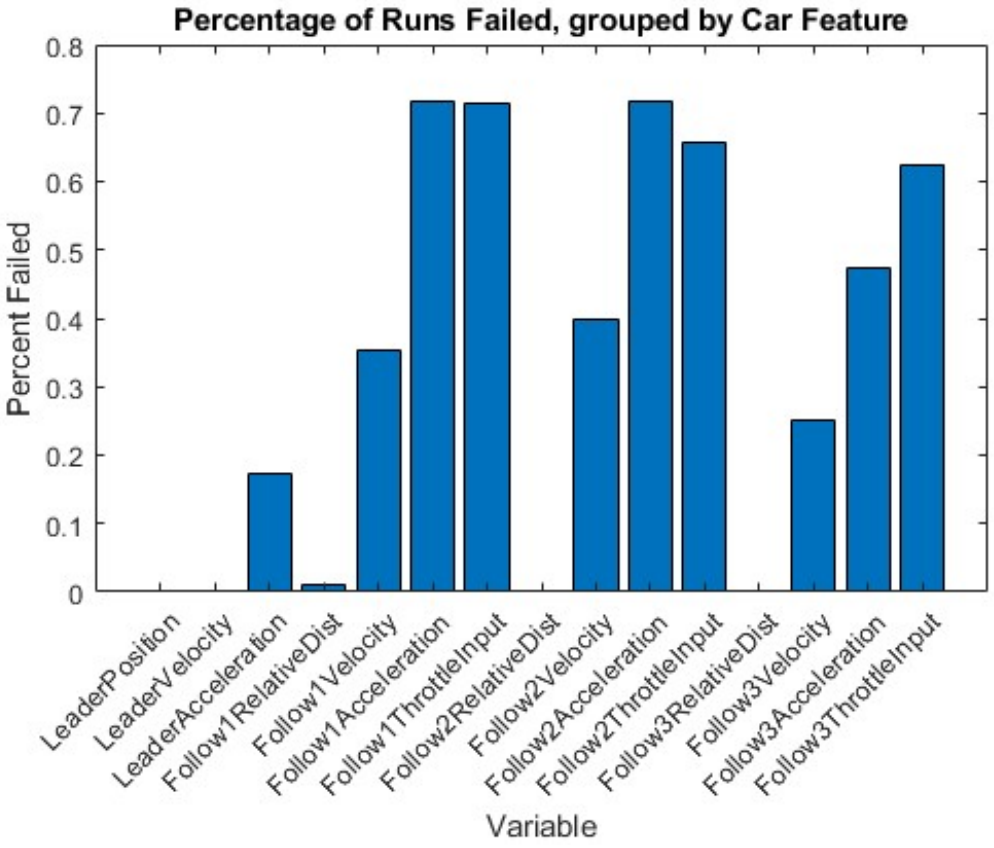


Figure 2.4: Percentage of failures by model feature.

feature sensitivity.

2.5.2.2 Average Number of Crashes

By adding additional specificity to our definition of sensitivity, it is possible to further differentiate model features with similar model sensitivities. Not all failures are created the same. A failure is characterized as at least one vehicle crashing into another. However, there are several cases where more than one vehicle crashes. Although these cases result in the same outcome, a model failure, it is important to differentiate between the two. It may be the case that two variables have similar fail rates for single vehicle crashes, but different fail rates for multiple vehicle crash scenarios. A feature that causes more total crashes is a more sensitive model feature. Figure 2.5 displays the average number of crashes per perturbed variable. Follower 1's throttle input is the most sensitive feature here by far, with 1.6 crashes on average. Other features are much less sensitive. Moreover,

stale data at the front of the platoon is the most sensitive as it can propagate to all vehicles, leading to the higher crash rate. Vehicles at the end of the platoon see the lowest crash rate.

2.5.2.3 Time to Fail

Time to fail is an important metric because it gives further insight into the sensitivity of model features. A failure that occurs after a long amount of time may not occur if injected late enough into a simulation. Additionally, that failure could possibly be avoided if the perturbation duration was decreased. Therefore, a feature with a short time to fail is sensitive to the model. This means that a feature with a short time to fail effects the model significantly in a short time frame. This means that for certain features, the error propagates throughout the model quickly. This usually means the error is propagating quickly within the original vehicle. It can also mean that error is propagating quickly vehicle to vehicle, causing errors in multiple vehicles, further shortening the time for a crash to occur. Figure 2.6 displays the average time to fail values for each variable. Velocity, acceleration, and throttle input are comparable in most cases, except for Follower 3. We believe the large difference in Follower 3 is due to it only neighboring one other vehicle. Without error propagation to other vehicles, the time to failure is extended. Excluding those outliers, observing this graphic does not give a clear, conclusive answer to most sensitive model feature in relation to average time to fail.

2.5.2.4 Error Propagation

In order to properly depict the sensitivity of each variable, an understanding of the error propagation is an integral means of comparing variable sensitivity. The first three result metrics allow us to observe the error propagation within the model. By observing Figure 2.3, we see that a perturbation below a certain threshold negatively affects the system, but eventually, the system self-corrects and reverts to normal. We define this threshold as the minimum amount of error needed for model failure. Therefore, Figure 2.4, the model features that propagate enough error to cause a failure. Figure 2.5 displays the magnitude of error introduced into the model for each feature. A larger number of crashed vehicles means that there is a large amount of error being propagated throughout the system, enough to make multiple vehicles crash. Figure 2.6 shows how fast errors propagate through the system for each model feature.

All model results are in some way related to the patterns associated with error propagation;

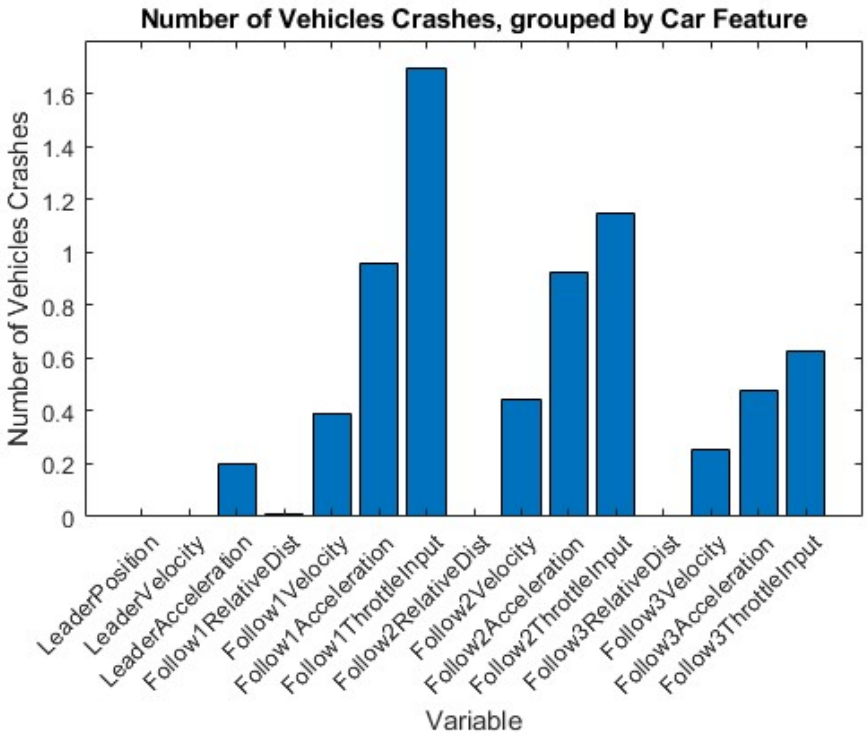


Figure 2.5: Number of crashes by car feature.

therefore, the information provided by error propagation plots is significant. By observing error propagation, we compare exactly how each variable negatively affects the model's output. Figure 2.7 displays the propagation of error through multiple vehicles when the Follower 1 has its throttle input perturbed. This is the average error in relative distance for the time period after a perturbation occurs. We see two patterns occur. The pattern in Follower Vehicle 1 is how the error usually propagates in the vehicle where the perturbation occurs. The error spikes, then slowly returns to zero after a couple peaks. The algorithm notices error within the data and tries to adjust the relative distance readings to their correct values. The peaks likely represent the algorithm's attempts to fix the relative distance while the values change in the opposite direction. For example, the algorithm knows that it needs to change data to depict true value so it may be decreasing relative distance values when the algorithm would normally increase relative distance at that time interval. This would cause a small spike in error. The other followers have a single peak of error that returns to zero after some time. The absence of multiple peaks is likely because the algorithm does not try to self-correct errors passed onto other vehicles, it allows those vehicles to fix themselves through additional calculations.

