Modified Trajectory Shaping Guidance for Autonomous Path Following Control of Platooning Ground Vehicles

Ishmaal T. Erekson

MODIFIED TRAJECTORY SHAPING GUIDANCE FOR AUTONOMOUS PATH FOLLOWING CONTROL OF PLATOONING GROUND VEHICLES

by

Ishmaal T. Erekson

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Mechanical Engineering

UTAH STATE UNIVERSITY
Logan, Utah

2016

ABSTRACT

Modified Trajectory Shaping Guidance for Autonomous Path Following Control of Platooning Ground Vehicles

by

Ishmaal T. Erekson, Master of Science

Utah State University, 2016

Major Professor: Dr. Rajnikant Sharma
Department: Mechanical and Aerospace Engineering

This thesis proposes a modification of trajectory shaping guidance to provide more accurate path convergence in curved paths. The object of this thesis is to apply this simple guidance law to platooning control to ensure all vehicles in the platoon converge to a desired constant radius path at a desired vehicle separation distance. To show the viability of this new guidance law, it is shown mathematically to be exponentially stable. It is also confirmed through simulations and on ground robots.

(86 pages)

PUBLIC ABSTRACT

Modified Trajectory Shaping Guidance for Autonomous Path Following Control of Platooning Ground Vehicles

by

Ishmaal T. Erekson, Master of Science

Utah State University, 2016

Major Professor: Dr. Rajnikant Sharma
Department: Mechanical and Aerospace Engineering

The purpose of this thesis is to develop a control method to keep a platoon of vehicles on a desired path and separated a desired distance from each other. There is already a control law developed for a single vehicle to follow a virtual target called trajectory shaping guidance. This thesis proposes a modification to trajectory shaping guidance to provide more accurate path convergence in curved paths. It is applied to a platoon of vehicles with the intent of ensuring all vehicles in the platoon converge to a desired path and are at the desired distance from each other. To ensure the viability of this new guidance law, it is shown mathematically that the vehicles will converge and then confirmed through simulations and on ground robots.

To my beloved wife Lucila.

ACKNOWLEDGMENTS

I would like to thank Dr. Rajnikant Sharma for his mentoring, funding, and help with my research. I am also grateful for his guidance and help in my life aside from my research. I express my gratitude for my other committee members, Dr. Rees Fullmer and Dr. David Geller, for their help and guidance as well. I would like to thank these men for the effort they put into helping their students in and out of the classroom and especially for the time they have spent to help me understand subject material that has aided me in completing this thesis. I would like to show appreciation to Dr. Don Cripps for his contagious enthusiasm for controls that got me hooked and has helped me find a great passion for engineering.

I would also like to thank Christine Spall, Kathy Phippen, and Tricia Brandenburg for the great effort they put into helping me succeed as a graduate student.

The members of the RISC Lab at USU also deserve my gratitude. Spencer Maughan not only set up the platform on which my thesis was run, but also worked with me on other research projects and has helped me gain an excitement for research.

My family has been an incredible support and without them I would not have made it here. I would like to thank my mother and father for instilling in me a desire to chase my dreams and for their many hours spent proof reading papers. Most of all I would like to thank my wife Lucila. Her patience and unfailing love and support have been my motivating factor in all I do.

Ishmaal T. Erekson

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACRONYMS

LQR	Linear Quadratic Regulator (an optimal control method)
PD	Proportional and Derivative (a control method)
PID	Proportional Integrator and Derivative (a control method)
RISC MAAP	Robust Intelligent Sensing and Control Multi-Agent Analysis Platform
RMS	Root Mean Square
ROS	Robot Operating System
USU	Utah State University

CHAPTER 1
INTRODUCTION

Autonomous Vehicles are becoming a popular platform for performing many tasks that could be dangerous, mind-numbing, difficult, or even impossible to achieve using a human driver. The ability to accurately control a system to follow a given path opens the door to many possibilities in both aerial and ground vehicles. Path-following control is being used in military, mining, agriculture, automotive, and aerial vehicles to increase safety, improve efficiency, and push the boundaries of what is currently possible. Many different path-following methods have been developed and studied and each has different advantages and disadvantages. Examples include pure-pursuit [1], virtual force framework [2], nonlinear guidance logic [3], and trajectory shaping [4].

Platooning is when a group of vehicles are controlled to move as a unit. There are generally two components of platooning control: longitudinal and lateral. Longitudinal control is the control of the spacing of the vehicles in the platoon, while lateral control is the control of the vehicles direction. It is important to study them together since they often affect each other. The ability to provide longitudinal and lateral control together in a platoon has the possibility of providing many positive impacts.

In [5], many of the benefits of platooning vehicles for an Automated Highway System are pointed out. These benefits would include increased highway capacity, which would also provide a smaller road size requirement, leading to better use of land; increased safety even in the presence of low-visibility conditions such as fog, snow, or rain; providing opportunity for those currently unable to drive to have personal transportation such as the blind, elderly, or disabled; increased fuel efficiency and decreased emissions; increased commercial efficiency; and better rider experience. This article points to the conclusion that an automated highway system is not just a fantasy, but a realizable future with great benefits. The benefits of platooning could be even further realized through mining operations, military border patrol, agricultural applications, and countless other possibilities. The purpose of this thesis is to propose a new control method to achieve longitudinal and lateral control of a platoon to obtain accurate path following.

1.1 Current Platooning Research

Much research has been done on longitudinal platooning control in a large variety of techniques [6–12] such as fuzzy-sliding mode [11], LQR [9], and many others. These also often show that there is a limitation in platoon size for achieving string stability. String stability means that if there is a disturbance in a vehicle of the platoon, the disturbance will not grow as it moves to the next vehicles. Furthermore, Jancović and Bamieh show that for some of the existing platooning controls the platoon stabilizability degrades as the number of vehicles in the platoon increases and will become unstabilizable within a limit [9].

Although, not as greatly studied, there is also research going on for lateral control of platoons using a variety of control techniques [10–12]. Of the lateral controls for platooning found, there seemed to be either a lack of accurate path following or a lack of simplicity in implementation.

Inherent in the control methods of LQR and PID is the difficulty of tuning parameters of the controller to appropriately run the system. Sliding mode control can be very robust to parameter uncertainty, which would be very advantageous in platooning vehicles, however, comes with complexity of derivation and may suffer from actuator chatter.

