THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF MECHANICAL ENGINEERING

AN IMPROVED ALGORITHM FOR AUTONOMOUS VEHICLE VELOCITY PROFILE CALCULATION

JONATHAN DAVID PARSONS
SPRING 2021

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate degree
in Mechanical Engineering
with honors in Mechanical Engineering

Reviewed and approved* by the following:

Sean N Brennan
Professor of Mechanical Engineering
Thesis Supervisor

Jean-Michel Mongeau
Professor of Mechanical Engineering
Honors Adviser

* Electronic Approvals on file

ABSTRACT

Autonomous Vehicles (AVs) are becoming a commodity of greater interest and importance to society in recent years. Much research and development has been spent in creating many operational prototypes, and currently some semi-autonomous vehicles have begun to be available to the average consumer. There are various benefits to owning and using an AV including the possibility of avoiding crashes due to driver error and lack of alertness, comfort of not having to drive and being able to fill the ride time with other activities, as well as enable the physically handicapped to be more independent. For many reasons, AVs show the potential of becoming an everyday appearance on the roadways in the upcoming years.

This research is motivated by a gap in AV research wherein AVs struggle to perform well in unpredictable and unique situations that can appear during a driving experience. One of the most common of these are construction zones. Each construction zone is unique based on layout, geography, random workers and equipment sporadically placed throughout and moving, as well as changing speed limits both within and around the construction zone. This thesis in particular examines speed-limit predictions and calculations in the situation where acceleration-limits, deceleration-limits, and speed-limits are all presented within segments of operation for an AV. The work here presents codes and methods suited for real-time implementation that calculate physically-feasible speed profiles for AVs to follow, with results and plots illustrating the performance of the algorithm including difficult edge-case situations.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF EQUATIONS

ACKNOWLEDGEMENTS

I would like to thank Dr. Sean Brennan for giving me the opportunity to join his autonomous vehicle research team in the development of this thesis. His countless hours working with me and my research enabled me to create and develop this thesis and the algorithm of which this thesis presents. This thesis could not have been accomplished without his support.

I would also like to thank the many faculty members within the Mechanical Engineering department and the Schreyer Honors College for all their help and support. It was truly a blessing to be surrounded by such knowledgeable and supportive faculty and staff at Penn State.

Chapter 1

This thesis develops an algorithm for calculating and plotting velocity profiles for Autonomous Vehicles operating with speed limit restrictions as well as limits on acceleration and deceleration. The algorithm is valuable to autonomous vehicle research and development because the code allows AVs to accurately calculate and travel at the maximum possible velocity through a designated segment distance. Since the efficiency and flow of travel can be increased with decreased travel times, this algorithm will ensure that an AV will travel at its maximum achievable velocity while abiding by posted speed limits and taking into account the vehicle's own acceleration and deceleration limits.

This research is motivated by a gap in AV research wherein AVs struggle to perform well in unpredictable and unique situations that can appear during a driving experience. One of the most common areas that cause difficulty for AV navigation are construction zones. Each construction zone is unique based on layout, geography, random workers and equipment sporadically placed throughout and moving, as well as changing speed limits both within and around the construction zone. This thesis in particular examines speed-limit predictions and calculations in the situation where acceleration-limits, deceleration-limits, and speed-limits are all presented within segments of operation for an AV. The work here presents codes and methods suited for real-time implementation that calculate physically-feasible speed profiles for AVs to follow, with results and plots illustrating the performance of the algorithm including difficult edge-case situations. The resulting algorithm allows AVs to accurately calculate and navigate in real-time at the maximum feasible velocity profile throughout a segmented distance which could

improve travel efficiency and safety in construction zones and any other area that has changing speed limits in a segmented distance.

The material of this thesis describing the improved algorithm for autonomous vehicles is laid out as follows. Chapter 2 contains the literature review summarizing past research that has been completed for autonomous vehicles and systems. Chapter 3 provides the physics and mathematical relationships used to derive the MATLAB code as well as key sections of the MATLAB code developed for the algorithm. Chapter 4 contains the results from running the algorithm using a test script demonstrating some predefined cases that could be inputted into the algorithm. Finally, the conclusion to the research and algorithm is presented in Chapter 5 with Appendix A and Appendix B containing the MATLAB test script and the MATLAB functions composing the algorithm respectively.

Chapter 2

Literature Review

This chapter presents four main summaries of the literature in the field of autonomous vehicles (AVs). Topics discussed include societal benefits and concerns to autonomous vehicles, mapping of roadways and proposed layout of roadway infrastructure for autonomous systems, proposed safety equipment for autonomous vehicles, as well as additional research that needs to be done in the field of autonomy. The first subsection will introduce the concept of AVs and the presumed benefits the technology has to offer. The following subsection will introduce the process of mapping roadways for AV navigation. Present safety systems that have been studied in relation to autonomy is discussed in the third subsection. To wrap up the discussion, the fourth subsection explains the gaps in current AV research and what still remains to be done in the field.

Societal Impacts of Autonomous Vehicles

This subsection presents the benefits, challenges, and concerns poised by the influx of autonomous vehicles. AVs can be used to accomplish and improve various tasks in the world. AVs can offer transportation to dependents including the disabled, elderly, and even children [1]. Besides the increase in access to mobility due to AVs, AVs might also improve travel times and reduce the number of high-risk situations on roadways including traffic jams or bottlenecks. According to Meyer et al. [1], road capacities could be increased anywhere from 80% to 370%

due to improved traffic flow and decreased reaction time necessary for AVs when compared to humans. AVs not only present an opportunity to increase access of mobility to more classes of people, AVs can also offer quicker and more convenient transportation to all of society.

Convenience is not the only benefit associated with AVs. There is also an economical benefit to an increase in AV usage. According to Chen et al. [2], the company nuTonomy built the first autonomous taxi in Singapore in 2015. The ability for nuTonomy to offer AV taxis allows the company to offer reduced taxi rates because of not having to pay a driver. Lower taxi prices have led to increased use of the taxi service and demonstrates the increased opportunity for people to be able to afford to travel [2].

Besides the improved access to mobility, AVs offer a safer driving environment. According to the National Highway Transportation Safety Administration (NHTSA), 90% of all collisions are due to some kind of driver error and 40% of fatal collisions include factors like alcohol, drugs, distraction, or fatigue [3]. Autonomous vehicles are less susceptible to driver-related errors; therefore, with proper planning and programming, collisions due to AV navigation error should be reduced. During the 2014-2015 year, only three AV accidents were reported where the AV was at-fault [4]. These accidents occurred at low speeds and the cause was determined to be because the AV either mis-calculated another driver's response, or the AV incorrectly read a traffic sign [4]. With improved vehicle-to-vehicle communication and vehicle-to-infrastructure communication, such as communication with traffic signals, AV response should become more predictable, eliminating most AV-responsible accidents.

Various methods have been proposed on how an AV can detect road departure. While the goal for many companies is to produce a fully autonomous vehicle, current AVs are being built to allow for both autonomous and human controlled driving. In both cases, human

monitoring is considered necessary. To help AVs assist in human error while driving, Yang et al. [5] have created a system to detect road departure using steering torque and yawrate. The system operates off the principle that if the steering is quickly adjusted, meaning a high torque is applied, and one or more of the wheels is spinning at a much different velocity or yawrate than the other tires, the AV can conclude that the vehicle has partially departed from the roadway (assuming a hard pavement) and needs to help correct the situation. A common issue in non-autonomous vehicles is that when accidently departing from the roadway, the driver will overcompensate by oversteering. The oversteering can cause the vehicle to go into a slide and if the tires come back into contact with the pavement, the spinning velocity of the tires can cause the vehicle to suddenly cross into the opposing lane of travel [5]. Safety devices like road departure detection could make an AV and our highways safer.

Mapping for Autonomous Vehicles

To improve AV operation and vehicle response, the AV must understand the terrain in which it is driving. Mapping provides an AV with a visual, topographical layout of the area. The map then allows the AV to more accurately predict road conditions, leading to improved handling and response as well as improved fuel economy [6]. Currently, the 2.5-dimensional (D) ray-tracing algorithm is the most cost-effective approach in navigating the mapped system [6].

There are several different types of mapping systems which include 3D, 2D, and 2.5D ray tracing algorithms. Ray tracing in 3D evaluates all three planes of the 3D world: width, depth, and height. In doing so, 3D ray tracing evaluates the different light rays that bounce off of objects to determine their location and size. The 3D ray tracing allows for the AV to identify

where objects are, how they are shaped, and what parts of the object are closer or further away from the AV which is otherwise known as depth perception. While 3D ray tracing would lead to the clearest and most accurate mapping system, 3D mapping is costly due to the sensors that are required[6].

Ray tracing in 2D operates in a two-dimensional field of width and height so depth perception is lost. The loss of depth perception causes the AV to no longer know how deep or dense the object is which limits the AV's ability to determine if an object is a threat. For example, a five-gallon bucket of concrete might appear the same as a trash bag blowing in front of the vehicle in 2D ray tracing. Both objects might have the same width and height dimensions which the 2D ray tracing will evaluate; however, 2D ray tracing will not allow for the AV to determine how thick the object is. Obviously, colliding with a thin trash bag is not a huge concern, but hitting a five-gallon bucket of concrete would likely come with repercussions. Therefore, in 2D ray tracing, the algorithm is programmed for safety reasons to assume more things in the map are obstacles than what actually are because the AV cannot determine the depth of an object [6].

The 2.5D ray tracing algorithm is a good medium between the two approaches of 2D and 3D without increasing the cost dramatically because 2.5D ray tracing provides an image in 2D that appears as a 3D image. The image appears to be three-dimensional because 2.5D ray tracing allows for the allusion of depth perception. A 2.5D image is like a 3D painting that appears to have depth as some things in the picture appear closer than other objects; however, the painting that hangs on the wall has only two dimensions of width and height. The painting alludes to having depth though it does not literally. For AVs, the 2.5D ray tracing algorithm takes multiple 2D images from different angles of the same location and then combines them to

form an equivalent image that appears as 3D. The 2.5D approach is more cost effective than using the equipment necessary for 3D mapping and provides additional information relating to depth when compared with 2D mapping [6]. Figure 1 shows the extensive equipment on an early autonomous vehicle which uses the 2.5D ray-tracing approach.

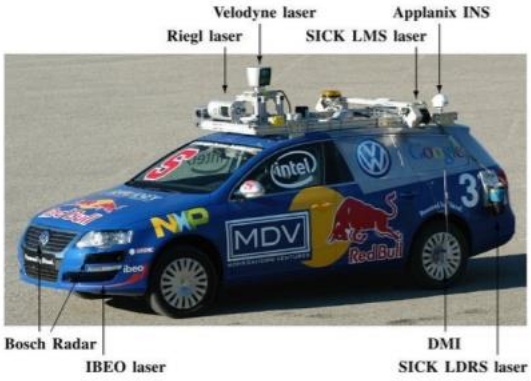

Figure 1: Picture of Mapping Vehicle for Autonomous Road Mapping [6]

Besides the need for mapping of current terrain, discussion about how roadways should be specifically designed for AV travel has also been considered. Chen et al. believe that creating separate roadways for AV use would help reduce traffic congestion and improve travel times for AVs [2]. The proposed solution would not prevent AVs from using other roadways that are not AV specific but rather provide additional roadways that only AVs can use. AVs may choose to travel on the AV designated roads if it leads to an improved travel time or more direct route of travel. It is important to note that in creating separate AV lanes, the goal is to increase travel flow for AVs and not hinder the travel of conventional vehicles that require human drivers. Because of the added AV driving zones, AV usage should increase as bottlenecks and travel times should be reduced.

Safety Systems for Autonomous Vehicles

In order to create safety systems for autonomous vehicles, studies have been conducted on human reaction to autonomous vehicles. In one study, 10 young adults were placed in an AV simulator which allowed for mixed autonomous and manual driving [7]. In each case, the "driver" slowed down upon receiving the warning from the vehicle that autonomation was about to resume control. This subconscious decision to slow the vehicle down is believed to be due to the public's general distrust of an AV. However, this perceived distrust can come with consequences. As this one car slows down, a domino effect can follow as all the cars behind the AV will be forced to slow in the bottleneck. Situations such as bottlenecks are known hazards because they can lead to accidents for distracted drivers [3]. Fortunately, as the simulation was repeated on each of the 10 subjects, the participants did not slow the vehicle down as much as in the previous experiment during AV takeover; this trend of drivers reacting less to the AV takeover is a sign that drivers become more trusting of AVs and become less of a traffic hazard as the drivers spend more time traveling in their AVs [7].

Another step in AV safety is how the vehicle itself is constructed. According to a study presented by Kocsis et al. [8], AVs should be constructed in the following fashion. First, all AVs must have multiple types of sensors at many locations throughout the vehicle, communication busses, collision detection equipment, a watchdog that monitors all vehicle operations, and an Electronic Control Unit (ECU) [8]. The sensors are the outer detection equipment that allow the computer to monitor the surroundings of the vehicle. These sensors vary in type depending on the company producing the vehicle. After the sensors pick up data, the data is then transmitted through the communication busses to the computer(s). The computer will use the data by applying it to the algorithms the computer has been programmed with. A decision will be made

by the computer based off of how the data interacts with the algorithm and this decision will be seen outwardly by a vehicle response. The vehicle's response is a large concern of this thesis, especially with relation to construction zones.

In addition to the basic sensor-to-computer autonomy design, autonomous vehicles should come equipped with another set or two of sensors to provide the same data. With the additional sensors, in the case that a sensor fails, which is an inevitable situation with age, the computer can still receive the data it needs to make an informed decision. A computer can typically detect a faulty sensor when the data from surrounding sensors are largely different than the data from one of the sensors. Also, defective sensors will typically send a voltage that is out of the specified range the computer is programmed to "see" and the computer will immediately notice the bad sensor and flag it for replacement.