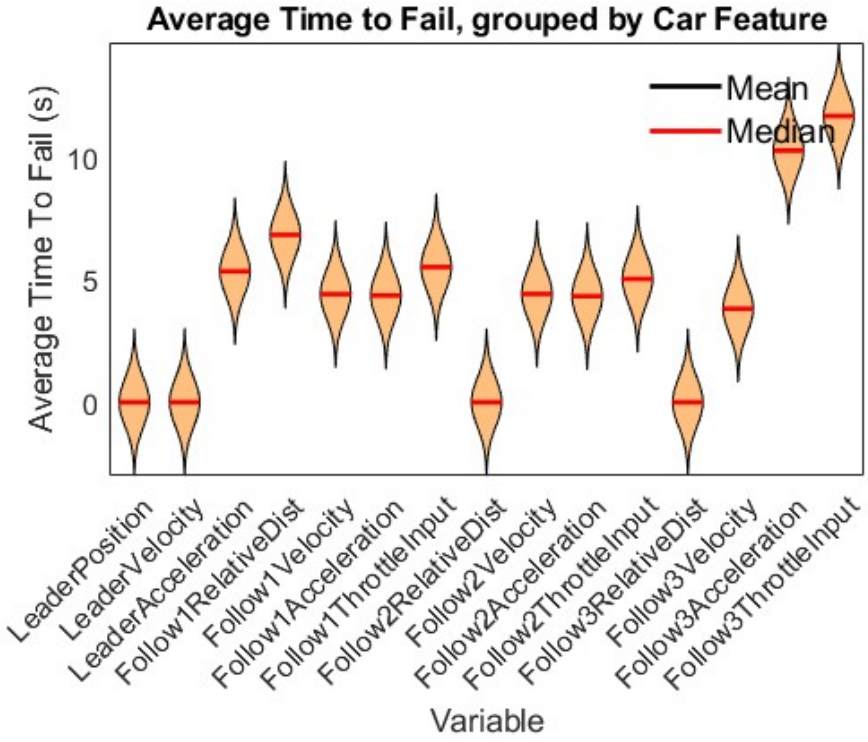


Figure 2.6: Average time to fail for car feature.

The most important aspect of the error propagation graphics is the magnitude of the initial error peak. The higher the error, the greater the chance of model failure. Whichever feature generates the most error should be the most sensitive model feature. After comparing the magnitudes of all model features, we have observed that throttle input in follower vehicle 1 has the greatest magnitude of error on average. Therefore, this feature is the most sensitive.

2.6 Discussion

Observing the model results, we make a couple of conclusions about the most sensitive model variables. Based on our results, we conclude that the most sensitive model features is vehicle throttle input. Regardless of vehicle, a perturbation in one of these model features consistently returns the highest rates of failures in the model and the highest number of car failures per iteration. Additionally, observing the error propagation graphics for these features convey a consistently higher magnitude of error for their perturbations.

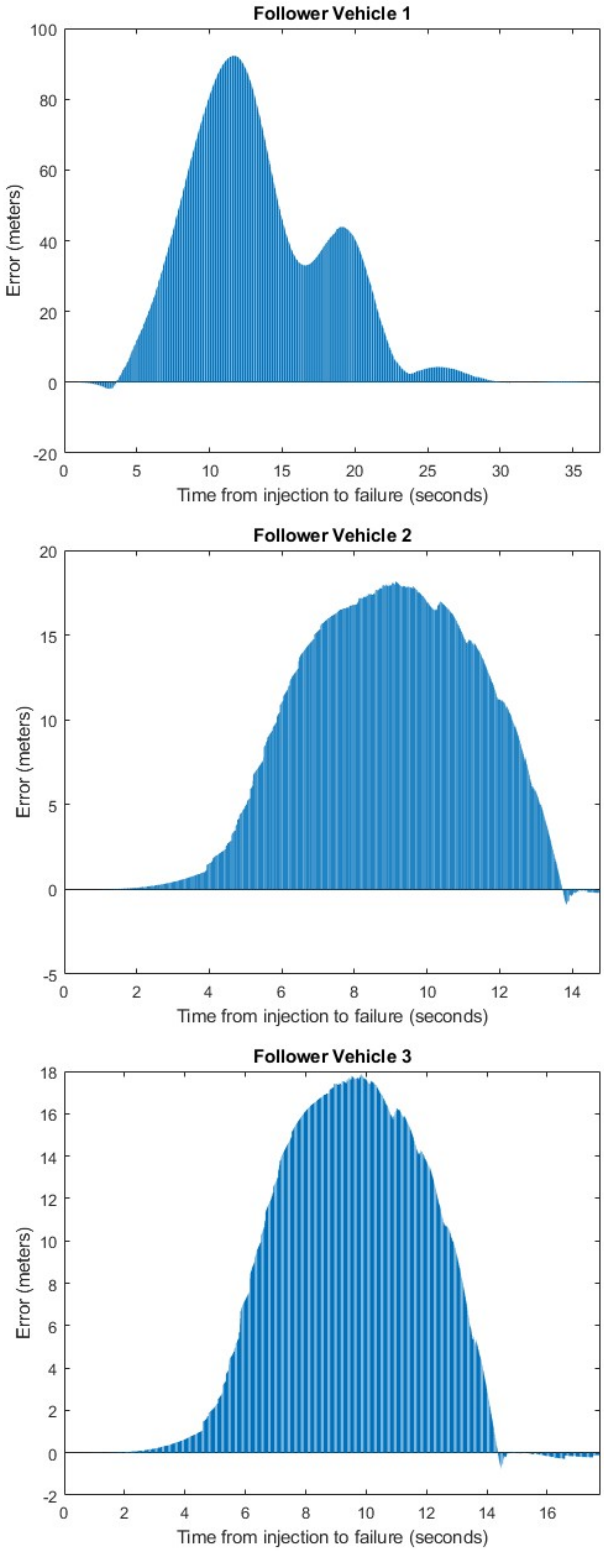


Figure 2.7: An example of the error propagation through different vehicles (Keep in mind the x and y axes are different).

2.7 Related Works

Lin et al. [36] provides a comprehensive overview of the world of ITS. They introduce several problems apparent in transportation now and how ITS alleviates them. The work gives a synopsis on the architecture of ITS, key technologies, and current challenges and opportunities in the area. Deng [8] observes the effects of heavy-duty vehicles (HDVs) on traffic flow. His work focuses more on the interaction of platooning vehicles to the surrounding environment. The framework used is complex and utilizes ACC/CACC algorithms. Applying stale data injections to a complex model like this would introduce additional factors to quantify model failure. Now the vehicles would have to consider other vehicles on the road, making a stale data attack more significant. Jin et al. [26] observes the macroscopic interactions between vehicle platoons and background traffic at highway bottlenecks. This work features multiple platoons operating at once. It is interesting to consider the implications of stale data on multiple communicating vehicle platoons.

2.8 Conclusion and Future Work

This chapter highlights the importance of robust ITS platooning systems that have sufficient situational awareness and fault tolerance. We introduce a 1D platooning model, in which we introduce perturbations into to simulate stale data. The model is simulated tens of thousands of times, introducing varying perturbation variables with each run. Our analysis shows that the most sensitive model features are vehicle acceleration and throttle input.

Future work includes extending our work to a 2D platooning model. The addition of a multidimensional model provides a more complex model equation. This will potentially change the importance of some model features, and subsequently their sensitivity to perturbations. In addition, we will explore techniques to detect and recover from stale data.

