THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF MECHANICAL AND NUCLEAR ENGINEERING

A CENTRALIZED COOPERATIVE DRIVING ALGORITHM FOR NON-SIGNALIZED INTERSECTIONS

TING XU
SPRING 2018

A thesis
submitted in partial fulfillment
of the requirements
for baccalaureate degrees in Mechanical Engineering and Mathematics
with honors in Mechanical Engineering

Reviewed and approved* by the following:

Sean N. Brennan
Professor of Mechanical Engineering
Thesis Supervisor

Daniel Cortes
Assistant Professor of Mechanical Engineering
Honors Adviser

* Signatures are on file in the Schreyer Honors College.
ABSTRACT

Connected and Autonomous Vehicles (CAVs) provide the opportunity for signal-free intersection navigation. This thesis introduces and demonstrates a centralized cooperative driving algorithm that considers two vehicles approaching a non-signalized multi-way intersection where the safe traversal can be negotiated. It is assumed that the incoming and outgoing directions are known, and individual vehicle velocities are controllable within a specified range of acceleration and for a specified range from the intersection.

The proposed algorithm is developed by first considering the time-space interval of possible intersections between the vehicles. This leads to the development of a set of collision patterns that predict intersection situations that do not need to be negotiated. It is shown that these patterns extend readily from two-way intersections to eight-way intersections. In cases where path conflicts are detected within the intersection, the algorithm seeks to minimize the complexity of multi-vehicle coordination by preventing any speed deviation of the first vehicle passing through the intersection. The proposed solution in the algorithm is to redesign velocity profiles of the second vehicle arriving at the intersection, thereby avoiding any interference in the planned trajectory of the first vehicle.

The algorithm is agnostic to the number of directions in/out of the intersection, and is readily generalized for ranges in acceleration limits and interaction ranges between vehicles. Based on the different cases where two vehicles’ original trajectories can cause potential collisions, simulation results show the effectiveness of the algorithm under different approaches, such as allowable velocity ranges, accelerations, and minimum algorithm starting distances.
# TABLE OF CONTENTS

TABLE OF CONTENTS.......................................................................................................................... ii

LIST OF FIGURES............................................................................................................................... iv

LIST OF TABLES....................................................................................................................................... vii

ACKNOWLEDGEMENTS........................................................................................................................... viii

Chapter 1 Introduction ........................................................................................................................... 1

1.1 Motivation ...................................................................................................................................... 1
1.2 Problem Statement ......................................................................................................................... 4
1.3 Organization of the thesis ............................................................................................................. 5

Chapter 2 Literature Review ................................................................................................................ 6

2.1 Vehicle-to-Vehicle and Vehicle-to-Infrastructure Communication Methods 6
2.2 Collision Avoidance Systems ........................................................................................................ 10
2.3 Time-Space Diagrams .................................................................................................................. 13
2.4 Cooperative Driving Algorithms at Non-Signalized Intersections .................................. 16

Chapter 3 The Collision Patterns .................................................................................................... 24

3.1 Intersection notation and assumptions ....................................................................................... 25
3.2 Understanding the Collision Patterns ......................................................................................... 26
3.3 Examples for Implementing Collision Patterns in a Four-Way intersection.......................... 27
3.4 Examples for Implementing Collision Patterns in a N-Way intersection.............................. 29
   3.4.1 Examples for Implementing Collision Patterns in a Three-Way intersection .............. 30
   3.4.2 Examples for Implementing Collision Patterns in a Five-Way intersection.............. 31
   3.4.3 Examples for Implementing Collision Patterns in an Eight-Way intersection ......... 33

Chapter 4 Cooperative Driving Algorithm for Two Vehicles at a Non-Signalized Intersection .... 35

4.1 Defining the Input Parameters ....................................................................................................... 36
4.2 Performing a Time Calculation ..................................................................................................... 38
4.3 Performing a Time Comparison .................................................................................................... 40
4.4 Evaluating Possible Collision Patterns ....................................................................................... 41
4.5 Taking Anti-Collision Actions if Necessary ............................................................................... 42
Chapter 5 Complete Examples of Using the Cooperative Driving Algorithm at a Four-Way Non-Signalized Intersection

5.1 An Example of Two Vehicles with Collision Patterns Passing Through the Intersection Without Changing Trajectories ........................................48
5.2 An Example of Two Vehicles Sharing Intersection Usage at the Same Time 50
5.3 An Example of Two Vehicles That Have a Potential Collision at the Intersection ............................................................................................. 53

Chapter 6 The Limitations of the Proposed Algorithm ...........................................57

6.1 Errors in the Outputs of the Proposed Algorithm Step Five ...........................57
6.2 Developing Dimensionless Parameters and Equations .................................59
6.3 Finding the Minimum Algorithm Starting Distance ......................................64
6.4 The Simulation Results ...................................................................................65

Chapter 7 Conclusions .................................................................................................69

Appendix A MATLAB Codes for Collision Patterns ................................................. 71

Appendix B MATLAB Codes for the Two-Vehicle Cooperative Driving Algorithm at A Four-Way Non-Signalized Intersection ................................................. 73

Appendix C MATLAB Codes for Generating 2-D Time-Space Diagrams .............. 78

Appendix D MATLAB Codes for Generating 3-D Time-Space Diagrams .............. 91

Appendix E MATLAB Codes for Dimensional Analysis .......................................107

E.1 Solving for S Matrix by Using Dimension Set Matrix Operation [52] ...........107
E.2 Dimensional Analysis Codes for the Case Where Two Vehicles Go Straight 107
E.3 Dimensional Analysis Codes for the Case Where Two Vehicles Have Conflict Turns ...........................................................................................................110
E.4 Dimensional Analysis Codes for the Case Where One Vehicles Have a Conflict Turn ...........................................................................................................114

BIBLIOGRAPHY ........................................................................................................ 119

ACADEMIC VITA ...................................................................................................... 125
LIST OF FIGURES

Figure 1-1. Illustration of the self-driving Tesla crashed with a tractor-trailer on a highway intersection.................................................................3

Figure 2-1. Travelling path for two vehicles with corresponding parameters..........11

Figure 2-2. A 3-D time space diagram of an intersection with vehicle 1 makes a large turn and vehicle 2 goes straight at the intersection. .............................................14

Figure 2-3. A 2-D time-space diagram with vehicle 1 goes straight and vehicle 2 makes a small turn at the intersection.................................................................15

Figure 2-4. Collision regions in distrusted reservation based cooperative driving systems [45]..............................................................................................................19

Figure 2-5. 2-D time-space diagram of vehicle A and B approaching an intersection [39].22

Figure 3-1. An illustration of a four-way, two-lane intersection setup with examples and notation for possible collision patterns......................................................25

Figure 3-2. Examples of direction pairs for vehicle 1 and 2 travelling from direction 2 and 3 respectively in a four-way intersection.................................................27

Figure 3-3. An illustration of a possible eight-way intersection..............................30

Figure 3-4. Examples of direction pairs vehicle 1 and 2 travelling from direction 1 and 2 respectively in a three-way intersection. .........................................................30

Figure 3-5. Examples of direction pairs vehicle 1 and 2 travelling from direction 2 and 4 respectively in a five-way intersection. ..........................................................32

Figure 3-6. Examples of direction pairs vehicle 1 and 2 travelling from direction 0 and 5 respectively in an eight-way intersection. .....................................................33

Figure 4-1. An illustration of all initial parameters at an intersection......................37

Figure 4-2. A 2-D time space diagram for two vehicles travelling straight towards an intersection with important time periods labeled........................................39

Figure 4-3. 2-D time space diagrams with the left diagram showing no time overlap area within the intersection and the right diagram showing the opposite. ..............41

Figure 4-4. Two possible velocity profiles for the vehicle 2. The left-side profile is designed for straight maneuver only. The right-side profile is designed for turning maneuvers.........................................................43
Figure 4-5. Possible redesigned velocity profile for vehicle 2 that has straight maneuver.

Figure 4-6. Possible redesigned velocity profile for vehicle 2 that has turning maneuver.

Figure 4-7. An example of adjusted trajectories in 2-D time-space diagram.

Figure 5-1. Vehicles' initial parameters and intersection setup in example 1.

Figure 5-2. A 2-D time space diagram demonstrates the step two of the algorithm.

Figure 5-3. A 2-D time space diagram demonstrates the final trajectories of two vehicles that do not need to take any actions.

Figure 5-4. Vehicles' initial parameters and intersection setup in example 2.

Figure 5-5. A 2-D time space diagram demonstrates the step two of the algorithm.

Figure 5-6. A 2-D time space diagram demonstrates the step two of the algorithm.

Figure 5-7. Vehicles' initial parameters and intersection setup used in example 3.

Figure 5-8. A 2-D time space diagram demonstrates the step two of the algorithm.

Figure 5-9. A 2-D time-space diagram demonstrates the final adjusted trajectories for two vehicles.

Figure 5-10. A 2-D time space diagram demonstrates the both the original and final adjusted trajectories of the two vehicles.

Figure 6-1. Three analysis cases for dimensional analysis include when both vehicle go straight (left), one vehicle has a conflict turn (middle), and both vehicles have conflict turns (right).

Figure 6-2. Dimension set matrix operation for finding the exponential values of repeating parameters that are used to represent \( \pi \) parameters [52].

Figure 6-3. Simplified analysis algorithm for finding the minimum algorithm starting distance.

Figure 6-4. Dimensional analysis results indicate the same dimensionless minimum algorithm starting distance for different acceleration values when both vehicles go straight.

Figure 6-5. Dimensional analysis results indicate the same dimensionless minimum algorithm starting distance for different acceleration values when both vehicles have conflict turns.
Figure 6-6. Dimensional analysis results indicate the same dimensionless minimum algorithm starting distance for different acceleration values when one vehicle has conflict turn.
LIST OF TABLES

Table 1-1. Categorized Traffic accidents in 2015, where the data is obtained from National Highway Traffic Safety Administration [1].................................2

Table 4-1. Vehicle Initial Parameters .................................................................37

Table 4-2. Collision Patterns ..............................................................................41

Table 6-1. Vehicle Initial Parameters .................................................................60

Table 6-2. Dimensions for initial parameters. .....................................................61

Table 6-3. Dimensional analysis matrix used in calculation. ...............................62
I have never thought I would be the author of a document that has more than a hundred pages by the time I graduate, when I began my undergraduate career at Penn State in Fall 2014. Now, sitting in front of my laptop and facing this thesis, I know that I would not be able to finish this work without help from so many people around me.

First of all, I would like to thank my thesis supervisor, Professor Sean Brennan, for providing guidance in my research project and helping build a strong foundation for my future in graduate studies. You have not only inspired me on how to become a real engineer and researcher, but also taught me numerous ways to improve communication and presentation skills.

I would like to thank the class of ME497H and all three instructors: Dr. Jacqueline O’Connor, Dr. Daniel Cortes, and Dr. Stephanie Stockar, for offering helpful suggestions in writing the proposal and literature review of this thesis. It was a great start that initiated my confident and happiness for continuing finishing this thesis.

I would also like to thank all my dear friends who allowed me to borrow their laptops for running MATLAB simulations. Though Zhesheng Shen’s windows crashed several times while running the codes, I still appreciate your generosity.

Finally, I would like to thank my honors advisor Dr. Daniel Cortes for taking the time to review this thesis and providing academic advices over the last two years.
Chapter 1

Introduction

1.1 Motivation

Traffic within intersections can be so complicated for human drivers that traffic control devices or methods are usually used, which in turn causes costs associated with infrastructure, driving time, and vehicle efficiency/emissions [1]. The challenge of intersections is to enable vehicles from all incoming directions to perform navigation behaviors individually such as going straight, and turning, yet with shared usage of the same physical space. Because of this complex traffic situation, intersections are a common location of human error that can lead to vehicle collisions.

According to crash data from National Highway Traffic Safety Administration, as shown in Table 1-1, a total of 11,275,000 traffic accidents happened in the United States in 2015 [2]. Within that data, about 47.2 % of total traffic accidents happened at intersections, regardless of traffic control device installations such as stop signs or traffic lights. During the same year, data in Table 1-1 also shows that about 67.5 % of accidents were at junctions where traffic control devices existed. This data set not only implies that intersection-related accidents are a significant portion of traffic accidents, but also suggests that traffic control devices are not highly effective at eliminating all accidents from happening. Consequently, there is the need to improve the management of traffic at intersections.
Table 1-1. Categorized Traffic accidents in 2015, where the data is obtained from National Highway Traffic Safety Administration [1].

<table>
<thead>
<tr>
<th>Relation to Junction</th>
<th>Traffic Control Device</th>
<th>Traffic Signal</th>
<th>Stop Sign</th>
<th>Other/unknown</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-junction</td>
<td>None</td>
<td>4,461,000</td>
<td>29,000</td>
<td>4,000</td>
<td>213,000</td>
</tr>
<tr>
<td>Junction</td>
<td>Traffic Signal</td>
<td>1,423,000</td>
<td>2,861,000</td>
<td>736,000</td>
<td>305,000</td>
</tr>
<tr>
<td>Other/Unknown</td>
<td>Stop Sign</td>
<td>1,068,000</td>
<td>47,000</td>
<td>48,000</td>
<td>79,000</td>
</tr>
<tr>
<td>Total</td>
<td>Other/unknown</td>
<td>6,952,000</td>
<td>2,937,000</td>
<td>788,000</td>
<td>597,000</td>
</tr>
</tbody>
</table>

Though traffic accidents keep happening over time and there is no perfect solution of avoiding crashes on the roads, the field of transportation is slowly adopting a new technological solution: intelligent transportation systems. Autonomous driving systems, such as the Tesla autopilot functionality, enable driver-assist systems on modern vehicles. These rely heavily on cameras, ultrasonic sensors and radar. However, these sensors cannot see through other vehicles, buildings or similar occlusions, and each sensor can experience faults or errors in processing. Further, the placement of sensors on a vehicle that may be advantageous on a straight highway situation could work poorly for detecting cross collisions in an intersection.

For example: on May 7th 2016, a tragedy happened at a highway intersection in Williston, Florida. Joshua Brown, the driver who was using the autopilot function on his Tesla Model S, was killed when his Tesla went under a tractor-trailer of a truck coming from his left side, as shown in Figure 1-1 [3]. While the truck driver should have looked carefully before turning, the sensors on the Tesla failed at detecting the tractor-trailer right in the front of the vehicle. This serious accident
is a warning of the reliance on purely vehicle-based automated driving systems and it reveals the dangers and challenges of possible future automated intersections.

![Figure 1-1. Illustration of the self-driving Tesla crashed with a tractor-trailer on a highway intersection.](image)

Connected vehicles and cooperative driving provide the opportunity for signal-free intersection navigation while reducing collision possibilities at the same time. New technologies such as Dedicated Short-Range Communication (DSRC) [4] radio systems in vehicles allow vehicle-to-vehicle (V2V) [5] or vehicle-to-infrastructure (V2I) [5] communication to coordinate motions around intersections. Historically, accidents at intersections occurred mostly because careless driving; drivers do not know there is another vehicle approaching or they do not have enough time to react when they see a vehicle is about to crash with their own vehicle. Benefits can be obtained through V2V- or V2I-based cooperative driving, since a driver or the computer system installed in the vehicle can receive information from others who are approaching the same intersection at the same time before they could physically see each other. However, the use of
these technologies and information to improve traffic requires algorithms for coordinating vehicle motion within an intersection.

1.2 Problem Statement

Assuming that vehicles are equipped with wireless communication systems and that a central intersection agent is built to exchange information with vehicles, this thesis presents the development of a centralized cooperative driving algorithm that allows two vehicles, coming from different directions, to safely navigate an intersection even in situations of possible conflicts. The control agent is assumed to function as a central information system which receives and sends data between vehicles around a local intersection. After reorganizing and arranging the information, the central agent performs the proposed algorithm steps and sends appropriate instructions back to individual vehicles. In addition, simulations are conducted to explore the performance of the cooperative driving algorithm in terms of its dependent input parameters.

A novelty of the approach is the rapid down-selection of maneuvers to consider only those causing collision patterns. And focusing only on these situations, the algorithm then takes necessary actions before vehicles arrive at the intersection to avoid potential collisions. By designing step-by-step procedures to manage this process, the algorithm also takes into consideration the need for comfortable human driving styles and feasible tire-friction limits, while eliminating superfluous driving behaviors at intersections such as stopping while there are no other vehicles at the same intersection. The system could improve traffic safety and efficiency, at the same time contributing to fuel efficiency at intersections. But most importantly, this work seeks to improve the reliability of autonomous driving at non-signalized intersections.
1.3 Organization of the thesis

This thesis presents the development and analysis of a cooperative driving algorithm at four-way non-signalized intersections for two vehicles. The collision patterns used in the algorithm are first presented. The proposed algorithm is then introduced by providing a step-by-step explanation. Furthermore, the performance of the algorithms is evaluated collaboratively by using dimensional analysis.

The remainder of the thesis is organized as follows: Chapter 2 is a literature review of V2V and V2I communication methods, collision avoidance systems, time-space diagrams, and cooperative driving systems. Chapter 3 focuses on the developed collision patterns that will be used in the cooperative driving algorithms that follow. Chapter 4 contains detailed description of the cooperative driving algorithm for two vehicles. Chapter 5 illustrates some complete examples of using the algorithm at a four-way non-signalized intersection. Chapter 6 provides analysis on the limitations of the developed algorithm under certain initial parameters, while examining specifically the infeasible cases and how these situations may arise, or not, in real-world driving. Chapter 8 is a summary of the thesis and the future work that can be performed.
Chapter 2

Literature Review

This chapter introduces three topics of research in the field of intelligent transportation systems including: 1) vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) wireless communication methods, 2) collision avoidance systems, and 3) cooperative driving systems for non-signalized intersections. Specifically, section 2.1 of this chapter describes V2V and V2I wireless communication methods based on radios. Section 2.2 gives an overview of the theories behind collision avoidance systems. Section 2.3 introduces the time-space diagrams that are commonly used in demonstrating intersection control problems. Section 2.4 presents a summary of designed cooperative driving systems for non-signalized intersections that combine the technologies described in sections 2.1 to 2.3.

2.1 Vehicle-to-Vehicle and Vehicle-to-Infrastructure Communication Methods

Before discussing the research related to V2V and V2I communication methods, it is useful to introduce the general concepts of these technologies. The idea of V2V and V2I is brought by Automatic Highway System (AHS) [5] research in 1990s [6]. V2V and V2I use wireless communication devices to exchange data and signals back and forth rapidly between moving vehicles and/or infrastructure. V2V and V2I communications have very low latency, and have the
ability to connect and disconnect seamlessly between devices in order to achieve efficient and safe driving conditions. The communication systems are increasingly cheap, and federal rule-making is in consideration to require these devices on future vehicles, which makes V2x systems feasible to use widely [6]. There are two competing technologies for V2V and V2I implementation: first, Dedicated Short Range Communications (DSRC) [4], and second, IEEE 802.11[7]. Both are explained in detail below.

In 1999, the U.S. Federal Communications Commission (FCC) first developed DSRC to be used in the intelligent transportation systems, such as V2V and V2I [4]. In 2003, ASTM and IEEE then adopted the DSRC standard [8]. Using radio channels within a 5.9 GHZ licensed band at a spectrum of 75 MHz, the goal of DSRC is to allow data transfer within a range of 1000 meters [4]. DSRC can help transmit messages between vehicles within milliseconds without delay [9]. DSRC also has high data transfer rate, typically ranges from 3 to 27 Mbps [10]. User Datagram Protocol (UDP), one of the network protocols used for the Internet, is the communication protocol of DSRC [11]. The radio channels are designed to achieve fast wireless communication between vehicles and infrastructure, as well as from a vehicle to another vehicle. DSRC is designed so that vehicles is drive at typical highway speeds and still be able to send or receive information successfully [12]. With increasing driver assistance systems being installed on vehicles, including for example collision warning systems and adaptive cruise control systems, the seven channels of the DSRC spectrum can be separated for communications between each of these different systems [12]. Examples of research that use DSRC radios are quite common. The Department of Transportation conducted The Connected Vehicle Safety Pilot Program, which developed safety applications based on V2V and V2I by using DSRC [6]. DSRC Biswas et al. utilized DSRC in
their Cooperative Collision Avoidance (CCA) system for V2V communication [12]. In developing a platooning system, DSRC was also used as the communication method by Bergenhem et al. [13].

IEEE 802.11 standard is similar to DSRC, and like DSRC the 802.11 standard is also based on the idea of using radios at certain frequencies to help information flow within a local area [7]. However, unlike DSRC, IEEE 802.11 standard also utilizes more frequencies, bandwidths and ranges in different protocols such as a, b, g, n, etc. [7]. Two configuration modes are defined in 802.11 standard for Basic Service Set (BSS): ad-hoc mode and infrastructure mode, which the ad-hoc is defined as the Independent Basic Service Set (IBSS) [14]. In the ad-hoc mode, nodes, representing subjects, communicate with each other in the same channel without an Access Point (AP), which acts like a central station and receives signals from or sends signals to each node in the infrastructure mode [15]. However, the IEEE 802.11 standards experiences delay problems in different areas, such as transmission delay, delay from interference, and congestion delay [16]. Wellnitz et al. also addressed packet loss while using IEEE 802.11 based wireless ad-hoc networks. Packet loss refers to losing parts of data during data transferring process [16]. As background traffic increases, delay also increases with higher packet loss rates. Wellnitz et al. ran experiments to compare the effect of background traffic in a wireless single-hop 802.11 based ad-hoc network [16]. While testing at 50 meters communication distance and 1400-byte packets without background traffic, the average data transferring time for round trip (RTT) was 5.27 milliseconds and packet loss was 2.98%. While testing with background traffic under the same communication distance and packet size, the RTT raised to 26.65 milliseconds and the packet loss rate increased to 17.41% [16]. The literature also includes examples of using 802.11 protocols in intelligent transportation systems [17]–[19]. For example, Wang et al. used specific IEEE 802.11b ad-hoc
mode radios to form wireless communication links between vehicles and roadside units in their collision warning system [17]. In this situation, the IEEE 802.11b ad-hoc mode radio operates at 2.4 GHZ and with different data rates according to signal strength [4]. With the radio, vehicles broadcast their information, such as velocity and direction, to nearby roadside units with an interval of 0.2 seconds periodically [9]. The roadside units then can regularly send back warning signals to vehicles at a time interval of 0.2 seconds [9]. A study has also shown that IEEE 802.11 has an appropriate beaconing mechanism to perform V2I communication [18]. Uchida et al. proposed to use IEEE 802.11 b/g/p in their Delay Tolerance Networks (DTN) for V2V [19].

DSRC and IEEE 802.11 both have their advantages and disadvantages. With low latency, DSRC also has distinguishable channels to exchange different levels of safety massages, designed specifically for V2V and V2I, which improves the efficiency [20]. Since DSRC uses licensed bandwidth, it provides safety and privacy [9]. However, DSRC can only transfer predefined instruction messages such as brake, turn right, and turn left indicators. Range standards for DSRC are also different between countries. The communication range is 30 meters in Japan, while that in Europe is about 15-20 meters, and that in USA is 1000 meters [20]. On the other hand, the IEEE 802.11 standards are unified worldwide. Even though 802.11 can pass over more specific behavior instructions to vehicles, it transfers data far slower than DSRC does [17] and was not purpose-built for high-speed vehicle usage. Packet loss and relatively high latency also makes IEEE 802.11 not ideal for V2X communication. Drivers and passengers inside moving vehicles cannot afford to have much delay or data lost while using wireless communication to receive traffic information.
2.2 Collision Avoidance Systems

One of the main goals of intersection navigation is to ensure traffic safety and prevent accidents from happening. While collision avoidance systems are commonly used to prevent collisions on straight road situations, the same ideas for roadway collision prevention can also be borrowed and implemented in automated intersection navigation. Collision avoidance systems also often use V2V or V2I data as discussed in the previous section to predict future collisions. If a possible collision exists, then the appropriate reactions to avoid it are to steer, decelerate or brake. Most collision avoidance systems focus on the use of deceleration or direct braking actuation, because steering control is much more difficult to plan and negotiate with a driver than speed.