Another back-up system to the AV's front-line sensors are sensors that detect impact. Although a collision is a worst-case scenario, if an AV has mis-interpreted the driving situation it is absolutely necessary that the AV know if it has been involved in an accident. Crash sensors, otherwise known as crush sensors, are imbedded in the bumpers of AV's and are a last-resort fail-safe preventing the AV from further injury. If an AV receives a signal from the crush sensors, the AV computer will immediately apply the brakes and bring the vehicle to a stop [8].

To improve AV safety between vehicles, some studies have been conducted using vehicle transmitters or vehicle-to-vehicle (V2V) communication. In one study, radio frequency transmitters (RFs) were placed on all construction vehicles, personnel, and construction automobiles on site. While none of the vehicles were autonomous, the sensors would alert the operator of the working vehicle and person that was located within close proximity of the working vehicle if both parties were too close to each other [9]. The study received positive

results; however, there were limitations to the system including geography, climate, and other obstructions that hindered the transmitter's signal. Further research and development, along with coupling the technology with autonomy, could lead to a safer construction zone environment.

In another study of construction zones by Kanan et al. [10], an IoT autonomous system was applied to autonomous vehicles in a construction zone. Within this study, three antennas and ultrasonic sensors were tested on the back of each construction vehicle. Also, RF transmitters were placed on the workers. Once again, results were positive as the vehicles could determine where a worker was located when in close proximity to the moving vehicles. The system was particularly focused on eliminating back-over accidents as 360 industrial workers died due to back-over accidents between 2005-2010 [10]. Also, 7681 deaths occurred between 1992-2010 due to construction vehicle accidents [10]. While this study was once again focused on construction vehicles, the technology could be tailored to a civilian AV. Also, communication between autonomous construction vehicles and civilian AVs could improve construction zone safety.

Gaps in Autonomous Vehicle Research

While there has been a great deal of research in autonomous systems, gaps in AV functionality remains a problem. AV's may need to disengage and require manual human driving when road situations become too complex for current algorithms to process. Another problem with the limitations of current algorithms is an unwanted vehicle response to a road situation which can lead to a collision.

One of the most problematic areas for current AV algorithms, and a problem motivating this thesis work, is navigation in constructions zones, particularly in selecting appropriate speeds for these zones. Construction zones present many challenges in navigation, even for manual human driving. Programming a car to navigate obstructed and altered highway lanes with modified traffic signs is challenging. Add construction workers and oddly shaped construction equipment working randomly throughout the construction zone only adds to the complexity of the environment. Current algorithms are not enough to safely and predictably navigate a construction zone. While studies relating to autonomy in construction zones have been conducted, the research has been limited to construction vehicles-to-worker communication and not civilian AV communication with the construction zone, equipment, and workers. The studies have improved the safety between construction workers and the equipment they operate; however, the safety of the workers in relation to the general public operating AVs through the construction zone remains an issue.

To improve the AV navigation in construction zones, several things need to be accomplished. First, each construction zone needs to be mapped using the mapping software for AVs. The mapping will provide each AV with clearly marked, new lanes of travel which will help ensure that the AV drives in the appropriate lane. Second, communication between AVs and construction equipment and workers should be installed. The added communication will allow the AVs to monitor the location of the various workers and equipment in the construction zones which will help AVs to avoid colliding with any of the workers or equipment. Thirdly, additional sensors may need to be added to detect various obstructions that might be present in the altered lanes. Finally, testing needs to be done tracking how the AV responds and altering the driving algorithms based on the vehicle's response. The end goal is to produce an algorithm

that an AV can use repeatedly to navigate a construction zone accurately, safely, and predictably. This algorithm could save lives and improve safety in construction zones for the workers as well as the occupants of the AVs.

This thesis will focus on how an AV responds to various construction zone setups relating to speed limits. AV's need to be able to travel appropriately fast on the road while being safe and abiding by all posted speed limits. An algorithm made to calculate the fastest velocity profile an AV can follow with given acceleration, deceleration, travel distance, initial and final velocities, and the applied speed limit prior to the construction zone will allow an AV to properly predict and navigate roadways with changing speed limits.

These needs motivate the work presented in this thesis, namely the goal to develop methods of predicting appropriate speed profiles. Basically, the proposed algorithm on the following pages will allow an AV to travel as fast as it can from an initial speed, up to the speed limit, and then back down to the speed limit of the construction zone (the final velocity) without breaking any traffic rules, and while maintaining acceleration and deceleration profile constraints. In the interest of increasing commerce and the efficiency of transportation, the proposed algorithm could increase travel efficiency for passengers in AV's and ensure that an AV will reliably and predictably enter a construction zone at the required speed limit, leading to a safer work zone environment.

Chapter 3

Autonomous Vehicles, Construction Zones, and Speed Limits

This chapter presents the main ideas for the goal of this thesis: to develop an algorithm for an autonomous vehicle to maximize travel velocity within zones of changing speed limits. Namely, the algorithm presented in this thesis will calculate a velocity profile for a vehicle when provided with a specified distance of traveled segment, acceleration, deceleration, initial velocity, final velocity, and the speed limit or max velocity the vehicle is allowed to achieve in a designated path. This algorithm is useful to automation in construction zones as speed limits on roadways typically change in construction zones. However, the ultimate goal of any passenger, and thus the autonomous vehicle, is to drive at the fastest possible allowed velocity to acquire the quickest travel time. Therefore, the algorithm presented in this thesis will allow the autonomous vehicle to calculate the fastest velocity profile it can achieve while obeying posted speed limits, acceleration and deceleration limits, and given initial and final velocities within a segmented distance. The first subsection will describe the mathematical relationships used to develop the algorithm. Next, the following subsection will describe the coding developed to create the algorithm.

Section 3.1 Mathematical Relationships of Velocity Using Physics

To plot the velocity profile, a definition of variables was defined. Given an initial velocity lower than the speed limit, a said speed limit, and a final velocity less than the speed limit, a trapezoidal shaped velocity profile of velocity versus distance will occur, as shown in Figure 2. The first segment of the velocity profile is a diagonal, linear line with a positive slope,

showing an increase in velocity over an increase in distance. This segment of increasing velocity is defined as "Fraction of P1" or the fraction of the distance traveled where the velocity increases. The point at which the velocity equals the speed limit is defined as P_1, with P_0 being the initial point or the beginning of the velocity profile. The segment following P_1 is the part at which the autonomous vehicle maintains a constant velocity with zero acceleration or deceleration. The vehicle will maintain the velocity equal to the speed limit until the vehicle must begin decelerating in order to meet the final velocity requirement within the said distance parameter. The point at which the vehicle begins to decelerate is defined as P_2. Once the vehicle begins decelerating, the third and final segment of the velocity profile is created by a linear, negative slope connecting P_2 to the final point where the end distance and final velocity is achieved, defined as P_3.

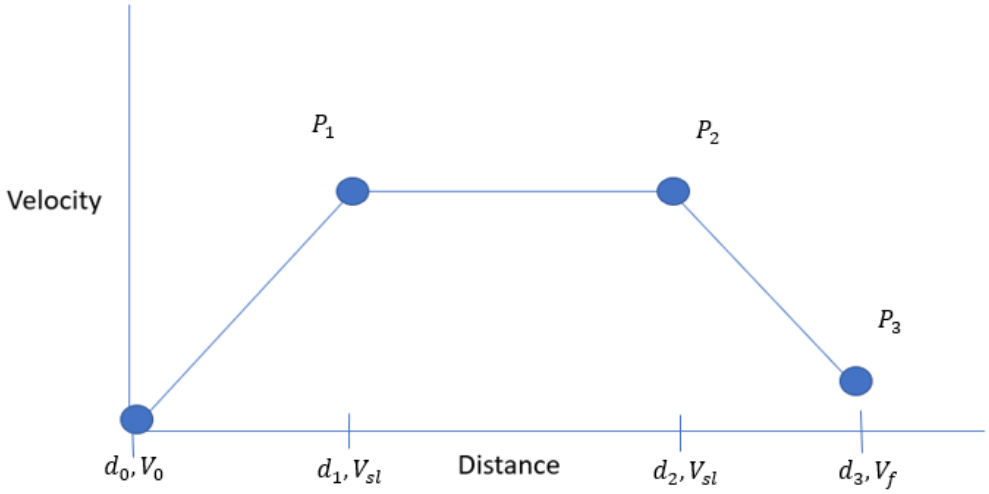

Figure 2: Sample Velocity Profile with Speed Limit

With the velocity profile visualized and new variables defined, a mathematical relationship using physics can be developed. Using Equation 1, where V is the end velocity we hope to obtain, V_0 is the initial velocity, a is the acceleration, d_1 is the distance from P_0 to P_1, we

can solve for d1 or the total distance of the first segment where the velocity increases. Equation 2 shows Equation 1 solved for d_1 with d_0 simplified out since the initial distance is zero because there is no distance to add to the existing, specified distance. V_{sl} is the speed limit or max velocity the vehicle should obtain and a_{max} is the positive acceleration value. Now that the distance of the first segment has been defined, Equation 3 shows the calculation for the total time of the segment.

$$V^2 = {V_0}^2 + 2a(d_1 - d_0)$$

Equation 1: Velocity, Distance, and Acceleration Relationship

$$d_1 = (\frac{{V_{sl}}^2 - {V_0}^2}{2a_{max}})$$

Equation 2: Distance of First Segment

$$t_1 = \frac{V_{sl} - V_0}{a_{max}}$$

Equation 3: Time Elapsed During First Segment

After solving for the distance of the first segment of the velocity profile, the deceleration portion of the velocity profile can be calculated. Using Equation 4, the distance of the final segment d_3 can be found much similar to the calculations done to find the distance of the velocity profile for the first segment using the difference between the final velocity V_f and the speed limit velocity divided by double the negative deceleration value a_{min}.

The calculation for the time interval is also much the same, substituting the deceleration value into the denominator and using the difference between the final and speed limit velocities as shown in Equation 5.

$$d_3 = (\frac{{V_f}^2 - {V_{sl}}^2}{2a_{min}})$$

Equation 4: Distance of Final Segment

$$t_3 = \frac{V_f - V_{sl}}{a_{min}}$$

Equation 5: Time Elapsed During Final Segment

Since the total distance is a known parameter, finding the distance segment from P_1 to P_2, or the constant acceleration part of the velocity profile where the velocity is constant, can be found by using Equation 6. Equation 6 takes the total distance and subtracts the first and third distance segments to find the resulting second distance segment. The time elapsed during the second segment can also be found using Equation 7 where the segmented distance is divided by the constant velocity of the segment.

$$d_2 = d_{total} - d_1 - d_3$$

Equation 6: Distance of Second Segment

$$t_2 = \frac{d_2}{V_{sl}}$$

Equation 7: Time Elapsed During Second Segment

Finally, the velocity over the segmented distances can be found using MATLAB's linearly spaced command which plots points starting at a specified initial point and linearly connecting the initial point to a specified final point using a specified number of points. Since the velocity profile is made of three segments, three linear space commands were used as shown in Figure 3. One command connects the initial velocity to the speed limit velocity. The second command connects P_1 to P_2 linearly at the speed limit. To finish, the third command connects the speed limit velocity to the final velocity. These velocity profiles can then be plotted against time or distance.

```
velocity_rising = linspace(V0, Vmax,num_points)';
velocity_steady = linspace(Vmax,Vmax,num_points)';
velocity_falling = linspace(Vmax, Vf,num_points)';
velocities = [velocity_rising; velocity_steady; velocity_falling];
```

Figure 3: MATLAB Coding of Velocity Profile Segments

Section 3.2 MATLAB Algorithm for Velocity Profile with Speed Limit

Using the mathematical relationships discussed in the previous subsection, an algorithm was created. Figure 4 contains the portion of the code in the function titled "fcn_costs_FastestVelFromConstAccelDeccelSpeedLimit" that deals with handling the speed limit requirement for the velocity profile. In order to enter the "if" statement, the algorithm first checks to see if the maximum velocity it wants to use at any time exceeds the speed limit. If the maximum velocity the algorithm calculates exceeds the speed limit then the algorithm will enter the "if" statement which recalculates the velocity profile and adjusts the maximum velocity to equal the speed limit.