Chapter 3

Data Perturbations in Intelligent Path Planning Models

3.1 Importance of Study

This study analyzes the effect of data perturbance in a path planning algorithm. With the inclusion of path planning algorithms in contemporary autonomous vehicle systems, it is increasingly important to ensure that researchers understand how path planners will react to flawed data. Path planners commonly use a pre-processing step where map terrain data is used to model the environment before a path is plotted. If map data is flawed, the path planner is very likely to produce dangerous scenarios by leading vehicles through non-traversable paths.

3.2 Introduction

In recent years, many research areas have been prioritized within the development of autonomous transportation systems. Specifically, the development of environmental perception, path planning, vehicle control, position localization, etc., are ares in which require the most effort to create and improve safe, reliable algorithms [12]. This chapter focuses on path planning as it is an integral aspect of an autonomous vehicle. Path planning is defined as developing a safe, obstacle free route between two points in an environment that has obstacles [31]. In an autonomous vehicle,

its function is to provide path motion operations for vehicles by utilizing environmental perception information [29]. Optimizing path planning algorithms is essential to ensuring safe travel, especially in off-road scenarios [22]. Before considering which of the many path planning algorithms that are suitable for a scenario, it is necessary to ensure the terrain data is accurate. A common method of doing so is taking the terrain map from satellite data. These terrain maps are called Digital Elevation Models (DEM) which yield grid data from remote sensing satellites [2]. For each grid point, there is a corresponding position and height; where position takes form as latitude and longitude location references and height as elevation. There are often additional characteristics for each point that determine whether that grid point is traversable or not.

Path planning algorithms are commonly divided into global and local path planning [2,6,37]. Global path planning focuses on static terrain data, while local path planning accounts for real-time environmental changes. An example of path planning variables that stay relatively constant are position and elevation. Other factors like slope, terrain density, and soil trafficability, on the other hand, can change quite frequently. If either of these values drop below or rise above a certain threshold, that can mean the difference between a grid point being traversable or not. There are several environmental factors like terrain density and soil trafficability that affect traversability. Anything from common weather phenomena like heavy rain in a thunderstorm to more extreme occurrences like hurricanes and tornadoes can cause significant change to an environment overnight. Small changes in the environment can potentially affect large changes in path planning routes.

3.2.1 Contributions

This work seeks to pinpoint the sensitivity in path planning algorithms after observing perturbances in local terrain data. We do not have perfect knowledge of the terrain data, so when our information is off, how is the path planning to be affected? Does the path planner devise a much longer path to dodge an obstacle that is not really there? Does a path planner plan a route through an obstacle that it thought was an open area? By perturbing a terrain map several times and observing aggregate output analysis, we are able to create some understanding of how a path planner reacts to specific changes. This chapter makes the following contributions:

- Provide an in-depth discussion of our proposed path planning algorithm.
- Identify key data elements that are most likely to be inaccurate and devise an injection frame-

work to test the A-Star (A*) algorithm's sensitivity.

- Conduct a fault injection campaign into the terrain data utilized by our A* algorithm. Analyzes impact of data perturbance in an A* path planning model.
- Results show that perturbing less than 1% of map points will result in more than 25% of models failing. Meaning that errors in terrain maps are very significant to path planning algorithms.

The following sections of this chapter are as follows. Section 3.3 presents a background on path planning, A* algorithms, and fault injection. Section 3.4 provides a short model definition along with the cost function utilized in our algorithm. It includes algorithm justification and introduces our injection methodology and evaluation metrics. Section 3.5 provides a detailed analysis of the effect of data perturbance on an A* path planning model. Section 3.6 discusses study trends. Section 3.7 concludes the chapter. Section 3.8 provides a detailed description of future work considerations.

3.3 Background

3.3.1 Digital Elevation Models

Since as early as the 1980s, researchers have been using digital elevation models (DEMs) to develop information about the morphology of land surfaces [25]. There are three types of DEMs, Raster DEM, TIN DEM, and 7.5-Minute DEM. A raster DEM is a two-dimensional image that depicts a map in a grid of pixels, it is also known as a heightmap when representing elevation [3]. The TIN DEM is a vector-based triangular irregular network. The TIN DEM is referred to as a primary (measured) DEM and the Raster DEM is referred to as a secondary (computed) DEM. A 7.5-Minute DEM covers a 30-by-30 meter data spacing. The different types of structure used by each DEM are regular square grids, triangulated irregular networks, and contours, respectively. Figure 3.1 displays the different types of DEMs. The information for DEMs typically come from remote sensing techniques; in some cases they are built from land surveying. Methods for obtaining elevation data used in DEMs include remote sensing techniques like LIDAR, stereo photogrammetry from aerial surveys, multi-view stereo applied to aerial photography, interferometry from radar data, real-time kinematic GPS, topographic maps, theodolite or total station, Doppler radar, surveying and mapping drones, and range imaging [3]. DEMs are commonly used to estimate map characteristics

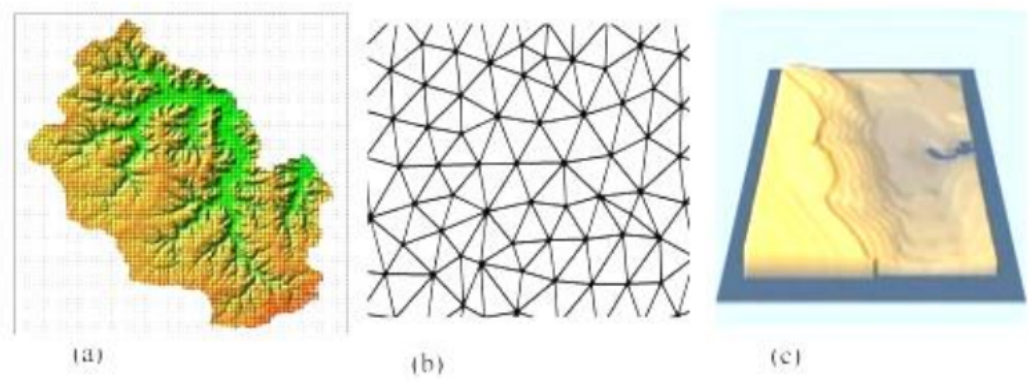


Figure 3.1: Types of DEMs (a) Gridded DEM (b) TIN DEM (c) Contour-based DEM [3].

like elevation, estimate slope and aspect, determine drainage networks, determine the watershed, determine terrain stability, to map soil, and create a profile graph from digitized features of a surface [3].

Our work utilizes a Raster DEM that splits the map into grid points of an arbitrary size. By dividing a terrain map into a collection of grid points, it becomes easy to differentiate between grid points on the basis of several characteristics. Raster processing systems are utilized to use neighborhood operations, making it possible to calculate slope, aspect, and shaded relief [25]. Neighborhood operations will group spatially adjacent cells in groups of eight and compare cell characteristics in order to distinguish certain terrain features like ridges, channels, watersheds, etc. Jenson et al. also states that the products of raster based algorithms are easily vectorized to be used for follow-on calculations [25].

DEMs require a certain accuracy to be accepted. This accuracy is affected by elevation quality and shape and topological quality. Elevation quality is generally defined in terms of absolute or relative accuracy. Shape and topological quality is related to DEM derivatives like slope, aspect, curvature, etc. [40]. External validation is utilized to assess the quality of a DEM; examples of external validation tools are a sparse cloud of points, contour lines, topographic profiles, or a much more accurate DEM [20]. There are three types altimetric errors that can occur in a DEM: gross, systematic, and random errors. Systematic errors are a bias between the modelled surface and the ground truth, and it depends on production technique, especially data acquisition configuration, but also on the interpolation method [23]. Random errors are usually due to the production technique

and are influenced by the quality of the raw data, the processing parameters, and the terrain morphology and vegetation [15]. Gross errors are outliers resulting from faults during the production of the DEM [40]. While the quality of DEMs has improved over the years, they are not perfect. Currently, most DEMs use RMSE or standard deviation of the elevation error to determine the quality of a DEM, [19]. Any reasonable perturbations applied to a DEM would have to not affect the RSME too drastically but can cause significant problems for path planning algorithms (e.g. a low RSME increase in certain variables can be the reason for a path planner to determine whether a point is traversable or not; traveling over a non-traversable point is a possible scenario that could arise from this.