The time-to-collision (TTC) [21] and the time-to-avoidance (TTA) metrics [21] are the main basic algorithms generally used for designing collision avoidance systems. Hayward, in 1972, defined the TTC to be the time that takes for two vehicles to collide while traveling at constant velocities following their original paths [21]. Later in 1993, the TTC was proved to be a reliable method to predict vehicle collisions for collision avoidance systems [22]. By using two vehicles’ travelling parameters, Equations 1.1 and 1.2 calculate the location of a potential collision:

\[ x_+ = \frac{(y_2-y_1)-(x_2\tan\theta_2-x_1\tan\theta_1)}{\tan\theta_1-\tan\theta_2} \]  
\[ y_+ = \frac{(x_2-x_1)-(y_2\cot\theta_2-y_1\cot\theta_1)}{\cot\theta_1-\cot\theta_2} \]  

where \( x_1, y_1, x_2, y_2 \) are the positions for vehicle 1 and vehicle 2, \( v_1 \) and \( v_2 \) are velocities, \( \theta_1 \) and \( \theta_2 \) are direction angles, and \((x_+, y_+)\) is where two vehicles collide as indicates in Figure 2-1[17][23].
With the predicted collision location \((x_+, y_+)\) and the velocities of two vehicles, the expected time for each vehicle to reach the collision location, defined as time-to-intersection (TTX), can thus be calculated by Equations:

\[
TTX_1 = \frac{|\vec{r} + (-\vec{r}_1)|}{|\vec{v}|} \cdot \text{sign}((\vec{r} + (-\vec{r}_1) \cdot \vec{v}_1)
\]

where \(\vec{v}_1\), \(\vec{v}_2\) are the velocities of the vehicles, \(\vec{r}_1\) is the vector representations of coordinate \((x_+, y_+)\), and \text{sign}() returns the sign of the equation inside the periapsis [23]. If \(TTX_1\) and \(TTX_2\) are equal, then the time is considered as TTC [23].

In contrast to the TTC, the TTA metric indicates the total time a vehicle takes to stop or decelerate so it can avoid collisions. The equation of TTA is defined by:

\[
TTA_{ort} = tr + \frac{\beta v}{\mu g}
\]

where \(tr\) is the reaction time of driver, \(v\) is the speed of vehicle, \(\mu\) is the friction coefficient of tires on the roads, \(g\) is the acceleration of gravity, and \(\beta\) is a speed reduction factor within a range of
(0,1). A value of \( \beta \) equal to 1 indicates that the subject vehicle is expected to stop completely for collision avoidance [23].

Combining TTC and TTA metrics, Miller and Huang developed an intersection collision warning system (ICWS) [23]. Based on the computed TTC and TTA, the ICWS will issue a warning to the driver if TTC is close to TTA under the circumstance that driver is still not braking [23]. No warnings will be issued if Equations 1.1 and 1.2 do not return a possible collision location or if the TTC is much greater than the TTA [23].

From the above TTC and TTA metrics, Wang et al. [17] developed the “refined” TTC and TTA metrics. Calculations of the refined TTC are similar with that of TTC introduced before, but include extra consideration of accelerations. Once the roadside units receive vehicles’ information such as velocities, acceleration, and directions, the system calculates the expected collision location by using Equations 1.1 and 1.2. Two time variables \( t_1 \) and \( t_2 \), which have the same idea as \( TTX_1 \) and \( TTX_2 \), are denoted as the time for each vehicle to reach the predicted collision location. Variables \( t_1 \) and \( t_2 \) can be calculated from the equations:

\[
s_1 = v_1 * t_1 + \frac{1}{2} a_1 t_1^2
\]

\[
s_2 = v_2 * t_2 + \frac{1}{2} a_2 t_2^2
\]

where \( s_1 \) is the distance between vehicle 1 and the predicted collision location, \( s_2 \) is the distance between vehicle 2 and the predicted collision location, \( v_1 \) and \( v_2 \) are the current speeds for vehicles 1 and 2, and \( a_1 \) and \( a_2 \) are the current accelerations for vehicles 1 and 2. If \( t_1 \) and \( t_2 \) are equal, then the refined TTC metric (\( TTC_{acc} \)) is the same as \( t_1 \) and \( t_2 \). The refined TTA metric (\( TTA_{acc} \)) is defined by:

\[
TTA_{acc} = \frac{\text{the current moving speed of the faster vehicle}}{\text{the maximum deceleration of the faster vehicle}}
\]
The refined TTA equation illustrates that the time to avoid a collision is inversely proportional to the deceleration limit of the faster vehicle. With the consideration of acceleration/deceleration, the refined equations can estimate the potential collision location and timing more accurately.

This thesis’s cooperative driving algorithm for non-signalized intersections does not directly use TTC, TTA or TTX calculations. However, the core ideas of those time metrics greatly inspired the calculation steps of the developed algorithm. For example, the TTX metric representing the time-to-intersection is used in the very first step of the algorithm; though, in this thesis, the “intersection” refers to a real four-way intersection instead of the point where two vehicles intersect.

2.3 Time-Space Diagrams

Before introducing the literature cooperative driving algorithm to coordinate vehicle behaviors within collision patterns, it is important to first explain time-space diagrams. These are coordinate representations commonly used for explaining intersection control, multi-vehicle path negotiation, and designing trajectories that meet both time and spatial constraints. Time-space plots, as the name implies, includes time and space as separate axes to plot trajectories of vehicles; if two trajectories intersect in a time-space plot, this means that two vehicles occupy the same space at the same time, which denotes a collision. Some scholars use three-dimensional representations, where x and y axis are the position vector of a vehicle in a local intersection area, and the z axis is time [43, 44]. Vehicle trajectories in the intersection area are then presented as lines or tubes that can be twisted or curved in x-y directions while rising in z direction, as shown in Figure 2-2. Though 3-D plots are easier to understand because they show the direct paths at an
intersection in 2-D space when the plots are viewed orthogonal to the time-axis, these plots are generally difficult to read from other views, particularly isometric views, due to the 3D to 2D projection of information.

Figure 2-2. A 3-D time space diagram of an intersection with vehicle 1 makes a large turn and vehicle 2 goes straight at the intersection.

Two-dimensional time-space diagrams are also commonly used [26]. To plot a vehicle’s time-space trajectory in 2D rather than 3D, the vehicles’ 2D space coordinates are plotted as their station distance along their respective paths, a representation called the s-coordinate. Possible collisions therefore occur when the time-space plots using the 1D s-coordinates intersect within the spatial region of an intersection, a region denoted as horizontal bars on the time-space plots that follow. In the 2D case, a vehicle’s trajectories will be curved or straight lines, depending on whether the vehicle’s velocity is changing or constant, respectively. Figure 2-3 is an example of a 2D time-space diagram where the y-axis is the distance between the vehicle and the start of an
intersection (shown in Fig.1) and x-axis is time. As vehicle 1 and 2 begins travelling at time, $t = 0s$, the values of $y$ start decreasing, meaning vehicles are approaching the intersection, then passing through the intersection, and eventually moving away from it. In this case, vehicle 2 is turning, therefore, this vehicle has to slow down to a constant turning speed and then speed up after turning. Such situations of changing velocity result in a curving trajectory in time-space. While the position $y = 0$ indicates where intersection starts, the duration of the intersection occupancy for each vehicle varies for each vehicle, since vehicle 1 goes straight and vehicle 2 turns. This variation will be explained in detail in section 4.2 by using equations for the respective vehicle velocities. The boxes around each trajectory highlights the specific entry-to-exit time periods when each vehicle is within the intersection. If these boxes overlap for two vehicles, a collision is possible.

Figure 2-3. A 2-D time-space diagram with vehicle 1 goes straight and vehicle 2 makes a small turn at the intersection.
2.4 Cooperative Driving Algorithms at Non-Signalized Intersections

While collision avoidance systems are able to effectively prevent collisions between two vehicles, investigators have also been developing V2I or V2V cooperative driving systems with a particular focus on non-signalized intersections to improve traffic efficiency and prevent collisions. While these are more detailed and complicated than the collision avoidance systems, those cooperative driving systems for intersection management, acting like traffic lights or policemen, not only avoid accidents but also help plan trajectories of vehicles. Analyzing the traffic information of all vehicles coming to a same intersection, such as initial velocities, initial and intended directions, the goal of those systems is to design appropriate travelling plans for individual vehicles to pass through an intersection safely. Although there are many intersection cooperative driving systems, they can be classified by specific criteria as noted below.

Generally, most cooperative intersection driving systems are divided into two main categories: distributed control methods [19–27] and centralized control methods [28–36]. As stated in section 2.1, a vehicle’s travel information can be sent via wireless communication protocols, such as DSRC or IEEE 802.11. However, important questions begin to arise: who exactly should the vehicle communicate with to ensure safety? Does one vehicle have broadcast and potentially negotiate with all surrounding vehicles? How often should this communication take place, and with what allowable latency? Should decision-making be shared among vehicles, or allocated to a local central agent who has all local traffic information? The last question – where responsibility lies in trajectory decision making – has most challenged the past research. Thus, the literature is divided between centralized and distributed vehicle control, and in both approaches scholars have investigated different cooperative intersection driving systems.
In the situation where there is no central master algorithm, this is characterized as distributed communication and/or control in the literature. Each vehicle is treated as an “agent” who exchanges their own information frequently with others nearby. Meanwhile, agents have to make their own decisions locally based on their current state and every massage they receive. The advantage of distributed communication is that all agents are able to obtain abundant information of surroundings and can make decisions freely by themselves. On the other hand, agents may in practice have to deal with hundreds of useless messages that do not affect their trajectories. The situation gets more and more complicated as the number of surrounding agents involved grows, and as the geographical range of decision-making is extended further away from the vehicle. Additionally, in cases where there may be failures in the algorithm or even collisions, it may be unclear where the failure-point occurred nor which agent would be responsible; this attribution and analysis of faults becomes especially difficult when distributed control systems are operated with each agent operating their own algorithm for negotiation, thus making certification of performance exceeding complex due to the need to analyze all permutations of distributed control algorithms among all agents.

Different from distributed communication, a centralized communication and/or control strategy assumes that all decisions are mediated by a local master agent. Theoretically, each intersection has its own master agent that manages and is responsible for all traffic information and trajectory decisions within a certain range of that interaction. All individual agents have to report to the master when they approach to that local area. The master analyzes the information related to each agent and sends individual instructions to each agent respectively. This centralized decision-making process is easier to organize since the master has an overall picture of the whole
local area, will only communicate when necessary, and will usually only update trajectories a small
number of times rather than as a constantly negotiated process. However, individual agents being
managed by the master lose freedom in decision-making, and the entire system is very fragile to
operational failures related to not only the master agent, but also if the other agents do not react to
master appropriately.

It is possible to organize past research as well based on planning methods, and there are
two popular organizational methods for cooperative driving systems at intersections: reservation-
based systems [28–30, 32, 33, 37] and trajectory-planning methods [24, 25, 35, 38–42]. The
definition of reservation in reality is the act of withholding a resource. Similarly, in reservation-
based cooperative driving systems, the vehicle’s “reservation” has the same definition of holding
a location in time-space by which only the vehicle can use an intersection. An intersection is thus
considered as a scare resource and is divided into multiple tiles based on its size [36] to enable
reservations. When used with centralized reservation systems, a local reservation agent is
presented to reserve tiles for all agents approaching to that intersection [36]. Each vehicle agent
has to request reservations with the reservation agent by providing its current status, such as
velocity, acceleration, current and intended direction. Therefore, if agent A reserves a particular
reservation unit for a specific time, no other agents are able to schedule that same tile at the same
time. Connecting back with time-space diagram, vehicle collisions can indeed be avoided at
intersection when no two vehicles are within the same tile at the same time. The reservation agent
also has rights to reject reservation requests if they conflict with any previous requests by other
agents. Then the vehicle agent must change their trajectory and then send a new request. Once the
request is approved, meaning no conflict exists with prior reservations, then the vehicle agent has
to confirm it will follow its reservation or cancel to make a new one if the trajectory is changed after the reservation is made [36].

Scholars have also used distributed communication in reservation-based cooperative driving system [19–21]. With distributed communication, the central reservation agent no longer exists. Vehicles then have to negotiate with each other directly. For example, VanMiddlesworth et al. [29] used simple massages like “claim” and “cancel” in the reservation system. Once a vehicle claims a tile at the intersection, no other vehicles can claim and occupy that tile until the “cancel” massage is released by the original vehicle. To avoid complexity, Nauman et al. [27] propose a concept of token reservation instead of tiles reservation. Each token represents actual collision regions, shown in Figure 2-4, which are the critical regions that vehicles may crash based on different trajectories. A vehicle approaching can claim an ownership of the token and broadcast the occupancy with respect to time. Therefore, other vehicles have to listen to tokens’ availabilities in order to avoid crashing.

Figure 2-4. Collision regions in distrusted reservation based cooperative driving systems [45].
Variations are commonly added to the centralized reservation-based cooperative driving systems to increase the reservation flexibility. In addition to a reservation agent, Schepperle et al. [32, 33] introduced the idea of an exchange agent whose role is to coordinate the reserved time slot. In this way, a particular unit space at a particular time can be exchanged between vehicle agents according to their own needs. The exchanging process can even be negotiated with economic incentives, where one agent may pay the other to complete a more advantageous exchange. Besides the time slot exchange, Schepperle et al. [42] introduced a mechanism named the Initial Time-Slot Auction (ITSA). In ITSA, vehicle agents can bid for their own time slots, and whoever has the highest bid wins while being required to make a payment equal to the second-highest bid. Some standard rules must apply to the bidders: vehicles can only bid after all leading vehicles have obtained time slots, and vehicles that have already obtained time slots earlier in time cannot re-enter the auction again [42]. The authors show that in some situations the average waiting time at intersections is shorter using ITSA than using the original centralized reservation method.

Trajectory-planning is also one of the major planning methods in cooperative driving systems at intersections. Instead of focusing on the concept of reservation, trajectory planning methods seek to change vehicle velocities to enforce an appropriate intersection entry order for all incoming vehicles. This method can have both challenges and advantages because vehicles from different directions can sometimes pass within the intersection safely and efficiently, even at the same time. The resulting trajectories can seem to perform like a virtual traffic light or a policeman. A vehicle’s travel parameters such as velocity, incoming and intended directions are still required from vehicles for trajectory planning. Rather than considering vehicles as individuals, some researchers group consecutive travelling vehicles together if multiple vehicles are approaching the
intersection from same directions [25]. Trajectory-planning objectives can be changed dynamically based on current traffic situation and different objectives; goals might include optimizing the time-usage efficiency of the intersection, optimizing an individual vehicle’s time passage (for example, for an emergency-responder), optimizing the fuel-economy of all vehicles passing the intersection, etc. For each situation, an objective function is usually formulated with equations of constraints that contain vehicles’ velocities, accelerations and time periods as variables [25, 39]. By solving the optimization problem, algorithms then can find out which groups of vehicles should pass the intersection first, and then instruct other groups of vehicles to pass through the intersection in order, with the ordering then sometimes requiring trajectory updates to be planned for certain vehicles.

Since the space of intersection is not divided into tiles anymore in trajectory-planning methods, systems can only rely on velocity and directions of the vehicles to determine whether there is a potential collision or not. Lee et al.[26] uses the comparison between times for vehicles to arrive at the beginning of an intersection to check if a potential collision exists. In the 2-D time-space diagrams in Figure 2-2, the distance \( l_w \) is considered the region of intersection. The times \( t_a \) and \( t_b \) are the times for Vehicle A and Vehicle B to arrive at the beginning of the intersection respectively. The box, indicating the period when the vehicle is in an intersection, is drawn corresponding to the entering and exiting intersection time for a vehicle. As long as the trajectories of two vehicles are in a same time-space box at the same time, like part a and b in Figure 2-5, then it means that they will be in the intersection at the same time. Therefore, a collision may happen and subsequent algorithms is to minimize the length of overlapping trajectories, with constraints of acceleration, deceleration, speeds and required headway. However, this situation may not cause
collisions because the intended directions of the vehicles also play an important role in determining whether a collision occurs between two vehicles at an intersection. The influence of the intended direction will be discussed in detail in the next chapter.

Compared with the reservation method, vehicles using a trajectory-planning method have less freedom in adjusting their own speeds and trajectories when they are approaching the intersection. For each vehicle, the acceleration, velocity, and acceleration period has been determined in a trajectory-planning method, and for all other vehicles nearby; these solutions are obtained by solving constrained trajectory equations via an optimization problem or via a rule-set. Therefore, a change in the trajectory of one vehicle can affect the trajectory of many other vehicles, requiring recalculations for all vehicles around that intersection. In the reservation method, a vehicle can adjust its own speed and trajectories, as long as it cancels the earlier requests and makes new reservation requests every time for passing the intersections. Those changes can mostly affect any vehicles following in the same lane, but those following can always choose to change lanes. The reservation method, while seemingly simpler than trajectory-planning, also may allow
vehicles from different directions to travel at the same time within an intersection. Although theoretically, no two vehicles using reservations will be on the same tile or a collision region at the same time, if one vehicle does not follow its original reservation, a disaster may occur in the middle of that intersection, causing serious accidents, without the ability of other vehicles to change their trajectory. Thus, trajectory-planning methods can be more complex and restrictive than reservation-based methods by enabling vehicles to react to each other, they may at the same time allow the entire traffic system to better react collectively to errors in trajectory-following by one vehicle.
Chapter 3
The Collision Patterns

In well-designed intersection algorithms, it is possible and in fact quite common that many vehicles can travel through the intersection simultaneously without collision, often with different goal directions. This fact cannot be ignored in order to improve traffic efficiency. While using trajectory planning method, Li et al. first proposed the idea of “safe patterns” that allow cars with no direction conflicts to pass through an intersection simultaneously with little to no negotiation required between vehicles [33]. For instance, vehicles with safe patterns can have usage directions (pair i in Figure 3-1) which simultaneously share intersection usage, but do not intersect with each other; whereas, in contrast, some usage directions (pair j) are not a safe pattern and could result in a collision.
This chapter extends the “safe pattern” idea into mathematical representations of collision patterns. Based on these safe pattern trajectories, it is possible to develop a mathematical prediction of the patterns that cause direction conflicts. An elegant outcome of this mathematical approach which follows is that it predicts collision patterns for N-way intersections, with N being an arbitrary integer of 3 or more, i.e. all realistic intersections. Thus, while the discussion focuses on 4-way intersections, the results apply to any intersection situation.

The rest of the chapter is organized as follows: first, the notation and assumptions are presented for an intersection. The collision patterns table is then introduced. Next, examples are presented showing how to predict safe or unsafe patterns in three-way, four-way, five-way and eight-way intersections.

### 3.1 Intersection notation and assumptions

This thesis considers an N-way, single-lane in the travel direction, non-signalized intersection where U-turns are not allowed. Although the physical dimensions of an intersection do not influence the safe collision patterns, the size of the intersection plays an important role in the cooperative driving algorithm later as it especially determines how long a vehicle will be within an intersection for a given velocity. AASHTO suggests lengths of 3.6 meters (12 feet) lanes on urban roadways [50]. Therefore, for calculation simplicity and using representative dimensions similar to existing roadways, each lane width (W) is assumed to be 4 meters and that the intersection forms an 8 by 8-meter region.
The directions at the intersection are hereafter defined numerically in modulo 4 for a four-way intersection, numbered from 0 to 3 counterclockwise as shown in Figure 3-1, with the 0 lane as the right road into the intersection. The original directions for vehicles are defined as M and N respectively by ascending order in their incoming direction numbers in modulo 4. In Figure 3-1, vehicle A is coming from 2 and vehicle B is coming from 3, thus, the original direction of vehicle A is \( M = 2 \) and that of vehicle B is \( N = 3 \). Because U-turns are disallowed in the analysis that follows, the final destinations, subscripted with \( f \), cannot be the incoming directions of either vehicle. Mathematically, this requires that \( M_f \neq M \) and \( N_f \neq N \).

### 3.2 Understanding the Collision Patterns

Table 3-1 illustrates the combinations of maneuvers that result in collision patterns, with collisions noted specifically in the last two rows. To use this table: whenever the intended final directions for vehicle A and B both satisfy the set membership within the specified range respectively from the same row, there is a potential collision at the intersection. The MATLAB codes written to check for these collision patterns are provided in Appendix A. All representations are in modulo 4 for a four-way intersection, where the smallest number is 0 and the largest number is 3. In order to avoid any confusions, an addition example in modulo 4 is provided here:

\[
3 + 1 = 4 \equiv 0 \pmod{4}
\]

The range representation follows the general mathematical rules for integer sets: the use of parenthesis means the end integer is not included in the set, and the bracket means the end integer is included. Note that counting for this 4-way intersection is in modulo 4, and that two numbers in the range representation denotes counting up counterclockwise from the left number to the right.
number at the intersection. For example, (0, 2] equals directions 1,2, whereas the notation [2,1] denotes directions 2,3,0,1.

<table>
<thead>
<tr>
<th>Collision patterns</th>
<th>Vehicle A</th>
<th>Vehicle B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original directions</td>
<td>( M )</td>
<td>( N )</td>
</tr>
<tr>
<td>Intended directions</td>
<td>( M_f )</td>
<td>( N_f )</td>
</tr>
<tr>
<td>Intended directions that cause collisions</td>
<td>((M,N + 1))</td>
<td>((M,M_f))</td>
</tr>
<tr>
<td>Intended directions that cause collisions</td>
<td>([N + 1, M])</td>
<td>([M_f, M])</td>
</tr>
</tbody>
</table>

3.3 Examples for Implementing Collision Patterns in a Four-Way intersection

Considering a four-way intersection situation, vehicle 1 and vehicle 2 is travelling from direction 2 and 3 towards the intersection respectively (i.e. \( M = 2, N = 3 \)). The following three examples of desired final direction pairs, as shown in Figure 3-2, are presented to demonstrate the implementations of the collision patterns.

![Figure 3-2. Examples of direction pairs for vehicle 1 and 2 travelling from direction 2 and 3 respectively in a four-way intersection.](image)

In the first example, vehicle 1 and vehicle 2 intend to go to directions 0 and 1 respectively, thus the desired final directions are \( M_f = 0, N_f = 1 \). This situation is the collision pattern shown
in part I of Figure 3-2. According to the fourth row in Table 3-1, the set of possible intended directions for vehicle 1 is given by \([N +1, M) = [0,2) = \{0,1\}\), denoted as set A, and that for vehicle 2 is given by \([M_f, M] = [0,2] = \{0,1,2\}\), denoted as set B. One can observe that, in this case \(M_f\) indeed belongs to set A, and \(N_f\) belongs to set B, and so there is a potential collision between vehicle 1 and 2.

In the second example, as shown in part II of Figure 3-2, the desired final directions are \(M_f = 1, N_f = 2\), with M and N as defined in example I. The set of possible intended directions, A, is still the same as before because M and N does not change: \([N +1, M) = [0,2) = \{0,1\}\). Similarly, the set of possible intended directions, B, can be formed by \([M_f, M] = [1,2] = \{1,2\}\). It is again the case that \(M_f\) is in set A and \(N_f\) is in set B; therefore, this direction pair results a potential collision at the intersection.