After adjusting the maximum velocity to equal the speed limit, the algorithm calculates the three segments of the velocity profile as discussed in the previous subsection. Using Equation 2, the algorithm starts by calculating the distance segment of the velocity profile for the first segment in which the velocity accelerates, referred to as "New_distance_P1". Following the distance segment calculation, the time for the same segment "t1" is coded in using Equation 3. Next, the algorithm calculates the third and final distance and time segments, "Downslope_distance" and "t3" respectively, of the velocity profile using Equations 4 and 5. To connect the first and third segments, the algorithm uses Equations 6 and 7, connecting the two distance segments by subtracting the sum of the two distance segments from the total distance and calculating the time segment by dividing the new found middle segment by the speed limit velocity. The second distance segment is coded as "distance_on_the_peak" because it is the segmented distance where the velocity is at its peak maximum and the time segment for this portion is referred to as "t2".

```
% Speed Limit Function Code
indices_speed_limit = find(V_max > Vsl);
if ~isempty(indices_speed_limit)    % Maximum velocity exceeds the speed limit

    % Update V_max to Vsl - the maximum speed will just be the speed limit
    V_max(indices_speed_limit,:) = Vsl(indices_speed_limit,:);

    % CALCULATE VALUES FOR THE UPSLOPE
    % P1 must be recalculated since platau cuts it off
    New_distance_P1(indices_speed_limit,:) = ...
        (Vsl(indices_speed_limit,:).^2 - V0(indices_speed_limit,:).^2)./(2.*a_max(indices_speed_limit,:));
    t1(indices_speed_limit,:) = (Vsl(indices_speed_limit,:)-V0(indices_speed_limit,:))./...
        a_max(indices_speed_limit,:);     % Time elapsed during increasing velocity in first frame

    % CALCULATE VALUES FOR THE DOWNSLOPE
    Downslope_distance(indices_speed_limit,:) = ...
        (Vf(indices_speed_limit,:).^2 - Vsl(indices_speed_limit,:).^2)./(2.*a_min(indices_speed_limit,:));
    t3(indices_speed_limit,:) = (Vf(indices_speed_limit,:)-Vsl(indices_speed_limit,:))./...
        a_min(indices_speed_limit,:);     % Time elapsed during decreasing velocity in third frame

    % CALCULATE VALUES FOR THE PLATEAU
    distance_on_the_peak = ...
        segment_lengths(indices_speed_limit,:) - New_distance_P1(indices_speed_limit,:)...
        - Downslope_distance(indices_speed_limit,:); % distance where velocity is steady
    t2(indices_speed_limit,:) = ...
    distance_on_the_peak./Vsl(indices_speed_limit,:);      % Time elapsed during constant velocity in second frame
```

Figure 4: MATLAB Code for Velocity Profile Using Speed Limit

With the mathematical relationships defined and coded, cases for each possible scenario of input data was created in the MATLAB function titled "fcn_plotcalc_kinematic_profile_from_key_vel_dist_speed_limit". Figure 5 shows a list of cases under an "if" statement to direct the algorithm to run the proper calculations to generate the correct velocity profile. The first case coded applies to the situation where there is only acceleration and deceleration with no portion of the velocity profile maintaining a constant speed. This situation can be created by setting a speed limit that is unrealistically high when evaluated with other parameters or when the acceleration and deceleration values are such that the speed limit is never reached.

The second case is for when the initial velocity equals the speed limit but the final velocity is lower than the speed limit. In the second case, there is no acceleration. The resulting velocity profile will have a flat, first segment of a constant velocity at the speed limit before decelerating to the final velocity. As shown in Figure 5, the segmented distance of the acceleration of the velocity profile is set equal to zero. The segmented distance for the

deceleration portion of the velocity profile is set equal to Equation 4 plus “d_P2”, the distance up to the deceleration which is calculated earlier in the code.

The third case is for the opposite situation to the second case. Here, the final velocity is equal to the speed limit and the initial velocity is lower than the speed limit. The resulting velocity profile will display an accelerating velocity curve until the velocity reaches the speed limit, at which time the velocity profile will become a straight, horizontal line to the end of the total distance segment. As shown in Figure 5, Equation 2 was used to calculate the segmented distance for the acceleration portion of the velocity profile and the deceleration portion of the curve was made nonexistent by setting it equal to the final point on the total distance vector.

Next, the fourth case is for when there is only acceleration. To enter this “if” statement, the speed limit would never be reached and the final velocity would have to be the maximum velocity achieved. As shown in the code in Figure 5, the distance during the deceleration portion of the velocity profile is set equal to the end position meaning there is no segmented distance for deceleration.

The fifth case is the opposite of the fourth case in which there is only deceleration. To enter this case, the speed limit would have to be equal to the final velocity and the deceleration low enough that is takes the whole distance segment to reach the final velocity. As shown in the code in Figure 5, the distance during the acceleration portion of the velocity profile is set equal to zero.

The sixth case is where the initial and final velocities are equal to the speed limit and the speed limit is a positive value greater than zero. As shown in the code in Figure 5, the acceleration section of the distance segment for when the velocity profile is increasing is set equal to zero and the deceleration portion of the distance segment is set equal to the final

distance value. Only the steady distance is generated and then linearly spaced between in initial and final position.

Finally, the seventh case is for the situation that was arbitrarily described and shown in Figure 3 where there is acceleration from an initial velocity to the speed limit, a segmented distance during a constant velocity at the speed limit, and then deceleration to a final velocity. As shown in Figure 5, the equations described in Section 3.1 are used.

```
% case where we accelerate and then decelerate (no steady speed)
if flag_onlyaccelerating == 1 && a_min~=0 && a_max~=0
    distances_rising = (velocity_rising.^2 - V0.^2)/(2*a_max);
    distances_falling = d_P2 + (velocity_falling.^2 - Vmax.^2)/(2*a_min);
    distances_steady =  d_P2*ones(num_points,1);
% case where we have a steady speed and then decelerate
elseif (fraction_at_P1==0) && (flag_onlyaccelerating == 0) && (fraction_at_P2 ~= 1)
    distances_rising = 0*ones(num_points,1);
    distances_falling = d_P2 + (velocity_falling.^2 - Vmax.^2)/(2*a_min);
    distances_steady = linspace(distances_rising(end),distances_falling(1),num_points)';
% case where we accelerate and then have steady speed
elseif (fraction_at_P2 == 1) && (flag_onlyaccelerating == 0) && (fraction_at_P1 ~= 0)
    distances_rising = (velocity_rising.^2 - V0.^2)/(2*a_max);
    distances_falling = d_total*ones(num_points,1);
    distances_steady = linspace(distances_rising(end),distances_falling(1),num_points)';
% case where we only accelerate
elseif flag_onlyaccelerating == 1 && fraction_at_P1 == 1
    distances_rising = (velocity_rising.^2 - V0.^2)/(2*a_max);
    distances_falling = d_total*ones(num_points,1);
    distances_steady = linspace(distances_rising(end),distances_falling(1),num_points)';
% case where we only deccelerate
elseif flag_onlyaccelerating == 1 && fraction_at_P1 == 0
    distances_rising = 0*ones(num_points,1);
    distances_falling = d_P2 + (velocity_falling.^2 - Vmax.^2)/(2*a_min);
    distances_steady = linspace(distances_rising(end),distances_falling(1),num_points)';
% case where we maintain a steady velocity from start to end
elseif (V0 == Vmax) && (Vmax == Vf)
    distances_rising = 0*ones(num_points,1);
    distances_falling = d_total*ones(num_points,1);
    distances_steady = linspace(distances_rising(end),distances_falling(1),num_points)';
% case where we accelerate, obtain steady speed, then decelerate
else
    distances_rising = (velocity_rising.^2 - V0.^2)/(2*a_max);
    distances_falling = d_P2 + (velocity_falling.^2 - Vmax.^2)/(2*a_min);
    distances_steady = linspace(distances_rising(end),distances_falling(1),num_points)';
end
```

Figure 5: MATLAB Cases with Speed Limit

Chapter 4

MATLAB Velocity Profile Results

This chapter will demonstrate the five different basic velocity profiles that can be generated. The first subsection will explain the case where the speed limit is much higher than the maximum velocity that can be achieved with the given distance and acceleration parameters. The second subsection will describe the second case where the initial velocity equals the speed limit with a lower final velocity. The next section will describe the reverse in which the final velocity equals the speed limit but the initial velocity starts at a lower value. Basically, the second and third cases discussed are cases where either the acceleration or deceleration is zero respectively. Next, Case 4, most similar to the one discussed in Chapter 3 and shown in Figure 2 where the initial and final velocities are both lower than the speed limit, there is an acceleration and deceleration segment as well as a constant velocity segment at the speed limit. The following subsection will describe a similar case to Case 4; however, Case 5 will use an initial and final velocity value of zero with a positive speed limit applied. The sixth subsection will describe the case where there is no acceleration or deceleration meaning the initial and final velocities equal the speed limit with a positive value for the velocity. Finally, the last subsection will note possible erroneous parameters that could be inputted into the algorithm and how the algorithm will properly catch and notify the operator that they have passed faulty data into the code.

All of the following test cases were generated using the same initial and final points which were identified in Cartesian coordinate form as (-10,10) and (10,10) respectively,

generating a travel distance of twenty meters. The initial and final velocities, speed limit, acceleration, and deceleration values were all varied on a case-by-case basis.

Section 4.1 Case 1: Speed Limit Exceeds Maximum Obtainable Velocity

In this case, the following parameters were passed into the algorithm. First, an initial and final velocity of zero was chosen. The speed limit was set to 10,000 meters per second and the acceleration and deceleration was set to six and negative seven meters per second squared respectively. Obviously, over the twenty-meter distance traversed, the autonomous vehicle cannot accelerate to 10,000 meters per second for multiple reasons. The acceleration and deceleration values do not allow the vehicle to reach such a high speed within the specified distance, especially with zero initial and final velocities. Affirmatively, the algorithm realizes that the velocity profile will not reach the maximum allowed speed limit and therefore finds the maximum obtainable velocity within the given acceleration, deceleration, initial, and final velocity parameters. The vehicle accelerated to a peak velocity for a brief moment before decelerating back to the final velocity. Since the speed limit is never reached, the vehicle only accelerates and decelerates as shown in Figure 6.

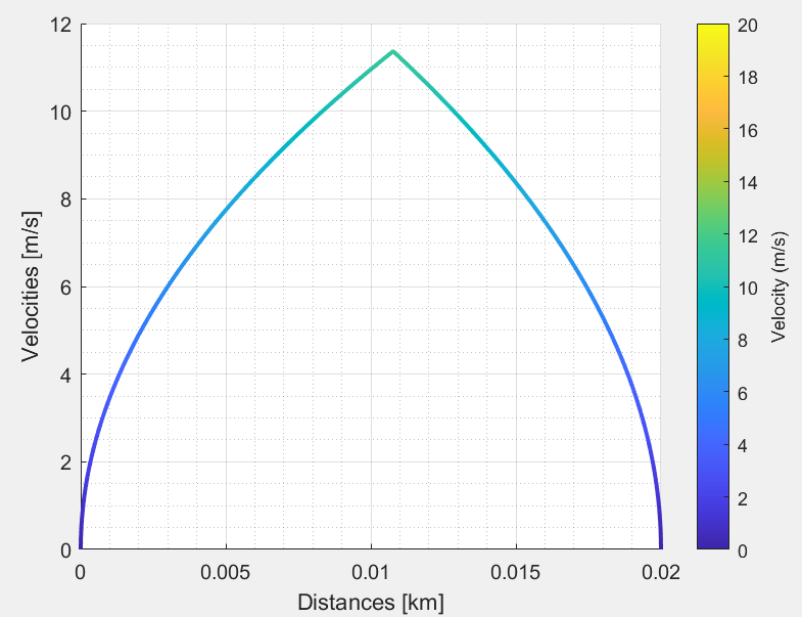

Figure 6: Only Acceleration and Deceleration, Velocity Profile

Section 4.2 Case 2: No Acceleration, Initial Velocity Equals Speed Limit Velocity Profile

Figure 7 shows the velocity profile with the initial velocity set equal to the speed limit at ten meters per second. Since the initial velocity is equal to the speed limit, there is no acceleration; however, a deceleration value of seven meters per second squared was applied and a finial velocity of zero. As expected, the resulting velocity profile maintains a steady velocity and then decelerates to the final velocity.

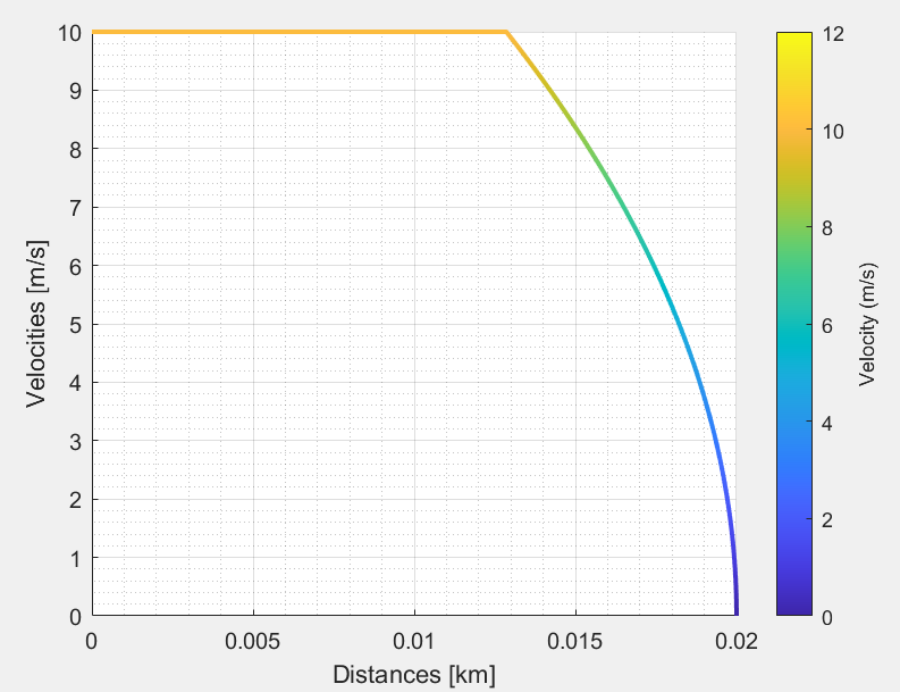

Figure 7: No Acceleration, Constant Velocity, Deceleration, Velocity Profile

Section 4.3 Case 3: No Deceleration, Final Velocity Equals Speed Limit Velocity Profile

Case 3 is the reverse situation to Case 2, though the parameter's magnitudes were changed. In this case, the initial velocity was set to zero and the final velocity was set equal to the speed limit of five meters per second. There was no deceleration since the final velocity is equal to the speed limit but there was an applied acceleration of one meter per second. Resultingly, Figure 8 shows a smaller acceleration in a smaller slope of the acceleration curve as the velocity profile increases. Then, when the travel velocity obtains the speed limit's value, the travel velocity holds steady to the end of the travel segment.