The desire of this thesis was to find a simple control method to achieve path convergence of all vehicles in the platoon at a set distance between each vehicle. Missile guidance control methods are very simple control laws and have been well studied for a long time. These very simple guidance laws often assume small angles since missiles will have trajectories with large radii. These missile guidance laws are often studied on ground robots for ease of experimentation and seem to work well, but can suffer in their ability to converge accurately to a path. Ferrin was able to use a version of pure pursuit on a ground vehicle to achieve path following of the vehicle to the path of its lead vehicle [13]. This experiment is a step in the direction of using simple missile guidance laws for the use of platooning control.

However, it was only used to control one vehicle to follow a lead target. The idea of using missile guidance laws for platooning control will be further pursued in this thesis specifically by modifying the technique of trajectory shaping guidance.

1.2 Contribution

Trajectory shaping guidance has been defined in [14] and [4], and will be shown in greater detail in Chapter 2. This is a missile guidance law that uses a virtual target to control the vehicle as shown in Fig. 2.1. The contributions of this thesis are as follows.

- A modified form of the trajectory shaping guidance law is developed to achieve path convergence for constant radius paths. This includes straight line paths. Chapter 2 discusses how this law is modified to provide more accurate path following. These results were submitted to the International Conference on Intelligent Robots and Systems (IROS).
- A mathematical analysis shows that using this modified law causes path convergence, while the original law converges to a slight offset. These conclusions are also shown in simulations and hardware results.
- The modified trajectory shaping law is applied to control a platoon of vehicles by making each vehicle the virtual target of the vehicle behind it as shown in Fig. 3.1. This provides a simple longitudinal and lateral control for a platoon of any size that will converge to a straight or curved path of constant radius. This is discussed in greater detail in Chapter 3.
- A mathematical analysis shows that using this modified law causes path convergence of all vehicles in the platoon. These conclusions are also shown in simulations and hardware results.
- Much of the platooning work in this thesis will be submitted to American Institute of Aeronautics and Astronautics (AIAA) Journal of Guidance, Control, and Dynamics.

- A multi-robot platform was developed using ROS and a motion capture system to implement the proposed trajectory shaping modification on a platoon of vehicles.

This thesis is organized in the following manner. Chapter 2 discusses a modification to the trajectory shaping guidance law to achieve more accurate path convergence. There are simulation and hardware results showing its advantage. Chapter 3 analytically shows the convergence of this modified guidance method and applies it to a platoon of vehicles. Chapter 4 shows and discusses the simulation and hardware results of this new guidance law on a platoon. Chapter 5 gives conclusions and possible future research.

CHAPTER 2
TRAJECTORY SHAPING GUIDANCE MODIFICATION FOR CONVERGENCE TO CURVED PATHS OF SMALL RADIUS

This chapter will discuss the benefits and limitations of trajectory shaping guidance. A modification will be proposed to achieve more accurate path convergence, even in the case of circular paths with small radii. This modified version will then be compared to the original trajectory shaping guidance law through simulations and hardware results to show the advantages of the modified version.

2.1 Trajectory Shaping Overview

Trajectory shaping is a simple missile guidance law that was developed in [14] as an "Optimal Guidance Law for Lag-Free Autopilot" and was tested in simulation and on hardware in [4]. Many path following methods, such as trajectory shaping, are developed as missile guidance laws and are meant to follow straight line and large radius paths. However, to apply this law to ground vehicles in an urban environment would necessitate the ability to follow a small radius trajectory while still maintaining accurate path convergence. Trajectory shaping guidance was chosen in this thesis for its simplicity to implement and its proven ability in path following on straight and curved paths. The purpose of this chapter is to modify the trajectory shaping algorithm to improve the accuracy of its path convergence on smaller radius paths and compare this modified version with the original.

As shown in Fig. 2.1, trajectory shaping guidance uses a virtual target with a position, velocity, and heading, to set the vehicle's lateral acceleration (a_{lat}). In [4], Ratnoo et al. showed that with the implementation of velocity control for the virtual target, these commands achieve convergence to a straight line and near convergence to a circular path.

As derived in [14] and [4], (2.1) is the commanded lateral acceleration of the vehicle, where V_v is the vehicle velocity, d is the distance from the vehicle to the virtual target,

γ_v is the vehicle heading angle measured from the positive x-axis, λ is the angle of the vector from the vehicle to the virtual target measured from the positive x-axis, and γ_t is the virtual target's heading measured from the positive x-axis. The vehicle velocity is assumed constant and thus the virtual target needs to change its velocity to achieve the desired distance between itself and the vehicle. This velocity control is given in (2.2) where V_t is the virtual target velocity and d^* is the desired distance between the vehicle and the virtual target.

$$a_{lat} = \frac{V_v^2}{d}\big(4(\lambda - \gamma_v) + 2(\lambda - \gamma_t)\big) \tag{2.1}$$

$$V_t = V_v \frac{d^*}{d} \tag{2.2}$$

2.1.1 Trajectory Shaping Limitation

Ratnoo et al. [4] showed that trajectory shaping guidance will cause the vehicle to