The third example is shown in part III of Figure 3-2. And while M and N stays the same as the prior two examples, the intended directions are now \(M_f = 3, N_f = 0\). The third row in Table 3-1 returns the set of possible intended directions for vehicle 1 as \((M, N +1) = (2,0) = \{3\}\), denoted as A, and that for vehicle 2 as \((M, M_f) = (2,3) = \{3\}\), denoted as B. In this case, \(M_f\) is in set A, but \(N_f\) is not in set B, and so this row does not predict a collision possibility. According to the fourth row in Table 3-1, the set of possible intended directions for vehicle 1 is the same as in the first and second examples: \([N +1, M) = [0,2) = \{0,1\}\), denoted as set C, and that for vehicle 2 is given by \([M_f, M] = [3,2] = \{3,0,1,2\}\), denoted as set D. In this case, although \(N_f\) is in set D, \(M_f\) is not in set C. Because neither row predicts a collision possibility, this direction pair of \(M_f = 3, N_f = 0\) does not satisfy any collision patterns in Table 3-1, which means that vehicles following these
trajectories would not cause any collision at the intersection even if both vehicles were in the intersection at the same time.

### 3.4 Examples for Implementing Collision Patterns in a N-Way intersection

Though the collision patterns are originally designed for four-way intersections, they also work well in three-way, five-way intersections and beyond. In the cases of N-way intersections, the collision patterns listed in Table 3-1 stay the same, while modulo representations should change into modulo N. Note that the MATLAB codes of checking collision patterns, again in Appendix A, also works for N-way intersections as long as the number of modulo is correctly defined in the beginning and all other set notations follow the same inclusion and modulo-counting rules defined earlier, except with the new modulo base system.

Three-way intersections are commonly seen in the real world, particularly where one road has a T-intersection into another road. For instance, highway entries can be considered as three-way intersections, and as well a driveway from a house can also form a three-way intersection.

Eight-way intersections may seem extreme but are nonetheless shown here to illustrate that the set notations mentioned previously continue to work. Figure 3-3 illustrates an example of a possible eight-way intersection where driveways from stores could be intersecting into a four-way intersection. The following three sections present examples for determining potential collision patterns in three-way, five-way and eight-way intersections.
3.4.1 Examples for Implementing Collision Patterns in a Three-Way intersection

With a three-way intersection, the different directions are labeled from 0 to 2 in modulo 3. Suppose vehicle 1 and vehicle 2 is travelling from direction 1 and 2 towards the intersection respectively (i.e. \( M = 1, N = 2 \)). The following two intended direction pairs, as shown in Figure 3-4, are presented to demonstrate the implementations of the collision patterns.

![Figure 3-4. Examples of direction pairs vehicle 1 and 2 travelling from direction 1 and 2 respectively in a three-way intersection.](image)

In part I of Figure 3-4, the intended directions are \( M_f = 0, N_f = 1 \). According to the fourth row in Table 3-1, the set of possible intended directions for vehicle 1 is: \([N + 1, M] = [0,1] = \{0\}\), denoted as set A, and that for vehicle 2 is given by \([M_f, M] = [0,1] = \{0,1\}\), denoted as set B. In
this case, $M_f$ belongs to set A and $N_f$ belongs to set B, which would result in a potential collision at the intersection.

In the second example, part II of Figure 3-4, while M and N stays the same, the intended directions are $M_f = 2, N_f = 1$. The third row in Table 3-1 returns the set of possible intended directions for vehicle 1 as $(M, N +1) = (1,0) = \{2\}$, denoted as A, and that for vehicle 2 as $(M, M_f) = (1,2) = \{2\}$, denoted as B. Clearly, even though $M_f$ is in set A, the direction $N_f$ is not in set B. According to the fourth row in Table 3-1, the set of possible intended directions for vehicle 1 is the same as in the first example: $[N +1, M) = [0,1) = \{0\}$, denoted as C, and that for vehicle 2 is given by $[M_f, M] = [2,1] = \{2,0,1\}$, denoted as set D. In this case, although $N_f$ is in set D, the direction $M_f$ is not in set C. Therefore, this direction pair of $M_f = 2, N_f = 1$ does not satisfy any collision patterns in Table 3-1, and these direction pairs would not cause any collisions at the intersection.

### 3.4.2 Examples for Implementing Collision Patterns in a Five-Way intersection

With a five-way intersection, the different directions are marked from 0 to 4 in modulo 5. Suppose vehicle 1 and vehicle 2 is travelling from direction 2 and 4 towards the intersection respectively (i.e. $M =2$, $N = 4$). The following two direction pairs, as shown in Figure 3-5, are presented to demonstrate the determination of possible collision patterns.
Figure 3-5. Examples of direction pairs vehicle 1 and 2 travelling from direction 2 and 4 respectively in a five-way intersection.

In the first example, part I of Figure 3-5, the starting directions are $M = 2, N = 4$ and the intended final directions are: $M_f = 0, N_f = 1$. According to the fourth row in Table 3-1, the set of possible intended directions for vehicle 1 is: $[N + 1, M) = [0, 2) = \{0, 1\}$, denoted as set A, and that for vehicle 2 is given by $[M_f, M] = [0, 2] = \{0, 1, 2\}$, denoted as set B. In this case, $M_f$ belongs to set A and $N_f$ belongs to set B. This therefore predicts that there is a potential collision at the intersection.

In the second example, as shown in part II of Figure 3-5, while $M$ and $N$ stays the same, the intended directions are: $M_f = 4, N_f = 0$. The third row in Table 3-1 returns the set of possible intended directions for vehicle 1 as $(M, N + 1) = (2, 0) = \{3, 4\}$, denoted as A, and that for vehicle 2 as $(M_f, M) = (4, 2) = \{4, 0, 1, 2\}$, denoted as B. Clearly, even though $M_f$ is in set A and $N_f$ is not in set B. According to the fourth row in Table 3-1, the set of possible intended directions for vehicle 1 is the same as in the first example: $[N + 1, M) = [0, 2) = \{0, 1\}$, denoted as set C, and that for vehicle 2 is given by $[M_f, M] = [4, 2] = \{4, 0, 1, 2\}$, denoted as set D. In this case, although $N_f$ is in set D, $M_f$ is not in set C. Therefore, this direction pair of $M_f = 4, N_f = 0$ does not satisfy any collision patterns in Table 3-1, which means that vehicles following these trajectories would not
cause any collision at the intersection, even if both vehicles were in the intersection simultaneously.

### 3.4.3 Examples for Implementing Collision Patterns in an Eight-Way Intersection

With an eight-way intersection, the different directions are marked from 0 to 7 in modulo 8. Suppose vehicle 1 and vehicle 2 are travelling from direction 0 and 5 towards the intersection respectively (i.e. $M = 0$, $N = 5$). The following two direction pairs, as shown in Figure 3-6, are presented to demonstrate the implementations of the collision patterns.

**EXAMPLE I**

In the first example, as shown in part I of Figure 3-6, the intended directions are: $M_f = 6, N_f = 0$. According to the fourth row in Table 3-1, the set of possible intended directions for vehicle 1 is: $[N + 1, M] = [6,0] = \{6,7\}$, denoted as set A, and that for vehicle 2 is given by $[M_f, M] = [6,0] = \{6,7,0\}$, denoted as set B. In this case, $M_f$ and $N_f$ belongs to set A and B respectively, which would result in a potential collision at the intersection.
In the second example, as shown in part II of Figure 3-6, while M and N stay the same, the intended directions are: $M_f = 3, N_f = 0$. The third row in Table 3-1 returns the set of possible intended directions for vehicle 1 as $(M, N+1) = (0,6) = \{1,2,3,4,5\}$, denoted as A, and that for vehicle 2 as $(M, M_f) = (0,3) = \{1,2,3\}$, denoted as B. Clearly, even though $M_f$ is in set A, but $N_f$ is not in set B. According to the fourth row in Table 3-1, the set of possible intended directions for vehicle 1 is the same as the first example: $(N+1, M) = (6,0) = \{6,7\}$, denoted as set C, and that for vehicle 2 is given by $[M_f, M] = [3,0] = \{3,4,5,6,7,0\}$, denoted as set D. In this case, although $N_f$ is in set D, $M_f$ is not in set C. Therefore, this direction pair of $M_f = 3, N_f = 0$ does not satisfy any collision patterns in Table 3-1, and these direction pairs would not cause any collisions at the intersection even if both vehicles were in the intersection simultaneously.
Chapter 4

Cooperative Driving Algorithm for Two Vehicles at a Non-Signalized Intersection

This chapter provides a detailed description of the proposed cooperative driving algorithm. The ordering vehicles in a collision-free and efficient way to pass an intersection is one of the ultimate goals for every traffic control device and cooperative driving algorithms. The proposed algorithm for two vehicles helps to organize vehicles by first using their initial velocity profiles to determine priority, then utilizing the collision patterns to determine whether trajectories should be modified. The algorithm is assumed to be deployed as a centralized trajectory planning method for intersection collision free navigation. The algorithm also assumes that each vehicle has a working, error-free wireless communication link to exchange information with a central agent.

The following five sections are separated by the five logical steps that are organized in the algorithm: 1) defining the input parameters, 2) performing a time calculation for each vehicle’s arrival, 3) performing a time comparison, 4) evaluating possible collision patterns, and 5) redesigning velocity profile of the secondary vehicle, as necessary, to avoid the primary vehicle.
4.1 Defining the Input Parameters

The collision-free intersection navigation system depends on specific situational characteristics that must be defined before use; the input parameters of the algorithm are listed in Table 4-1. Both the initial and intended directions are essential because they not only are required in evaluating collision patterns later, but they also determine if a vehicle is turning or not. The initial velocities are assumed to be constant for both vehicles. The algorithm starting distance is the distance between the initial communication with each vehicle and where the intersection starts, as shown in Figure 4-1; this is also where the algorithm starts the trajectory analysis for each vehicle. The vehicle length is an input for calculation because it is only when the tail of a vehicle leaves the intersection that one can assume the intersection is free for another vehicle’s use. For maneuvers involving turning within the intersection, the algorithm assumes that drivers usually slow down to a comfortable turning speed before they make a turn – in this algorithm, this speed is assumed to be 15 mph, but can be adjusted. Thus, the turning speed choice is important for this algorithm to be pragmatic. The allowable acceleration / deceleration is an input variable used for predicting the allowable changes in velocity profiles for vehicles that can have turn behaviors, and if potential collisions exist, for redesigning velocity profiles. The parameter representing the allowable acceleration may strongly depend on weather and road conditions: icy or wet roads may not allow fast velocity changes, and thus maneuvers that are collision-free in dry-road situations could cause collisions in adverse weather. This same parameter allows the algorithm to avoid sudden accelerations or decelerations, thereby keeping passengers comfortable. As a starting point for this parameter, highway design rules generally limit the value of a comfortable acceleration, $a$, to be within the less than 3.4 m / $s^2$ [51]. For simplicity in the analysis that follows, a constant value of $a$ is assumed whenever a vehicle needs to accelerate or decelerate.
Table 4-1. Vehicle Initial Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Vehicle 1</th>
<th>Vehicle 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Directions</td>
<td>$M$</td>
<td>$N$</td>
</tr>
<tr>
<td>Intended Directions</td>
<td>$M_f$</td>
<td>$N_f$</td>
</tr>
<tr>
<td>Initial Velocities (m/s)</td>
<td>$v_1$</td>
<td>$v_2$</td>
</tr>
<tr>
<td>Algorithm Starting Distance (m)</td>
<td>$d_1$</td>
<td>$d_2$</td>
</tr>
<tr>
<td>Vehicle Length (m)</td>
<td>$l_1$</td>
<td>$l_2$</td>
</tr>
<tr>
<td>Turning Speed (m/s)</td>
<td>$v_{\text{turn}}$</td>
<td></td>
</tr>
<tr>
<td>Acceleration (m/s^2)</td>
<td>$a$</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4-1. An illustration of all initial parameters at an intersection.
4.2 Performing a Time Calculation

The second step of the algorithm is to calculate the within-intersection time periods, as shown in Figure 4-2. Of particular interest is to determine which vehicle enters the intersection first, as this vehicle will become the primary vehicle, and the secondary vehicle will then be analyzed for potential conflicts with the primary vehicle. The MATLAB code for generating the 2D time-space diagram is provided in Appendix C for each vehicle based on the input variables.

This second step of the algorithm borrows the idea of calculating time metric of time-to-intersection (TTX) in collision avoidance system mentioned in the literature review. The TTX for the \( i \)th vehicle respectively is denoted as \( t_{entry,i} \). The time calculation method for \( t_{entry,i} \) depends of course on whether a straight-through or turning maneuver occurs for the vehicle within the intersection. If the \( i \)th vehicle travels straight through the intersection, the time \( t_{entry,i} \) for the vehicle is calculated simply as the starting distance divided by initial constant velocity. On the other hand, if the vehicle plans to turn, then Equation 4.1, which considers the behavior of slowing down before turning, is used for computing \( t_{entry,i} \). The first vehicle in the hierarchy, meaning the vehicle that enters the intersection first, is denoted as vehicle 1; this can be defined by definition such that vehicle 1 is the vehicle which has the smaller \( t_{entry,i} \), i.e. which enters the intersection first.

\[
t_{entry,i} = \frac{2ad_i-v_i^2+v_{turn}^2}{2av_i} + \frac{v_i-v_{turn}}{a}
\]  
(4.1)
The calculation for the time vehicle 1 takes to pass through the intersection, $t_{pass1}$, is next separated into three general cases as shown in Equation 4.2. Three different behaviors are possible: the vehicle goes straight; the vehicle makes a small turn (usually the right-hand turn in the US traffic system) or a large turn (usually a left-handed turn). Each situation results in a different actual travel distance within the intersection. This difference is also presented in Figure 2-3 where the end of the intersection is different for vehicle 1 and vehicle 2.

$$t_{pass1} = \begin{cases} \frac{2w + l_1}{v_1} & \text{straight} \\ \frac{w \pi}{4v_{\text{turn}}} + \frac{l_1}{v_{\text{turn}}} & \text{small turn} \\ \frac{3w \pi}{4v_{\text{turn}}} + \frac{l_1}{v_{\text{turn}}} & \text{large turn} \end{cases} \quad (4.2)$$

The total time vehicle 1 takes to exist the intersection is denoted as $t_{exit1}$. As shown in Fig.3, it is clear that $t_{exit1}$ is the sum of $t_{entry1}$ and $t_{pass1}$. This figure also illustrates a situation where
vehicle 2 is attempting to enter the intersection before vehicle 1 leaves, creating an area in the
time-space – shaded in the plot – where a collision is possible.

4.3 Performing a Time Comparison

After the entry-time calculations, the third step of the algorithm is to compare $t_{exit1}$ and
$t_{entry2}$, which is equivalent to checking whether an overlap area exists in time-space for the two
vehicles within the intersection. This step distinguishes two situations in Figure 4-3 that affect the
execution of remaining algorithm steps: either a collision is avoided because the times of
intersection occupancy are clearly separated (left plot of the figure), or these times overlap (right
plot of the figure). Specifically, if $t_{exit1} < t_{entry2}$, then this denotes that vehicle 1 can exit the
intersection before vehicle 2 reaches the intersection entry; therefore, there is no overlap area
between the two boxes in the time-space diagram as shown in the left diagram of Figure 4-3. Thus,
no collision would occur at the intersection and the two vehicles can proceed with their original
constant speeds without any speed modification required. In contrast, if $t_{exit1} > t_{entry2}$, an overlap
area is created between the vehicles in time-space, as shown in the right diagram of Figure 4-3.
The overlap denotes that vehicle 2 enters the intersection before vehicle 1 exits the intersection. In
this case, the two vehicles travelling in the intersection at the same time may cause a potential
collision. In this collision-possible situation, the algorithm proceeds to step four.
If the fourth step is entered after step three, then the two vehicles are confirmed to be within the intersection at the same time. Even in this case, there are common situations where, depending on the vehicle trajectories, the two vehicles can share the intersection usage. These non-collision situations were explained previously in the previous chapter. This fourth step of evaluating potential collision patterns helps eliminate velocity negotiation between vehicles that appear to conflict in 2-D time-space, but are not in conflict in actual 3-D space due to allowable shared usage of the intersection.

Based on the initial and intended directions of two vehicles, the collision patterns that are introduced in Chapter 3, in Table 4-2 are now evaluated. If the intended directions do not form a collision pattern, then the two vehicles can maintain their original trajectory without any risk in potential collisions. But if the intended directions form a collision pattern, then the algorithm continues to step five to prevent a collision.

Table 4-2. Collision Patterns
4.5 Taking Anti-Collision Actions if Necessary

This fifth step begins only if the unaltered trajectories of two incoming vehicles are confirmed to form a possible collision at the intersection. To avoid conflict and minimize velocity changes to vehicle 1, the algorithm seeks to redesign the velocity profile only for vehicle 2 to produce a trajectory that will arrive at the intersection secondly after vehicle 1. Because this calculation requires very clear knowledge of both vehicle situations, signalized intersections often require human drivers to slow down possibly to a complete stop in order to avoid collisions with others. In the extreme case of a 4-way stop sign, all incoming vehicles are expected to come to a complete stop; while this is certainly the most secure way to ensure collision-free occupancy of an intersection, these actions penalize traffic flow, fuel economy, and human comfort. In the case of two vehicles travelling into a potential conflict usage of the same intersection, the algorithm that follows assumes that is better to keep one vehicle on its original trajectory and impose velocity changes only on vehicle 2 to avoid collisions.

For simplicity, the algorithm slows down vehicle 2 using the velocity profiles provided in Figure 4-4, based on whether vehicle 2 is turning or not. The redesigned profiles are constrained by obtaining the total travel distance $d_2$, using the selected acceleration value, and then assigning $t_{exit1}$ as the new time $t_{entry2}$. If either vehicle is turning within the intersection, the user-selected
turning speed is used and is not allowed to change during the maneuver. These constraints ensure that vehicle 1 exits the intersection before vehicle 2 enters the intersection, thus eliminating the potential collision. In order to contribute to both fuel and traffic efficiency, the new velocity profile lets vehicle 2 slow down over a reasonable period, a deceleration which potentially allows regenerative braking on hybrid/electric vehicles, and then recover speed to the original velocity before vehicle 2 reaches the intersection entry when no turning is involved. If vehicle 2 intends to turn at the intersection, then it is slowed before turning and then is assumed to speed back to travel speed after intersection exit.

![Figure 4-4. Two possible velocity profiles for the vehicle 2. The left-side profile is designed for straight maneuver only. The right-side profile is designed for turning maneuvers.](image)

With those two possible velocity profiles, the objective of the fifth step is then to calculate the appropriate initial deceleration period \( t \), for vehicle 2 so that a potential collision can be avoided. This period also equals the final acceleration period since the acceleration and deceleration values are the same. This period, \( t \), can be computed based on the fact that total travel distance and time are, at this point, fully specified. In the situation where vehicle 2 goes straight within the intersection, Figure 4-5 labels each part of travel distances, \( d \), with subscripts corresponding to each velocity characteristic: deceleration (decel), constant velocity (cons), and acceleration (accel).
The total distance can be calculated from the following equations:

\[ d_2 = d_{decel} + d_{accel} + d_{cons} \]  \hspace{1cm} (4.3)

\[ d_{decel} = v_2 t - \frac{1}{2} a t^2 \]  \hspace{1cm} (4.4)

\[ d_{accel} = v_{cons} t + \frac{1}{2} a t^2 \]  \hspace{1cm} (4.5)

\[ d_{cons} = v_{cons}(t_{exit1} - 2t) \]  \hspace{1cm} (4.6)

where \( v_{cons} \) is the constant velocity that vehicle 2 experiences after deceleration. This velocity can be computed from the following equation:

\[ v_{cons} = v_2 - at \]  \hspace{1cm} (4.7)

By substituting Equation 4.7 into Equation 4.3 and simplifying the result, the initial deceleration period, \( t \), can then be solved from the resulting two-degree polynomial equation:

\[ at^2 - at_{exit1} + v_2 t_{exit1} = d_2 \]  \hspace{1cm} (4.8)

In the situation where vehicle 2 plans to turn at the intersection, Figure 4-6 labels each part of the distance following the same notation as Figure 4-5.
Even though the turning case is more complicated than the straight-through driving situation, the central idea is similar: determine the time duration necessary for velocity changes that meet all specifications on acceleration, velocity, and distance while requiring collision-free entry into the intersection. Equation 4.9 dominates the calculation process, and each part in side can be written from Equations 4.10 to 4.12:

\[
d_2 = d_{decel} + d_{decel2} + d_{cons} \tag{4.9}
\]

\[
d_{decel} = v_2 t - \frac{1}{2} at^2 \tag{4.10}
\]

\[
d_{decel2} = v_{cons} t_2 - \frac{1}{2} at_2^2 \tag{4.11}
\]

\[
d_{cons} = v_{cons}(t_{exit1} - t - t_2) \tag{4.12}
\]

where \( t_2 \) is denoted as the second deceleration period:

\[
t_2 = \frac{v_{cons} - v_{turn}}{a} \tag{4.13}
\]

and \( v_{cons} \) is the constant velocity that vehicle 2 experiences after the initial deceleration:

\[
v_{cons} = v_2 - at \tag{4.14}
\]
Substituting these equations into the governing Equation 4.9, both of the deceleration periods can then be calculated. The time for vehicle 2’s turning period can be calculated by using the Equation 4.2 in step two of the algorithm. The last time period, that for vehicle 2 to recover its speed back to the original travel speed, can also be easily computed from:

\[ t_{accel} = \frac{v_2 - v_{turn}}{a} \quad (4.15) \]

After these calculations, vehicle 2 has sufficient information to follow the redesigned profile with all accelerations, decelerations, velocities and time periods computed.

In the 2-D time-space diagram, Figure 4-7 illustrates the ultimate goal of the proposed algorithm: to produce a collision-causing intersection situation that can be readily and quickly recomputed. If a collision exists, the algorithm is able to shift the time-space intersection-occupancy box of vehicle 2 to where its front edge touches the rear edge of vehicle 1’s time-space intersection occupancy box. This geometry also indicates that vehicle 2 enters the intersection at exactly the same time when vehicle 1 exits the intersection, which perfectly avoids a pre-existed collision. The complete MATLAB script for all five steps of algorithm is provided in Appendix B.
Figure 4-7. An example of adjusted trajectories in 2-D time-space diagram.
Chapter 5

Complete Examples of Using the Cooperative Driving Algorithm at a Four-Way Non-Signalized Intersection

The goal of this chapter is to provide complete examples of how the designed cooperative driving algorithm generates instructions to each vehicle approaching a four-way intersection. By giving step-by-step descriptions with real parameters, the three examples present different situations that are common in real life. The intersection geometry in these examples remain the same as introduced in chapter 3. The first section demonstrates an example of two vehicles sharing intersection usage at the same time with no collisions patterns. Next, an example of two vehicles having collisions patterns but not sharing intersection usage is shown in the second section. In the third section, the situation where two vehicles sharing intersection usage with collision patterns is discussed.

5.1 An Example of Two Vehicles with Collision Patterns Passing Through the Intersection Without Changing Trajectories

The first step in the algorithm is to define the input parameters. Suppose two vehicles are approaching the intersection from directions 2 and 3 at their own constant speeds. Their initial parameters are shown in Figure 5-1. The constant acceleration is set to be 2 m/s² and the turning speed is set to be 4 m/s in the algorithm. The algorithm starting distances are set to be the same for
both vehicles at 100 m. Each of these parameters is chosen to be representative in value to real-world travel through intersections.

Figure 5-1. Vehicles’ initial parameters and intersection setup in example 1.

Then the algorithm continues to the second step, performing a time calculation to define the vehicle hierarchy. In this case, it is clear the vehicle 1 from direction 3 will arrive at the intersection first because of its larger velocity. With Equation 4.1, \( t_{\text{entry},i} \) can be calculated for both vehicles. Then the vehicle from direction 3 with the smaller \( t_{\text{entry},i} \) is thus defined as vehicle 1. By using Equation 4.2 with the small turn case, \( t_{\text{pass}1} \) can be found. Therefore, \( t_{\text{exit}1} \) is obtained from the summation of \( t_{\text{entry}1} \) and \( t_{\text{pass}1} \) as shown in Figure 5-2.