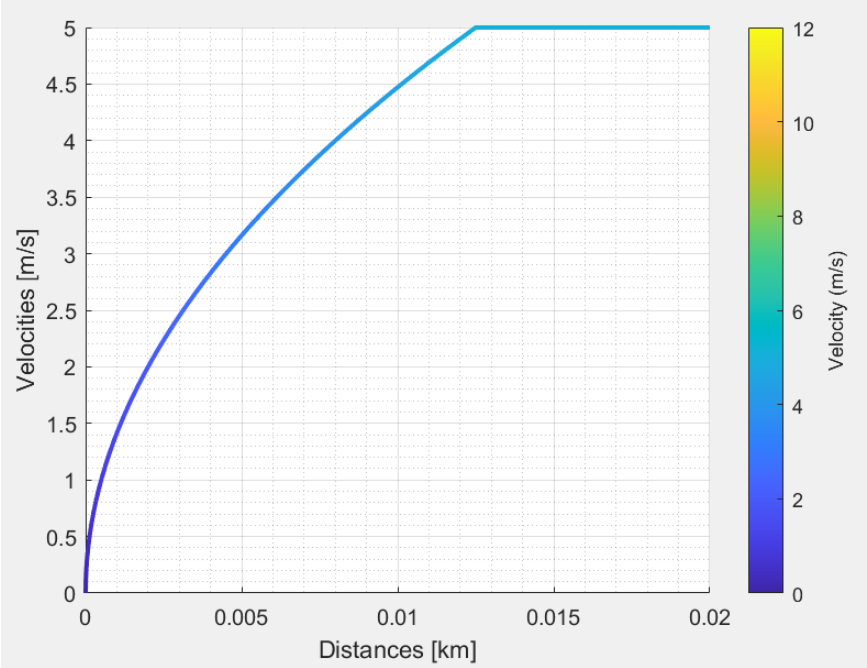

Figure 8: Acceleration, Constant Velocity, No Deceleration, Velocity Profile

Section 4.4 Case 4: Acceleration, Constant Velocity, and Deceleration Velocity Profile

In Case 4, there is both acceleration and deceleration as well as a segment of the velocity profile in which the velocity is held constant at the speed limit. Up to this point, Case 4 is most similar to the original example presented in Figure 3. Here, an acceleration and deceleration value of forty and forty-one meters per second squared was used respectively with an initial velocity, speed limit, and finial velocity of ten, eleven, and zero respectively. Since the acceleration and deceleration values were extremely large, the velocity profile displays steep slopes of increasing and decreasing velocities. Also, since the vehicle has such large acceleration and deceleration values, the vehicle can quickly accelerate to the speed limit, hold that speed limit for the maximum portion of the plot, before quickly decelerating to the final velocity as shown in Figure 9.

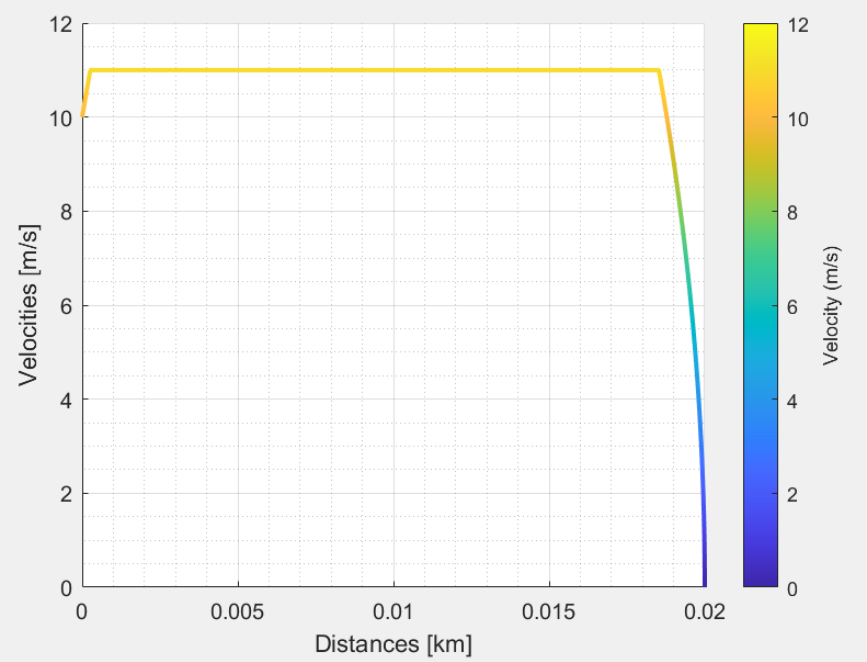

Figure 9: Acceleration, Constant Velocity, Deceleration, Velocity Profile

Section 4.5 Case 5: Acceleration, Constant Velocity, and Deceleration Velocity Profile with Zero Initial and Final Velocity Conditions

Case 5 is also similar to Case 4 and most similar to the arbitrary example displayed in Figure 2. However, Case 5 uses an initial and final velocity of zero with a speed limit of 10 meters per second and an acceleration and deceleration value of 10 meters per second squared. Since the acceleration and deceleration and the initial and final velocity values are equal, the velocity profile generated in Figure 10 is a symmetric plot. The smaller acceleration and deceleration values lead to a less steep slope for the first and third segments of the plot when compared to Case 4 but the velocity profile still displays a steady velocity segment equal to the speed limit.

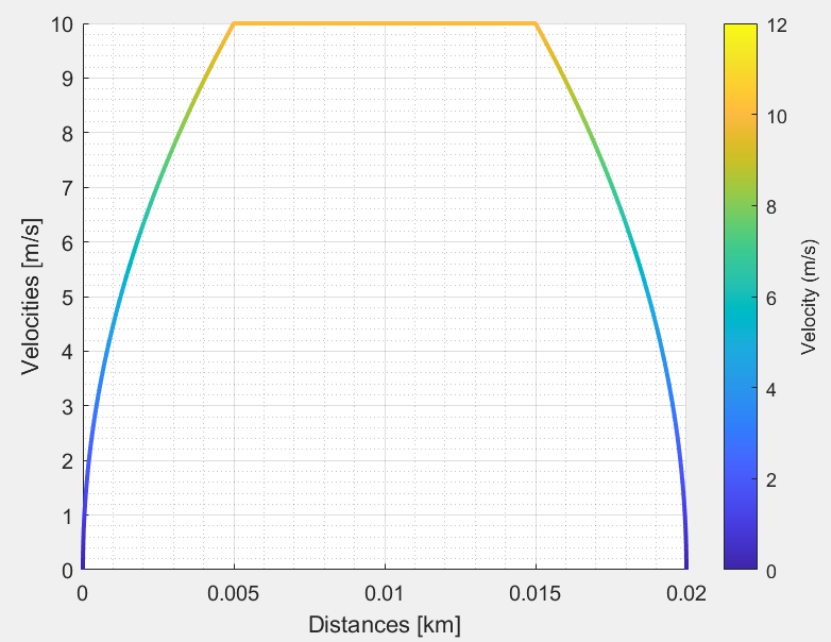

Figure 10: Acceleration, Constant Velocity, Deceleration, Zero Initial and Final Velocities, Velocity Profile

Section 4.6 Case 6: No Acceleration or Deceleration Velocity Profile

Case 6 presents the situation where there is no acceleration or deceleration as the initial and final velocities equal the speed limit, a value of ten meters per second. As expected, the velocity profile holds a steady value at the speed limit for the duration of the path as shown in Figure 11.

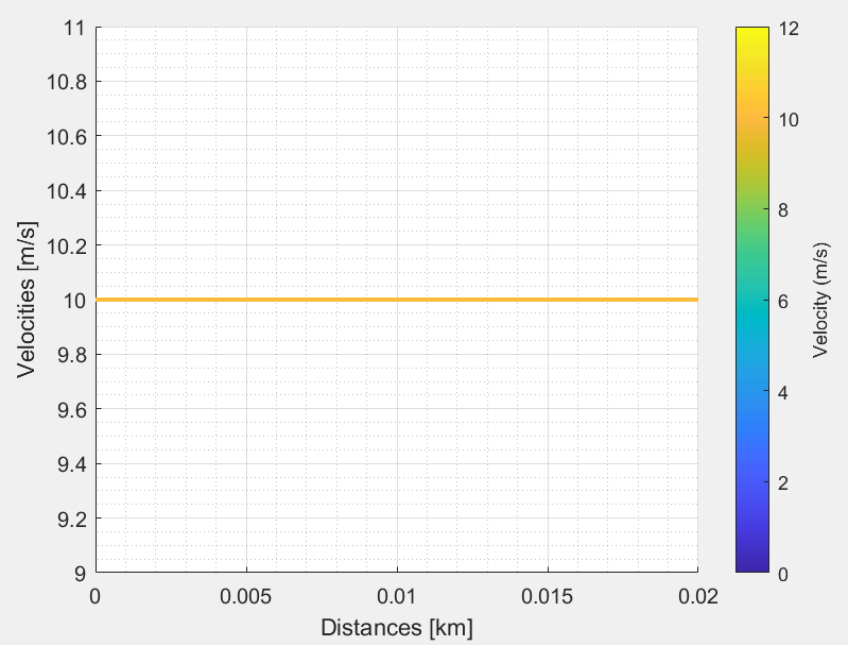

Figure 11: No Acceleration or Deceleration, Constant Velocity, Velocity Profile

Section 4.7: Test Cases for Erroneous Data

This subsection presents four possible cases of invalid inputs that the algorithm has been prepared to identify and will not generate a velocity profile since the inputs are invalid.

First, there could be a case in which the initial and final velocities are zero and the speed limit is zero for a segmented distance. In this case, the vehicle is not moving when it enters the segment, is not allowed to move in the segment, and does not move at the end of the segment. If this case is inputted into the algorithm, the algorithm will output an error saying “You are not allowed to move.”

Second, there is a possibility that the user could input an initial or final velocity greater than the speed limit. If the initial velocity is greater than the speed limit, the vehicle has already broken the speed limit which is not allowed. Also, if the final velocity is greater than the speed limit, there is no way for the vehicle to instantaneously accelerate at the end of the segment to

the final velocity value. Furthermore, if the vehicle does reach the final velocity, it will have broken the speed limit too. In either of these cases, a warning message will be displayed telling the user that the initial or final velocity cannot be higher than the speed limit.

Thirdly, a user could input conditions where, due to the acceleration or deceleration values, a velocity profile cannot be generated. For example, if a user inputs an extremely low acceleration with a final velocity that is higher than what can be achieved within the set distance, the algorithm will generate an error notifying the user that acceleration is too small to achieve the necessary final velocity. Likewise, an error about the deceleration value would occur if the deceleration is too low for the final velocity to be achieved.

Finally, an error will occur if a user inputs a negative speed limit. Since this algorithm is not designed for a vehicle going backwards since it does not seem applicable to autonomous vehicles navigating construction zones, the algorithm will simply display an error message stating “Speed Limit velocity should be a positive value.”

Chapter 5

Summary and Conclusion

This thesis has focused on the operation of autonomous vehicles in construction zones and proposed an algorithm to enable AVs to safely navigate construction zones at the specified speed limit. The proposed algorithm allows AVs to calculate from an initial velocity, how fast it can travel and for how long before it must begin decelerating in order to meet the new speed limit, otherwise known as the final velocity, of the upcoming construction zone. This algorithm can also be used at the end of a construction zone when the speed limit presumably increases back to a more normal speed of travel. In fact, this algorithm can be run by an AV for any change in speed limits, within or without a construction zone. With the speed limit algorithm, AVs will be able to travel at their max velocity while respecting speed limits and being fully aware of their own acceleration and deceleration limitations which will vary vehicle to vehicle