3.3.2 Path Planning

Contributions to path planning algorithms have emerged especially from the 1960s [18]. In recent years, with the advent of autonomous technology, path planning has become an increasingly important area of research [45]. Souissi et al. discuss that at its highest level, path planning problems are organized into three types: holonomic, nonholonomic, and kinodynamic problems. Holonomic problems have a platform where all degrees of freedom are controllable. Nonholonomic problems have differential constraints that cannot be fully integrated to remove time derivatives of the state variable [45]. Kinodynamic problems involve kinematic and dynamic constraints like: avoid obstacles and velocity bounds, and were first introduced in [11]. Souissi et al. further differentiate path planning problems by whether the algorithm requires the environment to be modeled before searching for an optimal/feasible path. Another level of differentiation is determining whether the algorithm is used online or offline. Usually, algorithms that require environmental modeling have to be used offline, as modeling the environment then path planning is computationally intensive. The last level used to distinguish path planning is split into two categories: deterministic and probabilistic models. Popular deterministic methods are the Dijkstra [9] and A* algorithms [18] that are applied with grids and visibility graphs [45]. These algorithms are not the most time effective (computationally) and are not commonly used in real-time environments. Examples of probabilistic algorithms that overcome real-time environment challenges are Particle Swarm Optimization [48], Ant Colony [5], Probabilistic Road Mapping [27], Randomly exploring Random Trees [33], and multi-agent path planning [50]. However, one of these more robust algorithms are not needed for our research, as we have map data provided to use in a pro-processing step.

3.3.3 A* Algorithms

The first version of the A* algorithm was the A1 algorithm introduced by Nils Nilsson, geared towards increasing the speed of Dijkstra's Shortest Path algorithm of 1959 [18]. In 1967, Bertram Raphael made improvements to the approach with his A2 algorithm, but could not display optimally [30]. In 1968, Pete E. Hart proved that A2 was optimal when using a consistent heuristic and the best algorithm with the given conditions [18]. He then coined the algorithm A* which is essentially a summation of all the A algorithms into one term. In respect to Dijkstra's algorithm, the A* algorithm is an improvement in performance, in respect to time, and is achieved with the use of heuristics. The A* algorithm is a search algorithm that takes an input, evaluates a number of possible paths and returns a solution [18]. The A* algorithm takes features of uniform-cost search and pure heuristic search to find optimal solutions [30]. Hart et al. describe two approaches, mathematical and heuristic, included in the formulation of the A* algorithm [18]. The A* algorithm follows the path of the lowest cost, recording a sorted priority queue of alternate path segments of alternate path segments [18, 30].

3.4 Methodology

3.4.1 Model Description

For our scenario, we have an unstructured off-road environment where agents can move to any position (x, y) if the land is traversable. We discretize the environment into a hexagonal grid, where certain environmental proprieties are mapped to the centroid of each grid point (node). An agent moves from the centroid of one hexagon $H(x_i, y_i)$ to another adjacent hexagon, $H(x_j, y_j)$. These properties are the presence of an obstacle and elevation. Each hexagon has a node id (H_i); therefore, the environment can be represented as a combination of nodes ($N = [H_1, H_2, ...H_i]$). The environmental properties are associated with each node and develop a specific cost criteria $H(x_i, y_i, Obstacle_i, Elevation_i)$. Khatiwada et al. design the algorithm to have the same number of paths as the number of agents [28]. Each path (k) can be represented by a series of hexagons ($[H_{start}, H_{k_1}, H_{k_2}, ...H_{target}]$). The algorithm chooses the next node with the help of the multi-criteria cost function. The paths are distinct but still have the ability for crossover if needed (e.g. when crossing a bridge or a narrow passage).

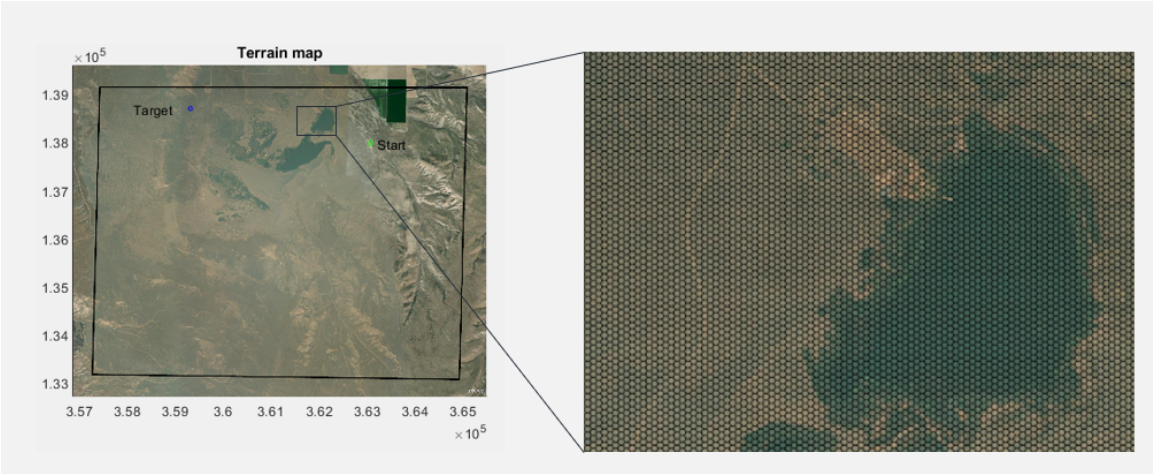


Figure 3.2: Hexagon grid representation of terrain map.

$$c_{ij} = \Sigma_a(W_a * X_a^{ij}) \tag{3.1}$$

Where c_{ij} is the cost to travel from node i to node j, X_a^{ij} is the value of the normalized cost parameter a (distance, obstacle, slope), and w_a is the value of weight assigned to the cost parameter a [28].

3.4.2 Algorithm Justification

Because we need a path planning algorithm for several agents with a cost function that considers several factors, we need to utilize a multi-criteria, multi-agent path planner. We use an A* algorithm because, as opposed to other probabilistic approaches, A* is grid based and the cost functions are determined on a node to node basis [28]. This makes it easier to associate terrain properties to each node and calculate costs based on these properties. It also uses a heuristic in the cost function which helps the algorithm converge to the solution faster. We overlay multi-criteria and multi-agent path planning features onto the A* algorithm for our model.

To summarize, we utilize a Raster DEM which is discretized into a grid of hexagons. Figure 3.2 displays the terrain map in hexagon grid form. We then run a multi-criteria, multi-agent A* path planning algorithm that develops several discrete best paths from start to end points on the DEM. Each grid point has a number of cell characteristics that allow the algorithm to understand whether the vehicle will be able to travel to neighboring cells. The algorithm is split into two parts.

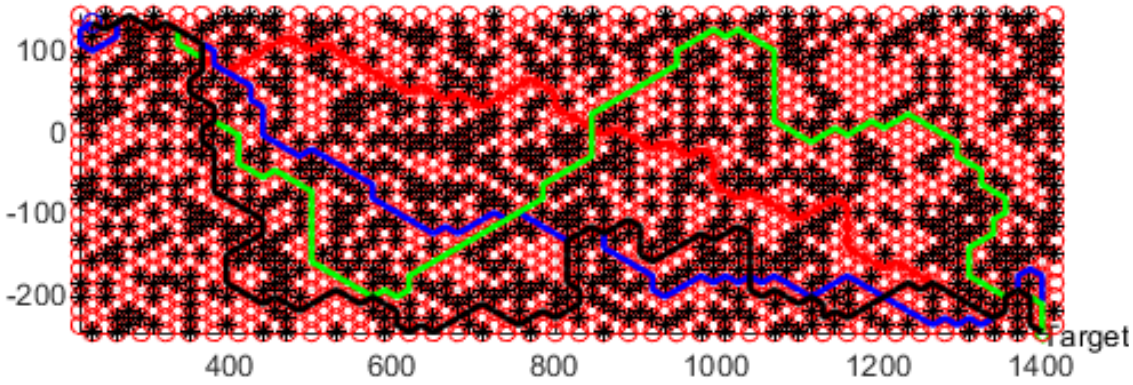


Figure 3.3: Terrain map in grid form with paths overlaid.