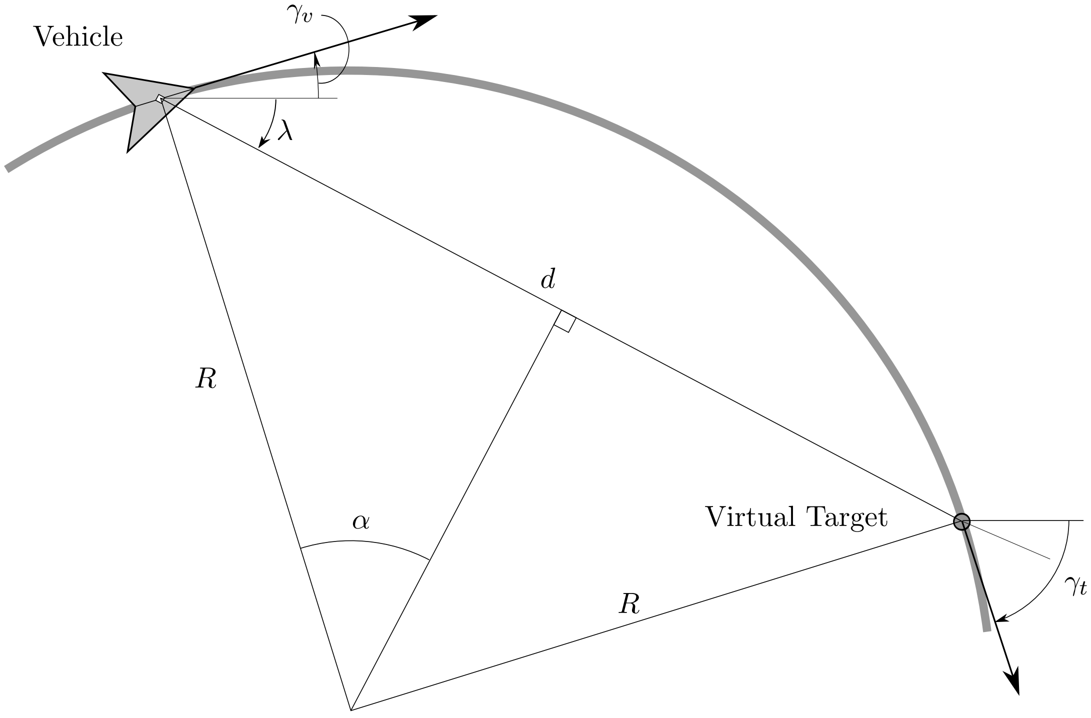


Fig. 2.1: Trajectory Shaping

converge to a path with constant radius. However, this is under the assumption of small angles and $d^* << 2R$. They found that in reality the vehicle will actually converge to an offset of the path radius when $d^* << 2R$ is not met. In many applications $d^* << 2R$ is a safe assumption, especially for missile guidance control, where the radius of the desired path will be very large. However, the purpose of this thesis is to apply this law to ground vehicle control where the path radius may be small. Furthermore, any offset to vehicle convergence would build with each vehicle in a platoon. With a small radius, theoretically the d^* value should be able to decrease such that it would provide $d^* << 2R$. However, decreasing d^* increases the effect of noise in the system and can cause instability. While d^* must be less than or equal to $2R$, there may be advantages to having its value close to $2R$ to help reduce the effects of noise in the system and increase stability. This chapter's purpose is to explore a modification of trajectory shaping to achieve the ability of the vehicle to closely follow a path of constant radius even when the condition $d^* << 2R$ is not met.

2.2 Modification

In order to obtain convergence to smaller radius paths, it is proposed that the following modification be made to the trajectory shaping guidance method.

$$a_{lat} = \frac{V_v^2}{d}\big(4\sin(\lambda - \gamma_v) + 2\sin(\lambda - \gamma_t)\big) \tag{2.3}$$

As can be seen, the only modification from (2.1) is to take the sine of the two angles of interest. In a small angle approximation the sine of the angle is assumed to be the angle ($\sin(\alpha) \approx \alpha$). The trajectory shaping guidance equation appears to possibly be a small angle approximation, thus the sine modification was an effort to not assume small angles. Throughout this thesis the original trajectory shaping method will be referred to as the regular method and the modified method will be referred to as the sine method.

There are several things that must occur for this to be a viable control method. The distance between the vehicle and virtual target (d) needs to converge to the commanded distance (d^*), the virtual target velocity (V_t) must converge to the commanded velocity

(V_v), and the vehicle position must converge to the desired path. These three errors are the criteria that will be analyzed to compare the sine method and regular method.

2.3 Simulation Results

Simulink was used to develop simulations to test this modified version of trajectory shaping on ground vehicles. A simplified kinematic bicycle model [15] was used as the ground vehicle model. The back wheels are assumed to be in line with the vehicle, thus the equations in [15] can be rewritten as

$$\dot{x} = V_v \cos(\gamma_v + \beta) \tag{2.4}$$

$$\dot{y} = V_v \sin(\gamma_v + \beta) \tag{2.5}$$

$$\dot{\gamma_v} = \frac{V_v \cos(\beta)}{\ell_f + \ell_r} \tan(\delta_f) \tag{2.6}$$

$$\beta = \tan^{-1}\left(\frac{\ell_r \tan \delta_f}{\ell_f + \ell_r}\right) \tag{2.7}$$

where x and y are the global x and y positions, δ_f is the wheel angle of the front wheels, ℓ_f is the distance from the vehicle point of interest (usually the center of gravity) to the front axle, ℓ_r is the distance from the vehicle point of interest to the back axle, and β is the vehicle slip angle.

This thesis assumes no slip in the vehicle wheels, thus the lateral acceleration was mapped to the desired wheel angle as follows, where R_t is the present turn radius and δ_f is the steering angle of the vehicle.

$$a_{lat} = \frac{V_v^2}{R_t} \quad \rightarrow \quad R_t = \frac{V_v^2}{a_{lat}}$$

$$\tan \delta_f = \frac{\ell_r + \ell_f}{\sqrt{R_t^2 - \ell_r^2}}$$

$$\tan\delta_f = \frac{\ell_r + \ell_f}{\sqrt{(\frac{V_v^2}{a_{lat}})^2 - \ell_r^2}}$$

$$\tan\delta_f = \frac{(\ell_r + \ell_f)a_{lat}}{\sqrt{V_v^4 - (\ell_r a_{lat})^2}}$$

$$\delta_f = \tan^{-1}\left(\frac{(\ell_f + \ell_r)a_{lat}}{\sqrt{V_v^4 - (\ell_r a_{lat})^2}}\right) \quad (2.8)$$

2.3.1 Straight Path

A straight line path was first simulated with the following parameters: $V_v = 0.5$m/s and $d^* = 0.2$m. A straight path is equivalent to an infinite radius, thus in this scenario, no matter what d^* is chosen to be, the condition $d^* << 2R$ will always be met. The results in Fig. 2.2, 2.3, and 2.4 show that, as expected, for a straight path, the original and modified versions of trajectory shaping will give similar results.

2.3.2 Circular Path

For the simulations on a radius path, the radius was selected to be 1m. This is to accommodate the test area for the hardware results discussed later in this chapter. Three different simulations were conducted all at the same velocity of 0.5 m/s. These simulations were with $d^* = 0.1$m, $d^* = 1$m, and $d^* = 1.9$m. This provides similar results to those found in [4], showing that when using the regular trajectory shaping guidance law, as the desired distance between the vehicle and the virtual target (d^*) approaches the diameter of the path ($2R$), the offset convergence increases. This means that the distance between the vehicle and target (d) will converge to an offset of the commanded distance (d^*), the velocity of the virtual target (V_t) will converge to an offset of the desired velocity (V_v), and the vehicle will converge to an offset of the actual path. However, as shown in 2.2, 2.3, and 2.4 by using the sine method, the distance, velocity, and path errors all converge to zero.