Figure 5-2. A 2-D time space diagram demonstrates the step two of the algorithm.
Now with all the time outputs defined from the second step, the third step is to compare \( t_{\text{entry}2} \) versus \( t_{\text{exit}1} \). Obviously, from Figure 5-2, it is clear that \( t_{\text{entry}2} \) is larger than \( t_{\text{exit}1} \). Thus, vehicle 1 will have already exited out the intersection before vehicle 2 will be entering the intersection. Therefore, the two vehicles will not share the intersection at any instants, meaning that there is no potential collision.

Even though Figure 5-1 shows that the intended directions of both vehicles may result a collision, the algorithm has yet analyzed that two vehicles may follow their original trajectories, shown in Figure 5-3. No further steps are needed for these two vehicles.

![Figure 5-3. A 2-D time space diagram demonstrates the final trajectories of two vehicles that do not need to take any actions.](image)

### 5.2 An Example of Two Vehicles Sharing Intersection Usage at the Same Time

The first step in the algorithm is to define the input parameters. Suppose two vehicles are approaching the intersection from direction 2 and 3 at their own constant speeds. Their initial parameters are shown in Figure 5-4. The constant acceleration is set to be 2 m/s\(^2\) and the turning
speed is set to be 4 m/s in the algorithm. The algorithm starting distances are set to be the same for both vehicles at 50 m.

Figure 5-4. Vehicles' initial parameters and intersection setup in example 2.

Then the algorithm continues to the second step, performing a time calculation to define the vehicle hierarchy. In this case, it is clear the vehicle from direction 3 will arrive at the intersection first because of its larger velocity. With Equation 4.1, the time $t_{entry,i}$ can be calculated for both vehicles. Then the vehicle from direction 3 with the smaller $t_{entry,i}$ is defined as vehicle 1. By using Equation 4.2 with the small turn case, the time $t_{pass1}$ can be found. As well, the time $t_{exit1}$ is obtained by adding $t_{entry1}$ and $t_{pass1}$ as shown in Figure 5-5.

Figure 5-5. A 2-D time space diagram demonstrates the step two of the algorithm.
Now with all the time outputs, the third step is to compare $t_{entry2}$ versus $t_{exit1}$. Obviously, from Figure 5-5, it is clear that $t_{entry2}$ is smaller than $t_{exit1}$. Thus, vehicle 1 would still be travelling in the intersection while vehicle 2 enters the intersection. The two vehicles are thus using the intersection at the same time, which may or may not cause a potential collision, which leads the algorithm to start the next step.

The fourth step is to evaluate possible collision patterns. Based on the input parameters shown in Figure 5-4, the approach directions for vehicle 1 and 2 are $M = 2$ and $N = 3$. The intended directions are $M_f = 1, N_f = 0$. The third row in the collision patterns table, Table 3-1, returns the set of possible intended directions for vehicle 1 as $(M, N+1) = (2,0) = \{3\}$, denoted as A, and that for vehicle 2 as $(M, M_f) = (2,1) = \{3,0,1\}$, denoted as B. Clearly, even though $N_f$ is in set B, the value of $M_f$ is not in set A. According to the fourth row in Table 3-1, the set of possible intended directions for vehicle 1 is: $[N+1, M) = [0,2) = \{0,1\}$, denoted as set C, and that for vehicle 2 is given by $[M_f, M] = [1,2] = \{1,2\}$, denoted as set D. In this case, although $M_f$ is in set C, $N_f$ is not in set D. Therefore, these original and intended directions of vehicles 1 and 2 do not match with any collision patterns in Table 3-1, and instead form a safe pattern. The appearance of this safe pattern indicates that the vehicles following these trajectories would not cause any collision at the intersection even though they travel within the intersection simultaneously.

The algorithm thus determines that the two vehicles may follow their original trajectories, shown in Figure 5-6, without taking any actions. This is because there will be no potential collisions between them, despite using the intersection at the same time.
5.3 An Example of Two Vehicles That Have a Potential Collision at the Intersection

For this third example, again the first step in the algorithm is to define the input parameters. Suppose two vehicles are approaching the intersection from directions 2 and 3 at their own constant speeds. Their initial parameters are shown in Figure 5-7. The constant acceleration is set to be 2 m/s$^2$ and the turning speed is set to be 4 m/s in the algorithm. The algorithm starting distances are set to be the same for both vehicles at 100 m.

Figure 5-6. A 2-D time space diagram demonstrates the step two of the algorithm.

Figure 5-7. Vehicles’ initial parameters and intersection setup used in example 3.
Next the algorithm continues to the second step, performing a time calculation to define the vehicle hierarchy. In this case, it is clear that the vehicle from direction 3 will arrive at the intersection first because of its larger velocity value. With Equation 4.1, the time $t_{\text{entry},i}$ can be calculated for both vehicles. Then the vehicle from direction 3 with the smaller time, $t_{\text{entry},i}$, is defined as vehicle 1. By using Equation 4.2 with the smaller turn case, the time $t_{\text{pass}}$ can be found. The sum of $t_{\text{entry}}$ and $t_{\text{pass}}$ gives the time $t_{\text{exit}}$ as shown in Figure 5-8.

![2-D time space diagram demonstrating the step two of the algorithm.](image)

Once all the time outputs are calculated, the third step is to compare the times $t_{\text{entry},2}$ versus $t_{\text{exit}}$. From Figure 5-9, it is obvious that $t_{\text{entry},2}$ is smaller than $t_{\text{exit}}$. Thus, vehicle 1 would still be travelling in the intersection while vehicle 2 enters the intersection. The two vehicles using the intersection at the same time may or may not cause a potential collision, and this potential causes the algorithm to proceed to the next step.

Next, the fourth step of the algorithm is to evaluate whether there is a possible collision pattern. Based on the input parameters shown in Figure 5-7, the approach directions are from M =
2 and N = 3. The intended directions are \( M_f = 1, N_f = 2 \). The third row in the collision patterns table, Table 3-1, returns the set of possible intended directions for vehicle 1 as \((M, N + 1) = (2,0) = \{3\}\), denoted as A, and that for vehicle 2 as \((M_f, M) = (2,1) = \{3,0,1\}\), denoted as B. Clearly, \( M_f \) is not in set A and \( N_f \) is not in set B. According to the fourth row in Table 3-1, the set of possible intended directions for vehicle 1 is: \([N + 1, M) = [0,2) = \{0,1\}\}, denoted as set C, and that for vehicle 2 is given by \([M_f, M] = [1,2] = \{1,2\}\}, denoted as set D. In this case, \( M_f \) is in set C and \( N_f \) is in set D. Therefore, these original and intended directions of vehicles 1 and 2 form a collision pattern. This predicted potential collision allows the algorithm to proceed to the last step, taking anti-collision actions if necessary.

In this last step, the algorithm aims to redesign the velocity profile for vehicle 2. The input parameters in step indicate that vehicle 2 is turning at the intersection. Thus, the redesigned velocity profile is assumed to follow the template as shown in Figure 4-5, where the vehicle will experience two periods of decelerating. By implementing Equations 4.9 to 4.15, the algorithm calculates the time periods for decelerating, travelling at a constant speed, decelerating for turning, turning, and then accelerating back to the original speed for vehicle 2. Following those time periods, and requiring that the deceleration/acceleration and turning speed values be met, vehicle 2 now has sufficient information to pass through the intersection safely without any collision with vehicle 1. Figure 5-9 illustrates the final travel trajectories of both vehicles. It is clear that the two time-space boxes showing both vehicles’ intersection time periods within the intersection do not have any overlap but are touching. This means that vehicle 2 enters the intersection immediately as vehicle 1 exits. This demonstration in 2-D time-space diagram indeed proves that the adjusted trajectories avoid the possible collision that could have occurred without the trajectory modification.
Figure 5-9. A 2-D time-space diagram demonstrates the final adjusted trajectories for two vehicles.

Figure 5-10 may be helpful to understand the goal of the calculation process in step five more clearly. This figure shows a plot of the original and final adjusted trajectories for both vehicles together. Geometrically, the redesigned trajectory of vehicle 2 shifts the original red box in time-space which represents the intersection occupancy, which has overlap with the blue box, to the right. When considering the time it takes vehicle 2 to proceed through the intersection, vehicle 2 gives up its original intersection entry time and takes $t_{exit1}$ as its new entry time to avoid conflicts with vehicle 1.

Figure 5-10. A 2-D time space diagram demonstrates the both the original and final adjusted trajectories of the two vehicles.
Chapter 6

The Limitations of the Proposed Algorithm

Ensuring safety is among the most important goals of every driving-related algorithm. Therefore, it is necessary to determine whether and when the proposed algorithm may fail or cause improper instructions for vehicles. This chapter provides a dimensional analysis to specify the limitations of the algorithm starting distance \(d\) with changing input variables. The first section explains the general situations of when proposed algorithm may return errors. The second section introduces the analysis process of developing dimensionless parameters and equations. The third section discusses the analysis method. The last section presents the results and findings based on three different study cases: both vehicles going straight, one vehicle has a conflict turn, and both vehicles have conflict turns.

6.1 Errors in the Outputs of the Proposed Algorithm Step Five

In the fifth step of the algorithm, the situation is considered where prior steps of algorithm predict that the two vehicles’ original trajectories could result a collision at the intersection and thus that vehicle 2 needs to yield and change its velocity profile. The objective of the fifth step is to calculate an appropriate initial decelerating period for vehicle 2 to avoid this possible collision. Other time periods in the trajectory, such as travelling at a constant speed and/or initiating a second deceleration in the case of when turning behavior is involved, are then computed from this initial
decelerating period, as shown earlier in Chapter 4. However, the calculation of the initial deceleration time in step five may return unrealistic time periods (i.e. negative or imaginary times) for vehicle 2, depending on the initial input variables. These situations indicate that there are no possible solutions available that both avoid the potential collision and meet the trajectory constraints.

To understand how situations could arise where no solutions are possible, it is helpful to revisit the underlying equations and how they are solved. In the case where both vehicles go straight, Equation 4.8 is the final governing equation for solving the initial decelerating period when vehicle 2 needs to change its velocity profile. This equation is a polynomial of degree 2. It is possible that the discriminant part in the quadratic formula, as shown in Equation 6.1, for solving Equation 4.8 returns a negative number based on the combinations of velocity and acceleration values, or on the starting distances entered. For example, when \( v_2 = 10 \text{ m/s}, d_1 = d_2 = 50 \text{ m}, a = 0.5 \text{ m/s}^2, \) and \( t_{exit1} = 6.2 \text{ s} \) (resulting from \( v_1 = 10 \text{ m/s} \) and calculations in section 4.2), Equation 6.1 returns a determinant equal to -14.39.

\[
b^2 - 4ac = (at_{exit1})^2 - 4a(v_2t_{exit1} - d_2) \tag{6.1}
\]

This negative discriminant part can then result in an imaginary initial decelerating period.

In the case where turning behavior is involved, while imaginary initial decelerating periods may appear with the same reason as both vehicles go straight, the situation is more complicated because now there are two different decelerating periods. According to Equation 4.13, if \( v_{cons} \) is less than \( v_{turn} \), then the second decelerating period, \( t_2 \), will be negative. Equation 4.14 gives the process of computing \( v_{cons} \), which indicates the possibility of \( v_{cons} \) being small with certain combinations of deceleration/acceleration values and initial decelerating periods. For example, suppose both vehicles are making larger turns and their initial velocities are all 20 m/s. Setting
\(d_1 = d_2 = 200 \text{ m}, v_{\text{turn}} = 4 \text{ m/s} \text{ and } a = 1 \text{ m/s}^2\). By following the calculation process in section 4.5, the initial decelerating period is 17.9 s. In this case, it is clear that \(v_{\text{cons}} = v_2 - at = 20 - 17.9 < v_{\text{turn}}\). Thus, the constraints predict that this maneuver is infeasible.

These situations illustrate cases where there is not enough time for vehicle 2 to follow a velocity profile change while meeting the deceleration limits selected, given the distance available between the vehicle and the intersection. In order to avoid the output time periods from being negative or imaginary, it is possible to find the minimum algorithm starting distance, \(d\), for every selected acceleration value. The ability to mathematically define these situations is exceedingly important as it allows careful, weather-dependent definition of the relationship between the negotiated distance ahead of the intersection, versus the available deceleration of the vehicle which will depend very strongly on road friction.

### 6.2 Developing Dimensionless Parameters and Equations

Because all initial parameters are correlated and affect the algorithm outputs, it is difficult to demonstrate the effects of those parameters’ simultaneous variations on the minimum algorithm starting distance. Dimensional analysis is thus performed to simplify the analysis outcome and minimize parameter dependency as much as possible, but at the same time showing the minimum algorithm starting distance for all combinations of initial parameters that cause unrealistic algorithm outputs.

The first step in this analysis is to identify all initial parameters that affect the final algorithm outputs, as listed in Table 6-1 and to determine their unit dimensions. Note that for simplicity both vehicles are assumed to have the same algorithm starting distance and vehicle
length. With the units listed for all parameters, the next procedure is to produce dimensionless parameters for the situation. Because the provided directions do not have units, they separate the analysis into three different cases, as shown in Figure 6-1: both vehicles go straight, both vehicles have conflict turns, one vehicle has a conflict turn.

Table 6-1. Vehicle Initial Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Vehicle 1</th>
<th>Vehicle 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Directions</td>
<td>( M )</td>
<td>( N )</td>
</tr>
<tr>
<td>Intended Directions</td>
<td>( M_f )</td>
<td>( N_f )</td>
</tr>
<tr>
<td>Initial Velocities (m/s)</td>
<td>( v_1 )</td>
<td>( v_2 )</td>
</tr>
<tr>
<td>Algorithm Starting Distance</td>
<td>( d )</td>
<td>( d )</td>
</tr>
<tr>
<td>(m)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vehicle Length (m)</td>
<td>( l )</td>
<td>( l )</td>
</tr>
<tr>
<td>Turning Speed (m/s)</td>
<td></td>
<td>( v_{\text{turn}} )</td>
</tr>
<tr>
<td>Acceleration (m/s^2)</td>
<td></td>
<td>( a )</td>
</tr>
</tbody>
</table>

Figure 6-1. Three analysis cases for dimensional analysis include when both vehicle go straight (left), one vehicle has a conflict turn (middle), and both vehicles have conflict turns (right).
Though the three cases considered here are different, the unit dimensions of the core parameters are the same for all cases. The three basic dimensions are length (m), time (s) and mass (kg) in international standard units. The notation used hereafter will be to write the dimensions of a parameter as the exponential value of its unit. For example, if the unit of velocity is m/s, then this is equal to $length^1 time^{-1} mass^0$. From the unit of input parameters, it is obvious that there is no mass unit, therefore, no mass dimension. Table 6-2 lists all parameters in their dimensional basis forms.

<table>
<thead>
<tr>
<th></th>
<th>t</th>
<th>$v_1$</th>
<th>$v_2$</th>
<th>d</th>
<th>l</th>
<th>$v_{turn}$</th>
<th>w</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>time</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>-2</td>
</tr>
</tbody>
</table>

By selecting the lane width ($w$) and acceleration/deceleration ($a$) as repeating parameters, the next step is to group all other parameters into dimensionless forms by multiplying or dividing by these two repeating parameters. The converted dimensionless parameters are denoted as $\pi$ parameters. The dimension set matrix operation [52], as shown in Figure 6-2, is performed to solve for the lower right rectangular S matrix, which contains the exponential values of repeating parameters that are used to represent $\pi$ parameters. With the B and A matrices defined as the left and right part of first two rows in Table 6-3, and E matrix defined as an identity matrix, a MATLAB script is written (in Appendix E.1) to solve for S matrix. The results showing the dimensional exponents of each parameter that give a dimensionless $\pi$ parameter are listed in the lower right rectangle of Table 6-3.
Figure 6-2. Dimension set matrix operation for finding the exponential values of repeating parameters that are used to represent $\pi$ parameters [52].

Table 6-3. Dimensional analysis matrix used in calculation.

<table>
<thead>
<tr>
<th></th>
<th>$t$</th>
<th>$v_1$</th>
<th>$v_2$</th>
<th>$d$</th>
<th>$l$</th>
<th>$v_{\text{turn}}$</th>
<th>$w$</th>
<th>$a$</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>time</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>-2</td>
</tr>
<tr>
<td>$\pi 1$</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\pi 2$</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\pi 3$</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\pi 4$</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>$\pi 5$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>$\pi 6$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>-0.5</td>
<td>-0.5</td>
</tr>
</tbody>
</table>

From the above calculations, the parameters listed in Table 6-3 can be written into dimensionless forms as functions of $w$ and $a$. Taking $t$ as an example, one can denote the dimensionless form of time, $t$, as $t^*$ which is given by:

$$t^* = t \, w^{-0.5} \, a^{0.5} = t \, \sqrt{\frac{a}{w}}$$  \hspace{1cm} (6.2)
It is also easy to verify that $t^*$ is indeed dimensionless by unit cancelation:

$$ t \sqrt{\frac{a}{w}} = \text{seconds} \sqrt{\frac{\text{meters}}{\text{seconds}^2 \text{meters}}} = \text{seconds}^0 \text{meters}^0 \quad (6.3) $$

Similarly, the remaining dimensionless parameters are given by:

$$ v_1^* = \frac{v_1}{\sqrt{aw}} \quad (6.4) $$

$$ v_2^* = \frac{v_2}{\sqrt{aw}} \quad (6.5) $$

$$ d^* = d \frac{1}{w} \quad (6.6) $$

$$ l^* = l \frac{1}{w} \quad (6.7) $$

$$ v_{\text{turn}}^* = \frac{v_{\text{turn}}}{\sqrt{aw}} \quad (6.8) $$

By using these dimensionless parameters, all the equations in chapter 4 can now be rewritten into dimensionless forms as well. Taking $t_{\text{entry},i}$ as an example when vehicle $i$ is going straight (where $i = 1$ or 2, referring to vehicle 1 or vehicle 2):

$$ t_{\text{entry},i} = \frac{d}{v_i} = \frac{ws(d)}{\sqrt{aw} (v_i)} \Rightarrow t_{\text{entry},i} \sqrt{\frac{a}{w}} = \frac{d^*}{v_i^*} = t_{\text{entry},i}^* \quad (6.9) $$

Similarly,

$$ t_{\text{entry},i}^* = \frac{d^*}{v_i^*} + \frac{1}{2} v_i^* + \frac{v_{\text{turn}}^*}{2 v_2^*} - v_{\text{turn}}^* \quad , \text{if vehicle } i \text{ is turning} \quad (6.10) $$

$$ t_{\text{pass},i}^* = \begin{cases} 
\frac{2 + l^*}{v_i^*} & \text{straight} \\
\frac{\pi}{4v_{\text{turn}}^*} + \frac{l}{v_{\text{turn}}^*} & \text{small turn} \\
\frac{3\pi}{4v_{\text{turn}}^*} + \frac{l}{v_{\text{turn}}^*} & \text{large turn} 
\end{cases} \quad (6.11) $$

And the governing equation when solving for initial deceleration period can be written as:

$$ (v_2^* - t^*) \left( \frac{d^*}{v_i^*} + \frac{l^*}{v_i^*} + \frac{2}{v_1^*} \right) + t^*^2 = d^* \quad , \text{if vehicle 2 is going straight} \quad (6.12) $$
\[ v_2^* t^* - \frac{t^*^2}{2} + (t_{exit1}^* - t^*) (v_2^* - t^*) - \frac{(v_2^* - t^* - v_{turn})^2}{2} = d^*, \text{ if vehicle 2 is turning} \quad (6.13) \]

Now that all parameters, time metrics and equations are in dimensionless forms, it is much simpler to proceed to find the minimum algorithm starting distance in different cases.

### 6.3 Finding the Minimum Algorithm Starting Distance

The fundamental approach used here to find the minimum starting distance is the same for analyzing those three different cases: both vehicles go straight, both vehicles have conflict turns, and one vehicle has a conflict turn. The initial parameters are chosen to be reasonable constants as before, except for the two vehicles’ initial velocities and algorithm starting distances. Specifically, acceleration \( a = 2 \text{ m/s}^2 \), turning speed \( v_{turn} = 4 \text{ m/s} \), vehicle length \( l = 4 \text{ m} \) and lane width \( w = 4 \text{ m} \). The general analysis method is shown in Figure 6-3, and the detailed MATLAB script is in Appendix E.2-4 for different cases. The analysis considers situations varying both \( v_1 \) and \( v_2 \) from 3 to 45 m/s with a step size of 1 m/s, and algorithm starting distance \( d \) from 0 to 600 meters with a step size of 1 meter. For every velocity combination, the algorithm finds the first entry distance, \( d \), that will not return any imaginary or negative time periods for vehicle 2.

```
for v1 = 3:0.45:45 (m/s)
  for v2 = 3:0.45:45 (m/s)
    for d = 0:0.0600 (m)
      Step II
      Step III
      Step IV (returns time periods for car 2)
      if neither of the time periods is imaginary nor negative
        d_min = d
        break
      end
    end
  end
end
```

Figure 6-3. Simplified analysis algorithm for finding the minimum algorithm starting distance.
In this analysis algorithm, everything is simplified from the original algorithm in chapter 4 in order to save simulation time. The simplification skips step four of the algorithm because it is assumed that two vehicles will have a potential collision trajectory at the intersection, which is the situation by construction to explore the limitations in step five mentioned previously. The straightforward dimensionless equations from section 6.2 are used. The analysis scripts are created separately for three different cases to avoid any if conditions to check turning behaviors; due to this simplification, some equations may change from script to script depending on whether the script is focusing on turning behavior. Again, details of the scripts are shown in the Appendix.

6.4 The Simulation Results

The simulation results show different trends in the minimum algorithm starting distance ($d$) for the three different cases: both vehicles go straight, both vehicles have conflict turns, and one vehicle has a conflict turn. However, the results all proved that in dimensionless forms, there is an invariant relationship between the deceleration settings for an intersection, versus the distance where vehicle speeds would need to be negotiated. Figure 6-4 to 6-6 illustrates the contour plots for the three different cases respectively. For each case, vehicles’ lengths are set to 4 meters and lane width is 4 meters. The plots show that with different acceleration values, the dimensionless minimum $d$ can be the same for all dimensionless velocity combinations.
Figure 6-4. Dimensional analysis results indicate the same dimensionless minimum algorithm starting distance for different acceleration values when both vehicles go straight.

Figure 6-5. Dimensional analysis results indicate the same dimensionless minimum algorithm starting distance for different acceleration values when both vehicles have conflict turns.
Figure 6-6. Dimensional analysis results indicate the same dimensionless minimum algorithm starting distance for different acceleration values when one vehicle has conflict turn.

The use of the simulation results and these plots allow the proposed cooperative driving algorithm to avoid infeasible outputs and to initialize vehicle negotiations of the intersection usage at appropriate algorithm starting distance. The results also show how this negotiation must change based on weather-dependent acceleration values. While it is well-known that deceleration value smaller than 3 m/s give passengers comfortable decelerating experiences, it is not as well known that under extreme weather conditions such as snowing, vehicles should decelerate at even smaller values to prevent unstable tire behaviors. However, those small deceleration values require vehicles to start trajectory modification actions further away from the intersection to avoid any potential collisions. The presented simulation results allow engineers to select a safe distance to start such negotiation, based on weather conditions.

After converting the initial velocities of both vehicles into dimensionless form by implementing Equation 6.4 and 6.5 with known initial parameters, the dimensionless entry distance, $d^*$, can be obtained from the contour plots depending on the conflict cases. Then using Equation 6.6, the dimensionless $d^*$ can be converted back to a dimensional $d$ in meters. This
process ensures safe usage of the proposed algorithm, even allowing it to work under any weather conditions and/or for different kinds of vehicles that may require especially cautious driving behaviors.
This thesis proposed a cooperative driving algorithm that presents a simple methodology to prevent collisions in a two-vehicle signal-free intersection. The proposed algorithm focuses on simplicity by utilizing conflict-free shared usage of an intersection. The use of those discovered patterns allows vehicles with safe trajectories to travel through an intersection simultaneously without collision in order to improve traffic efficiency. An elegant outcome of this mathematical approach which follows is that it predicts collision patterns for N-way intersections, with N being an arbitrary integer. Thus, while the discussion focuses on 4-way intersections, it is proved that the collision patterns apply to any intersection situation.