1. A pre-processing step that calculates the cost values for each node

2. A multi-criteria, multi-agent path planning paired with a penalty based A* algorithm. In this step, the algorithm also utilizes a multi-criteria cost function weighted with an entropy method described below.

3.4.3 Multi-criteria Weighting & Multi-agent Path Planning

The algorithm utilizes terrain properties (e.g. longitudinal and latitudinal slopes) and the presence of obstacles to understand which neighbors are traversable from the current node. These criteria are normalized using an entropy weighting method. Entropy weighting is an objective weighting method that assigns weights based on uncertainty; this uncertainty is represented by a discrete probability distribution of the cost criteria [24]. This approach produces a different set of weights for each step. Each step will calculate a variable's importance based on its relevance to its neighboring points [41].

The first path is generated using the A* algorithm [18], then every additional path does the same but applies a penalty to all the nodes of the previous path. The path colors correspond to a path number between one and four. In order from one to four, the path colors are red, blue, green, black. Figure 3.3 displays a terrain map in grid form with real paths overlaid on top.

3.4.4 Perturbation Description and Injection Methodology

We currently perturb one variable in the terrain data, the presence of an obstacle. The variable is binary, confirming or denying the presence of an obstacle. Perturbations will be entered into the map data. In the pre-processing step, the map is read into the algorithm to develop the cost functions for each node. If cost variables are perturbed, the costs from node to node will change. Changes in cost between neighboring nodes likely result in the path planner developing a new path. This can either turn an 'go' node into 'no go' node or vice-versa. This can cause the path to go through non-traversable nodes or ignore traversable nodes.

The model is run for ten-thousand iterations. Each iteration, there are 10 grid points perturbed (i.e., a little less than 1% of the total grid points). We run all experiments on a Windows 11 workstation with an Intel i9-12900K processor with 64.0 GB of RAM. The software leveraged MATLAB version 9.12.0 and Simulink version R2022a Update 1.

3.4.5 Evaluation Metrics

There are several evaluation metrics.

- First, we check to see if the path goes through any non-traversable points. If the path goes through a non-traversable point, we consider it an automatic failure.
- Second, we compare the path lengths of the original path, used with the unperturbed map, to the perturbed path.
- Third, we compare path similarity by comparing the number of similar points.
- Fourth, we compare the path costs of the original path to the perturbed path.

After observing the conditions of each evaluation metric, we should be able to understand how the model is affected in the presence of certain perturbations.

3.5 Experimental Results

Our perturbations cause 24.9%, 21.7%, 37.42%, and 35.94% of runs result in a failure for paths 1 through 4 respectively. This is significant because it conveys that a small perturbation will result in an increasingly large amount of model failures the more paths are plotted. To further

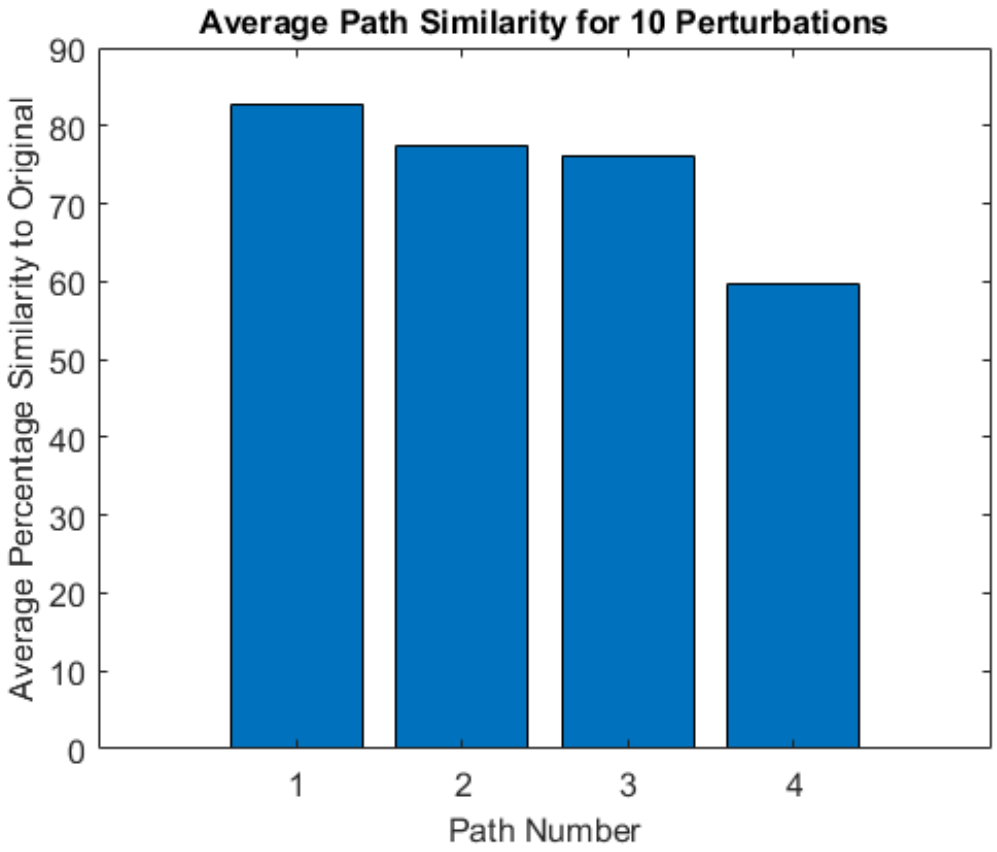


Figure 3.4: Average path similarity, grouped by path.

understand model characteristics, we conduct more analysis by observing three metrics: (1) Average percent similarity of the perturbed path to the original, (2) Average difference in distance between perturbed path and original, (3) Average difference in cost between perturbed path and original. All metrics are grouped by path. All graphics depict the averaged values for each path through ten-thousand trial runs.

3.5.1 Average Percent Similarity

This information is important because it tells us how similar the path is to the original. If it is changed significantly, we know the path planner had difficult constructing the same path. However, in some cases the path planner could have found a better path, so it is important to incorporate other metrics. We know that the original paths are the 4 best paths our path planner developed for an unperturbed map. If many of the paths developed after a perturbance are not