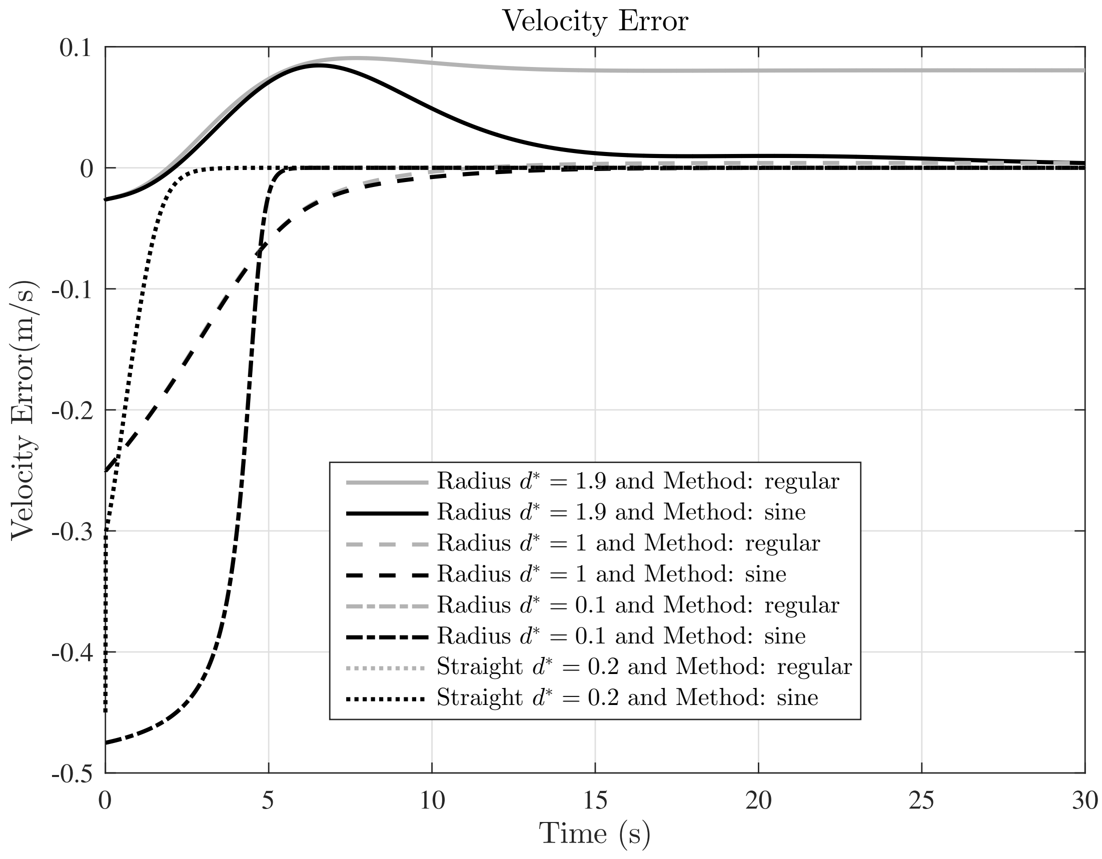


Fig. 2.2: Virtual Target Velocity Error in Simulation

2.3.3 Results

The RMS value was taken for all these runs to evaluate how much improvement was made using the sine modification. The values were only counted for the RMS calculation after 15 seconds of simulation to allow the vehicle to converge to steady state. The results shown in Table 2.1 show there is significant improvement using the sine modification.

2.4 Hardware Results

Using the same test criteria as in the simulations, experiments were also performed on hardware. Pololu's m3pi robot was selected for these experiments. This is a differential steering two-wheeled vehicle, so instead of mapping the lateral acceleration to a wheel angle, the lateral acceleration was mapped to the differential velocity of the right and left side of the robot as shown in (2.9) and (2.10) where V_r is the velocity of the right side of the robot,

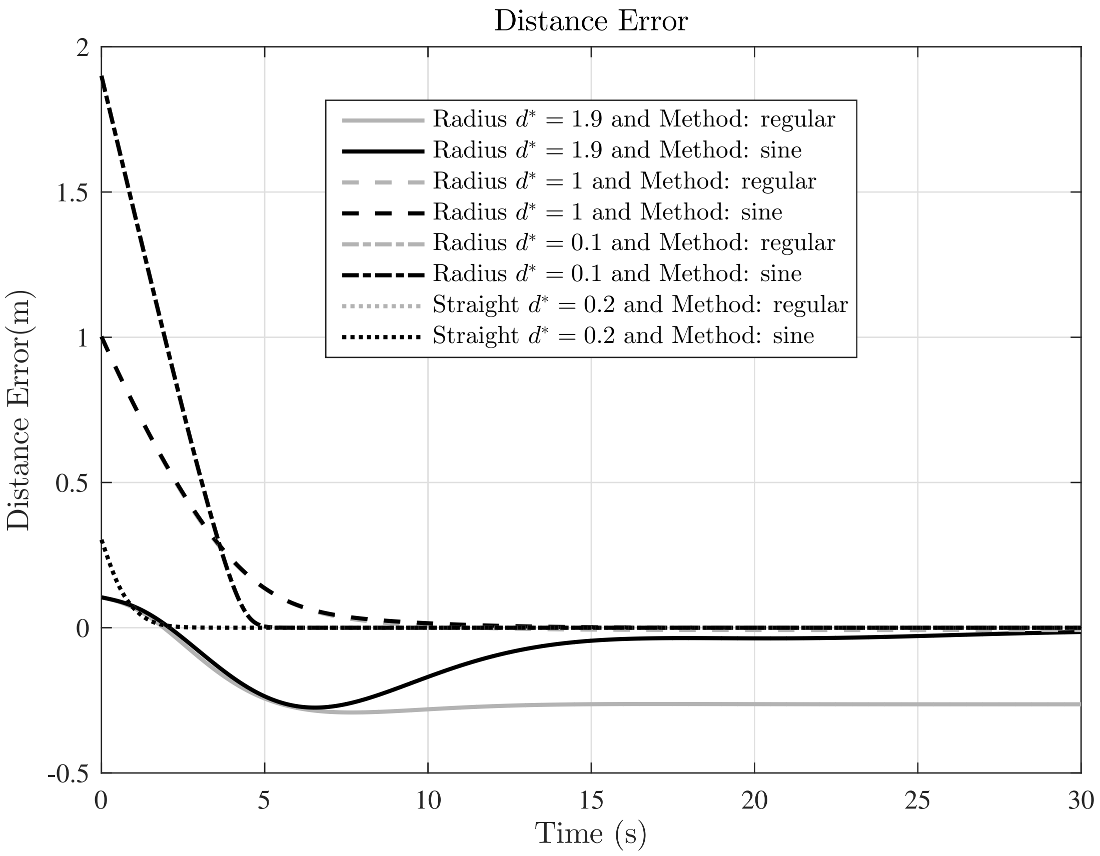


Fig. 2.3: Vehicle to Virtual Target Distance Error in Simulation

V_l is the velocity of the left side of the robot, V_c is the commanded velocity, and W is the distance between the two wheels.