By dividing intersection coordination into five logical steps, the proposed algorithm is concise and easy to understand. The five steps are introduced in detail in Chapter 4 as follows: defining the input parameters, performing a time calculation and defining the vehicle hierarchy, performing a time comparison, evaluating possible collision patterns, and taking anti-collision actions if necessary. With those steps, the algorithm for two vehicles specifically seeks to minimize speed variations in the vehicles with priority to cause zero trajectory variation in the first vehicle to enter the intersection.

While analyzing the performance of the algorithm, it is proved that the equations used in the step of taking anti-collision actions may return imaginary or negative time outputs while
redesigning velocity profiles for a vehicle. Dimensional analysis is performed to find the minimum algorithm starting distance that return realistic velocity profiles with changing inputs by looking into three different collision cases: both vehicles go straight, both vehicles have conflict turns, one vehicle has a conflict turn. The simulation results for those three cases shows that, even with different acceleration values minimum algorithm starting distances, in dimensionless form the limiting cases all remain the same, for all dimensionless velocity combinations. The contour plots from the simulation are significant because they have mathematically defined the limitations of the algorithm based on initial parameters. This provides engineers the opportunity to change the algorithm settings for different users under various environmental conditions and thus to avoid algorithm failure particularly due to inclement weather.

The instructions generated by the proposed algorithm are simple enough to be implemented in real-time after appropriate testing, with flexibility to allow different user-defined settings for deceleration rates, vehicle sizes, number of lanes, and intersection dimensions. While the algorithm may not generate the most usage-optimized solutions in terms of time-space negotiation, it does minimize – to zero communication – the interaction between vehicle 1 and vehicle 2. While not discussed here, this algorithm is easily extended to situations with multiple vehicles passing through the intersection by renegotiating the vehicle hierarchy for each new incoming vehicle. More specifically, this two-vehicle algorithm is simple enough to be readily extended to multi-vehicle implementations with proper adjustments.
MATLAB Codes for Collision Patterns

% Important note to remember before using this code: the order of M,N must
% be increasing where M < N
clc,clear

%% the inputs
roadN = 4; % modulo number, is N is the number of N-way intersection
M = 2; % number of original direction for vehicle A/vehicle 1
N = 3; % number of original direction for vehicle B/vehicle 2
Mf = 0; % number of intended direction for vehicle A
Nf = 1; % number of intended direction for vehicle B

%% setting up
N_1=N+1; % create this variable for later
M_1=M+1; % create this variable for later
N_1 = rem(N_1,roadN);  % constraint the variable within [0,roadN-1]
M_1 = rem(M_1,roadN);  % constraint the variable within [0,roadN-1]

%% collision pattern check
[range1] = colli_pat2(M,N_1,0,0,roadN); % create collision pattern range for the third row,
% second column in the collision pattern table
[range2] = colli_pat2(M,Mf,0,1,roadN); % create collision pattern range for the third row, third
% column
Mi_c2=find(range3==Mf);  % check if the intended direction for vehicle A is in range 1
Ni_c2=find(range4==Nf); % check if the intended direction for vehicle B is in range 2

%% collision pattern check
[range3] = colli_pat2(N_1,M,1,0,roadN);
[range4] = colli_pat2(Mf,M,1,1,roadN);
Mi_c2=find(range3==Mf);
Ni_c2=find(range4==Nf);

% check collision patterns
if (isempty(Mi_c2)==0) && (isempty(Ni_c2)==0) % if the intended direction for vehicle A and
B matches one of the collision pattern in the third row
    action = 1 % collision pattern formed
elseif (isempty(Mf_c)==0) && (isempty(Nf_c)==0) % if the intended direction for vehicle A
and B matches one of the collision pattern in the fourth row
    action = 2 % collision pattern formed
else
    action = 3 % collision pattern not formed
action = 1 % collision pattern formed
else
    action = 0 % no collision pattern
end

function [range] = colli_pat2(a,b,right,left,roadN)
    % creat the range first
    if a <= b % left and right bound are in order from small to large
        range = a:b;
    else % left bound is larger than right bound
        range = a:b+roadN ; % rescale smaller left bound and make a list
        range = rem(range,roadN); % rescale back within [0,roadN]
    end

    % include or exclude the first or end value, [], () or [], ()?
    if length(range)>1 % if there is more than one number in the range
        if (right ==0) && (left ==0) % both ends are not included
            range(1) = []; % range stays the same as its original one number
            range(end) =[];
        elseif right == 0 % if the right end is not included
            range(1) = []; % range stays the same as its original one number
            range(end) =[];
        elseif left == 0 % if the left end is not included
            range(1) = []; % range stays the same as its original one number
            range(end) =[];
        end
    else % if there is only one number in the range
        if (right ==0) && (left ==0) % if no ends are included
            range = []; % the range should be empty
        else
            % range stays the same as its original one number
        end
    end
end
MATLAB Codes for the Two-Vehicle Cooperative Driving Algorithm at A Four-Way Non-Signalized Intersection

clear
%% setting up direction, DON'T change, this is here so that users can directly enter x,y,-x,-y in initials parameters
x = 1; % direction
y = 2; % direction
ori = [x y -x -y];

%% initials parameters that can be changed
direction_i = [-x -y]; % original direction, follow the order in "ori" matrix, if x & -y, do [-y +x]
direction_f = [+x +y]; % intended direction, follow the order as well
vi = [10 10]; % m/s, initial velocity
position = [50 50]; % m, initial distance to intersection
lw = 4; % m, lane width
v_length = 4; % m, length of vehicles
vturning = 24; % kph, ~10mph, turning speed
vturning = vturning*10/36; % m/s
safety_zone = 0; % setting a safety range between vehicle1 and vehicle2
accel = 2; % m/s^2, generalized acceleration for all vehicles
a_turnprep1 = accel; % m/s^2, acceleration to slow down for turning
a_turnrecover = accel; % m/s^2, acceleration to recover to original speed after turning
inter_range_str = lw*2; % range of the intersection, if going straight

%% conversion
%vi = vi.*10./36; % m/s
M = find(ori==direction_i(1)); % index for original direction
N = find(ori==direction_i(2));
Mi = find(ori==direction_f(1)); % index for intended direction
Ni = find(ori==direction_f(2));
N_1=N+1; % create this variable for later
M_1=M+1; % create this variable for later
if N_1>4
N_1 = rem(N_1,4); % constraint the variable within [1,4]
end
if M_1>4
\[ M_1 = \text{rem}(M_1, 4); \] constraint the variable within \([1, 4]\)
\end

%%% time calculation
\begin{verbatim}
for i = 1:length(direction_i)
    diridx_i(i) = find(direction_i(i)==ori); % find the index of directions
    diridx_f(i) = find(direction_f(i)==ori);
end
t = abs(position)./vi; % calculate time for vehicles to arrive at the intersection
t_org = zeros(2);
t_turnrecover = zeros(1,2); % time it takes to recover to original speed after turning
s_turnrecover = zeros(1,2);
for i = 1:length(direction_f)
    w(i)=find(ori==direction_f(i)); % index of intended direction
    q(i)=find(ori==direction_i(i)); % index of initial direction
    % count for slowing down period for turning
    if abs(direction_f(i)) ~= abs(direction_i(i)) % turning
        s = abs(position(i)); % distance to intersection
        t_turnprep = (vi(i)-vturning)/a_turnprep1; % s, time period of slowing down for turning
        s_turnprep = vi(i)*t_turnprep-0.5*a_turnprep1*t_turnprep^2; %m, slowing down distance
        t1 = (s-s_turnprep)/vi(i); % s, time period for constant speed travelling
        t_org(i,1:2) = [t1 t_turnprep] % s, time organization till intersection, vehicle has constant speed during t_org(i,1) seconds, vehicle slows down during t_org(i,2)
        if t_org(i,1) <=0 % if deceleration is so small that the vehicle does not have enough time to slow down for turning
            disp('please pick another acceleration value, because deceleration is so small that the vehicle does not have enough time to slow down for turning')
        end
        trtt = t1+t_turnprep; % s, travelling total time to reach intersection
        t(i) = trtt;
        t_turnrecover(i) = (vi(i)-vturning)/a_turnrecover;
        s_turnrecover(i) = vturning*t_turnrecover(i)+0.5*a_turnrecover*t_turnrecover(i)^2;
    else  % no turning
        t_org(i,1) = t(i) % update for time organization for the noun turning vehicle, only has t_org(i,1), time for constant speed
        % t_org(i,2)== 0 meaning no slowing down
    end
end
% find out who goes first
a = find(t==min(t)); % faster vehicle index
if length(a)==2
    a=1;
end
if a==1
    a2=2; % slower vehicle index
else
a2=1;

end

% to calculate time for passing the intersection (from one stop sign to another stop sign)
q1=q+1; % right side of initial direction
q_1=q-1; % left side of initial direction

s_tr_temp = s_turnrecover;

for i = 1:length(w)
    % keep all numbers representing directions in [1,4]
    if q1(i)>=5;
        q1(i)=q(i)+1-4;
    elseif q_1(i)==0;
        q_1(i)=q(i)-1+4;
    end
    if q1(i)==w(i);
        tpass = pi*(lw/2)/2/vturning ; % small turn
        texd = v_length/vturning; % time for the end of vehicle to exist intersection, vehicle length is assumed to be 4 meters
    elseif q_1(i)==w(i);
        tpass = pi*(3*lw/2)/2/vturning ; % large turn
        texd = v_length/vturning;
    else
        tpass = (inter_range_str)/vi(i);
        texd = v_length/vi(i);
    end
    t_pass(i)=tpass; % time period inside the intersection
    t_exd(i) = texd; % time for the end of vehicle to exist intersection
    t_end(i) = t_pass(i)+t(i) ; % total time to exit the intersection
end

firstv_t = t(a)+t_pass(a); % time for the fastest vehicle to pass through the intersection

if length(vi) == 1  % if there is only one vehicle
    % all final velocity and time still the same
    vf = vi;
    t_new = t;
else
    if t_others > firstv_t % 1st situation, time shorter
        % all final velocity and time still the same
        vf = vi ;
        t_new(a) = t_others;
        t_new(a2) = t(a);
    else % collision pattern check, this assumes its an four-way intersection with no modulo input
disp('check pattern')
[range1] = colli_pat(M,N_1,0,0); % create collision pattern range for the third row, second column in the collision pattern table
[range2] = colli_pat(M,Mi,0,1); % create collision pattern range for the third row, third column
Mi_c = find(range1 == Mi); % check if the intended direction for vehicle A is in range 1
Ni_c = find(range2 == Ni); % check if the intended direction for vehicle B is in range 2

% repeat the last four lines for the fourth row in the collision pattern table
[range3] = colli_pat(N_1,M,1,0); % create collision pattern range
[range4] = colli_pat(Mi,M,1,1);
Mi_c2 = find(range3 == Mi);
Ni_c2 = find(range4 == Ni);

% check collision patterns
if (isempty(Mi_c2) == 0) && (isempty(Ni_c2) == 0) % if the intended direction for vehicle A and B matches one of the collision pattern in the third row
    action = 1 % collision pattern formed
elseif (isempty(Mi_c) == 0) && (isempty(Ni_c) == 0) % if the intended direction for vehicle A and B matches one of the collision pattern in the fourth row
    action = 1 % collision pattern formed
else
    action = 0 % no collision pattern formed
    % all final velocity and time still the same
    vf = vi;
    t_new(a2) = t_others;
    t_new(a) = t(a);
    end
end

%%% change acceleration
t_accel = zeros(1,2);
t_cons = zeros(1,2);
t accel_turn = zeros(1,2);
t_cons_turn = zeros(1,2);
t_adjust = firstv_t + safety_zone + t_exd(a); % the time assign for the slower vehicle so that when it can avoid meeting the faster vehicle at the intersection
t_turning_adjust = [];
if action == 1
    if abs(direction_f(a2)) == abs(direction_i(a2)) % no turning involved
        syms t_accel
        t_cons = t_adjust - 2*t_accel; % constant velocity time period
        t_accel = solve(2*vi(a2)*t_accel - accel*t_accel^2 + vi(a2)*t_cons - accel*t_accel*t_cons == position(a2), t_accel);
        t_accel = double(t_accel);
        t_cons = t_adjust - 2*t_accel; % get numerical constant velocity time period
    end
end
f = find(t_accel==min(t_accel)); % find the minimum time period for deceleration that will not return a negative constant velocity time period
    t_accel = t_accel(f);
    t_cons = t_cons(f);
    taccel(a2)=t_accel % final matrix for deceleration time periods of each vehicle that's going straight
    tcons(a2)=t_cons % final matrix for constant velocity time periods of each vehicle that's going straight
else % vehicle is turning
    syms t2 % initial slow down period
    t4 = (vi(a2)-accel*t2 - vturning) /a_turnprep1; % slow down period before turning
    t3 = t_adjust - t2 - t4; % constant period
    ssum = vi(a2)*t2-0.5*accel*t2^2+(vi(a2)-accel*t2)*t3+(vi(a2)-accel*t2)*t4-0.5*a_turnprep1*t4^2; % total distance
    t2 = solve(ssum==s,t2); % setting total distance to actual distance till intersection to solve for initial slowing down time period
    t2 = double(t2);
    t4 = (vi(a2)-accel*t2 - vturning) /a_turnprep1; % slowing down time period to reach turning speed
    t3 = t_adjust - t2 - t4; % going constant time period
    f = find(t3>0);
    t_turning_adjust = [t2(f) t3(f) t4(f)] % time periods for vehicle that's turning
    % the first variable is slowing down time period, the second is going constant time period, and the third is another slowing down time period to reach turning speed
end
    t_new(a2) = t_adjust; % new time to arrive at the intersection
    t_new(a) = t(a);
    t_end_new = t_new+t_pass; % updating exiting intersection time for the slowing down vehicle
end
% plotting starts from line 202, where you can adjust the limits of x,y axis on plots, as well as the font size
clc,clear
%% setting up, DON'T change, this is here so that users can directly enter x,y,-x,-y in initials
parameters
x = 1; % direction
y = 2; % direction
ori = [x y -x -y];

%% entering initials that can be changed
vi = [20 20]; % m/s, initial velocity
position = [200 200]; % m, initial distance to intersection
direction_i = [-x -y]; % follow the order in "ori" matrix
direction_f = [+x +y]; % intended direction, follow the order as well
lw = 4; % m, lane width
v_length = 4; % m, length of vehicles
vturning = 4; % m/s, turning speed
safety_zone = 0; % setting a safety range between vehicle1 and vehicle2
accel = 2; % m/s^2, generalized acceleration for all vehicles
a_turnprep1 = accel; % m/s^2, acceleration to slow down for turning
a_turnrecover = accel; % m/s^2, acceleration to recover to original speed after turning
inter_range_st = pi*(lw/2)/2; % range of the intersection, small turn
inter_range_lt = pi*(3*lw/2)/2; % range of the intersection, large turn
inter_range_str = lw*2; % range of the intersection, if going straight

%% unit conversion
%vi = vi.*(1609.34)./3600; % m/s
% vturning1 = vturning1*(1609.34)/3600; % m/s
% vturning2 = vturning2*(1609.34)/3600; % m/s

%% time calculation
t = abs(position)./vi; % calculate time for vehicles to arrive at the intersection
t_org = zeros(2);
t_turnrecover = zeros(1,2); % time it takes to recover to original speed after turning
s_turnrecover = zeros(1,2);
for i = 1:length(direction_f)
    w(i)=find(ori==direction_f(i)); % index of intended direction
q(i)=find(ori==direction_i(i)); % index of initial direction

% count for slowing down period for turning
if abs(direction_f(i))~=abs(direction_i(i)) % turning
    s = abs(position(i)); % distance to intersection
    t_turnprep = (vi(i)-vturning)/a_turnprep1; % s, time period of slowing down for turning
    s_turnprep = vi(i)*t_turnprep-0.5*a_turnprep1*t_turnprep^2; %m, slowing down distance
    t1 = (s-s_turnprep)/vi(i); % s, time period for constant speed travelling
    t_org(i,1:2) = [t1 t_turnprep] % s, time organization till intersection, vehicle has constant speed during t_org(i,1) seconds, vehicle slows down during t_org(i,2)
    if t_org(i,1) <=0 % if deceleration is so small that the vehicle does not have enough time to slow down for turning
        disp('please pick another acceleration value, because deceleration is so small that the vehicle does not have enough time to slow down for turning')
    end
    trtt = t1+t_turnprep; % s, travelling total time to reach intersection
    t(i) = trtt;
    t_turnrecover(i) = (vi(i)-vturning)/a_turnrecover;
    s_turnrecover(i) = vturning*t_turnrecover(i)+0.5*a_turnrecover*t_turnrecover(i)^2;
else % no turning
    t_org(i,1) = t(i) % update for time organization for the noun turning vehicle, only has t_org(i,1), time for constant speed
    % t_org(i,2)= 0 meaning no slowing down
    end
end

% find out who goes first
a = find(t==min(t)); % faster vehicle index
if length(a)==2
    a=1;
end
if a==1
    a2=2; % slower vehicle index
else
    a2=1;
end

% to calculate time for passing the intersection (from one stop sign to another stop sign)
q1=q+1; % right side of initial direction
q_1=q-1; % left side of initial direction
s_tr_temp = s_turnrecover;
for i = 1:length(w)
    % keep all numbers representing directions in [1,4]
    if q1(i)>5;
        q1(i)=q(i)+1-4;
    elseif q_1(i)==0;
        q_1(i)=q(i)-1+4;
    end
end
end
if q1(i)==w(i)
    tpass = pi*(lw/2)/vturning ; % small turn
    texd = v_length/vturning;
elseif q_1(i)==w(i)
    tpass = pi*(3*lw/2)/vturning ; % large turn
    texd = v_length/vturning;
else
    tpass = (inter_range_str)/vi(i);
    texd = v_length/vi(i);
end
    t_pass(i)=tpass;
    t_exd(i)=texd; % time for the end of vehicle to exist intersection
    t_end(i)=t_pass(i)+t(i); % total time to exit the intersection
end

firstv_t = t(a)+t_pass(a)+t_exd(a); % time for the fastest vehicle to pass through the intersection
t_others = t(a2); % time til intersection matrix for vehicles that are slower

%% pass without stop, check collision patterns, this is an older version of coding collision patterns, the newer and shorter one is in 3D time-space code
t_new = []; % new time to arrive at the intersection
t_end_new = t_end;
action = 0;
if length(vi) == 1
    vf = vi;
    t_new = t;
elseif t_others > firstv_t % 1st situation, time shorter
    vf = vi ;
    t_new(a2) = t_others;
    t_new(a) = t(a);
elseif abs(direction_i(1)) == abs(direction_i(2)) % 2nd situation part 1, intended directions do not intersect
    b = find(ori==direction_i(1));
    c = find(ori==direction_i(2));
    d = find(ori==direction_f(1));
    e = find(ori==direction_f(2));
    b2 = b+1;
    b3 = b+2;
    b4 = b+3;
    c2 = c+1;
    c3 = c+2;
    c4 = c-1;
    bc = [b2 b3 c2 c3 b4 c4];
    for i=1:length(bc)
        if bc(i)>4
bc(i) = bc(i) - 4;
end
end
if find(bc(1:2)==d)>0 & find(bc(3:4)==e)>0
    vf = vi;
    t_new(a2) = t_others;
    t_new(a) = t(a);
elseif bc(5)==d & bc(6)==e
    vf = vi;
    t_new(a2) = t_others;
    t_new(a) = t(a);
else
    action = 1;  % need to change acceleration
end
elseif abs(direction_i(1)) ~= abs(direction_i(2)) % 2nd situation part 2, intended directions do not intersect
    b = find(ori==direction_i(1));
    c = find(ori==direction_i(2));
    d = find(ori==direction_f(1));
    e = find(ori==direction_f(2));
    b2 = b+1;
    b3 = b+3;
    c2 = c+1;
    bc = [b2 b3 c2];
    for i=1:length(bc)
        if bc(i)>4
            bc(i) = bc(i) - 4;
        end
    end
if bc(1) == d
    vf = vi;
    t_new(a2) = t_others;
    t_new(a) = t(a);
elseif bc(2) == d & bc(3)==e
    vf = vi;
    t_new(a2) = t_others;
    t_new(a) = t(a);
else
    action = 1;  % need to change acceleration
end
end

%% change acceleration
taccel = zeros(1,2);
tcons = zeros(1,2);
taccel_turn = zeros(1,2);
tcons_turn = zeros(1,2);
t_adjust = firstv_t+safety_zone;
t_turning_adjust = [];
if action == 1 % if collision patterns match
    if abs(direction_f(a2)) == abs(direction_i(a2)) % no turning involved
        syms t_accel
        t_cons = t_adjust-2*t_accel; % constant velocity time period
        t_accel = solve(2*vi(a2)*t_accel-accel*t_accel^2+vi(a2)*t_cons-accel*t_accel*t_cons ==
        position(a2), t_accel); % solve to deceleration time period
        t_accel = double(t_accel);
        t_cons = t_adjust-2*t_accel;% get numerical constant velocity time period
        f = find(t_accel==min(t_accel)); % find the minimum time period for deceleration that will
        not return a negative constant velocity time period
        t_accel = t_accel(f);
        t_cons = t_cons(f);
        taccel(a2)=t_accel  % final matrix for deceleration time periods of each vehicle that's going
        straight
        tcons(a2)=t_cons  % final matrix for constant velocity time periods of each vehicle that's
        going straight
    else % vehicle is turning
        syms t2 % initial slow down period
        t4 = (vi(a2)-accel*t2 - vturning) /a_turnprep1; % slow down period before turning
        t3 = t_adjust - t2 - t4; % constant period
        ssum = vi(a2)*t2-0.5*accel*t2^2+(vi(a2)-accel*t2)*t3+(vi(a2)-accel*t2)*t4-
        0.5*a_turnprep1*t4^2; % total distance
        t2 = solve(ssum==position(a2),t2);% setting total distance to actual distance till intersection
        t2 = double(t2);
        t4 = (vi(a2)-accel*t2 - vturning) /a_turnprep1;
        t3 = t_adjust - t2 - t4;
        f = find(t3>0);
        t_turning_adjust = [t2(f) t3(f) t4(f)]; % time periods for vehicle that's turning
        % the first variable is slowing down time period, the second is going constant time period,
        and the third is another slowing down time period to reach turning speed
    end
    t_new(a2) = t_adjust; % new time to arrive at the intersection
    t_new(a) = t(a);
    t_end_new = t_new+t_pass; % updating exiting intersection time for the slowing down vehicle
end