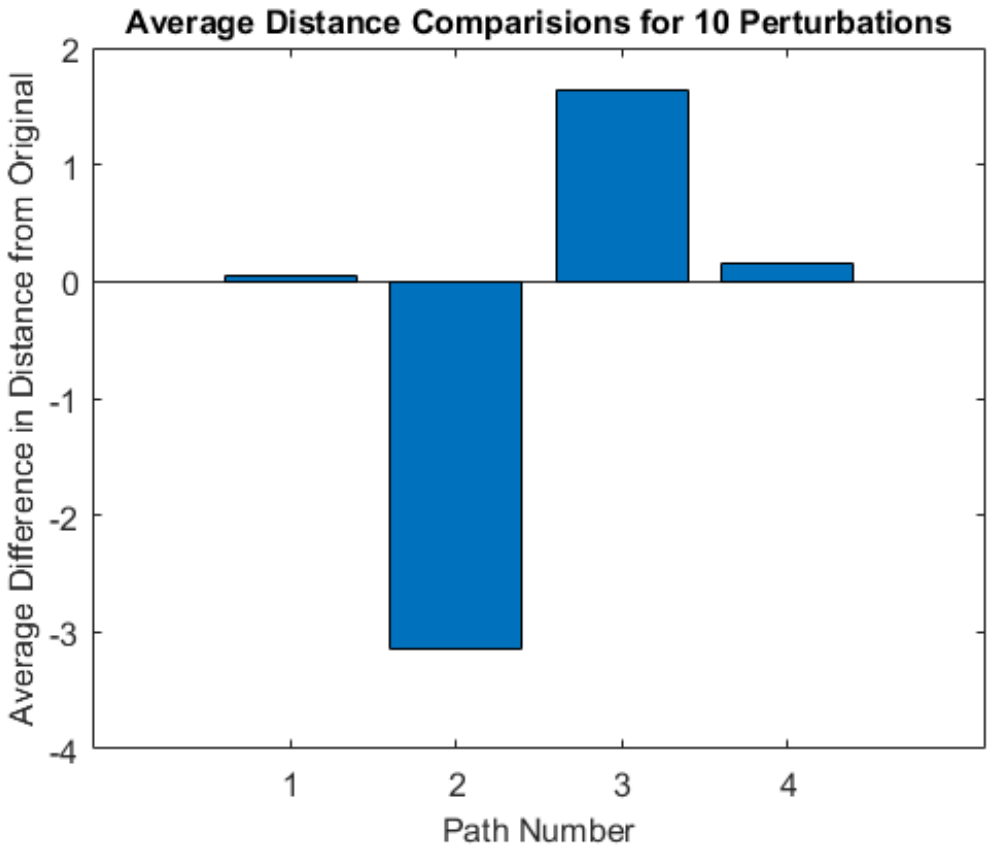


Figure 3.5: Average path similarity, grouped by path.

similar to their originals, we know that the path planner was affected significantly. We can also assume that several of the paths are not optimal, and if they are better than their original, that is likely because it is taking shortcuts through non-traversable points.

Figure 3.4 displays the average percent similarity each perturbed path has with its original path. In this case, similarity is the number of points that occur in both paths. The common trend is that the first path will be the most similar, and as we go from the first to last path, similarity drops in a descending manner. This makes sense because it displays how the first path will likely follow similar points. Because the map is perturbed slightly, it will have to adapt to that change. It makes sense how the other paths have descending similarity because as more paths are created, the less amount of routes is available. Also, fewer points should be available, so the path planner will be forced to find a new path with new points entirely.

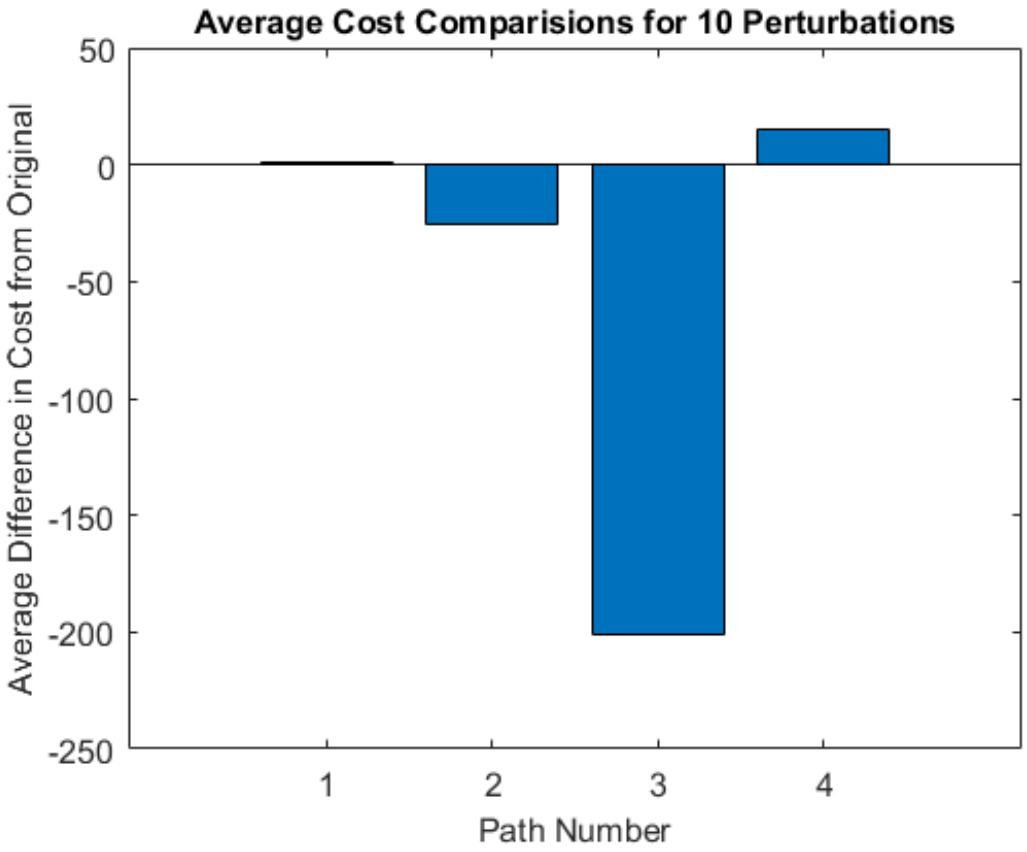


Figure 3.6: Average path similarity, grouped by path.

3.5.2 Average Difference in Distance

This variable is important because it tells us whether the path planner was able to generate a shorter path or not. However, if a shorter path was generated, it does not necessarily mean that path is better than the original. Path length is not indicative of how long it will take to traverse a path (e.g. a 50m path up a mountain will likely take longer to travel than a 100m path on flat ground.) Cost will need to be considered to compare fully.

Figure 3.5 displays the average difference in distance each perturbed path has compared to its original path. There does not look to be any common trend in this data other than when the model does not fail, there is not much variation. The second path seems to be shorter on average and the third path longer. The first and last paths do not seem to be changed.

3.5.3 Average Difference in Cost

This variable is the most important because it tells us whether the path is truly a better path than the original. Cost considers various elements like soil-trafficability, slope, etc. for each point and returns a cost variable for "goodness" for the path. Essentially, a point that would present less problems for a vehicle will have a smaller cost associated with it. While a path may be shorter, if the cost function is higher, we know that that path may have incorporated additional difficulty. This means the vehicle was able to traverse the path but had to go through areas of high slope or low trafficability, likely forcing the vehicle to slow down.

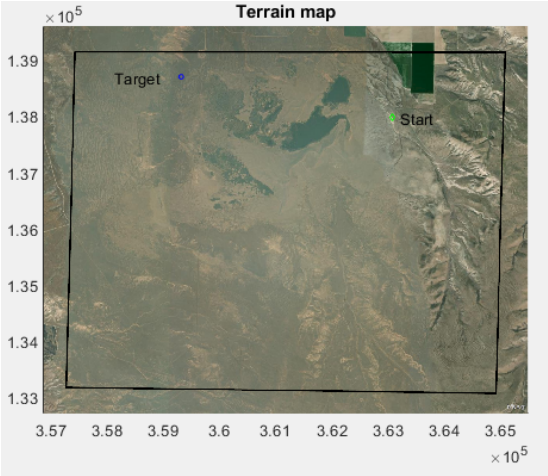
Figure 3.6 displays the average difference in cost each perturbed path has compared to its original path. There does not seem to be a common trend, but the cost for the third path seems to be significantly less on average. This can mean that the first two paths, are finding similar paths to their originals, cost-wise, but also opening up more space for the third path to develop a more cost effective path. Further research will need to be conducted to understand exactly why this activity occurs.

3.6 Discussion

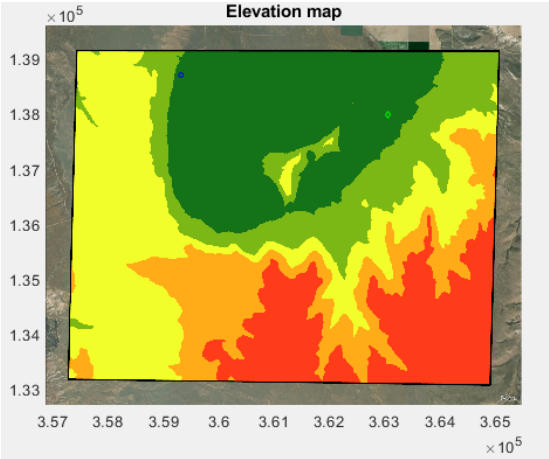
Observing model results, we conclude that testing is necessary. The difference in path distances do not seem significant enough to describe anything. The cost differences also do not make much sense. Additional testing to include additional variables to perturb will likely result in better and more understandable output data.