$$V_r = V_c\left(1 + \frac{W}{2R_t}\right) \tag{2.9}$$

$$V_l = V_c\left(1 - \frac{W}{2R_t}\right) \tag{2.10}$$

The right and left velocities were then mapped to motor commands on the m3pi using experimental data. While not perfect, the m3pi motors were powerful enough and consistent enough to map the desired left and right velocities to left and right motor PWM signals. A proportional feedback controller was used to ensure the vehicle velocity converged to its commanded velocity. The gain was chosen to be very small so as to rely more heavily on the

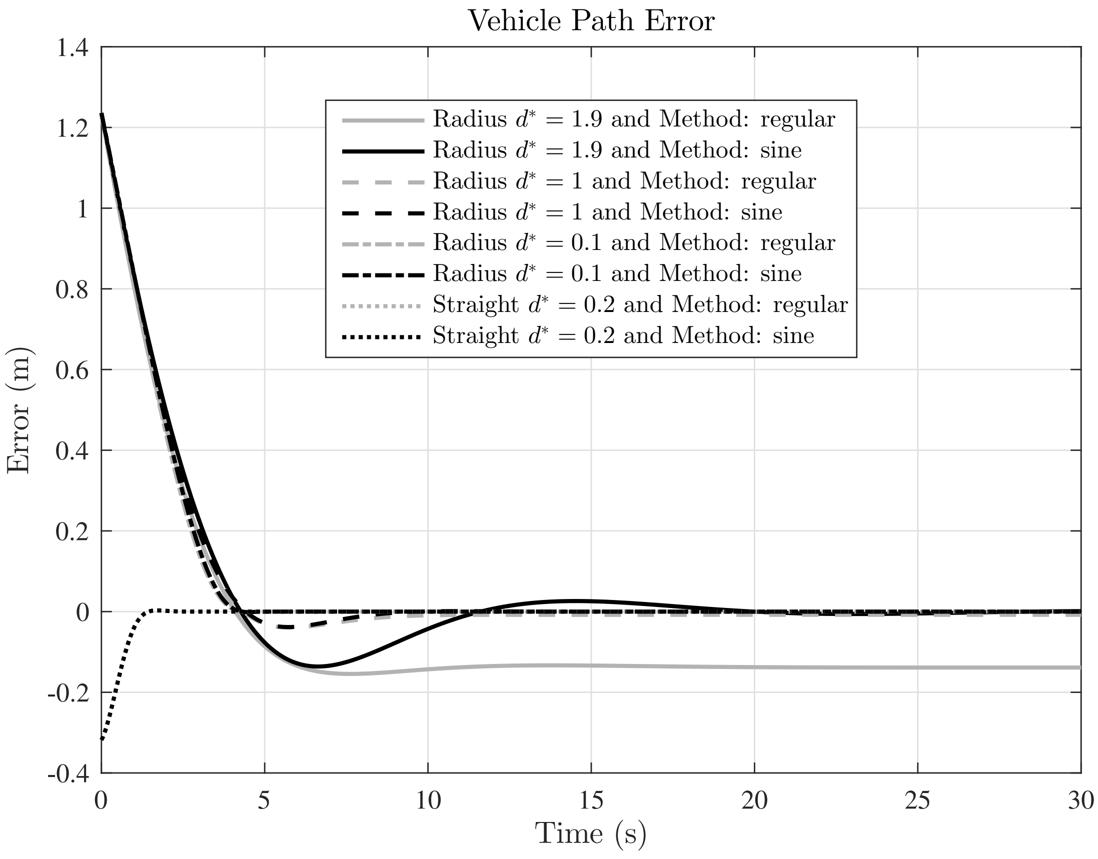


Fig. 2.4: Vehicle to Path Error in Simulation

above mentioned velocity mapping. This allowed the system to act similarly to having an open loop velocity control based on the mapping. However, any steady state errors would converge to zero.

Using ROS (Robot Operating System) and Utah State University's (USU) Robust Intelligent Sensing and Control Multi-Agent Analysis Platform (RISC MAAP) [16], the m3pi robot was able to receive position data from a motion capture system which required a unique template of reflective markers as seen in Fig. 2.5. This information was then used to apply the trajectory shaping tests. This system is able to give position data within around 2mm accuracy and heading data within about 2 degrees. Fig. 2.6 shows the general layout of how the RISC MAAP System works. The wireless communication was achieved using XBee Series 1 RF Modules. The ROS framework provides for easy passing of state information from the motion capture system to the main computer with ROS. The state

Table 2.1: RMS of Error in Simulation

	Control Method		
	Regular	Sine	Improvement
Velocity Errors (m/s)			
Straight Path	3.28e−14	3.13e−14	4.75%
Radius Path ($d^* = 0.1$m)	3.15e−7	2.85e−10	99.9%
Radius Path ($d^* = 1$m)	0.0039	0.0003	93.4%
Radius Path ($d^* = 1.9$m)	0.0804	0.0085	89.4%
Distance Errors (m)			
Straight Path	1.31e−14	1.25e−14	4.77%
Radius Path ($d^* = 0.1$m)	6.26e−8	5.70e−11	99.9%
Radius Path ($d^* = 1$m)	0.0078	5.21e−4	93.3%
Radius Path ($d^* = 1.9$m)	0.2633	0.0318	87.9%
Path Errors (m)			
Straight Path	1.15e−14	0	N/A
Radius Path ($d^* = 0.1$m)	6.30e−7	2.16e−9	99.7%
Radius Path ($d^* = 1$m)	0.0080	1.28e−5	99.8%
Radius Path ($d^* = 1.9$m)	0.1377	0.0095	93.1%

information could then be translated into motor commands based on the control algorithms and then sent wirelessly to the m3pi robots.

2.4.1 Straight Path

As expected, the straight path tests showed no decipherable difference between the sine method and regular method. Thus, the results of the straight path experiments on hardware will not be shown in this thesis.