%% time space diagram
% plot the range for an intersection
timelim = 25; % x-axis limit
time=[0:1:timelim]; % range of x-axis
intersection_start = time*0; % intersection starts at space=0
intersection_end = time*0-(inter_range_str+lw); % intersection ends at space = 3*lw
color = {{[0.6 0.6 0.9] [0.9 0.6 0.6]}}; % the colors for plotting original routes
if a == 1 % there is an bug for using the color for adjusted trajectories so that the lighter and
darker color match, creating this if statement is the fastest way to fix this bug
    color2 = {{[0 0 0.8] [1 0 0]}}; % the colors for plotting adjusted routes
elseif a==2
    color2 = {{[1 0 0] [0 0 0.8]}}; % the colors for plotting adjusted routes
end
linestyle = {'-',':'};
linew = 3; % plotting line width
FigHandle = figure('Position',[0,0,600,500]);
axis([0 timelim -100 100]);
%axis([5 20 -30 30]);
xlabel('time (s)');
ylabel({'distance to where','intersection starts (m)'})
% to adjust the label fonts
ax = gca;
ax.FontSize = 14;
ax.FontWeight = 'bold';
box on;
hold on
i1=plot(time,intersection_start,'-.','color','k','Linewidth',2);
hold on
i2=plot(time,intersection_end,'--','color','k','Linewidth',2);
hold on
order = [a a2];
% plot for initial routes
for i = 1:2
    if abs(direction_f(order(i))) == abs(direction_i(order(i))) % no turning
        space1 = 200-(vi(order(i)).*time + (200-position(order(i))));
        i3{i} = plot(time,space1,'LineStyle',linestyle{i},'color',color{i},'Linewidth',linew);
    else %if the vehicle is turning
        [ ft,fs1,ft2,fs2,ft3,fs3,ft4,fs4,ft5,fs5 ] =
        distance_steps(vi(order(i)),position(order(i)),time,t_org(order(i),1),t(order(i)),t_end(order(i)),t_end(order(i)),t_e
        xd(order(i)),vturning,a_turnprep1,a_turnrecover,t_turnrecover(order(i)));
        plot(ft,fs1,'LineStyle',linestyle{i},'color',color{i},'Linewidth',linew);
        hold on
        plot(ft2,fs2,'LineStyle',linestyle{i},'color',color{i},'Linewidth',linew);
        hold on
        plot(ft3,fs3,'LineStyle',linestyle{i},'color',color{i},'Linewidth',linew);
        hold on
        plot(ft4,fs4,'LineStyle',linestyle{i},'color',color{i},'Linewidth',linew);
        hold on
        plot(ft5,fs5,'LineStyle',linestyle{i},'color',color{i},'Linewidth',linew);
        hold on
        i3{i} = plot(ft5,fs5,'LineStyle',linestyle{i},'color',color{i},'Linewidth',linew);
    end
% plot the horizontal line for intersection exit distance
if q1(order(i))==w(order(i))
    inter_range = inter_range_st+v_length;  % small turn
    i9(order(i)) = plot(time,time*0-inter_range,'--o','Linewidth',2);
elseif q_1(order(i))==w(order(i))
    inter_range = inter_range_lt+v_length; % large turn
    i9(order(i)) = plot(time,time*0-inter_range,'k-o','Linewidth',2);
else
    inter_range = inter_range_str+lw;  % range of the intersection
    i9(order(i)) = 0;
end
hold on

% plot the box around the within intersection periods for both vehicles
box_h = [-inter_range:0.01:0];
box_p1= box_h*0 + t(order(i));
line{order(i)} = box_p1;
plot(box_p1,box_h,'color',color{i},'Linewidth',4)
hold on
box_len = t(order(i)):0.01:t_end(order(i))+t_exd(order(i));
box_p2 = 200-(box_len*0 + 200);
plot(box_len,box_p2,'color',color{i},'Linewidth',4)
hold on
box_p3 = box_h*0+t_end(order(i))+t_exd(order(i));
line2{order(i)} = box_p3;
plot(box_p3,box_h,'color',color{i},'Linewidth',4)
hold on
box_p4 = 200-(box_len*0+200+inter_range);
plot(box_len,box_p4,'color',color{i},'Linewidth',4)
hold on
end

% create the light shadowed area for two boxes overlaping, only when the
% two vehicles are going to physically crash
if 1==action
    templ = line{a2};
    templ2 = line2{a};
    f = min(length(templ),length(templ2));
    x2 = [templ(1:f),fliplr(templ2(1:f))];
    inBetween = [box_h(length(box_h)-f+1:end),fliplr(box_h(length(box_h)-f+1:end))];
    fillhandle = fill(x2,inBetween,'b');
    set(fillhandle,'EdgeAlpha',0.3,'FaceAlpha',0.3);
end

%
if i9(1) ~=0 && i9(2)==0  
    legend([i1 i2 i9(1) i3 {1} i3 {2} ],'intersection starts','intersection ends (straight)','intersection ends (large turn)','initial vehicle1','initial vehicle2','location','northeast');
elseif i9(2)~=0  && i9(1)==0  
    legend([i1 i2 i9(2) i3 {1} i3 {2} ],'intersection starts','intersection ends (straight)','intersection ends (small turn)','initial vehicle1','initial vehicle2','location','northeast');
elseif i9(2)~=0  && i9(1)~=0  
    legend([i1 i2 i9(2) i9(1) i3 {1} i3 {2} ],'intersection starts','intersection ends (straight)','intersection ends (small turn)','intersection ends (large turn)','initial vehicle1','initial vehicle2','location','northeast');
else  
    legend([i1 i2 i3 {1} i3 {2} ],'intersection starts','intersection ends (straight)','initial vehicle1','initial vehicle2','location','northeast');
end

% uncomment the next nuclear comment sign if needs to plot the adjusted  
% movements

% plot for adjusted movements  
if 1==action  
% if a conflict exsits, and one of the vehicles need to change speed  
    % first plot the trajectory for the vehicle that maintain its original speed  
    if abs(direction_f(a)) == abs(direction_i(a))  
        space2 = 200-(vi(a)*time + (200-position(a)));  
        i5 = plot(time,space2,'color',color2{a},'Linewidth',linew,'LineStyle',linestyle{a});  
        hold on  
    else  
        [ ft,fs1,fs2,ft3,fs3,ft4,fs4,ft5,fs5 ] =  
            distance_steps(vi(a),position(a),time,t_org(a,1),t(a),t_end_new(a),t_exd(a),vturning,a_turnprep1,  
                a_turnrecover,t_turnrecover(a));  
        plot(ft,fs1,'color',color2{a},'Linewidth',linew,'LineStyle',linestyle{a});  
        hold on  
        plot(ft2,fs2,'color',color2{a},'Linewidth',linew,'LineStyle',linestyle{a});  
        hold on  
        plot(ft3,fs3,'color',color2{a},'Linewidth',linew,'LineStyle',linestyle{a});  
        hold on  
        plot(ft4,fs4,'color',color2{a},'Linewidth',linew,'LineStyle',linestyle{a});  
        hold on  
        plot(ft5,fs5,'color',color2{a},'Linewidth',linew,'LineStyle',linestyle{a});  
        hold on  
        i5 = plot(ft5,fs5,'color',color2{a},'Linewidth',linew,'LineStyle',linestyle{a});  
        hold on  
    end
end
% then plot the trajectory for the vehicle that yields
if abs(direction_f(a2)) == abs(direction_i(a2))  % if the vehicle going straight
    time_accel = [0:0.0001:round(taccel(a2),4)]; % deacceleration time period
    space2_accel = 200 - (vi(a2) .* time_accel - 0.5 * accel .* time_accel.^2 + (200 - position(a2))) ; % travelling with deacceleration
    vtemp = vi(a2) - accel * time_accel; % decreased velocity
    ttemp = time_accel + tcons(a2); % time to start acceleration
    time_cons = [round(taccel(a2),4):0.0001:round(ttemp,4)]; % constant velocity time period
    space2_cons = space2_accel(end) - vtemp .* (time_cons - time_accel); % travelling with constant velocity
    time_accelp = [round(ttemp,4):0.0001:round(2 * time_accel + tcons(a2),4)]; % acceleration time period
    space2_accelp = space2_cons(end) - vtemp .* (time_accelp - ttemp) + 0.5 * accel .* (time_accelp - ttemp).^2; % travelling with acceleration
    vtemp2 = vtemp + accel * time_accel; % final velocity
    time_rest = [round(2 * time_accel + tcons(a2),4):0.0001:timelim];
    space2_rest = 0 - vtemp2 .* (time_rest - t_new(a2)); % travelling with initial speed
else  % if the vehicle is turning
    [tt,ts1,tt2,ts2,tt3,ts3,tt4,ts4,tt5,ts5,tt6,ts6] = turning_steps(vi(a2),position(a2),time,t_turning_adjust(1),t_turning_adjust(2),t_turning_adjust(3),t_pass(a2),t_exd(a2),accel,vturning,a_turnprep1,a_turnrecover,t_turnrecover(a2));
    plot(tt,ts1,'color',color2{a2},'Linewidth',2,'LineStyle',linestyle{a2});
    hold on
    plot(tt2,ts2,'color',color2{a2},'Linewidth',2,'LineStyle',linestyle{a2});
    hold on
    plot(tt3,ts3,'color',color2{a2},'Linewidth',2,'LineStyle',linestyle{a2});
    hold on
    plot(tt4,ts4,'color',color2{a2},'Linewidth',2,'LineStyle',linestyle{a2});
    hold on
    plot(tt5,ts5,'color',color2{a2},'Linewidth',2,'LineStyle',linestyle{a2});
    hold on
    else  % if the vehicle is turning
        [tt,ts1,tt2,ts2,tt3,ts3,tt4,ts4,tt5,tt6] = turning_steps(vi(a2),position(a2),time,t_turning_adjust(1),t_turning_adjust(2),t_turning_adjust(3),t_pass(a2),t_exd(a2),accel,vturning,a_turnprep1,a_turnrecover,t_turnrecover(a2));
    plot(tt,ts1,'color',color2{a2},'Linewidth',2,'LineStyle',linestyle{a2});
    hold on
    plot(tt2,ts2,'color',color2{a2},'Linewidth',2,'LineStyle',linestyle{a2});
    hold on
    plot(tt3,ts3,'color',color2{a2},'Linewidth',2,'LineStyle',linestyle{a2});
    hold on
    plot(tt4,ts4,'color',color2{a2},'Linewidth',2,'LineStyle',linestyle{a2});
    hold on
    plot(tt5,ts5,'color',color2{a2},'Linewidth',2,'LineStyle',linestyle{a2});
    hold on
end
i4 = plot(tt6,ts6,'color',color2{a2},'Linewidth',2, 'LineStyle',linestyle{a2});
hold on
end

% plot for the box
for i = 1:2
  if q1(order(i))==w(order(i))
    inter_range = inter_range_st+v_length; % small turn
    i9(i) = plot(time,time*0-inter_range,'--o','Linewidth',2);
  elseif q_1(order(i))==w(order(i))
    inter_range = inter_range_lt+v_length; % large turn
    i9(i) = plot(time,time*0-inter_range,'k-o','Linewidth',2);
  else
    inter_range = inter_range_str+lw; % range of the intersection
    i9(i) = 0;
  end

  box_h = [-inter_range:0.01:0];
  box_p1= box_h*0 + t_new(order(i));
  plot(box_p1,box_h,'color',color2{order(i)},'Linewidth',2)
  hold on
  box_len = t_new(order(i)):0.01:t_end_new(order(i))+t_exd(order(i));
  box_p2 = box_len*0;
  plot(box_len,box_p2,'color',color2{order(i)},'Linewidth',2)
  hold on
  box_p3 = box_h*0+t_end_new(order(i))+t_exd(order(i));
  plot(box_p3,box_h,'color',color2{order(i)},'Linewidth',2)
  hold on
  box_p4 = 200-(box_len*0 + 200+inter_range);
  plot(box_len,box_p4,'color',color2{order(i)},'Linewidth',2)
  hold on
end

if i9(1) ~=0 && i9(2)==0
  legend([i1 i2 i9(1) i3{1} i3{2} i5 i4],'intersection starts','intersection ends (straight)','intersection ends (large turn)','initial vehicle1','initial vehicle2','vehicle that maintains its original speed', 'yielding vehicle','location','northeast');
elseif i9(2)==0 && i9(1)==0
  legend([i1 i2 i9(2) i3{1} i3{2} i5 i4],'intersection starts','intersection ends (straight)','intersection ends (small turn)','initial vehicle1','initial vehicle2','vehicle that maintains its original speed', 'yielding vehicle','location','northeast');
else i9(2)==0 && i9(1)==0
legend([i1 i2 i9(2) i3 {1} i3 {2} i5 i4],'intersection starts','intersection ends (straight)','intersection ends (small turn)','intersection ends (large turn)','initial vehicle1','initial vehicle2','vehicle that maintains its original speed', 'yielding vehicle','location','northeast');
else
legend([i1 i2 i3 {1} i3 {2} i5 i4],'intersection starts','intersection ends (straight)','initial vehicle1','initial vehicle2','vehicle that maintains its original speed', 'yielding vehicle','location','northeast');
end
else % plot when two vehicles remain their initial speeds
    for i = 1:2
        if abs(direction_f(i)) == abs(direction_i(i)) % no turning
            space1 = 200-(vi(i).*time + (200-position(i)));
i3{i}=plot(time,space1,'color',color2{i},'Linewidth',linew);
else
[ ft,fs1,ft2,fs2,ft3,fs3,ft4,fs4,ft5,fs5 ] = distance_steps(vi(i),position(i),time,t_org(i,1),t(i),t_end(i),t_exd(i),vturning,a_turnprep1,a_turnrecover,t_turnrecover(i));
        plot(ft,fs1,'color',color2{i},'Linewidth',linew, 'LineStyle',linestyle{i});
        hold on
        plot(ft2,fs2,'color',color2{i},'Linewidth',linew,'LineStyle',linestyle{i});
        hold on
        plot(ft3,fs3,'color',color2{i},'Linewidth',linew,'LineStyle',linestyle{i});
        hold on
        plot(ft4,fs4,'color',color2{i},'Linewidth',linew,'LineStyle',linestyle{i});
        hold on
        plot(ft5,fs5,'color',color2{i},'Linewidth',linew,'LineStyle',linestyle{i});
        hold on
i3{i} = plot(ft5,fs5,'color',color2{i},'Linewidth',linew,'LineStyle',linestyle{i});
        hold on
    end
if q1(i)==w(i)
    inter_range = inter_range_st+v_length;  % small turn
i9(i) = plot(time,time*0-inter_range,'--o','Linewidth',2);
elseif q_1(i)==w(i)
    inter_range = inter_range_lt+v_length; % large turn
i9(i) = plot(time,time*0-inter_range,'k-o','Linewidth',2);
else
    inter_range = inter_range_str+lw;  % range of the intersection
i9(i) = 0;
end
box_h = -inter_range:0.01:0;
box_p1= box_h*0 + t(i);
plot(box_p1,box_h,'color',color2{i},'Linewidth',2)
hold on
box_len = t(i):0.01:t_end(i)+t_exd(i);
box_p2 = 200-(box_len*0 + 200);
plot(box_len,box_p2,'color',color2{i},'Linewidth',2) hold on
box_p3 = box_h*0+t_end(i)+t_exd(i);
plot(box_p3,box_h,'color',color2{i},'Linewidth',2) hold on
box_p4 = 200-(box_len*0+200+inter_range);
plot(box_len,box_p4,'color',color2{i},'Linewidth',2) hold on
end
if i9(1) ~=0 && i9(2)==0
    legend([i1 i2 i9(1) i3{1} i3{2} ],'intersection starts','intersection ends (straight)','intersection ends (large turn)','initial & current vehicle1','initial & current vehicle2','location','northeast');
elseif i9(2)~=0  && i9(1)==0
    legend([i1 i2 i9(2) i3{1} i3{2}],'intersection starts','intersection ends (straight)','intersection ends (small turn)','initial & current vehicle1','initial & current vehicle2','location','northeast');
elseif i9(2)~=0  && i9(1)~=0
    legend([i1 i2 i9(2) i9(1) i3{1} i3{2}],'intersection starts','intersection ends (straight)','intersection ends (small turn)','intersection ends (large turn)','initial & current vehicle1','initial & current vehicle2','location','northeast');
else
    legend([i1 i2 i3{1} i3{2}],'intersection starts','intersection ends (straight)','initial & current vehicle1','initial & current vehicle2','location','northeast');
end
end

function [ ft,fs1,ft2,fs2,ft3,fs3,ft4,fs4,ft5,fs5 ] = distance_steps(v,position,time,t2,t3,t_end,t_exd,vturning,a_prep,a_re,t_turnrecover)
%UNTITLED2 Summary of this function goes here

% travel at constant speed
  ft = [0:0.0001:t2];
  fs1 = 200-(v.*ft + (200-position));

% slowing down period for turning
  ft2 = [t2:0.0001:t3];
  fs2 = fs1(end) - (v.*(ft2-t2)-0.5*a_prep.*(ft2-t2).^2);

% constant turning period
  ft3 = [t3:0.0001:(t_end+t_exd)];
  fs3 = fs2(end) - vturning.*(ft3-t3);

% recovering period to its original speed
  ft4 = [t_end+t_exd:0.0001:t_end+t_exd+t_turnrecover];
fs4 = fs3(end)-(vturning.*(ft4-t_end-t_exd)+0.5*a_re.*(ft4-t_end-t_exd).^2);

ft5 = [t_end+t_exd+t_turnrecover:0.0001:time(end)]; % constant travelling
fs5 = fs4(end) - v.*(ft5-t_end-t_turnrecover-t_exd);

end

function [ tt,ts1,tt2,ts2,tt3,ts3,tt4,ts4,tt5,ts5,tt6,ts6 ] = 
turning_steps(v,position,time,t1,tcons,tde,t_passing,texd,accel,vturning,a_prep,a_re,t_turnrecover)

%UNTITLED3 Summary of this function goes here
% Detailed explanation goes here

 tt = [0:0.0001:t1]; % deacceleration time period t1 = t_turning_adjust(1)
ts1 =200 -(v.*tt-0.5*accel.*tt.^2+(200-position)); % travelling with deacceleration
vtemp = v-accel*t1; % decreased velocity
ttemp = t1 + tcons;

 tt2 = [t1:0.0001:ttemp]; % constant velocity time period, ttemp = t_turning_adjust(1+2)
ts2 =ts1(end) - vtemp*(tt2-t1); % travelling with constant velocity

 t2 = ttemp + tde;
 tt3 = [ttemp:0.0001:t2]; % deacceleration time period t2 = t_turning_adjust(1+2+3)
ts3 =ts2(end)- (vtemp.*(tt3-ttemp)-0.5*a_prep.*(tt3-ttemp).^2); %travelling with deacceleration

 t3 = t2+t_passing+texd;
 tt4 = [t2:0.0001:t3]; % constant turning velocity t3 = t2 +t_passing
 ts4 = ts3(end) - vturning.*(tt4-t2); % t_passing is t_pass(a2)

 t4 = t3 + t_turnrecover;
 tt5 = [t3:0.0001:t4]; % recovering period
 ts5 = ts4(end) - (vturning.*(tt5-t3)+0.5*a_re.*(tt5-t3).^2);

 tt6 = [t4:0.0001:time(end)];
 ts6 = ts5(end) - v.*(tt6-t4); % travelling with original speed

end
MATLAB Codes for Generating 3-D Time-Space Diagrams

clc, clear
%% setting up direction, DON'T change, this is here so that users can directly enter x,y,-x,-y in
initials parameters
x = 1; % direction
y = 2; % direction
ori = [x y -x -y];

%% initials parameters that can be changed
position = [50 50]; % m, initial distance to intersection
lw = 4; % m, lane width
v_length = 4; % m, length of vehicles
vturning = 24; % kph, ~10mph, turning speed
vturning = vturning*10/36; % m/s
safety_zone = 0; % setting a safety range between vehicle1 and vehicle2
accel = 2; % m/s^2, generalized acceleration for all vehicles
a_turnprep1 = 2; % m^2/s, acceleration to slow down for turning
a_turnrecover = 2; % m^2/s, acceleration to recover to original speed after turning
inter_range_str = lw*2; % range of the intersection, if going straight

%% User input
direction_i = [-x -y]; % original direction, follow the order in "ori" matrix, if x & -y, do [-y +x]
direction_f = [-y +x]; % intended direction, follow the order as well
vi = [15 10]; % m/s, initial velocity

%% conversion
%vi = vi.*10./36; % m/s
M = find(ori==direction_i(1)); % index for original direction
N = find(ori==direction_i(2));
Mi = find(ori==direction_f(1)); % index for intended direction
Ni = find(ori==direction_f(2));
N_1=N+1; % create this variable for later
M_1=M+1; % create this variable for later
if N_1>4
N_1 = rem(N_1,4); % constraint the variable within [1,4]
end
if $M_1 > 4$
$M_1 = \text{rem}(M_1, 4)$;  % constraint the variable within [1,4]
end

%% time calculation
for $i = 1:\text{length}(\text{direction}_i)$
    diridx$_i(i) = \text{find}(\text{direction}_i(i)==\text{ori});$  % find the index of directions
    diridx$_f(i) = \text{find}(\text{direction}_f(i)==\text{ori});$
end
t = \text{abs(position).}/\text{vi}$;  % calculate time for vehicles to arrive at the intersection
\text{t}_\text{org} = \text{zeros}(2);
\text{t\_turnrecover} = \text{zeros}(1,2);$  % time it takes to recover to original speed after turning
\text{s\_turnrecover} = \text{zeros}(1,2);
for $i = 1:\text{length}(\text{direction}_f)$
    w(i)=\text{find}(\text{ori}\text{=}\text{direction}_f(i));  % index of intended direction
    q(i)=\text{find}(\text{ori}\text{=}\text{direction}_i(i));  % index of initial direction
% count for slowing down period for turning
    \text{if abs(\text{direction}_f(i))==abs(\text{direction}_i(i)) % turning
    \text{s} = \text{abs(position}(i));  % distance to intersection
    \text{t\_turnprep} = (\text{vi}(i)-\text{vturning})/\text{a\_turnprep1};  % s, time period of slowing down for turning
    \text{s\_turnprep} = \text{vi}(i)*\text{t\_turnprep}-0.5*\text{a\_turnprep1}\text{t\_turnprep}^2;  %m, slowing down distance
    \text{t1} = (\text{s}-\text{s\_turnprep})/\text{vi}(i);  % s, time period for constant speed travelling
    \text{t\_org}(i,1:2) = [\text{t1} \text{t\_turnprep}];  % s, time organization till intersection, vehicle has constant speed during t\_org(i,1) seconds, vehicle slows down during t\_org(i,2)
\text{if t\_org}(i,1) <=0  % if deceleration is so small that the vehicle does not have enough time to slow down for turning
    \text{disp('please pick another acceleration value, because deceleration is so small that the vehicle does not have enough time to slow down for turning')}
end
    \text{trtt} = \text{t1}+\text{t\_turnprep};  % s, travelling total time to reach intersection
    \text{t(i)} = \text{trtt};
    \text{t\_turnrecover}(i) = (\text{vi}(i)-\text{vturning})/\text{a\_turnrecover};
    \text{s\_turnrecover}(i) = \text{vturning}*\text{t\_turnrecover}(i)+0.5*\text{a\_turnrecover}\text{t\_turnrecover}(i)^2;
\text{else}  % no turning
    \text{t\_org}(i,1) = \text{t(i)};  % update for time organization for the noun turning vehicle, only has t\_org(i,1), time for constant speed
\text{if t\_org}(i,1) = 0  % meaning no slowing down
end
end
% find out who goes first
\text{a} = \text{find}(\text{t==min(t)});  % faster vehicle index
\text{if length(a)==2}
    \text{a}=1;
end
\text{if a==1}
    \text{a2}=2;  % slower vehicle index
else
    a2=1;
end

% to calculate time for passing the intersection (from one stop sign to another stop sign)
q1=q+1; % right side of initial direction
q_1=q-1; % left side of initial direction
s_tr_temp = s_turnrecover;
for i = 1:length(w)
    % keep all numbers representing directions in [1,4]
    if q1(i)>=5;
        q1(i)=q(i)+1-4;
    elseif q_1(i)==0;
        q_1(i)=q(i)-1+4;
    end
    if q1(i)==w(i);
        tpass = pi*(lw/2)/2/vturning ; % small turn
        texd = v_length/vturning; % time for the end of vehicle to exist intersection, vehicle length
        is assumed to be 4 meters
    elseif q_1(i)==w(i);
        tpass = pi*(3*lw/2)/2/vturning ; % large turn
        texd = v_length/vturning;
    else
        tpass = (inter_range_str)/vi(i);
        texd = v_length/vi(i);
    end
    t_pass(i)=tpass; % time period inside the intersection
    t_exd(i) = texd; % time for the end of vehicle to exist intersection
    t_end(i) = t_pass(i)+t(i) ; % total time to exit the intersection
end

firstv_t = t(a)+t_pass(a); % time for the fastest vehicle to pass through the intersection