3.7 Conclusion

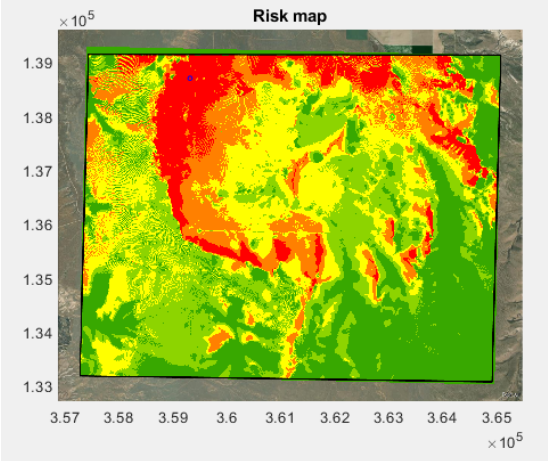
This chapter highlights the importance of accurate terrain map data for path planning algorithms. Our path planning model displays the effects of errored terrain data on the model's success, along with, cost, length, and path similarity. With outdated or incorrect information, path planners are likely to lead vehicles into dangerous situations or will cause unideal path circumstances.



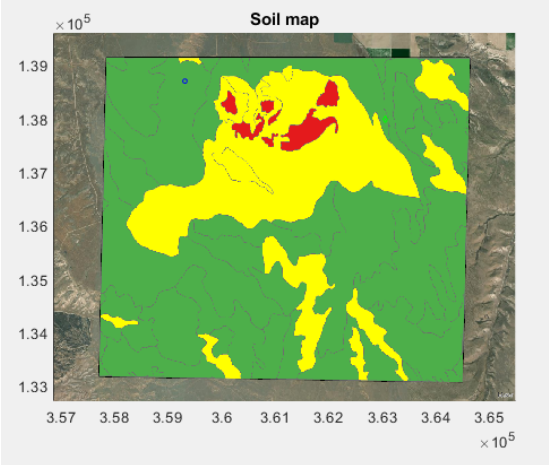
(a) Terrain map.



(b) Elevation layer overlaid over terrain map.



(c) Risk layer overlaid over terrain map.



(d) Soil layer overlaid over terrain map.

Figure 3.7: Terrain map and variable layers.

3.8 Future Work

In the future, the algorithm will include soil trafficability and risk in its criteria. There will be a number of additional variables present in our map data. By measuring their values at each grid point, we can develop a number of layers to our map.

Figure 3.7 displays a number of map layers generated and georeferenced using the Geographic Information System(GIS). For the elevation layer, red corresponds to the highest elevations (e.g. around 1000m) and green corresponds to the lowest elevations (e.g. around 750m). The colors are organized in order red, orange, yellow, light green, dark green. The risk layer depicts colors the same colors as the elevation map, with red being the areas with the most risk and green with the

Soil trafficability rating	Probability of traversing the area
Excellent	$90\% \leq P \leq 100\%$
Good	$75\% \leq P \leq 90\%$
Fair	$50\% \leq P \leq 75\%$
Poor	$0\% \leq P \leq 50\%$

Table 3.1: Soil trafficability rating and corresponding probability of traversing the area.

least. The soil layer depicts the soil trafficability within the terrain map. Green represents a high soil trafficability soil rating, yellow represents decent conditions, and red represents a low trafficability rating. Table 3.1 displays the range of values for soil trafficability rating. The information was obtained from the web soil database on the USDA National Resources Conservation Service [4]. Risk will not be considered as perturbation variable as it relates to a nodes' line-of-sight exposure to other nodes. This feature mainly describes the risk an agent will incur when going through highly visible locations with possible combatant threats in the area.

We will perturb soil trafficability, elevation, and the presence of obstacles. Perturbing soil trafficability will reflect a change in the soil trafficability rating to reflect either excellent or poor soil trafficability. Elevation will be perturbed by a value of plus or minus 50m, effecting neighboring slope calculations. The presence of obstacles is a binary value that will be changed to the opposite of the current value.

Chapter 4

Conclusion

This paper covers two significant intelligent transportation systems. By entering perturbations and analyzing the results, we can pinpoint their sensitive areas. Using this information, follow on research can improve the robustness and reliability of these systems.

In 1D platooning, we highlight the importance of robust ITS platooning systems that have sufficient situational awareness and fault tolerance. We display a 1D platooning model that incorporates several variables. Through our perturbation of the model, we are able to determine that throttle input is the most sensitive model variable to perturbations. We also discover that our method of perturbing the model results in failures nearly 34% of the time. This information can be utilized in future research to further improve robustness of platooning systems.

In path planning, we highlight the importance of accurate terrain map data for path planning algorithms. Our path planning model displays the effects of stale/inaccurate terrain data on a path planner's success, along with, cost, length, and path similarity. With outdated or incorrect information, path planners are likely to lead vehicles into dangerous situations or will cause unideal path circumstances. With our method of fault injection, we can see that the path planner will fail approximately 25% to 37% of the time, depending on the path number. With follow on research, we should be able to determine which variable is most sensitive to path planning (e.g. soil trafficability, obstacles, elevation, slope, etc.)

By exploring these two systems, we are able to further extend our knowledge on model sensitivity for intelligent transportation systems while providing a stepping stone for follow on safety/danger mitigation techniques.

Bibliography

[1] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, 2004.

[2] Ji Hoon Bai and Young-Jae Oh. Global path planning of lunar rover under static and dynamic constraints. *International Journal of Aeronautical and Space Sciences*, pages 1–9, 2020.

[3] A Balasubramanian. Digital elevation model (dem) in gis. *University of Mysore*, 2017.

[4] Colin PD Birch, Sander P Oom, and Jonathan A Beecham. Rectangular and hexagonal grids used for observation, experiment and simulation in ecology. *Ecological modelling*, 206(3-4):347–359, 2007.

[5] Michael Brand, Michael Masuda, Nicole Wehner, and Xiao-Hua Yu. Ant colony optimization algorithm for robot path planning. In *2010 international conference on computer design and applications*, volume 3, pages V3–436. IEEE, 2010.

[6] Norlida Buniyamin, W Wan Ngah, Nohaidda Sariff, Zainuddin Mohamad, et al. A simple local path planning algorithm for autonomous mobile robots. *International journal of systems applications, Engineering & development*, 5(2):151–159, 2011.

[7] Jon Calhoun, Luke Olson, and Marc Snir. Flipit: An llvm based fault injector for hpc. In *European Conference on Parallel Processing*, pages 547–558. Springer, 2014.

[8] Qichen Deng. A general simulation framework for modeling and analysis of heavy-duty vehicle platooning. *IEEE Transactions on Intelligent Transportation Systems*, 17(11):3252–3262, 2016.

[9] Edsger W Dijkstra. A note on two problems in connexion with graphs. In *Edsger Wybe Dijkstra: His Life, Work, and Legacy*, pages 287–290. 2022.

[10] George Dimitrakopoulos and Panagiotis Demestichas. Intelligent transportation systems. *IEEE Vehicular Technology Magazine*, 5(1):77–84, 2010.

[11] Bruce Donald, Patrick Xavier, John Canny, and John Reif. Kinodynamic motion planning. *Journal of the ACM (JACM)*, 40(5):1048–1066, 1993.

[12] Shang Erke, Dai Bin, Nie Yiming, Zhu Qi, Xiao Liang, and Zhao Dawei. An improved a-star based path planning algorithm for autonomous land vehicles. *International Journal of Advanced Robotic Systems*, 17(5):1729881420962263, 2020.