2.4.2 Circular Path

The radius tests were conducted just as they were in the simulation tests. Setting $d^* = 0.1$ shows that the noise in the position and angle was enough to cause great amounts of error in the control of the robot as seen in Fig. 2.7. This is seen in both methods, but shows that there is advantage to increasing the d^* term.

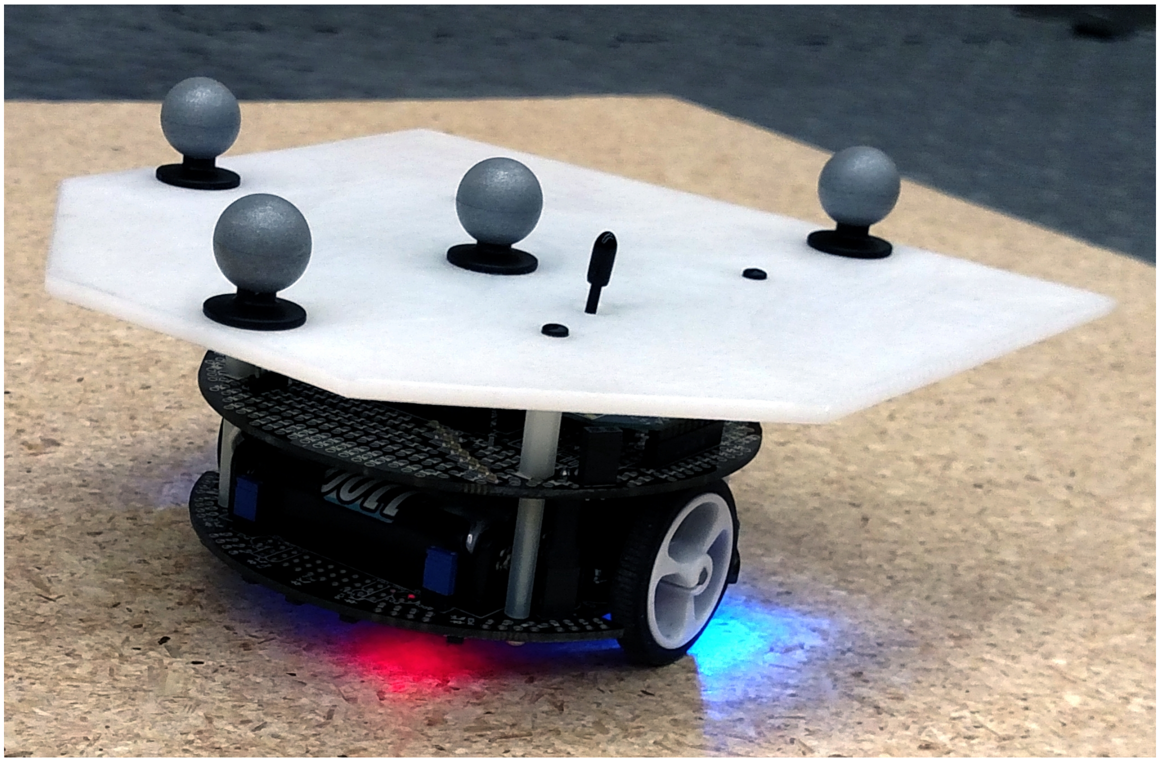


Fig. 2.5: Pololu m3pi Robot with Custom Reflector Template

2.4.3 Results

The results when increasing the d^* term show the advantages of using the sine modification. For $d^* = 1$m, the velocity, distance, and path errors are very similar, but the sine modification is slightly better as seen in Fig. 2.8(a), 2.8(b), and 2.8(c) respectively.

At $d^* = 1.9$m, the differences become significant. Fig. 2.9(a), 2.9(b), and 2.9(c) show the offset in the distance, velocity, and path convergence. There is also an offset in the sine method on the hardware due to noise in the system, physical parameter measurement uncertainties, disturbances such as unevenness in the ground, and communication delays. However, it is easily seen that the sine method achieves smaller errors.

Again, the RMS values of the errors were calculated only using values after 15 seconds to allow the system to reach steady state. These results shown in Table 2.2 confirm that using the sine modification improves convergence of the vehicle to the desired path.