% pass without stop, check directions
if length(vi) == 1  % if there is only one vehicle
    % all final velocity and time still the same
    vf = vi;
    t_new = t;
elseif t_others > firstv_t % 1st situation, time shorter
    % all final velocity and time still the same
    vf = vi ;
    t_new(a2) = t_others;
    t_new(a) = t(a);
else  % collision pattern check, this assumes its an four-way intersection with no modulo input
    disp('check pattern')
    [range1] = colli_pat(M,N_1,0,0); % create collision pattern range for the third row, second
    column in the collision pattern table
    [range2] = colli_pat(M,Mi,0,1); % create collision pattern range for the third row, third
    column
    Mi_c = find(range1==Mi); % check if the intended direction for vehicle A is in range 1
    Ni_c = find(range2==Ni); % check if the intended direction for vehicle B is in range 2

    % repeat the last four lines for the fourth row in the collision pattern table
    [range3] = colli_pat(N_1,M,1,0); % create collision pattern range
    [range4] = colli_pat(Mi,M,1,1);
    Mi_c2 = find(range3==Mi);
    Ni_c2 = find(range4==Ni);

    % check collision patterns
    if (isempty(Mi_c2)==0) & (isempty(Ni_c2)==0) % if the intended direction for vehicle A
        action = 1 % collision pattern formed
        elseif (isempty(Mi_c)==0) & (isempty(Ni_c)==0) % if the intended direction for vehicle A
            action = 1 % collision pattern formed
        else
            action = 0 % no collision pattern formed

    end

    %% change acceleration
    taccel = zeros(1,2);
    tcons = zeros(1,2);
    taccel_turn = zeros(1,2);
    tcons_turn = zeros(1,2);
    t_adjust = firstv_t+safety_zone+t_exd(a); % the time assign for the slower vehicle so that when
    it can avoid meeting the faster vehicle at the intersection
    t_turning_adjust = [];
    if action == 1
        if abs(direction_f(a2)) == abs(direction_i(a2)) % no turning invloved
            syms t_acc
            t_cons = t_adjust-2*t_acc; % constant velocity time period
            t_acc = solve(2*vi(a2)*t_acc-accel*t_acc^2+vi(a2)*t_cons-accel*t_acc*t_cons ==
            position(a2), t_acc);
            t_acc = double(t_acc);
t_cons = t_adjust-2*t_accel;  % get numerical constant velocity time period
f = find(t_accel==min(t_accel));  % find the minimum time period for acceleration
    t_accel = t_accel(f);
    t_cons = t_cons(f);
    taccel(a2)=t_accel  % final matrix for acceleration time periods of each vehicle that's going
straight
    tcons(a2)=t_cons  % final matrix for constant velocity time periods of each vehicle that's
going straight
else  % vehicle is turning
    syms t2  % initial slow down period
    t4 = (vi(a2)-accel*t2 - vturning) /a_turnprep1;  % slow down period before turning
    t3 = t_adjust - t2 - t4;  % constant period
    ssum = vi(a2)*t2-0.5*accel*t2^2+(vi(a2)-accel*t2)*t3+(vi(a2)-accel*t2)*t4-0.5*a_turnprep1*t4^2;  % total distance
    t2 = solve(ssum==s,t2);  % setting total distance to actual distance till intersection to solve
    for initial slowing down time period
        t2 = double(t2);
        t4 = (vi(a2)-accel*t2 - vturning) /a_turnprep1;  % slowing down time period to reach
        turning speed
        t3 = t_adjust - t2 - t4;  % going constant time period
        f = find(t3>0);
        t_turning_adjust = [t2(f) t3(f) t4(f)]  % time periods for vehicle that's turning
        and the third is another slowing down time period to reach turning speed
    end
    t_new(a2) = t_adjust;  % new time to arrive at the intersection
    t_new(a) = t(a);
    t_end_new = t_new+t_pass;  % updating exiting intersection time for the slowing down vehicle
end

%% 3D time space animation
color = {[0.9 0.6 0.6] [0.6 0.6 0.9]} ;  % color for original trajectories
color2 = {[1 0 0] [0 0 0.8]};  % color for adjusted trajectories
linewidth = 8;  % linewidth in the plot (pretending it's the vehicle width)
linewidth2 = 8;
timelim = 40;  % z-axis limit
    time=0:0.1:timelim;  % time matrix
time2 = 0:0.0001:timelim;  % larger time matrix

% matrix for plotting roads
road_x1 =[-104:0.1:104];
road_y1 = road_x1*0-4;
road_y2 = road_x1*0+4;
time_p1 = road_x1*0;
% ensure plot size and location
FigHandle = figure('Position',[100,100,900,800]);
axis([-204 204 -204 204 0 timelim]);
xlabel('x (m)')
ylabel('y (m)')
zlabel('time (s)')
ax = gca;
ax.FontSize = 22;
ax FontWeight = 'bold';
box on;
hold on

% plot roads
plot3(road_x1,road_y1,time_p1,'y',road_x1,road_y2,time_p1,'y','Linewidth',2);
hold on
plot3(road_y1,road_x1,time_p1,'y',road_y2,road_x1,time_p1,'y','Linewidth',2);
hold on

% plot origial routes

% some direction index
d_i = [M N]; % initial directions
d_f = [Mi Ni]; % final directions
N_m1 = N-1;
M_m1 = M-1;
if N_m1 ==0
    N_m1 = 4;
elseif M_m1==0
    M_m1 = 4;
end
d_i = [M_m1 N_m1];
d_i = [M_1 N_1];

% find the initial position
p_i_mtx = [];
dir_sn_mtx1 = [];
for i = 1:length(d_i)
    if (d_i(i)==3) || (d_i(i) ==4)% original direction at -y,-x
        p_i = -position(i)-4; % initial position
        dir_sign1 = 1;
    elseif (d_i(i)==1) || (d_i(i) ==2)% original direction at y,x
        p_i = position(i)+4; % initial position
        dir_sign1 = -1;
end
p_i_mtx = [p_i_mtx p_i];
dir_sn_mtx1 = [dir_sn_mtx1 dir_sign1];
end

% different initial directions cause variation in +y,-y,+x,-x, find the
% signs
dir_sn_mtx2 = [];
for i= 1:length(d_i)
    if (d_i(i)==1) || (d_i(i) ==4)
        dir5 = 1;
    elseif (d_i(i)==2) || (d_i(i) ==3)
        dir5 = -1;
    end
    dir_sn_mtx2 = [dir_sn_mtx2 dir5]
end
radius = zeros(1,2);
dir = zeros(1,2);
dir2 = zeros(1,2);
dir3 = zeros(1,2);

% plot for vehicle 3
for i = 1:length(d_i)
    if abs(direction_f(i)) ~= abs(direction_i(i))  % turning
        % different initial directions cause variation in +y,-y,+x,-x,while travelling
        % this step is to find the signs to make plotting look right
        if d_f(i) == di_m1(i) % making large turn
            radius(i) =3*lw/2;
            if (d_i(i) ==3)
                dir(i) =1;
                dir2(i) = -1;
                dir3(i) = 1;
            elseif (d_i(i) ==2)
                dir(i) =1;
                dir2(i) = -1;
                dir3(i) = -1;
            elseif (d_i(i) ==4)
                dir(i) =-1;
                dir2(i) = 1;
                dir3(i) = 1;
            elseif (d_i(i) ==1)
                dir(i) = -1;
            end
        end
    end
dir2(i) = 1;
dir3(i) = -1;
end
elseif d_f(i) == di_1(i)  % making small turn
    radius(i) = lw/2;
    if (d_i(i) ==3)
        dir(i) =-1;
        dir2(i) = -1;
        dir3(i) = -1;
    elseif (d_i(i) ==2)
        dir(i) =-1;
        dir2(i) = -1;
        dir3(i) = 1;
    elseif (d_i(i) ==4)
        dir(i) =1;
        dir2(i) = 1;
        dir3(i) = -1;
    elseif (d_i(i)==1)
        dir(i) =1;
        dir2(i) = 1;
        dir3(i) = 1;
    end
end

% get data points by calling the function turnpath2
[cont_p3,route_y_p3,route_x_p3,turnprep_p3,route_y2_p3,route_x2_p3,turning_p3,route_y3_p3,route_x3_p3,turnprep2_p3,route_y2_p3,route_x2_p3,turning2_p3,route_y3_p3,route_x3_p3,rest_p3,route_y5_p3,route_x5_p3,temp3] =
    turnpath2(p_i_mtx(i),vi(i),t_org(i,1),t_org(i,2),t_pass(i),t_turnrecover(i),timelim,vturning,a_turn
    prep1,a_turnrecover,d(i),dir2(i),dir3(i),radius(i),dir_sn_mtx1(i));

% create the master time and x,y coordinate matrix
    ttemp1 = cont_p3;  % time period of constant velocity
    ttemp2 = turnprep_p3 + ttemp1(end); % time period of slowing down for turning
    ttemp3 = turning_p3 + ttemp2(end);% time period of turning
    ttemp4 = recover_p3 + ttemp3(end);% time period of recovering to normal speed
    ttemp5 = rest_p3 + ttemp4(end);  % rest of the journey
    time_total = [ttemp1,ttemp2,ttemp3,ttemp4,ttemp5];
    x_total = [route_x_p3,route_x2_p3,route_x3_p3,route_x4_p3,route_x5_p3];
    y_total = [route_y_p3,route_y2_p3,route_y3_p3,route_y4_p3,route_y5_p3];
else    % going straight
    route_x_p3 = p_i_mtx(i) + dir_sn_mtx1(i)*vi(i).*time2;
    route_y_p3 = time2*0 + dir_sn_mtx2(i)*2;
    x_total = route_x_p3;
    y_total = route_y_p3;

% flip x and y for vehicles that are original on y axis
if d_i(i)==2 || d_i(i)==4
    x_totaltmp = x_total;
    x_total = y_total;
    y_total = x_totaltmp;
end

% store data into array
x_array{1,i} = x_total;
y_array{1,i} = y_total;

end

% extract data points for each vehicles, just so it's easier later
x_total = x阵列{1,1};
y_total = y_array{1,1};
x_total2 = x_array{1,2};
y_total2 = y_array{1,2};

i1 = plot3(x_total2,y_total2,time2,'color',color{2},'Linewidth',linewidth);
hold on
i2 = plot3(x_total,y_total,time2,'color',color{1},'Linewidth',linewidth);
hold on

% plot for the box
[fx3,fy3,fz3] = intsbox(t(1),t_end(1)+t_exd(1));
fill3(fx3,fy3,fz3,color{1},'FaceAlpha',0.4);
[fx3,fy3,fz3] = intsbox(t(2),t_end(2)+t_exd(2));
fill3(fx3,fy3,fz3,color{2},'FaceAlpha',0.4);
legend([i1,i2],'initial vehicle1','initial vehicle2')

%%% plot for current routes
% uncomment the next nuclear comment sign if needs to plot the adjusted
% movements
%
if action ==0 % plot for new motion but covers the original movements
    % real time plotting
    aline2 = animatedline('color',color2{2},'Linewidth',linewidth2);
    aline1 = animatedline('color',color2{1},'Linewidth',linewidth2);
    for ii = 1:length(time2)
        if rem(ii,2000)==0
            addpoints(aline1,x_total(ii),y_total(ii),time2(ii));

end

addpoints(aline2,x_total(ii),y_total(ii),time2(ii));
end

end
addpoints(aline2,x_total2(ii),y_total2(ii),time2(ii));
drawnow
end
end
i3 = aline1;
i4 = aline2;

% plot for boxes
% [fx3,fy3,fz3] = intsbox(t_new(1),t_end_new(1));
% fill3(fx3,fy3,fz3,color{1},'FaceAlpha',0.4);
% [fx3,fy3,fz3] = intsbox(t_new(2),t_end_new(2));
% fill3(fx3,fy3,fz3,color{2},'FaceAlpha',0.4);
legend([i1,i2,i3,i4],'initial vehicle1','initial vehicle2','current vehicle1','current vehicle2')
else % action ==1, plot for adjusted movements
    % data points for the vehicle that has the same speed
    if a == 1 % if vehicle 3
        x_totala = x_total;
y_totala = y_total;
    else % if vehicle 4
        x_totala = x_total2;
y_totala = y_total2;
    end

    % data points for the vehicle that slows down
    if abs(direction_f(a2)) == abs(direction_i(a2)) % going straight
        [slow_p,routey_p,routeex_p,cont_p,route2_p,routeex2_p,up_p,routey3_p,routeex3_p,cont2_p,routey4_p,routeex4_p] = ...
            adjustedstraight3d2(d_i(a2),vi(a2),taccel(a2),tcons(a2),taccel(a2),timelim-(round(taccel(a2),4)*2+round(tcons(a2),4)),p_i_mtx(a2),dir2(a2),dir_sn_mtx1(a2),accel);
        % create the master time and x,y coordinate matrix
        time_total2 = [slow_p,cont_p+taccel(a2),up_p+round(taccel(a2),4)+round(tcons(a2),4),cont2_p+round(taccel(a2),4)];
        x_total2a = [routeex_p,routeex2_p,routeex3_p,routeex4_p];
y_total2a = [routey_p,routey2_p,routey3_p,routey4_p];
    else % turning adjustments
        [slow_p,routey_p,routeex_p,cont_p,route2_p,routeex2_p,slow2_p,routey3_p,routeex3_p,turning_p,route2_p,routeex2_p,routey4_p,routeex4_p,up_p,route5_p,routeex5_p,cont2_p,route6_p,routeex6_p] = ...
            adjustedturn3d2(d_i(a2),p_i_mtx(a2),vi(a2),t_turning_adjust(1),t_turning_adjust(2),t_turning_adjust(3),t_pass(a2),t_turnrecover(a2),timelim-t_end_new(a2)-
t_turnrecover(a2), vturning, a_turnprep1, a_turnrecover, accel, dir(a2), dir2(a2), dir3(a2), dir_sn_mtx, l(a2), radius(a2));

% create the master time and x,y coordinate matrix

% create the master time and x,y coordinate matrix

% create the master time and x,y coordinate matrix

time_total2 = [slow_p, cont_p + round(t_turning_adjust(1), 4), slow2_p + round(t_turning_adjust(1) + t_turning_adjust(2), 4), turning_p + round(t_new(a2), 4), up_p + t_end_new(a2), cont2_p + round(t_end_new(a2), 4) + round(t_turnrecover(a2), 4)];

x_total2a = [routex_p, routex2_p, routex3_p, routex4_p, routex5_p, routex6_p];
y_total2a = [routey_p, routey2_p, routey3_p, routey4_p, routey5_p, routey6_p];

end

% mid points between current trajectory and original trajectory for vehicle that slows down
%
if a2 == 1 % if its vehicle3
    midx = (x_total + x_total2a) ./ 2;
    midy = (y_total + y_total2a) ./ 2;
else
    midx = (x_total2 + x_total2a) ./ 2;
    midy = (y_total2 + y_total2a) ./ 2;
end

% animation plotting

aline1 = animatedline('color', color2{a}, 'Linewidth', linewidth2); % current trajectory for vehicle 3
aline2 = animatedline('color', color2{a2}, 'Linewidth', linewidth2); % current trajectory for vehicle 4
%
aline3 = animatedline; % mid points between current trajectory and original trajectory for vehicle that slows down

for ii = 1:length(time2)
    if rem(ii, 3000) == 0 % change the number here to adjust animation speed
        addpoints(aline1, x_totala(ii), y_totala(ii), time2(ii));
        addpoints(aline2, x_total2a(ii), y_total2a(ii), time2(ii));
        % addpoints(aline3, midx(ii), midy(ii), time2(ii));
        drawnow
    end
end
i3 = aline1; % variable for legends
i4 = aline2;

% plot for boxes
%
[fx3, fy3, fz3] = intsbox(t_new(a), t_end_new(a));
% fill3(fx3, fy3, fz3, color{a}; 'FaceAlpha', 0.4);
[fx3, fy3, fz3] = intsbox(t_new(a2), t_end_new(a2));
fill3(fx3, fy3, fz3, color{a2}, 'FaceAlpha', 0.4);
legend([i1, i2, i3, i4], 'initial vehicle1', 'initial vehicle2', 'vehicle that maintains its initial speed', 'yielding vehicle');
end

% function for checking collision patterns
function [range] = colli_pat(a, b, right, left)
    % create the range first
    if a <= b % right and left bound are in order from small to large
        range = a:b;
    else % left bound is smaller than right bound
        range = a:b+4; % rescale smaller left bound and make a list
        range = rem(range, 4); % rescale back
        f = find(range==0); % find variables that were 4, rescale last step made them 0
        range(f) = 4; % get 4 back
    end

    % include or exclude the first or end value, [], () or [], or ()?
    if length(range)>1 % if there is more than one number in the range
        if (right == 0) && (left == 0) % both ends are not included
            range(1) = []; % the range stays the same as its original one number
        elseif right == 0 % if the right end is not included
            range(1) = []; % the range stays the same as its original one number
        elseif left == 0 % if the left end is not included
            range(end) = []; % the range stays the same as its original one number
        end
    end

    else % if there is only one number in the range
        if (right == 0) && (left == 0) % if no ends are included
            range = []; % the range should be empty
        else
            % range stays the same as its original one number
        end
    end

% function for generating turning path data
function [cont_p3, route_y_p3, route_x_p3, turnprep_p3, route_y2_p3, route_x2_p3, turning_p3, route_y3_p3, route_x3_p3, recover_p3, route_y4_p3, route_x4_p3, rest_p3, route_x5_p3, route_y5_p3, temp] = ...
turnpath2(p3_x, vi, t1, t2, t3, t4, timelim, vturning, a_turnprep1, a_turnrecover, dir, dir2, dir3, radius, dir4)

    cont_p3 = 0:0.0001:round(t1,4);
    routey_p3 = cont_p3*0 + dir2*2;
    routex_p3 = p3_x + dir4*vi.*cont_p3; % constant velocity

    turnprep_p3 = 0.0001:0.0001:round(t2,4); % time period of slowing down for turning
    routey2_p3 = turnprep_p3*0+dir2*2;
    routex2_p3 = routex_p3(end) + dir4*(vi.*turnprep_p3-0.5.*a_turnprep1.*turnprep_p3.^2); % slowing down for turning

    turning_p3 = 0.0001:0.0001:round(t3,4); % time period of turning
    routey3_p3 = radius*(-dir)*cos(vturning.*turning_p3/radius)+(dir)*4;
    routex3_p3 = dir4*radius*sin(vturning.*turning_p3/radius)-dir4*4;

    recover_p3 = 0.0001:0.0001:round(t4,4); % time period of recovering to normal speed
    routex4_p3 = recover_p3*0+dir3*2;
    routey4_p3 = dir*4 + dir*(recover_p3.*vturning + 0.5.*a_turnrecover.*recover_p3.^2);

    temp = cont_p3(end) + turnprep_p3(end)+turning_p3(end)+recover_p3(end); % rest of the journey
    rest_p3 = 0.0001:0.0001:timelim-temp;
    routex5_p3 = rest_p3*0+dir3*2;
    routey5_p3 = routey4_p3(end) + dir*vi.*rest_p3;

end

function [fx3, fy3, fz3] = intsbox(t, t_end) % function for helping plot the box
fx3 = [ -4 -4 -4 -4 -4 4; 4 -4 4 4 4 4; 4 -4 4 4 4 4; -4 -4 -4 -4 -4 4;];
fy3 = [ -4 -4 -4 -4 -4 4; -4 4 -4 -4 4 4; -4 4 4 4 4 4; -4 -4 4 4 4 4;];
fz3 = [ t t t t end t t; t t t end t t; t_end t_end t_end t_end t End t_end;];
% function for generating adjusted trajectory data for turning behavior
function [slow_p, routey_p, routex_p, cont_p, routey2_p, routex2_p, slow2_p, routey3_p, routex3_p, turning_p, routey4_p, routex4_p, up_p, routey5_p, routex5_p, cont2_p, routey6_p, routex6_p] = ...
adjustedturn3d2(a2, p_x, vi, t1, t2, t3, t4, t5, t6, vturning, a_turnprep1, a_turnrecover, accel, dir, dir2, dir3, dir4, radius)
% slowing down
slow_p = 0:0.0001:round(t1,4);
route_y_p = slow_p*0 + dir2*2;
routex_p = p_x + dir4*(vi.*slow_p - 0.5.*accel.*slow_p.^2);
x1 = route_y_p;
y1 = routex_p;

vtemp = vi-accel*t1; % decreased velocity
% stays constant
cont_p = 0.0001:0.0001:round(t2,4);

route_y2_p = cont_p*0 + dir2*2;
routex2_p = routex_p(end) + dir4*vtemp.*cont_p;
x2 = route_y2_p;
y2 = routex2_p;

% slowing down for turning
slow2_p = 0.0001:0.0001:round(t3,4);

route_y3_p = slow2_p*0 + dir2*2;
routex3_p = routex2_p(end) + dir4*(vtemp.*slow2_p - 0.5.*a_turnprep1.*slow2_p.^2);
x3 = route_y3_p;
y3 = routex3_p;

% constant turning velocity
turning_p = 0.0001:0.0001:round(t4,4);

route_y4_p = radius*(-dir)*cos(vturning.*turning_p/radius) + (dir)*4;
routex4_p = dir4*radius*sin(vturning.*turning_p/radius)-dir4*4;
x4 = route_y4_p;
y4 = routex4_p;
% speed up

up_p = 0.0001 : 0.0001 : round(t5,4);

routex5_p = up_p*0 + dir3*2;
routey5_p = dir*4 + dir*(vturning.*up_p + 0.5.*a_turnrecover.*up_p.^2);
x5 = routey5_p;
y5 = routex5_p;

% constant velocity

cont2_p = 0.0001:0.0001:round(t6,4);
routex6_p = cont2_p*0 + dir3*2;
routey6_p = routey5_p(end) + dir*vi.*cont2_p;
x6 = routey6_p;
y6 = routex6_p;

if (a2 == 2) || (a2 == 4)
    routey_p = y1;
routex_p = x1;
routey2_p = y2;
routex2_p = x2;
routey3_p = y3;
routex3_p = x3;
routey4_p = y4;
routex4_p = x4;
routey5_p = y5;
routex5_p = x5;
routey6_p = y6;
routex6_p = x6;
    end
end

% function for generating adjusted trajectory data for straight
function[slow_p,routey_p,routex_p,cont_p,routey2_p,routex2_p,up_p,routey3_p,routex3_p,cont2_p,routey4_p,routex4_p] = adjustedstraight3d2(a2,vi,t1,t2,t3,t4,p3_x,dir2,dir4,accel)

% slowing down
slow_p= 0:0.0001:round(t1,4);
routey_p = slow_p*0 + dir2*2;
routex_p = p3_x + dir4*(vi.*slow_p - 0.5.*accel.*slow_p.^2);
x1 = routey_p;
y1 = routex_p;

vtemp = vi-accel*t1; % decreased velocity

% constant velocity
cont_p = 0.0001:0.0001:round(t2,4);
routey2_p = cont_p*0 +dir2*2;
routex2_p = routex_p(end)+ dir4*vtemp.*cont_p;
x2 = routey2_p;
y2 = routex2_p;

% speeding up
up_p = 0.0001 : 0.0001 : round(t3,4);
routey3_p = up_p*0 +dir2*2;
routex3_p = routex2_p(end) + dir4*(vtemp.*up_p + 0.5.*accel.*up_p.^2) ;
x3 = routey3_p;
y3 = routex3_p;

% constant velocity
cont2_p = 0.0001:0.0001:round(t4,4);
routey4_p = cont2_p*0 +dir2*2;
routex4_p = routex3_p(end) + dir4*vi.*cont2_p;
x4 = routey4_p;
y4 = routex4_p;

if (a2 == 2) || (a2 == 4)
    routey_p = y1;
routex_p = x1;
routey2_p = y2;
routex2_p = x2;
routey3_p = y3;
routex3_p = x3;
routey4_p = y4;
routex4_p = x4;
end
end
Appendix E