Appendix A

MATLAB Speed Limit Test Script Code

```
%
script_test_fcn_plot_pathVelocitiesVsDistanceConstA
ccel - this is is a script written
% to test the function:
fcn_plot_pathVelocitiesVsDistanceConstAccel.m
%
% USAGE of the base function:
%
fcn_plot_pathVelocitiesVsDistanceConstAccel(P0,Pf,V
0,Vf,Vmax,fraction_at_max_speed,fig_num,max_colorba
r_speed)
%
% Written by S. Brennan, sbrennan@psu.edu if there
are any questions
% Eddited by J. Parsons, jup381@psu.edu is there
are any questions
% Revision history:
% 2020_07_09 - first write of the script
% 2021_03_15 - added speed limit restrictions

% Prep the workspace
close all;
clear all;
clc;

%% CASE 1: Example showing usage with cost
calculations
% Vf = V0 = 0, Vsl is unobtainably high
velocity_fig_num = 1;

% (NOTE: it will only work if the function below is
in the path)
P0= [-10 10]; % Initial Position (in meters)
```

```
Pf= [10 10]; % Final Position (in meters)
V0= 0;       % Initial Velocity (in meters/second)
Vf= 0;       % Final Velocity (in meters/second)
Vsl= 10000;  % Speed Limit (meters/second)

a_max= 6;    % Max Constant Acceleration (in
meters/second^2)
a_min= -7;   % Max Constant Deceleration (in
meters/second^2)
max_colorbar_speed = 20;

[Vmax,~,~,~,P1,P2,fraction_at_P1,fraction_at_P2] =
fcn_costs_FastestVelFromConstAccelDecelSpeedLimit(P
0,Pf,V0,Vf,Vsl,a_max,a_min);
fcn_plot_pathVelVsDistConstAccelSpeedLimit(P0,Pf,V0
,Vf,Vmax,fraction_at_P1,fraction_at_P2,velocity_fig
_num,max_colorbar_speed)


%% CASE 2: The following three calls are all with
non-zero initial velocity
% Vf = 0, V0 = Vsl = 10
velocity_fig_num = 2;  % (optional) plot results in
figure
max_colorbar_speed = 12;

% Basic example - harder decel than accel
P0= [-10 10]; % Initial Position (in meters)
Pf= [10 10]; % Final Position (in meters)
V0= 10;      % Initial Velocity (in meters/second)
Vf= 0;       % Final Velocity (in meters/second)
Vsl= 10;     % Speed Limit (meters/second)
a_max= 0.00000001;    % Max Constant Acceleration
(in meters/second^2)
a_min= -7;   % Max Constant Deceleration (in
meters/second^2)
```

```
[Vmax,~,~,~,P1,P2,fraction_at_P1,fraction_at_P2] =
fcn_costs_FastestVelFromConstAccelDecelSpeedLimit(P
0,Pf,V0,Vf,Vsl,a_max,a_min);
fcn_plot_pathVelVsDistConstAccelSpeedLimit(P0,Pf,V0
,Vf,Vmax,fraction_at_P1,fraction_at_P2,velocity_fig
_num,max_colorbar_speed)

%% CASE 3: Basic example - harder accel than decel
with final velocity > speed limit
% Vf = Vsl, V0 = 0
velocity_fig_num = 3;  % (optional) plot results in
figure

P0= [-10 10]; % Initial Position (in meters)
Pf= [10 10]; % Final Position (in meters)
V0= 0;       % Initial Velocity (in meters/second)
Vf= 5;       % Final Velocity (in meters/second)
Vsl= 5;      % Speed Limit (meters/second)
a_max= 1;    % Max Constant Acceleration (in
meters/second^2)
a_min= -1;   % Max Constant Deceleration (in
meters/second^2)
[Vmax,~,~,~,P1,P2,fraction_at_P1,fraction_at_P2] =
fcn_costs_FastestVelFromConstAccelDecelSpeedLimit(P
0,Pf,V0,Vf,Vsl,a_max,a_min);
fcn_plot_pathVelVsDistConstAccelSpeedLimit(P0,Pf,V0
,Vf,Vmax,fraction_at_P1,fraction_at_P2,velocity_fig
_num,max_colorbar_speed)

%% CASE 4: Basic example - equal accel and decel
% Vsl doesn't equal V0 or Vf
velocity_fig_num = 4;
max_colorbar_speed = 12;
P0= [-10 10]; % Initial Position (in meters)
Pf= [10 10]; % Final Position (in meters)
V0= 10;      % Initial Velocity (in meters/second)
Vf= 0;       % Final Velocity (in meters/second)
```

```
Vsl= 11;       % Speed Limit (meters/second)
a_max= 40;     % Max Constant Acceleration (in
meters/second^2)
a_min= -41;    % Max Constant Deceleration (in
meters/second^2)
[Vmax,~,~,~,P1,P2,fraction_at_P1,fraction_at_P2] =
fcn_costs_FastestVelFromConstAccelDecelSpeedLimit(P
0,Pf,V0,Vf,Vsl,a_max,a_min);
fcn_plot_pathVelVsDistConstAccelSpeedLimit(P0,Pf,V0
,Vf,Vmax,fraction_at_P1,fraction_at_P2,velocity_fig
_num,max_colorbar_speed)


%% CASE 5: The following three calls show the max /
min limits being applied
% zero start and ending velocities, positive speed
limit value applied

velocity_fig_num = 5;  % (optional) plot results in
figure 1
max_colorbar_speed = 12;
P0= [-10 10]; % Initial Position (in meters)
Pf= [10 10]; % Final Position (in meters)
V0= 0;       % Initial Velocity (in meters/second)
Vf= 0;        % Final Velocity (in meters/second)
Vsl= 10;      % Speed Limit (meters/second)
a_max= 10;     % Max Constant Acceleration (in
meters/second^2)
a_min= -10;    % Max Constant Deceleration (in
meters/second^2)
[Vmax,~,~,~,P1,P2,fraction_at_P1,fraction_at_P2] =
fcn_costs_FastestVelFromConstAccelDecelSpeedLimit(P
0,Pf,V0,Vf,Vsl,a_max,a_min);
fcn_plot_pathVelVsDistConstAccelSpeedLimit(P0,Pf,V0
,Vf,Vmax,fraction_at_P1,fraction_at_P2,velocity_fig
_num,max_colorbar_speed)

%% CASE 6: V0 = Vsl = Vf = 10
```

```
velocity_fig_num = 6;  % (optional) plot results in
figure
P0= [-10 10]; % Initial Position (in meters)
Pf= [10 10]; % Final Position (in meters)
V0= 10;       % Initial Velocity (in meters/second)
Vf= 10;        % Final Velocity (in meters/second)
Vsl= 10;      % Speed Limit (meters/second)
a_max= 1;     % Max Constant Acceleration (in
meters/second^2)
a_min= -1;    % Max Constant Deceleration (in
meters/second^2)
[Vmax,~,~,~,P1,P2,fraction_at_P1,fraction_at_P2] =
fcn_costs_FastestVelFromConstAccelDecelSpeedLimit(P
0,Pf,V0,Vf,Vsl,a_max,a_min);
fcn_plot_pathVelVsDistConstAccelSpeedLimit(P0,Pf,V0
,Vf,Vmax,fraction_at_P1,fraction_at_P2,velocity_fig
_num,max_colorbar_speed)

%% Intentionally breaking code items follow
% Uncomment to inentionally break the code
if 1 == 0
    %% V0 = Vsl = Vf = 0 INTENTIONALL BREAKS CODE
SINCE THERE IS NO MOTION
    P0= [-10 10]; % Initial Position (in meters)
    Pf= [10 10]; % Final Position (in meters)
    V0= 0;       % Initial Velocity (in
meters/second)
    Vf= 0;        % Final Velocity (in
meters/second)
    Vsl= 0;      % Speed Limit (meters/second)
    a_max= 1;     % Max Constant Acceleration (in
meters/second^2)
    a_min= -1;    % Max Constant Deceleration (in
meters/second^2)

[Vmax,~,~,~,P1,P2,fraction_at_P1,fraction_at_P2] =
```

```
fcn_costs_FastestVelFromConstAccelDecelSpeedLimit(P
0,Pf,V0,Vf,Vsl,a_max,a_min);

fcn_plot_pathVelVsDistConstAccelSpeedLimit(P0,Pf,V0
,Vf,Vmax,fraction_at_P1,fraction_at_P2,velocity_fig
_num,max_colorbar_speed)


    %% Basic example - harder accel than decel -
INTENTIONALLY BREAKS CODE DUE TO SPEED LIMITS OUT
OF RANGE
    % starting velocity with lower speed limit and
zero ending velocity
    P0= [-10 10]; % Initial Position (in meters)
    Pf= [10 10]; % Final Position (in meters)
    V0= 10;      % Initial Velocity (in
meters/second)
    Vf= 0;       % Final Velocity (in
meters/second)
    Vsl= 2;      % Speed Limit (meters/second)
    a_max= 20;     % Max Constant Acceleration (in
meters/second^2)
    a_min= -2.5;    % Max Constant Deceleration (in
meters/second^2)

[Vmax,~,~,~,P1,P2,fraction_at_P1,fraction_at_P2] =
fcn_costs_FastestVelFromConstAccelDecelSpeedLimit(P
0,Pf,V0,Vf,Vsl,a_max,a_min);

fcn_plot_pathVelVsDistConstAccelSpeedLimit(P0,Pf,V0
,Vf,Vmax,fraction_at_P1,fraction_at_P2,velocity_fig
_num,max_colorbar_speed)


    %% The following three calls are all with non-
zero final velocity - INTENTIONALLY BREAKS CODE,
ACCEL TOO LOW
```

```
    % Vf = Vsl, V0 = 0, acceleration is too low
    velocity_fig_num = 11;  % (optional) plot
results in figure 1
    max_colorbar_speed = 12;

    % Basic example - harder decel than accel
    P0= [-10 10]; % Initial Position (in meters)
    Pf= [10 10]; % Final Position (in meters)
    V0= 0;      % Initial Velocity (in
meters/second)
    Vf= 5;       % Final Velocity (in
meters/second)
    Vsl= 5;      % Speed Limit (meters/second)
    a_max= 0.0000000001;    % Max Constant
Acceleration (in meters/second^2)
    a_min= -10;   % Max Constant Deceleration (in
meters/second^2)

[Vmax,~,~,~,P1,P2,fraction_at_P1,fraction_at_P2] =
fcn_costs_FastestVelFromConstAccelDecelSpeedLimit(P
0,Pf,V0,Vf,Vsl,a_max,a_min);

fcn_plot_pathVelVsDistConstAccelSpeedLimit(P0,Pf,V0
,Vf,Vmax,fraction_at_P1,fraction_at_P2,velocity_fig
_num,max_colorbar_speed)

    %% Basic example - equal accel and decel with
negative speed limit - INTENTIONALLY BREAKS CODE,
NEG SPEED LIMIT
    P0= [-10 10]; % Initial Position (in meters)
    Pf= [10 10]; % Final Position (in meters)
    V0= 0;      % Initial Velocity (in
meters/second)
    Vf= 10;       % Final Velocity (in
meters/second)
    Vsl= -5;      % Speed Limit (meters/second)
```

```
    a_max= 3;     % Max Constant Acceleration (in
meters/second^2)
    a_min= -3;    % Max Constant Deceleration (in
meters/second^2)

[Vmax,~,~,~,P1,P2,fraction_at_P1,fraction_at_P2] =
fcn_costs_FastestVelFromConstAccelDecelSpeedLimit(P
0,Pf,V0,Vf,Vsl,a_max,a_min);

fcn_plot_pathVelVsDistConstAccelSpeedLimit(P0,Pf,V0
,Vf,Vmax,fraction_at_P1,fraction_at_P2,velocity_fig
_num,max_colorbar_speed)
end
```