[13] Amr Farag, Ahmed Hussein, Omar M. Shehata, Fernando García, Hadj Hamma Tadjine, and Elmar Matthes. Dynamics platooning model and protocols for self-driving vehicles. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 1974–1980, 2019.

[14] David Fiala, Frank Mueller, Christian Engelmann, Rolf Riesen, Kurt Ferreira, and Ron Brightwell. Detection and correction of silent data corruption for large-scale high-performance computing. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 78:1–78:12, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.

[15] Peter F Fisher and Nicholas J Tate. Causes and consequences of error in digital elevation models. *Progress in physical Geography*, 30(4):467–489, 2006.

[16] Yosra Fraiji, Lamia Ben Azzouz, Wassim Trojet, and Leila Azouz Saidane. Cyber security issues of internet of electric vehicles. In *2018 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6, 2018.

[17] Juan Guerrero-Ibáñez, Sherali Zeadally, and Juan Contreras-Castillo. Sensor technologies for intelligent transportation systems. *Sensors*, 18(4):1212, 2018.

[18] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[19] Felix Hebeler and Ross S Purves. The influence of elevation uncertainty on derivation of topographic indices. *Geomorphology*, 111(1-2):4–16, 2009.

[20] Joachim Höhle and Marketa Potuckova. *Assessment of the quality of digital terrain models.* European Spatial Data Research, 2011.

[21] Cavender Holt and Jon C. Calhoun. Stale data analysis in intelligent transportation platooning models. In *2022 IEEE 13th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, pages 0313–0320, 2022.

[22] Zhonghua Hong, Pengfei Sun, Xiaohua Tong, Haiyan Pan, Ruyan Zhou, Yun Zhang, Yanling Han, Jing Wang, Shuhu Yang, and Lijun Xu. Improved a-star algorithm for long-distance off-road path planning using terrain data map. *ISPRS International Journal of Geo-Information*, 10(11), 2021.

[23] Peng Hu, Xiaohang Liu, and Hai Hu. Accuracy assessment of digital elevation models based on approximation theory. *Photogrammetric Engineering & Remote Sensing*, 75(1):49–56, 2009.

[24] Jingwen Huang. Combining entropy weight and topsis method for information system selection. In *2008 ieee conference on cybernetics and intelligent systems*, pages 1281–1284. IEEE, 2008.

[25] Susan K Jenson and Julia O Domingue. Extracting topographic structure from digital elevation data for geographic information system analysis. *Photogrammetric engineering and remote sensing*, 54(11):1593–1600, 1988.

[26] Li Jin, Mladen Čičić, Saurabh Amin, and Karl H. Johansson. Modeling the impact of vehicle platooning on highway congestion: A fluid queuing approach. In *Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (Part of CPS Week)*, HSCC '18, page 237–246, New York, NY, USA, 2018. Association for Computing Machinery.

[27] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580, 1996.

[28] Sachet Khatiwada, Pamela Murray-Tuite, and Mattias Schmid. Multi-criteria multi-agent path planning in unstructured off-road environments. 2023.

[29] Junsoo Kim, Kichun Jo, Wonteak Lim, and Myoungho Sunwoo. A probabilistic optimization approach for motion planning of autonomous vehicles. *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, 232(5):632–650, 2018.

[30] Sai Varsha Konakalla. A star algorithm. *Document, Indiana State University, Bloomington*, 2014.

[31] Mehmet Korkmaz and Akif Durdu. Comparison of optimal path planning algorithms. In *2018 14th International Conference on Advanced Trends in Radioelecrtronics, Telecommunications and Computer Engineering (TCSET)*, pages 255–258. IEEE, 2018.

[32] Marina Krotofil, Alvaro Cárdenas, Jason Larsen, and Dieter Gollmann. Vulnerabilities of cyber-physical systems to stale data—determining the optimal time to launch attacks. *International Journal of Critical Infrastructure Protection*, 7(4):213–232, 2014.

[33] Steven M LaValle et al. Rapidly-exploring random trees: A new tool for path planning. 1998.

[34] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert. Statistical fault injection: Quantified error and confidence. In *2009 Design, Automation & Test in Europe Conference & Exhibition*, pages 502–506, 2009.

[35] Dong Li, Jeffrey S. Vetter, and Weikuan Yu. Classifying soft error vulnerabilities in extreme-scale scientific applications using a binary instrumentation tool. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 57:1–57:11, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.

[36] Yangxin Lin, Ping Wang, and Meng Ma. Intelligent transportation system(its): Concept, challenge and opportunity. In *2017 ieee 3rd international conference on big data security on cloud (bigdatasecurity), ieee international conference on high performance and smart computing (hpsc), and ieee international conference on intelligent data and security (ids)*, pages 167–172, 2017.

[37] Qinghe Liu, Lijun Zhao, Zhibin Tan, and Wen Chen. Global path planning for autonomous vehicles in off-road environment via an a-star algorithm. *International Journal of Vehicle Autonomous Systems*, 13(4):330–339, 2017.

[38] T. C. May and Murray H. Woods. Alpha-particle-induced soft errors in dynamic memories. *Electron Devices, IEEE Transactions on*, 26(1):2–9, January 1979.

[39] Vicente Milanés, Steven E. Shladover, John Spring, Christopher Nowakowski, Hiroshi Kawazoe, and Masahide Nakamura. Cooperative adaptive cruise control in real traffic situations. *IEEE Transactions on Intelligent Transportation Systems*, 15(1):296–305, 2014.

[40] Laurent Polidori and Mhamad El Hage. Digital elevation model quality assessment methods: A critical review. *Remote Sensing*, 12(21), 2020.

[41] Yesy Diah Rosita, Erly Ekayanti Rosyida, and Muhammad Adik Rudiyanto. Implementation of dijkstra algorithm and multi-criteria decision-making for optimal route distribution. *Procedia Computer Science*, 161:378–385, 2019.

[42] Imre J Rudas and János Fodor. Intelligent systems. *International Journal of Computers, Communications & Control*, 3(3):132–138, 2008.

[43] Raja Sengupta, Shahram Rezaei, Steven E. Shladover, Delphine Cody, Susan Dickey, and Hariharan Krishnan. Cooperative collision warning systems: Concept definition and experimental implementation. *Journal of Intelligent Transportation Systems*, 11(3):143–155, 2007.

[44] Susan Shaheen and Rachel Finson. Intelligent transportation systems. 2013.

[45] Omar Souissi, Rabie Benatitallah, David Duvivier, AbedlHakim Artiba, Nicolas Belanger, and Pierre Feyzeau. Path planning: A 2013 survey. In *Proceedings of 2013 International Conference on Industrial Engineering and Systems Management (IESM)*, pages 1–8, 2013.

[46] Bart van Arem, Cornelie J. G. van Driel, and Ruben Visser. The impact of cooperative adaptive cruise control on traffic-flow characteristics. *IEEE Transactions on Intelligent Transportation Systems*, 7(4):429–436, 2006.

[47] Yu Xuan and Mohammad Naghnaeian. Detection and identification of cps attacks with application in vehicle platooning: a generalized luenberger approach. In *2021 American Control Conference (ACC)*, pages 4013–4020, 2021.

[48] Maryam Yarmohamadi, H Haj Seyyed Javadi, and Hossein Erfani. Improvement of robot path planning using particle swarm optimization in dynamic environments with mobile obstacles and target. *Advanced Studies in Biology*, 3(1):43–53, 2011.

[49] Zuobin Ying, Maode Ma, Zijun Zhao, Ximeng Liu, and Jianfeng Ma. A reputation-based leader election scheme for opportunistic autonomous vehicle platoon. *IEEE Transactions on Vehicular Technology*, 71(4):3519–3532, 2022.

[50] Jingjin Yu and Steven M LaValle. Multi-agent path planning and network flow. In *Algorithmic Foundations of Robotics X: Proceedings of the Tenth Workshop on the Algorithmic Foundations of Robotics*, pages 157–173. Springer, 2013.