MATLAB Codes for Dimensional Analysis

E.1 Solving for S Matrix by Using Dimension Set Matrix Operation [52]

\[
\begin{align*}
B_d &= \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 \\
1 & -1 & -1 & 0 & 0 & -1 \end{bmatrix}; \quad \text{% the B matrix} \\
A_d &= \begin{bmatrix} 1 & 1 \\
0 & -2 \end{bmatrix}; \quad \text{% the A matrix} \\
E &= \text{eye}(6); \quad \text{% the identity matrix} \\
S &= (-\text{inv}(A_d) \times B_d \times E)' \quad \text{% the s matrix, solving for exponential values for repeating parameters for representing \( \pi \) parameters}
\end{align*}
\]

E.2 Dimensional Analysis Codes for the Case Where Two Vehicles Go Straight

```matlab
clear
h = waitbar(0,'running');
w = 4; \quad \text{% m, lane width} \\
a = 3.0; \quad \text{% m/s, acceleration} \\
vturn = 5; \quad \text{% m/s, turning speed} \\
l = 4; \quad \text{% m/s, vehicle length} \\
di = 0:300; \quad \text{%m, algorithm starting distance} \\
i4 = 0; \quad \text{i3 = 0; \% index for loop later} \\
v_set=[2,15]; \quad \text{% set the dimensionless v} \\
v_set = v_set.*sqrt(a*w); \quad \text{% convert that into actual v} \\
v_set = round(v_set); \quad \text{% round each values into integers} \\
size = length(v_set(1):v_set(2)); \quad \text{% find the size of velocity set} \\
v_all = v_set(1):v_set(2); \quad \text{% generate a velocity matrix with a step size of 1} \\
dmin = zeros(size);

for i2 = 1:size \quad \text{% first loop for vehicle 1 velocity} \\
    waitbar(i2/size)
```

for i2 = 1:size \quad % first loop for vehicle 1 velocity
    waitbar(i2/size)
v1_lop = v_all(i2); % m/s velocity for vehicle

for i3 = i2:size % second loop for vehicle 2 velocity
    i4 = i4+1;
    v2_lop = v_all(i3); % m/s velocity for vehicle 2
i = 0;
    v = [v1_lop v2_lop]; % m/s, creat a velocity matrix
%    disp(v)
    v = sort(v,'descend'); % order the velocity in descend so that first vehicle in order is the faster vehicle
    v1 = v(1);
    v2 = v(2);
    v1_str = v1/sqrt(a*w); % dimensionless velocity
    v2_str = v2/sqrt(a*w);
    l_str = l/w; % dimensionless vehicle length

    for d = 0:300 % algorithm starting distance
        i = i+1;
        syms t %s, initial slowing down period, set t as an unknown, solve for t later
        d_str = d/w; %dimensionless algorithm starting distance
        t_str = t*sqrt(a/w); % dimensionless t
        t_entry2_str = d_str / v2_str; % dimensionless time period until the second vehicle to enter the intersection
        t_entry1_str = d_str / v1_str;
        t_pass1_str = 2/v1_str; % dimensionless time period for first vehicle to pass through the intersection
        t_exd1_str = l_str / v1_str; % dimensionless time period for first vehicle's end to exit the intersection
        t_exit1 = d/v1 + w*2/v1 + l/v1; % s, total time period for the first vehicle to exit the intersection
        t_exit1_str = t_entry1_str + t_pass1_str + t_exd1_str;

        if t_entry2_str >= t_exit1_str % if there is no collision, let them pass
            tt = 1i;
            tcons = 1i;
            dmin(i2,i3) = di(i); % store the current d as the minimum d
%            disp('I')
            break % exit the loop, run the next velocity combination
        else % if there is a collision, slow down the second vehicle
            t1 = solve((v2_str-t_str)*(d_str/v1_str + l_str/v1_str + 2/v1_str) + (t*sqrt(a/w))^2 == d_str,t); % solve for initial slowing down period if a vehicle needs to take action
            t1 = double(t1); % convert that into an actual number, sometimes the line above returns equations
            f = find(t1==min(t1));
if length(f) == 2
    f = f(1);
end

%                 disp(d)
    tt = t1(f);
    tcons = t_exit1 - 2*tt;
end

t_all(i,1) = tt;
t_all(i,2) = tcons;

if isreal(tt)==1  % if the initial decelerating time period is real
    if tt>0 % if the initial decelerating time period is positive
        if tcons>0 % if the resulting constant speed time period is positive
            dmin(i2,i3) = di(i); % take the current d as the minimum algorithm starting distance
            disp('II')
            break
        end
    end
end

end

t_all = [];
t_all = [];
end
end

close(h)

% mirror the upper triangular matrix to lower part because we know that this
% plot is symmetric

dmin2= dmin;
dmin2 = rot90(dmin2,2);
dmin2 =flipud(dmin2);
dmin2 = rot90(dmin2,1);
dmin2 = tril(dmin2,-1);
dminall = dmin+dmin2;  % this is the final minimum algorithm starting distance for all velocity combinations

%% save and plot
%{
save('dmless_straighta3_setvstr.mat') % save the whole workspace

%% a3
v_plot = v_all
v_plot_dimless = v_plot./sqrt(a*w);
dmin_dimless = dminall./w;
fig = contour_no_color_dimless(v_plot_dimless,v_plot_dimless,dmin_dimless,{\textquoteleft minimum algorithm starting distance,\textquoteleft with acceleration at 3 m/s}^2 (dimensionless)\textquoteleft });

%% a2
v_plot = v_all
v_plot_dimless = v_plot./sqrt(a*w);
dmin_dimless = dminall./w;
fig = contour_no_color_dimless(v_plot_dimless,v_plot_dimless,dmin_dimless,{\textquoteleft minimum algorithm starting distance,\textquoteleft with acceleration at 2 m/s}^2 (dimensionless)\textquoteleft });

function FigHandle = contour_no_color_dimless(a,b,c,d)
    FigHandle = figure('Position',[0,0,800,700]);
    [c,h]=contourf(a,b,c,'fill','off');
    axis([4 12 4 12]);
    xlabel('vehicle 1 velocity (dimensionless)');
    ylabel('vehicle 2 velocity (dimensionless)');
    clabel(c,h,'manual','FontSize',30,'Color','red')
    title(d)
    ax = gca;
    ax.FontSize = 18;
    ax.FontWeight = 'bold';
end

E.3 Dimensional Analysis Codes for the Case Where Two Vehicles Have Conflict Turns

clc,clear
h = waitbar(0,'running');
w = 4 ; % m, lane width
a = 1.0 ; % m/s
vturn = 10 ; % m/s
l = 4 ; % m/s
di = 0:300; %m, algorithm starting distance
i4 = 0; % index for loop later
v_set=[1,15]; % set the dimensionless v
v_set = v_set.*sqrt(a*w); % convert that into actual v
v_set = round(v_set); % round each values into integers
size = length(v_set(1):v_set(2)); % find the size of velocity set
v_all = v_set(1):v_set(2); % generate a velocity matrix with a step size of 1
dim = zeros(size);

for i2 = 1:size % first loop for velocity
    waitbar(i2/size)
    v1_lop = v_all(i2); % m/s velocity for one vehicle

    for i3 = i2:size % second loop for velocity
        i4 = i4+1;
        v2_lop = v_all(i3);% m/s velocity for another vehicle

        i = 0;
        v = [v1_lop v2_lop]; % m/s, creat a velocity matrix
        disp(v)
        v = sort(v,'descend'); % order the velocity in descend so that first vehicle in order is the faster vehicle
        v1 = v(1); % m/s velocity for vehicle 1
        v2 = v(2); % m/s velocity for vehicle 2
        % dimensionless velocities
        v1_str = v1/sqrt(a*w);
        v2_str = v2/sqrt(a*w);
        l_str = l/w;  % dimensionless vehicle length
        vturn_str = vturn/sqrt(a*w); % dimensionless turning speed

        for d = 0:300 % algorithm starting distance
            i = i+1;
            syms t %s, initial slowing down period, set t as an unknown, solve for t later
            d_str = d/w; %dimensionless algorithm starting distance
            t_str = t*sqrt(a/w); % dimensionless t
            t_entry2_str = d_str/v2_str+0.5*v2_str+0.5*(vturn_str)^2/v2_str-vturn_str; % dimensionless time period until the second vehicle to enter the intersection
            t_entry1_str = d_str/v1_str+0.5*v1_str+0.5*(vturn_str)^2/v1_str-vturn_str; t_pass1_str = 3*pi/4/vturn_str; % dimensionless time period for first vehicle to pass through the intersection
            t_exd1_str = l_str / vturn_str; % dimensionless time period for first vehicle's end to exit the intersection
            t_entry1 = (d-(v1*((v1-vturn)/a)-0.5*a*((v1-vturn)/a)^2))/v1 + (v1-vturn)/a; %time period until the second vehicle to enter the intersection
            t_exit1 = t_entry1 + (3*pi*w/4/vturn) + (l/vturn); % s, total time period for the first vehicle to exit the intersection
            t_exit1_str = t_entry1_str + t_pass1_str + t_exd1_str;
if t_entry2_str >= t_exit1_str % if there is no collision, let them pass
    tt = 1i;
    tcons = 1i;
    dmin(i2,i3) = di(i); % store the current d as the minimum d
    break % exit the loop of varying d, run the next velocity combination
else % if there is a collision, slow down the second vehicle
    % solve for initial slowing down period if a vehicle needs to take action
    t1 = solve(v2_str*(t*sqrt(a/w)) - 0.5*(t*sqrt(a/w))^2 + t_exit1_str-
               (t*sqrt(a/w))*(v2_str-(t*sqrt(a/w)))-0.5*(v2_str-(t*sqrt(a/w))-vturn_str)^2 == d_str,t);
    t1 = double(t1);
    % second decelerating period for turning
    t4 = (v2-t1*a-vturn)/a;
    % constant speed period
    t3 = t_exit1 - t1 - t4;
    % if multiple results find the one that will not result
    % negative second decelerating period
    f = find(t3>0);
    if length(f) == 2
        f = f(1);
    end
    tt = t1(f);
    t3 = t3(f);
    t4 = t4(f);
end
% checking if all time outputs are real and positive
if isreal(tt)==1
    if (tt>0)
        if (t3>0)
            if (t4>0)
                dmin(i2,i3) = di(i); % record the current d as minimum algorithm starting distance
            end
        end
    end
end
end

t_all = []; 
t_all = [];
% mirror the upper triangular matrix to lower part because we know that this
% plot is symmetric
dmin2 = dmin;
vecnorm2 = rot90(dmin2,2);
vecnorm2 = flipud(dmin2);
vecnorm2 = rot90(vecnorm2,-1);
dvecnorm2 = tril(vecnorm2,-1);
dvecnormall = dvecnorm+dvecnorm2;
save('dmless_2turnsa2vturn10_2.mat') % save the whole workspace

%% plotting
% 2 turns a2
v_plot = v_all;
v_plot_dimless = v_plot./sqrt(a*w);
dvecnorm_dimless = dvecnormall./w;
fig = contour_no_color_dimless(v_plot_dimless,v_plot_dimless,dvecnorm_dimless,{'minimum
algorithm starting distance',' with acceleration at 2 m/s^2'});

% 2 turns a3
v_plot = v_all;
v_plot_dimless = v_plot./sqrt(a*w);
dvecnorm_dimless = dvecnormall./w;
fig = contour_no_color_dimless(v_plot_dimless,v_plot_dimless,dvecnorm_dimless,{'minimum
algorithm starting distance',' with acceleration at 3 m/s^2 (dimensionless)'});

% function for contour plot with no color and manually click for labels
function FigHandle = contour_no_color_dimless(a,b,c,d)
    FigHandle = figure('Position',[0,0,800,700]);
    [c,h]=contourf(a,b,c,'fill','off');
    axis([4 12 4 12]);
    xlabel('vehicle 1 velocity (dimensionless)');
    ylabel('vehicle 2 velocity (dimensionless)');
    clabel(c,h,'manual','FontSize',30,'Color','red')
    title(d)
    ax = gca;
    ax.FontSize = 18;
    ax.FontWeight = 'bold';
end
E.4 Dimensional Analysis Codes for the Case Where One Vehicles Have a Conflict Turn

% suppose vehicle 1, the outer loop always turns, making a large turn
clear
h = waitbar(0,'running');

w = 4; % m, lane width
a = 2.0; % m/s
vturn = 4; % m/s
l = 4; % m/s
di = 0:600; %m, algorithm starting distance
i4 = 0; % index for loop later
v_set=[1,15]; % set the dimensionless v
v_set = v_set.*sqrt(a*w); % convert that into actual v
v_set = round(v_set); % round each values into integers
size = length(v_set(1):v_set(2)); % find the size of velocity set
v_all = v_set(1):v_set(2); % generate a velocity matrix with a step size of 1

for i2 = 1:size  % first loop for velocity
    waitbar(i2/size)
v1_lop = v_all(i2); % m/s velocity for vehicle A

    for i3 = 1:size  % second loop for velocity
        v2_lop = v_all(i3); % m/s velocity for vehicle B, turning vehicle
        i4 = i4+1;
        v = [v1_lop v2_lop]; % m/s, creat a velocity matrix
        v = sort(v,'descend'); % order the velocity in descend so that first vehicle in order is the faster vehicle
        v1 = v(1); % m/s velocity for vehicle 1
        v2 = v(2); % m/s velocity for vehicle 2
        % dimensionless velocities
        v1_str = v1/sqrt(a*w);
        v2_str = v2/sqrt(a*w);
        l_str = l/w; % dimensionless vehicle length
        vturn_str = vturn/sqrt(a*w); % dimensionless turning speed
        % check if turning vehicle is slower or straight vehicle
        if v(1) == v1_lop
            cs = 1; % slow down turning vehicle, vehicle B
        else
            cs = 0;
        end

    end

    for d = 0:600  % algorithm starting distance
        if cs == 1 % slow down turning vehicle
\[ i = i+1; \]
\[ \text{syms } t \ % \text{ initial slowing down period, set } t \text{ as an unknown, solve for } t \text{ later} \]
\[ d_{\text{str}} = d/w; \ % \text{dimensionless algorithm starting distance} \]
\[ t_{\text{str}} = t\sqrt{a/w}; \ % \text{dimensionless } t \]
\[ t_{\text{entry2}}_{\text{str}} = d_{\text{str}}/v_{2\text{str}}+0.5*v_{2\text{str}}+0.5*(v_{\text{turn}}_{\text{str}})^2/v_{2\text{str}}-v_{\text{turn}}_{\text{str}}; \ % \text{dimensionless time period until the second vehicle to enter the intersection} \]
\[ t_{\text{entry1}}_{\text{str}} = d_{\text{str}}/v_{1\text{str}}+0.5*v_{1\text{str}}+0.5*(v_{\text{turn}}_{\text{str}})^2/v_{1\text{str}}-v_{\text{turn}}_{\text{str}}; \]
\[ t_{\text{pass1}}_{\text{str}} = 3*pi/4/v_{\text{turn}}_{\text{str}}; \ % \text{dimensionless time period for first vehicle to pass through the intersection} \]
\[ t_{\text{exd1}}_{\text{str}} = l_{\text{str}} / v_{\text{turn}}_{\text{str}}; \ % \text{dimensionless time period for first vehicle's end to exit the intersection} \]
\[ t_{\text{entry1}} = (d-(v_{1\text{str}}((v_{1\text{str}}-v_{\text{turn}})/a)-0.5*a*((v_{1\text{str}}-v_{\text{turn}})/a)^2))/v_{1\text{str}} + (v_{1\text{str}}-v_{\text{turn}})/a; \ % \text{time period until the second vehicle to enter the intersection} \]
\[ t_{\text{exit1}} = t_{\text{entry1}} + (3*pi*w/4/v_{\text{turn}}) + (l/v_{\text{turn}}); \ % \text{s, total time period for the first vehicle to exit the intersection} \]
\[ t_{\text{exd1}}_{\text{str}} = t_{\text{entry1}}_{\text{str}} + t_{\text{pass1}}_{\text{str}} + t_{\text{exd1}}_{\text{str}}; \]
\[ \text{if } t_{\text{entry2}}_{\text{str}} \geq t_{\text{exit1}}_{\text{str}} \ % \text{if there is no collision, let them pass} \]
\[ tt = 1i; \]
\[ t\text{cons} = 1i; \]
\[ d_{\text{min}}(i2,i3) = d_{\text{i}}(i); \ % \text{store the current } d \text{ as the minimum } d \]
\[ \text{break} \ % \text{exit the loop of varying } d, \text{run the next velocity combination} \]
\[ \text{else} \ % \text{if there is a collision, slow down the second vehicle} \]
\[ \% \text{solve for initial slowing down period if a vehicle needs to take action} \]
\[ t1 = \text{solve}(v_{2\text{str}}(t\sqrt{a/w}) - 0.5*(t\sqrt{a/w})^2 + (t_{\text{exit1}}_{\text{str}}-t\sqrt{a/w})*(v_{2\text{str}}-(t\sqrt{a/w}))-0.5*(v_{2\text{str}}-(t\sqrt{a/w})-v_{\text{turn}}_{\text{str}})^2 = d_{\text{str}}, t); \]
\[ t1 = \text{double}(t1); \ % \text{second decelerating period for turning} \]
\[ t4 = (v_{2\text{str}}-t1*a-v_{\text{turn}})/a; \ % \text{constant speed period} \]
\[ t3 = t_{\text{exit1}} - t1 - t4; \ % \text{if multiple results find the one that will not result} \]
\[ \% \text{negative second decelerating period} \]
\[ f = \text{find}(t3>0); \]
\[ \text{if length}(f) == 2 \]
\[ f = f(1); \]
\[ \text{end} \]
\[ tt = t1(f); \]
\[ t3 = t3(f); \]
\[ t4 = t4(f); \]
\[ \text{end} \ % \text{checking if all time outputs are real and positive} \]
\[ \text{if isreal}(tt)==1 \]
\[ \text{if } (tt>0) \]
\[ \text{if}(t3>0) \]
\[ \text{if}(t4>0) \]
\texttt{dmin(i2,i3) = di(i); % record the current d as minimum algorithm starting distance}
for this velocity set
\begin{verbatim}
  break
  end
end
end
\end{verbatim}

\begin{verbatim}
else  % slow down going straight vehicle
  i = i+1;
syms t %s, initial slowing down period, set t as an unknown, solve for t later
d_str = d/w; %dimensionless algorithm starting distance
t_str = t*sqrt(a/w); % dimensionless t
t_entry2_str = d_str / v2_str; % dimensionless time period until the second vehicle to
enter the intersection
t_entry1_str = d_str/ v1_str;
t_pass1_str = 2/v1_str; % dimensionless time period for first vehicle to pass through the
intersection
t_exd1_str = l_str / v1_str; % dimensionless time period for first vehicle's end to exit the
intersection
t_exit1 = d/v1 + w*2/v1 + l/v1; % s, total time period for the first vehicle to exit the
intersection
t_exit1_str = t_entry1_str + t_pass1_str + t_exd1_str;
if t_entry2_str >= t_exit1_str  % if there is no collision, let them pass
  tt = 1i;
tcons = 1i;
dmin(i2,i3) = di(i); % store the current d as the minimum d
  break  % exit the loop, run the next velocity combination
else % if there is a collision, slow down the second vehicle
  t1 = solve((v2_str-t_str)*(d_str/v1_str + l_str/v1_str + 2/v1_str) +(t*sqrt(a/w))^2 ==
  d_str,t); % solve for initial decelerating time period if a vehicle needs to take action
  t1 = double(t1); % convert that into an actual number, sometimes the line above returns
  equations
  f = find(t1==min(t1));
  if length(f) == 2
    f = f(1);
  end
  tt = t1(f);
tcons = t_exit1 - 2*tt;
end
t_all(i,1) = tt;
t_all(i,2) = tcons;
if isreal(tt)==1  % if the initial decelerating time period is real
\end{verbatim}
if tt>0 % if the initial decelerating time period is positive
    if tcons>0 % if the resulting constant speed time period is positive
        dmin(i2,i3) = di(i); % take the current d as the minimum algorithm starting distance
        break
    end
end
end
end

t_all = [];
t_all = [];
end
end

close(h)
save('dmless_1turnsa2.mat')  % save the whole workspace

%%% plotting
% 1turn a3
v_plot = v_all;
    v_plot_dimless = v_plot./sqrt(a*w);
    dmin_dimless = dminall./w;
    fig = contour_no_color_dimless(v_plot_dimless,v_plot_dimless,dmin_dimless,{'minimum algorithm starting distance',' with acceleration at 3 m/s^2 (dimensionless)'});

% 1tunr a2
v_plot = v_all;
    v_plot_dimless = v_plot./sqrt(a*w);
    dmin_dimless = dmin./w;
    fig = contour_no_color_dimless(v_plot_dimless,v_plot_dimless,dmin_dimless,{'minimum algorithm starting distance',' with acceleration at 2 m/s^2 (dimensionless)'});

% function for contour plot with no color and manually click for labels
function FigHandle = contour_no_color_dimless(a,b,c,d)
    FigHandle = figure('Position',[0,0,800,700]);
    [c,h]=contourf(a,b,c,'fill','off');
    axis([4 12 4 12]);
    xlabel('vehicle 1 velocity (dimensionless)');
    ylabel('vehicle 2 velocity (dimensionless)');
    clabel(c,h,'manual','FontSize',30,'Color','red')
title(d)
ax = gca;
ax.FontSize = 18;
ax.FontWeight = 'bold';
end
BIBLIOGRAPHY


ACADEMIC VITA

Academic Vita of Ting Xu
xting0831@gmail.com

EDUCATION

The Pennsylvania State University
Schreyer Honors College
Bachelor of Science in Mechanical Engineering
Bachelor of Science in Mathematics – Systems Analysis option

May 2018

Experience

Undergraduate Research Assistant
Spring 2016 - Present
Intelligent Vehicles and Systems Group, Penn State Dept. of Mechanical and Nuclear Engineering (Prof. Sean Brennan)

- Develop a centralized cooperative driving algorithm for two collision free automated vehicles at non-signalized intersections
- Demonstrate the four-way intersection scenario using MATLAB Graphical User Interface (GUI) animations
- Apply the developed algorithm to multi-vehicles at non-signalized intersections
- Modified realistic vehicles models in Blender for building a Highway Driving Simulator
- Programmed Python scripts to generate random vehicle selections in simulated highway autonomous traffic

Undergraduate Research Assistant
Summer 2017
Systems for Hybrid-Additive Process Engineering Lab, Penn State Dept. of Mechanical and Nuclear Engineering (Prof. Guha Manogharan)

- Enhanced fundamental knowledge of finite element analysis
- Wrote MATLAB scripts to perform finite element analysis for both one dimension and two-dimensional geometries
- Proposed an algorithm for applying manufacturing constraints of minimum wall and wall to wall thicknesses in topology optimization for additive manufacturing

Teaching Assistant
Summer 2016
The Penn State University, Engineering Design 100

- Lectured on modeling in SolidWorks and assisted students with tutorials
- Facilitated group design projects and assignments with innovative ideas

Teaching Assistant
Summer 2015
Penn State Engineering Study Abroad Program, Impact of Culture on Engineering in China

- Promoted Chinese culture to domestic students and demonstrated leadership and translation skills
- Organized activities for student and ensure their safety

Skills

MATLAB/Simulink, Python, SAS, Arduino UNO, CATIA, AutoCAD, SolidWorks, Photoshop, Blender, 3D Printing
Scholarships

Louis A. Harding Memorial Scholarship
2017 – 2018

Activities and Affiliations

Volunteer, Penn State Advanced Vehicle Team 2016 - 2017
EcoCAR3 control algorithm subgroup
- Created Simulink models for potential failure modes and control solutions

Coordinator of Public Relation Department 2014 – 2015
Chinese International Culture and Communication Association
- Conducted activities that help build relationships between international and domestic students
- Communicated with professors and organization leaders to coordinate events