Appendix A. 1: Test Script File for Testing Speed Limit MATLAB Functions

Appendix B

MATLAB Speed Limit Function Code

```
function
[V_max,t_tot,t1,t2,P1,P2,fraction_at_P1,fraction_at
_P2] =
fcn_costs_FastestVelFromConstAccelDecelSpeedLimit(P
0,Pf,V0,Vf,Vsl,a_max,a_min,varargin)
%
% INPUTS:
%      P0: a vector (N x 2) of [x y] initial
position
%      Pf: a vector (N x 2) of [x y] final position
%      V0: a vector (N x 1) of the initial velocity
in the direction of the
%      path from P0 to Pf
%      Vf: a vector (N x 1) of the final velocity
in the direction of the
%      path from P0 to Pf
%      Vsl: a vector (N x 1) of the speed limit
(maximum, legally allowed,
%      velocity on the roadway) that must be
respected
%      a_max: a vector (N x 1) of the maximum
acceleration of the object
%      a_min: a vector (N x 1) of the maximum
deceleration of the object
%
%
% OUTPUTS:
%      V_max: a vector (N x 1) of the peak velocity
in the direction of the
%      path from P0 to Pf
%      t_tot: a vector (N x 1) of the total time
for the object to travel from P0 to Pf
```

```
%       t_peak_start: a vector (N x 1) of the time
it takes for the object to reach its first peak
velocity
%       t_peak_end: a vector (N x 1) of the time it
takes for the object to reach its second peak
velocity
%       P1: a vector (N x 2) of the [x y] position
at which the object
%       reaches its first peak velocity of each of
the N segments
%       P2: a vector (N x 2) of the [x y] position
at which the object
%       reaches its second peak velocity of each of
the N segments
%       fraction_at_P1: a vector (N x 1) of the
fraction (0 to 1) of
%       the segment length where the maximum speed
occurs for the first
%       segment
%       fraction_at_P2: a vector (N x 1) of the
fraction (1 to 2) of
%       the segment length where the maximum speed
occurs for the second
%       segment
%
% Examples:
%
% Basic Example
%           P0= [-10 10]; % Initial Position (in
meters)
%           Pf= [10 10];  % Final Position (in
meters)
%           V0= 0;        % Initial Velocity (in
meters/second)
%           Vf= 0;        % Final Velocity (in
meters/second)
%           Vsl=
```

```
%            a_max= 7;     % Max Constant
Acceleration (in meters/second^2)
%            a_min= -3;    % Max Constant
Deceleration (in meters/second^2)
%            fig_num = 1;  % (optional) plot results
in figure 1
%
%
fcn_costs_FastestVelFromConstAccelDecel(P0,Pf,V0,Vf
,a_max,a_min,fig_num)
%
% Advanced examples: See
script_test_fcn_costs_FastestVelFromConstAccelDecel
%
% This function was written on 2020_06_09 by M.
Taylor, mods by S. Brennan
% Questions or comments? mft5296@psu.edu or
sbrennan@psu.edu
%
% This function was edited on 2021_03_01 by J.
Parsons
% Questions or comments? jup381@psu.edu
%
% Revision history:
% 2020_06_12 - Wrote the code by M. Taylor
% 2020_06_14 - Minor cleanup of comments by S.
Brennan
%            - Added varargin to allow optional
figure number to be sent
%              into function
%            - Added more input checking (for
negative velocities)
%            - Renamed function to not confuse it
with others in future
%            - Added a do_debug flag for debugging,
inlcuding color plot of
%              velocities
```

```
%             - Added a bounds check on initial and
final velocities to be
%               greater than zero
%             - Fixed typo on m-file name in example
%             - Fixed inequality check on
a_max/a_min to avoid zero values,
%               which cause division by zero errors
in the code
%             - Fixed big in minimum allowable
velocity calculation
%             - Vectorized the function to allow
multiple inputs
% 2020_07_08 - Fixed error in velocity plotting
where missing velocity
%               segments between points
% 2021_02_01
% - Added more comments and link to main script
(sbrennan@psu.edu)
% 2020_03_01 - Added Speed Limit Constraints (J.
Parsons)

flag_check_inputs = 1; % Set equal to 1 to check
the input arguments
flag_do_debug = 0;     % Set equal to 1 for
debugging

%% check input arguments?
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%    ______                        _
%   |_    _|                      | |
%     | |   _ __   _ __   _    _  | |_ ___
%     | | | '_ \| '_ \| | | | | __/ __|
%    _| |_| | | | |_) | |_| | |_\__ \
%   |_____|_| |_| .__/ \__,_|\__|___/
%               | |
%               |_|
```

```
% See:
http://patorjk.com/software/taag/#p=display&f=Big&t
=Inputs
% or www.askapache.com/online-tools/figlet-ascii/
(standard setting)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

N_segments = length(P0(:,1));

if flag_check_inputs
    % Are there the right number of inputs?
    if nargin<6 || nargin>7
        error('Incorrect number of input
arguments')
    end

    % Are the input vectors with the correct number
of columnts
    if length(P0(1,:)) ~= 2
        error('P0 must have 2 columns');
    end
    if length(Pf(1,:)) ~= 2
        error('Pf must have 2 columns');
    end
    if length(V0(1,:)) ~= 1
        error('V0 must be a Nx1 vector in the
direction of the path');
    end
    if length(Vf(1,:)) ~= 1
        error('Vf must be a Nx1 vector in the
direction of the path');
    end
    if length(Vsl(1,:)) ~= 1
        error('Vsl must be a Nx1 vector in the
direction of the path');
    end
    if length(a_max(1,:)) ~= 1
```

```
        error('a_max must be a Nx1 vector in the
direction of the path');
    end
    if length(a_min(1,:)) ~= 1
        error('a_min must be a Nx1 vector in the
direction of the path');
    end

    % Check consistency of lengths of input vectors
    if length(Pf(:,1)) ~= N_segments
        error('Pf must have same number of rows as
P0');
    end
    if length(V0(:,1)) ~= N_segments
        error('V0 must have same number of rows as
P0');
    end
    if length(Vf(:,1)) ~= N_segments
        error('Vf must have same number of rows as
P0');
    end
    if length(Vsl(:,1)) ~= N_segments
        error('Vsl must have same number of rows as
P0');
    end
    if length(a_max(:,1)) ~= N_segments
        error('a_max must have same number of rows
as P0');
    end
    if length(a_min(:,1)) ~= N_segments
        error('a_min must have same number of rows
as P0');
    end


    % Do the accelerations / decelerations make
sense?
```

```
    if any(V0<0)
        error('Initial velocity should be a zero or
positive value.')
    end
    if any(Vf<0)
        error('Final velocity should be a zero or
positive value.')
    end
    if any(Vsl<0)
        error('Speed Limit velocity should be a
positive value.')
    end
    if any(V0>Vsl)
        error('Initial velocity should be less than
or equal to the Speed Limit velocity.')
    end
    if any(Vf>Vsl)
        error('Final velocity should be less than
or equal to the Speed Limit velocity.')
    end
    % Do the initial and final velocities make
sense?
    if any(a_max<=0)
        error('Max constant acceleration should be
a positive value.')
    elseif any(a_min>=0)
        error('Max constant deceleration should be
a negative value.')
    end
        if any(Vsl == 0)
        error('Vsl = 0, You are not allowed to
move.')
        end
end

% Does user want to show the plots?
```

```
flag_do_plots = 0;  % Default to zero
if 8 == nargin  % A seventh optional argument
indicates a figure is given
    fig_num = varargin{1};  % The fig number is the
first optional argument
    figure(fig_num);  % Open the figure
    flag_do_plots = 1; % Set debug on (to make the
figure appear)
else
    if flag_do_debug  % Also make a figure if raw
debugging
        flag_do_plots = 1; %#ok<UNRCH>
        fig = figure; %  create new figure with
next default index
        fig_num = fig.Number;
    end
end


%% Solve for Peak velocity and time
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%
%
%   __  __       _
%  |  \/  |     (_)
%  | \  / | __ _ _ _ __
%  | |\/| |/ _` | | '_ \
%  | |  | | (_| | | | | |
%  |_|  |_|\__,_|_|_| |_|
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%

% This method uses constant acceleration kinematic
equations relating
% distance, velocity, and acceleration. See the PPT
file accompanying this
```

```
% function for derivation and details of where the
code comes from.

segment_lengths= sum((Pf-P0).^2,2).^0.5; % Distance
from point 0 to point 2, e.g. total distance of
path
segment_distance_at_max_speed = (Vf.^2-V0.^2-
2*a_min.*segment_lengths)./(2*(a_max-a_min)); %
Distance from point 0 to point 2 where velocity is
maximum

% Initialize V_max vectors
V_max = 0*a_max;  % Set it equal to zero to start

% is the peak velocity location beyond the length
of the path? Check this.
indices_at_limits =
find(segment_distance_at_max_speed>segment_lengths)
;
if ~isempty(indices_at_limits)
    % If S1>S2 than the object accelerates
indefinitely through the final
    % position and therefore the peak velocity is
at S2 and is equal to Vf
    V_max(indices_at_limits) =
(V0(indices_at_limits).^2 +
2*a_max(indices_at_limits).*segment_lengths(indices
_at_limits)).^0.5;

segment_distance_at_max_speed(indices_at_limits)=se
gment_lengths(indices_at_limits);

    % Check for errors, e.g. that the max velocity
is less than the
    % demanded final velocity, within 100 times the
numerical tolerance
    % (eps)
    threshold = 0.001;
```

```
    if
any(V_max(indices_at_limits)<(Vf(indices_at_limits)
-threshold))
        for i=1:length(indices_at_limits)
            fprintf(1,'Index: %d \t  Max allowable
velocity, V_max: %.2f \t Demanded final velocity,
Vf:
%.2f\n',indices_at_limits(i),V_max(indices_at_limit
s(i)),Vf(indices_at_limits(i)));
        end
        error('The final velocity, Vf, could not be
met with the given acceleration constraints, which
limit the maximum final velocity V_max. See the
workspace for V_max and Vf.')
    end
end

% Is the peak velocity point before the path
begins?
indices_at_limits =
find(segment_distance_at_max_speed<=0);

% Yes, the peak velocity occurs BEFORE the path
starts.
if ~isempty(indices_at_limits)
    %If S1<S0 than the object decelerates
indefinitely to the final
    % position and therefore the peak velocity is
at S0 and is equal to V0

segment_distance_at_max_speed(indices_at_limits)=0;
    V_max(indices_at_limits)=
V0(indices_at_limits);

    % Calculate the minimum velocity we achieve
with full deceleration
```

```
    V_min = (V0.^2 +
2*a_min.*segment_lengths).^0.5;  % Fixed this bug
on 2020_06_14

    % Sometimes V_min will give complex numbers,
for example with segments
    % where we are not checking. We need to just
look at the real parts
    V_min = real(V_min);

    % Check to see if the minimum allowable
velocity is still greater than
    % the final velocity. The only way this can
happen is if we are
    % commanding full deceleration, then getting to
a speed that is SLOWER
    % than the minimum velocity requested at the
end point. If that
    % actually happens, then there SHOULD have been
a point in the segment
    % where we could have sped up even slightly,
rather than full
    % deceleration along the entire path. So we
should NOT be entering the
    % next section of code ever, unless there's
something really wrong with
    % the code.

    % So... we check for errors, e.g. that the max
velocity is less than the
    % demanded final velocity, within 100 times the
numerical tolerance
    % (eps)
    threshold = 0.001; %
    if any(V_min>(Vf+threshold))  % Fixed this bug
on 2020_06_14
```

```
        fprintf(1,'ERROR DETECTED - Final velocity
was able to slow down more than min velocity limit:
\n');
        fprintf(1,'\tINPUTS: \n');
        fprintf(1,'\t P0\n');
        fprintf(1,'\t\t %.2f\n',P0);
        fprintf(1,'\t Pf\n');
        fprintf(1,'\t\t %.2f\n',Pf);
        fprintf(1,'\t V0\n');
        fprintf(1,'\t\t %.2f\n',V0);
        fprintf(1,'\t Vf\n');
        fprintf(1,'\t\t %.2f\n',Vf);
        fprintf(1,'\t a_max: %.2f\n',a_max);
        fprintf(1,'\t a_max: %.2f\n',a_min);
        fprintf(1,'\t RESULT: \n');
        fprintf(1,'\t V_min\n');
        fprintf(1,'\t\t %.14f\n',V_min);
        fprintf(1,'\t Vf\n');
        fprintf(1,'\t\t %.14f\n',(Vf+100*eps));
        fprintf(1,'\t V_min - Vf\n');
        fprintf(1,'\t\t %.14f\n',V_min-
(Vf+threshold));

        for i=1:length(indices_at_limits)
            fprintf(1,'Index that broke: %d
\n',indices_at_limits(i));
        end
        error('The final velocity could not be met
with the given deceleration constraints, which
limit the minimum final velocity. See the workspace
for V_min and Vf.')
        %         V_max = nan;
        %         t_tot = nan;
        %         t_peak = nan;
        %         P1 = nan;
        %         fraction_at_max_speed = nan;
        %         return;
```

```
    end
end


% Finally, handle the valid cases
indices_valid =
find((segment_distance_at_max_speed>0)&(segment_dis
tance_at_max_speed<=segment_lengths));
if ~isempty(indices_valid)
    % If 0<S1<S2 than the max velocity is
calculated based on max
    % acceleration, initial velocity and distance
traveled to reach peak
    % velocity.
    V_max(indices_valid) =
(V0(indices_valid).^2+(2*a_max(indices_valid).*segm
ent_distance_at_max_speed(indices_valid))).^0.5;

end

% Total time (also minimum time) for obect to
travel from point 0 to point 2
t_tot = ((V_max-V0)./a_max)+((Vf-V_max)./a_min);

% Fix the special cases
t_tot(segment_distance_at_max_speed==segment_length
s)=
((V_max(segment_distance_at_max_speed==segment_leng
ths)-
V0(segment_distance_at_max_speed==segment_lengths))
./a_max(segment_distance_at_max_speed==segment_leng
ths));
t_tot(segment_distance_at_max_speed==0) =
((Vf(segment_distance_at_max_speed==0)-
V_max(segment_distance_at_max_speed==0))./a_min(seg
ment_distance_at_max_speed==0));
```

```
t_peak= (V_max-V0)./a_max; % Time to reach peak
velocity

% Find the point of max speed
P1 = (Pf-
P0).*segment_distance_at_max_speed./segment_lengths
+ P0; % Linearly interpolates to find P1 based on
distance from P0 to S1

% Find the fraction of the path where max speed
occurs
fraction_at_P1 =
segment_distance_at_max_speed./segment_lengths;

% All code up to this point assumes that there is
one peak max velocity.
% The following code evaluates if there are two max
peak velocities ("flat
% top")
% Speed Limit Function Code
indices_speed_limit = find(V_max > Vsl);
if ~isempty(indices_speed_limit)    % Maximum
velocity exceeds the speed limit

    % Update V_max to Vsl - the maximum speed will
just be the speed limit
    V_max(indices_speed_limit,:) =
Vsl(indices_speed_limit,:);

    % CALCULATE VALUES FOR THE UPSLOPE
    % P1 must be recalculated since platau cuts it
off
    New_distance_P1(indices_speed_limit,:) = ...
        (Vsl(indices_speed_limit,:).^2 -
V0(indices_speed_limit,:).^2)./(2.*a_max(indices_sp
eed_limit,:));
```

```
    t1(indices_speed_limit,:) =
(Vsl(indices_speed_limit,:)-
V0(indices_speed_limit,:))./...
        a_max(indices_speed_limit,:);     % Time
elapsed during increasing velocity in first frame

    % CALCULATE VALUES FOR THE DOWNSLOPE
    Downslope_distance(indices_speed_limit,:) = ...
        (Vf(indices_speed_limit,:).^2 -
Vsl(indices_speed_limit,:).^2)./(2.*a_min(indices_s
peed_limit,:));
    t3(indices_speed_limit,:) =
(Vf(indices_speed_limit,:)-
Vsl(indices_speed_limit,:))./...
        a_min(indices_speed_limit,:);     % Time
elapsed during decreasing velocity in third frame

    % CALCULATE VALUES FOR THE PLATEAU
    distance_on_the_peak = ...
        segment_lengths(indices_speed_limit,:) -
New_distance_P1(indices_speed_limit,:)...
        -
Downslope_distance(indices_speed_limit,:); %
distance where velocity is steady
    t2(indices_speed_limit,:) = ...

distance_on_the_peak./Vsl(indices_speed_limit,:);
% Time elapsed during constant velocity in second
frame


    % Total time (also minimum time) for obect to
travel from point 0 to point 2
    t_tot = ...
        t1(indices_speed_limit,:) + ...
        t2(indices_speed_limit,:) + ...
        t3(indices_speed_limit,:);
```

```
    % Update the segment distances
    segment_distance_to_P1 = New_distance_P1; %
Distance from point 0 to point 1 where velocity is
accelerating
    segment_distance_to_P2 = distance_on_the_peak +
New_distance_P1; % Distance from point 0 to point 2

    % Find the fraction of the path where max speed
occurs
    fraction_at_P1 =
segment_distance_to_P1./segment_lengths;
    fraction_at_P2 =
segment_distance_to_P2./segment_lengths;

    % Update P1 and P2 calculations
    P1 = P0 + fraction_at_P1*(Pf-P0);
    P2 = P0 + fraction_at_P2*(Pf-P0);

else % no platau, just one peak
    t1 = t_peak;
    t2 = t_peak;
    P2 = P1;
    fraction_at_P2 = fraction_at_P1;
end



%% Plot the results (for debugging)?
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%
%    ______        _
%   |  __ \      | |
%   | |  | | ___| |__   _   _  __ _
%   | |  | |/ _ \ '_ \| | | |/ _` |
%   | |__| |  __/ |_) | |_| | (_| |
%   |_____/ \___|_.__/ \__,_|\__, |
%                             __/ |
%                            |___/
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%
if flag_do_plots
    max_colorbar_speed = 40;
    %
fcn_plot_pathVelocitiesVsDistanceConstAccel(P0,Pf,V
0,Vf,a_max,a_min,V_max,segment_distance_at_max_spee
d, fraction_at_max_speed,
fig_num,max_colorbar_speed);

fcn_plot_pathVelVsDistConstAccelSpeedLimit(P0,Pf,V0
,Vf,V_max,fraction_at_P1,fraction_at_P2,fig_num,max
_colorbar_speed);
    %figure(1)

%plot(t1,Velocities_1,t2,Velocities_2,t3,Velocity_3
)
end

end

%% Script code used to plot velocity vs time curves
% g = 9.81;  % m/s^2
% v0 = 10;  % m/s
% v2 = 5;   % m/s
% amax = 0.7*g;
% amin = 0.4*g;
% t = 1:0.1:10;
% v_increasing = v0+t*amax;
% v_decreasing = v2+fliplr(t)*amin;
% figure(38738);plot(t,v_increasing,'g-
',t,v_decreasing,'r-');
```