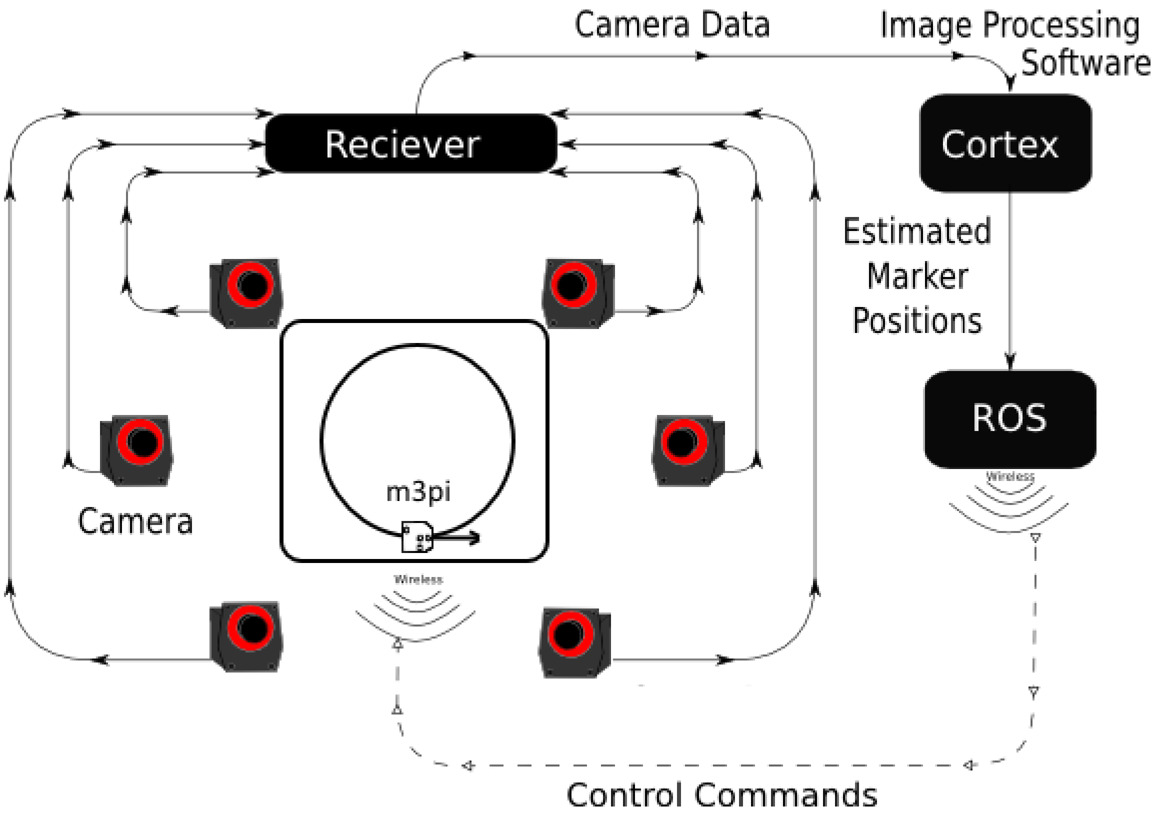


Fig. 2.6: USU's RISC MAAP System

Table 2.2: RMS of Error on Hardware

	Control Method		
	Regular	Sine	Improvement
Velocity Errors			
Radius Path ($d^* = 1$)	0.0110	0.0100	8.93%
Radius Path ($d^* = 1.9$)	0.1001	0.0478	52.3%
Distance Errors			
Radius Path ($d^* = 1$)	0.0265	0.0249	5.88%
Radius Path ($d^* = 1.9$)	0.3210	0.1689	47.4%
Path Errors			
Radius Path ($d^* = 1$)	0.0233	0.0186	20.0%
Radius Path ($d^* = 1.9$)	0.1688	0.0804	52.3%

2.5 Conclusion

This chapter shows that the proposed sine modification to the trajectory shaping guidance method improves path convergence in small radius paths. This allows for greater

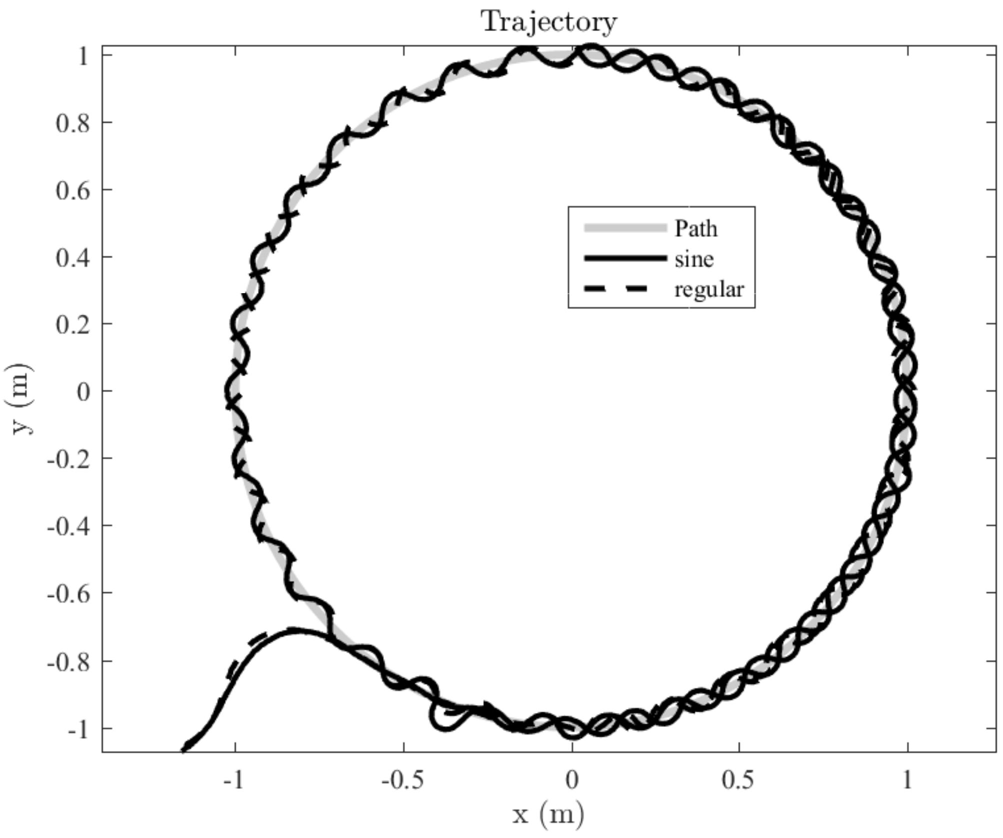
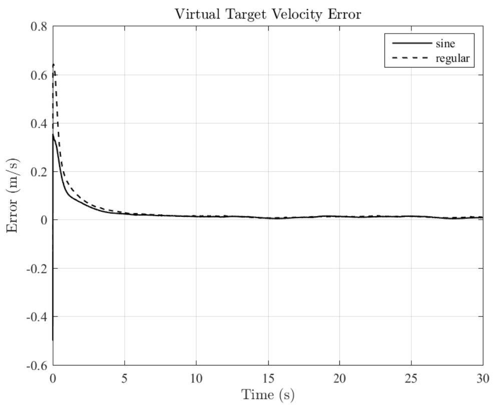


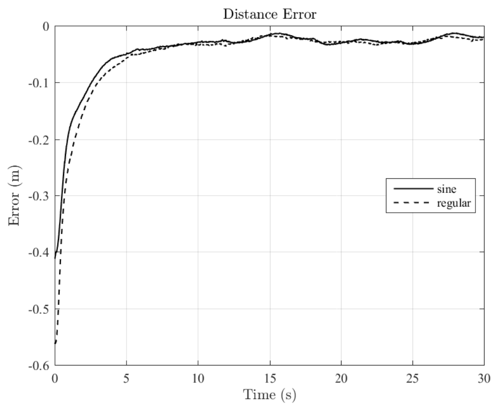
Fig. 2.7: Trajectory for 1(m) Path Radius at $d^* = 0.1$(m) on Hardware

flexibility in choosing the desired distance from the vehicle to the virtual target. In cases of small radius paths, this allows for a greater value d^* which helps to reduce the effects of noise in the system and to keep it stable.