Appendix B. 1: Outermost Function fcn_costs_FastestVelFromConstAccelDecelSpeedLimit

```
function
fcn_plot_pathVelVsDistConstAccelSpeedLimit(P0,Pf,V0
```

```
,Vf,Vmax,fraction_at_P1,fraction_at_P2,fig_num,max_
colorbar_speed)
%
% INPUTS:
%      P0: a vector (N x 2) of [x y] initial
position (meters)
%      Pf: a vector (N x 2) of [x y] final position
(meters)
%      V0: a vector (N x 1) of the initial velocity
in the direction of the
%      path from P0 to Pf
%      Vf: a vector (N x 1) of the final velocity
in the direction of the
%      path from P0 to Pf
%      V_max: a vector (N x 1) of the maximum
velocity in the direction of
%      the path from P0 to Pf
%      fraction_at_P1: a vector (N x 1) of the
fraction (0 to 1) of
%      the segment length where the maximum speed
occurs.
%      fraction_at_P2: a vector (N x 1) of the
fraction (1 to 2) of
%      the segment length where the maximum speed
occurs.
%      fig_num: The figure number to plot to.
%      max_colorbar_speed: A scalar representing
the max speed to use on
%      the colorbar axis.
%
% Examples:
%
% Basic Example
%         fig_num = 1;  % plot results in figure 1
%         P0= [-10 10]; % Initial Position (in
meters)
```

```
%         Pf= [10 10];  % Final Position (in
meters)
%         V0= 0;        % Initial Velocity (in
meters/second)
%         Vf= 0;        % Final Velocity (in
meters/second)
%         Vmax = 20;    % Max Velocity (in
meters/second)
%         fraction_at_P1 = 0.2;  % Fraction length
of max vel wrt path
%         fraction_at_P2 = 0.8;  % Fraction length
of max vel wrt path
%         max_colorbar_speed = 20; % Top speed
shown in color bar
%
fcn_plot_pathVelocitiesVsDistanceConstAccelSpeedLim
it(P0,Pf,V0,Vf,Vmax,fraction_at_P1,fraction_at_P2,f
ig_num,max_colorbar_speed)
%
%
% This function was written on 2020_07_08 by S.
Brennan
% Questions or comments? sbrennan@psu.edu

% This function was edited on 2021_03_01 by J.
Parsons
% Questions or comments? jup381@psu.edu

% Revision history:
%
% 2020_07_08 - Fixed error in velocity plotting
where missing velocity
%              segments between points
% 2020_03_01 - Added Speed Limit Constraints (J.
Parsons)
```

```
flag_check_inputs = 1; % Set equal to 1 to check
the input arguments

%% check input arguments?
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%    ______                          _
%   |_    _|                        | |
%     | |   _ __   _ __   _    _| |_ ___
%     | | | '_ \| '_ \| | | | __/ __|
%    _| |_| | | | |_) | |_| | |_\__ \
%   |_____|_| |_| .__/ \__,_|\__|___/
%               | |
%               |_|
% See:
http://patorjk.com/software/taag/#p=display&f=Big&t
=Inputs
% or www.askapache.com/online-tools/figlet-ascii/
(standard setting)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

N_segments = length(P0(:,1));

if flag_check_inputs
    % Are there the right number of inputs?
    if nargin~=9
        error('Incorrect number of input
arguments')
    end
    % Are the input vectors with the correct number
of columnts
    if length(P0(1,:)) ~= 2
        error('P0 must have 2 columns');
    end
    if length(Pf(1,:)) ~= 2
        error('Pf must have 2 columns');
    end
    if length(V0(1,:)) ~= 1
```

```
        error('V0 must be a Nx1 vector in the
direction of the path');
    end
    if length(Vf(1,:)) ~= 1
        error('Vf must be a Nx1 vector in the
direction of the path');
    end
    if length(Vmax(1,:)) ~= 1
        error('V_max must be a Nx1 vector in the
direction of the path');
    end
    if length(fraction_at_P1(1,:)) ~= 1
        error('fraction_at_P1 must be a Nx1 vector
in the direction of the path');
    end
    if length(fraction_at_P2(1,:)) ~= 1
        error('fraction_at_P3 must be a Nx1 vector
in the direction of the path');
    end

    % Check consistency of lengths of input vectors
    if length(Pf(:,1)) ~= N_segments
        error('Pf must have same number of rows as
P0');
    end
    if length(V0(:,1)) ~= N_segments
        error('V0 must have same number of rows as
P0');
    end
    if length(Vf(:,1)) ~= N_segments
        error('Vf must have same number of rows as
P0');
    end
    if length(Vmax(:,1)) ~= N_segments
        error('V_max must have same number of rows
as P0');
    end
```

```
    if length(fraction_at_P1(:,1)) ~= N_segments
        error('fraction_at_P1 must have same number
of rows as P0');
    end
    if length(fraction_at_P2(:,1)) ~= N_segments
        error('fraction_at_P3 must have same number
of rows as P0');
    end

    % Do the velocities make sense?
    if any(V0<0)
        error('Initial velocity should be a zero or
positive value.')
    end
    if any(Vf<0)
        error('Final velocity should be a zero or
positive value.')
    end
    if any(Vmax<0)
        error('Maximum velocity should be a zero or
positive value.')
    end


end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%
%
%   __  __           _
%  |  \/  |         (_)
%  | \  / | __ _ _ _ _ __
%  | |\/| |/ _` | | | '_ \
%  | |  | | (_| | | | | | |
%  |_|  |_|\__,_|_|_|_| |_|
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%
```

```
% Decide how many points are to be plotted in each
segment's rising and
% falling portions
num_points = 50;

% Initialize arrays for distances, velocities, x
and y points
all_velocities = [];
all_points     = [];

% Loop over each segment, building vectors of
distance, velocity, x, and y
% Note that the distances along each point may be
wrong for curved arcs
for i_segment = 1:N_segments
    [points, velocities] =
fcn_plotcalc_kinematic_profile_from_key_vel_dist_sp
eed_limit(...
        num_points,...
        P0(i_segment,:),...
        Pf(i_segment,:),...
        V0(i_segment,1),...
        Vf(i_segment,1),...
        Vmax(i_segment,1),...
        fraction_at_P1(i_segment,1),...
        fraction_at_P2(i_segment,1));

    all_points = [all_points; points]; %#ok<AGROW>
    all_velocities = [all_velocities; velocities];
%#ok<AGROW>

end

% % FOR DEBUGGING
% figure(999);
% clf; hold on; grid minor; xlabel('X (km)');
ylabel('Y (km)');
% plot(all_points(:,1),all_points(:,2),'r.');
```

```
% plot(P0(:,1),P0(:,2),'gx');
% plot(Pf(:,1),Pf(:,2),'go');

% Create vectors that connect adjacent points
vectors        = (all_points(2:end,:) -
all_points(1:end-1,:));

% Calculate the length of the vectors
vector_lengths = sum(vectors.^2,2).^0.5;

% Calculate the distance by adding up all the
vector lengths
all_distances = [0;cumsum(vector_lengths)];

%% Make the plots using the data calculated above
% OPTIONAL - plot the figure now? No reason to do
this since identical
if 1 == 0
    figure(fig_num+100); % Add 100 to prevent
overlap with other fig numbers
    hold on % allow multiple plot calls
    xlabel('Distances [km]');
    ylabel('Velocities [m/s]');
    grid on; grid minor;
    plot(all_distances/1000,all_velocities);   %
Plot velocities
    hold off
end

figure(fig_num);
hold on % allow multiple plot calls
xlabel('Distances [km]');
ylabel('Velocities [m/s]');
grid on; grid minor;

% PLot the distance versus velocity data in a plain
plot
```

```
plot(all_distances/1000,all_velocities);   % Plot
velocities

% Create a multi-colored plot
x = all_distances'/1000;
y = all_velocities';
z = zeros(size(x));
col = all_velocities';  % This is the color, vary
with velocity
h_surf = surface([x;x],[y;y],[z;z],[col;col],...
    'facecol','no',...
    'edgecol','interp',...
    'linew',2);
% [max_vel,ind_max_vel] = max(all_velocities);
%
text(all_distances(ind_max_vel),max_vel,path_type)
caxis([0 max_colorbar_speed]);
colorbar;


% FOR DEBUGGING:
% % Plot the input data (NOTE: clean this up!)
% for i_segment = 1:N_segments
%     calc_velocities = [V0(i_segment,1);
V_max(i_segment,1); Vf(i_segment,1)];
%     calc_distances = [0;
segment_distance_at_max_speed(i_segment,1);
segment_lengths(i_segment,1)];
%     if i_segment>1
%         jump_distance = sum((P0(i_segment,:) -
Pf(i_segment-1,:)).^2,2).^0.5;
%         calc_distances = calc_distances +
last_distance + jump_distance;
%     end
%
plot(calc_distances/1000,calc_velocities,'ro');  %
Plot velocities at key calculation points
```

```
%     last_distance = calc_distances(end);
% end

end % Ends the function
```

Appendix B. 2: Second Deepest Function fcn_plot_pathVelVsDistConstAccelSpeedLimit

```
function [points, velocities] = ...

fcn_plotcalc_kinematic_profile_from_key_vel_dist_sp
eed_limit(...

num_points,P0,Pf,V0,Vf,Vmax,fraction_at_P1,fraction
_at_P2)
% [points, velocities] =
fcn_plotcalc_kinematic_profile_from_key_vel_dist(nu
m_points,P0,Pf,V0,Vf,Vmax,fraction_at_P1,fraction_a
t_P2)
% Fills in the velocity and points using constant
acceleration assumptions
% between key points given in velocity
distributions for a single segment.
% The key points are the initial and final points
of the segment, plus the
% location in the segment where a maximum velocity
is achieved represented
% as a fraction of the distance from the start to
the end point. If the
% initial, maximum, and final velocity of the
segment is given, then this
% code fills in an additional number of points
(given by num_points) on
% both the rising and falling edges. This is useful
to plot the points on
% the segment, the distances, or the velocities
within the segment.
```

```
%
% INPUTS:
%       num_points: the number of points to use on
the rising and falling
%       portions of the kinematic trajectory
%
%       P0: a vector (N x 2) of [x y] initial
position
%
%       Pf: a vector (N x 2) of [x y] final position
%
%       V0: a vector (N x 1) of the initial velocity
in the direction of the
%       path from P0 to Pf
%
%       Vf: a vector (N x 1) of the final velocity
in the direction of the
%       path from P0 to Pf
%
%       Vmax: a vector (N x 1) of the maximum
velocity in the direction of
%       the path from P0 to Pf
%
%       fraction_at_P1: a vector (N x 1) of the
fraction (0 to 1) of
%       the segment length where the maximum speed
occurs.
%
%       fraction_at_P2: a vector (N x 1) of the
fraction (1 to 2) of
%       the segment length where the maximum speed
occurs.
%
% Examples:
%
% Basic Example
```

```
%           num_points = 50; % Use 50 points in up,
another 50 in down
%           P0= [-10 10]; % Initial Position (in
meters)
%           Pf= [10 10];  % Final Position (in
meters)
%           V0= 0;        % Initial Velocity (in
meters/second)
%           Vf= 0;        % Final Velocity (in
meters/second)
%           Vmax= 7;      % Max speed
%           fraction_at_P1= 0.3; % Hit peak speed
at 30% of dist
%           fraction_at_P2= 0.7; % Hit peak speed
at 70% of dist
%
%      [points,velocities] =
fcn_plotcalc_kinematic_profile_from_key_vel_dist(nu
m_points,P0,Pf,V0,Vf,Vmax,fraction_at_P1,fraction_a
t_P2)
%
%
% This function was written on 2020_07_08 by S.
Brennan
% Questions or comments? sbrennan@psu.edu

% This function was edited on 2021_03_01 by J.
Parsons
% Questions or comments? jup381@psu.edu

% Revision history:
%
% 2020_07_08 - Fixed error in velocity plotting
where missing velocity
%              segments between points
% 2021_03_01 - Added Speed Limit Constraints (J.
Parsons)
```

```
flag_check_inputs = 1; % Set equal to 1 to check
the input arguments

%% check input arguments?
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%    ______                          _
%   |_    _|                        | |
%     | |   _ __   _ __   _   _  ___| |_ ___
%     | |  | '_ \| '_ \| | | |/ __| __/ __|
%    _| |_ | | | | |_) | |_| | (__| |_\__ \
%   |_____|_| |_| .__/ \__,_|\___|\__|___/
%               | |
%               |_|
% See:
http://patorjk.com/software/taag/#p=display&f=Big&t
=Inputs
% or www.askapache.com/online-tools/figlet-ascii/
(standard setting)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

N_segments = length(P0(:,1));

if flag_check_inputs
    % Are there the right number of inputs?
    if nargin~=8
        error('Incorrect number of input
arguments')
    end
    % Are the input vectors with the correct number
of columnts
    if length(P0(1,:)) ~= 2
        error('P0 must have 2 columns');
    end
    if length(Pf(1,:)) ~= 2
        error('Pf must have 2 columns');
    end
    if length(V0(1,:)) ~= 1
```

```
        error('V0 must be a Nx1 vector in the
direction of the path');
    end
    if length(Vf(1,:)) ~= 1
        error('Vf must be a Nx1 vector in the
direction of the path');
    end
    if length(Vmax(1,:)) ~= 1
        error('V_max must be a Nx1 vector in the
direction of the path');
    end
    if length(fraction_at_P1(1,:)) ~= 1
        error('fraction_at_P1 must be a Nx1 vector
in the direction of the path');
    end
    if length(fraction_at_P2(1,:)) ~= 1
        error('fraction_at_P2 must be a Nx1 vector
in the direction of the path');
    end

    % Check consistency of lengths of input vectors
    if length(Pf(:,1)) ~= N_segments
        error('Pf must have same number of rows as
P0');
    end
    if length(V0(:,1)) ~= N_segments
        error('V0 must have same number of rows as
P0');
    end
    if length(Vf(:,1)) ~= N_segments
        error('Vf must have same number of rows as
P0');
    end
    if length(Vmax(:,1)) ~= N_segments
        error('V_max must have same number of rows
as P0');
    end
```

```
    if length(fraction_at_P1(:,1)) ~= N_segments
        error('fraction_at_P1 must have same number
of rows as P0');
    end
    if length(fraction_at_P2(:,1)) ~= N_segments
        error('fraction_at_P2 must have same number
of rows as P0');
    end
    % Do the velocities make sense?
    if any(V0<0)
        error('Initial velocity should be a zero or
positive value.')
    end
    if any(Vf<0)
        error('Final velocity should be a zero or
positive value.')
    end
    if any(Vmax<0)
        error('Maximum velocity should be a zero or
positive value.')
    end


end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%
%    __  __           _
%   |  \/  |      (_)
%   | \  / | __ _ _ _ __
%   | |\/| |/ _` | | '_ \
%   | |  | | (_| | | | | |
%   |_|  |_|\__,_|_|_| |_|
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%
```

```
% Decide how many points are to be plotted in each
segment's rising and
% falling portions
num_points = 50;

velocity_rising = linspace(V0, Vmax,num_points)';
velocity_steady = linspace(Vmax,Vmax,num_points)';
velocity_falling = linspace(Vmax, Vf,num_points)';
velocities = [velocity_rising; velocity_steady;
velocity_falling];

% Find the distance traversed
d_total = sum((Pf-P0).^2,2).^0.5;
d_P1 = fraction_at_P1*d_total;
d_P2 = fraction_at_P2*d_total;

% Find a_max and a_min
a_max = (Vmax.^2 - V0.^2)/(2*d_P1);
a_min = (Vf.^2 - Vmax.^2)/(2*(d_total - d_P2));

if a_min==0
    fraction_at_P1 = 1;
end
if a_max==0
    fraction_at_P2 = 0;
end
if fraction_at_P1 == fraction_at_P2
    flag_onlyaccelerating = 1;  % This is a special
case where the answer is to ONLY accelerate - there
is no flat portion nor drop at the end
else
    flag_onlyaccelerating = 0;
end
% case where we accelerate and then decelerate (no
steady speed)
if flag_onlyaccelerating == 1 && a_min~=0 &&
a_max~=0
```

```
    distances_rising = (velocity_rising.^2 -
V0.^2)/(2*a_max);
    distances_falling = d_P2 + (velocity_falling.^2
- Vmax.^2)/(2*a_min);
    distances_steady =  d_P2*ones(num_points,1);
% case where we have a steady speed and then
decelerate
elseif (fraction_at_P1==0) &&
(flag_onlyaccelerating == 0) && (fraction_at_P2 ~=
1)
    distances_rising = 0*ones(num_points,1);
    distances_falling = d_P2 + (velocity_falling.^2
- Vmax.^2)/(2*a_min);
    distances_steady =
linspace(distances_rising(end),distances_falling(1)
,num_points)';
% case where we accelerate and then have steady
speed
elseif (fraction_at_P2 == 1) &&
(flag_onlyaccelerating == 0) && (fraction_at_P1 ~=
0)
    distances_rising = (velocity_rising.^2 -
V0.^2)/(2*a_max);
    distances_falling = d_total*ones(num_points,1);
    distances_steady =
linspace(distances_rising(end),distances_falling(1)
,num_points)';
% case where we only accelerate
elseif flag_onlyaccelerating == 1 && fraction_at_P1
== 1
    distances_rising = (velocity_rising.^2 -
V0.^2)/(2*a_max);
    distances_falling = d_total*ones(num_points,1);
    distances_steady =
linspace(distances_rising(end),distances_falling(1)
,num_points)';
% case where we only deccelerate
```

```
elseif flag_onlyaccelerating == 1 && fraction_at_P1
== 0
    distances_rising = 0*ones(num_points,1);
    distances_falling = d_P2 + (velocity_falling.^2
- Vmax.^2)/(2*a_min);
    distances_steady =
linspace(distances_rising(end),distances_falling(1)
,num_points)';
% case where we maintain a steady velocity from
start to end
elseif (V0 == Vmax) && (Vmax == Vf)
    distances_rising = 0*ones(num_points,1);
    distances_falling = d_total*ones(num_points,1);
    distances_steady =
linspace(distances_rising(end),distances_falling(1)
,num_points)';
% case where we accelerate, obtain steady speed,
then decelerate
else
    distances_rising = (velocity_rising.^2 -
V0.^2)/(2*a_max);
    distances_falling = d_P2 + (velocity_falling.^2
- Vmax.^2)/(2*a_min);
    distances_steady =
linspace(distances_rising(end),distances_falling(1)
,num_points)';
end

distances = [distances_rising; distances_steady;
distances_falling];

if any(isnan(distances))
    warning('Nan detected in distance calculations
within plotting function:
fcn_plotcalc_kinematic_profile_from_Key_vel_dist');
end
```

```
% Now find the xy points by interpolating between
end and start points
points = (Pf-P0).*(distances./d_total) + P0;
end
```

Appendix B. 3: Deepest Function fcn_plotcalc_kinematic_profile_from_key_vel_dist_speed_limit

BIBLIOGRAPHY

[1] J. Meyer, H. Becker, P. M. Bösch, and K. W. Axhausen, "Autonomous vehicles: The next jump in accessibilities?," *Res. Transp. Econ.*, vol. 62, pp. 80–91, Jun. 2017, doi: 10.1016/j.retrec.2017.03.005.

[2] Z. Chen, F. He, Y. Yin, and Y. Du, "Optimal design of autonomous vehicle zones in transportation networks," *Transp. Res. Part B Methodol.*, vol. 99, pp. 44–61, May 2017, doi: 10.1016/j.trb.2016.12.021.

[3] D. J. Fagnant and K. Kockelman, "Preparing a nation for autonomous vehicles: opportunities, barriers and policy recommendations," *Transp. Res. Part A Policy Pract.*, vol. 77, pp. 167–181, Jul. 2015, doi: 10.1016/j.tra.2015.04.003.

[4] V. V. Dixit, S. Chand, and D. J. Nair, "Autonomous Vehicles: Disengagements, Accidents and Reaction Times," *PLoS One*, vol. 11, no. 12, p. e0168054, Dec. 2016, doi: 10.1371/journal.pone.0168054.

[5] H. Yang, D. McBlane, C. Boyd, C. Beal, and S. Brennan, "Vehicle road departure detection using anomalies in dynamics," in *2016 American Control Conference (ACC)*, 2016, pp. 6314–6319, doi: 10.1109/ACC.2016.7526662.

[6] D. Dolgov and S. Thrun, "Autonomous driving in semi-structured environments: Mapping and planning," in *2009 IEEE International Conference on Robotics and Automation*, 2009, pp. 3407–3414, doi: 10.1109/ROBOT.2009.5152682.

[7] H. Clark and J. Feng, "Semi-Autonomous Vehicles," *Proc. Hum. Factors Ergon. Soc. Annu. Meet.*, vol. 59, no. 1, pp. 781–785, Sep. 2015, doi: 10.1177/1541931215591241.

[8] M. Kocsis, N. Susmann, J. Buyer, and R. Zollner, "Safety concept for autonomous vehicles that operate in pedestrian areas," in *2017 IEEE/SICE International Symposium on System Integration*

(SII), 2017, pp. 841–846, doi: 10.1109/SII.2017.8279327.

[9] J. Teizer, B. S. Allread, C. E. Fullerton, and J. Hinze, "Autonomous pro-active real-time construction worker and equipment operator proximity safety alert system," *Autom. Constr.*, vol. 19, no. 5, pp. 630–640, Aug. 2010, doi: 10.1016/j.autcon.2010.02.009.

[10] R. Kanan, O. Elhassan, and R. Bensalem, "An IoT-based autonomous system for workers' safety in construction sites with real-time alarming, monitoring, and positioning strategies," *Autom. Constr.*, vol. 88, pp. 73–86, Apr. 2018, doi: 10.1016/j.autcon.2017.12.033.

ACADEMIC VITA

Jonathan Parsons
Jup381@psu.edu

EDUCATION

The Pennsylvania State University and **Schreyer Honors College (SHC)**, State College, PA
Bachelor of Science in Mechanical Engineering (Graduation: May 2021)
Honor's Student (SHC) **Dean's List 6 Semesters**
President's Freshman Award (Fall 2017)

TECHNICAL SKILLS

AutoCAD	Solid Works	Mechanical Skills	SAP (Volvo CE)	Prosis
Enovia SMARTEAM		Microsoft Office	MATLAB	
CI Fundamentals	Creo	Windchill		

RELEVANT EXPERIENCE

Design Engineering Intern **JLG Hagerstown, MD (June 1, 2020-August 2020)**

- Created drawings in Creo to prototype an ergonomic machine securement design
- Collaborated with Logistics, the Design Department, and Shipping to develop a safe and ergonomic machine tie-down design
- Developed communicational skills through creating and presenting various machine securement designs in cross-departmental meetings
- Presented a complete machine securement design for both the boom and scissor lifts to the company

Operations Engineering Intern **JLG McConnellsburg, PA (June 3, 2019-August 16, 2019)**

- Developed a sophisticated hose kitting system and racks to minimize footprint in cell and improve functionality of the hose kitting/assembly process to increase efficiency leading to increased production
- Assisted my team in the implementation of Project Ignite to revolutionize the whole assembly line process, eliminating excess parts/shortages, improve material presentation, material flow, and layout of worker cells to increase uptime
- Collaborated with assembly workers weekly to develop efficient and user-friendly procedures leading to increased production and a happier work environment
- Worked alongside Industrial Engineers to develop new machinery for the assembly line as well as floor layout and work instructions to maintain a safe and efficient work environment
- Insured the safety of team members in the hose kitting process by completing a Job Safety Analysis (JSA) making all employees in the area (current and future) aware of all potential hazards and proper procedures to ensure the well being of themselves and those around them

Quality Assurance Engineering Intern **Volvo CE Shippensburg, PA (April 2018-May 3, 2019)**

- Repaired and replaced 10-150 broken parts monthly from the line
- Created Quality Notifications on inferior parts to alert Supplier Development (SD) Engineers to increase vendor production performance
- Collaborated with SD Engineers several times a month to identify and improve vendor production quality
- Updated Standard Kaizen reports for weekly production meetings to increase awareness on vendor quality to help decrease line shutdowns due to inferior or late parts and improve vendor performance
- Transferred material systematically within the Shippensburg plant to improve warehouse storage accuracy and increase production uptime
- Programed and updated Secondary Information Displays (SID Co-Pilot) to new operation certificate which increased customer satisfaction in the more user-friendly interface

- Created re-work and new parts orders for suppliers to repair parts that were broken from the line to increase productivity

Farm Equipment Operator & Technician Parsons Farm Shade Gap, PA (2012-present)

- Operated various models of John Deere, Farmall, International, Case, and Steiner tractors as well as implements including; spreaders, industrial snow blower, dumping station, Freightliner triaxle, and Ingersoll Rand telehandler
- Performed routine maintenance including tire replacement, battery testing and replacement, belt and chain adjustment/replacement, and oil changes and lubrication to increase uptime and reduce costs in operation

HOBBIES/INTERESTS

Vehicle Repair and Restoration

- Rebuilt a 1961 Thunderbird including full drivetrain removal, interior removal and replacement, and body work and paint
- Tore down and replaced blown head gaskets in a Ford 390 cubic inch engine
- Rebuilt a gear-reduction Mopar starter
- Restored a 1985 Dodge Diplomat that had been sitting parked outside for 15 years
- Removed and rebuilt a "Swing Away" steering column in a 1965 Thunderbird which involved tearing out the driver's side of the dashboard and removing the swing away track and mechanisms
- Repaired/rebuilt the whole side of a plastic bumper cover on a 1991 Mercury with fiberglass and body filler since reproductions were no longer available, the hole was about a 6in*8in from an accident and previous to the repair the owner was told by a body shop that it was unrepairable and would likely no longer pass inspection, the car is still in use today
- Performed Engine Diagnostics on a 3.8 Liter Ford V6 equipped with first generation Onboard Diagnostics (OBD) system, engine had a very rough idle and was prone to stalling, local mechanics were unable to diagnose because they no longer had the equipment to interface with the computer in the car, after extensive research I discovered an alternative method to interface with the car and found the faulty sensors

Certified Steam Engine Operator

- Graduate of the Williams Grove PA Steam School and Cumberland Valley Antique Engine and Machinery Association in the operation of Steam Engines
- Operate a 1916 Peerless Geiser steam traction engine and a 1926 Rumely Oil-Pull (kerosene tractor) at various antique tractor shows in PA

ADDITIONAL EXPERIENCE

Co-Organist/Pianist, Choir Leader, Quartet Accompanist/Tenor Singer, Spring Run, PA (2015-present)

Co-Organist and Chorale Accompanist, Greencastle, PA (January 6, 2019-August 4, 2019)