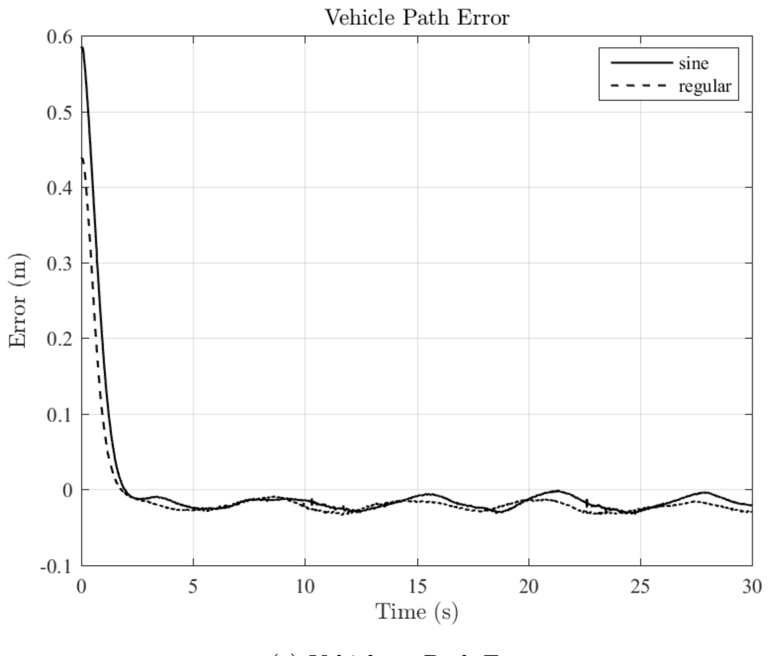
The advantage of being able to more accurately converge to a given path is helpful in applying this law to a platoon of vehicles. If a vehicle converges to an offset of the desired path, the vehicle behind would converge to an offset of that offset and the last vehicle in the platoon will be furthest from the path. It is desired to analyze the implementation of this modified trajectory shaping on a platoon of multiple vehicles to achieve accurate path convergence for each vehicle in the platoon.



(a) Virtual Target Velocity Error

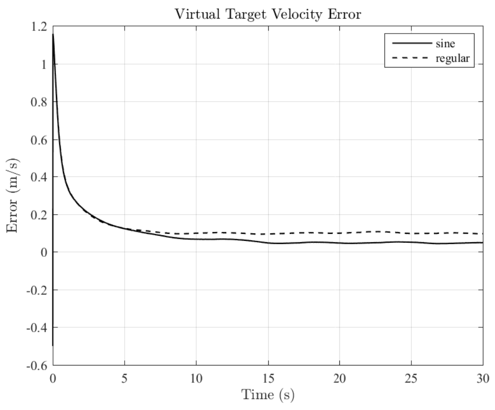


(b) Vehicle to Virtual Target Distance Error

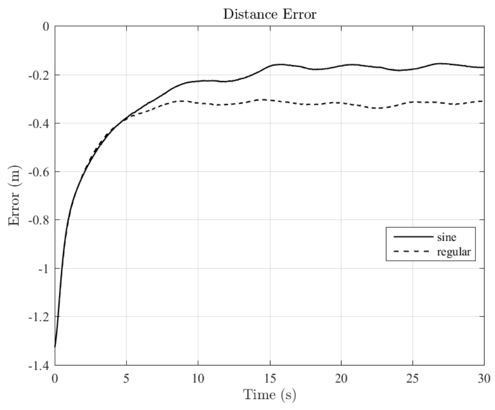


(c) Vehicle to Path Error

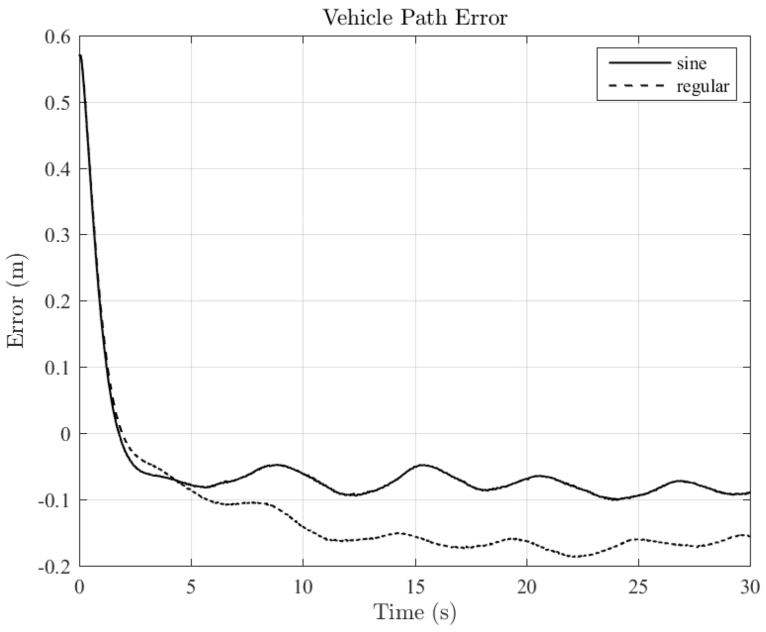
Fig. 2.8: Virtual Target Errors at $d^* = 1$(m) on Hardware



(a) Virtual Target Velocity Error



(b) Vehicle to Virtual Target Distance Error



(c) Vehicle to Path Error

Fig. 2.9: Virtual Target Errors at $d^* = 1.9$(m) on Hardware

CHAPTER 3
TRAJECTORY SHAPING APPLIED TO PLATOON CONTROL

This chapter mathematically shows the convergence of all vehicles to the desired path and separation distance for both the single vehicle case and for a platoon of n vehicles.

3.1 Definition of Variables

Trajectory shaping guidance is applied to platooning by treating each vehicle as the target of the vehicle behind it (see Fig. 3.1). Fig. 3.2 and 3.1 define the needed variables:

veh_i	$=$	Vehicle i
n	$=$	Number of Vehicles in the platoon
d^*	$=$	The desired distance between all vehicles and their targets
d_i	$=$	The actual distance between veh_i and its target (veh_{i-1})
γ_{v_i}	$=$	The heading angle of veh_i
γ_t	$=$	The virtual target's heading angle, also written γ_0
V_i	$=$	The velocity of veh_i
V_t	$=$	The virtual target velocity, also written V_0
λ_i	$=$	The angle of the line of sight vector from veh_i to veh_{i-1}
α_{t_i}	$=$	$\gamma_{v_{i-1}} - \lambda_i$
α_{v_i}	$=$	$\lambda_i - \gamma_{v_i}$
ω_{v_i}	$=$	The angular rate of the heading vector of veh_i
ω_{t_i}	$=$	The angular rate of the heading vector of veh_{i-1}
ω_{λ_i}	$=$	The angular rate of the line of sight vector of from veh_i to veh_{i-1}

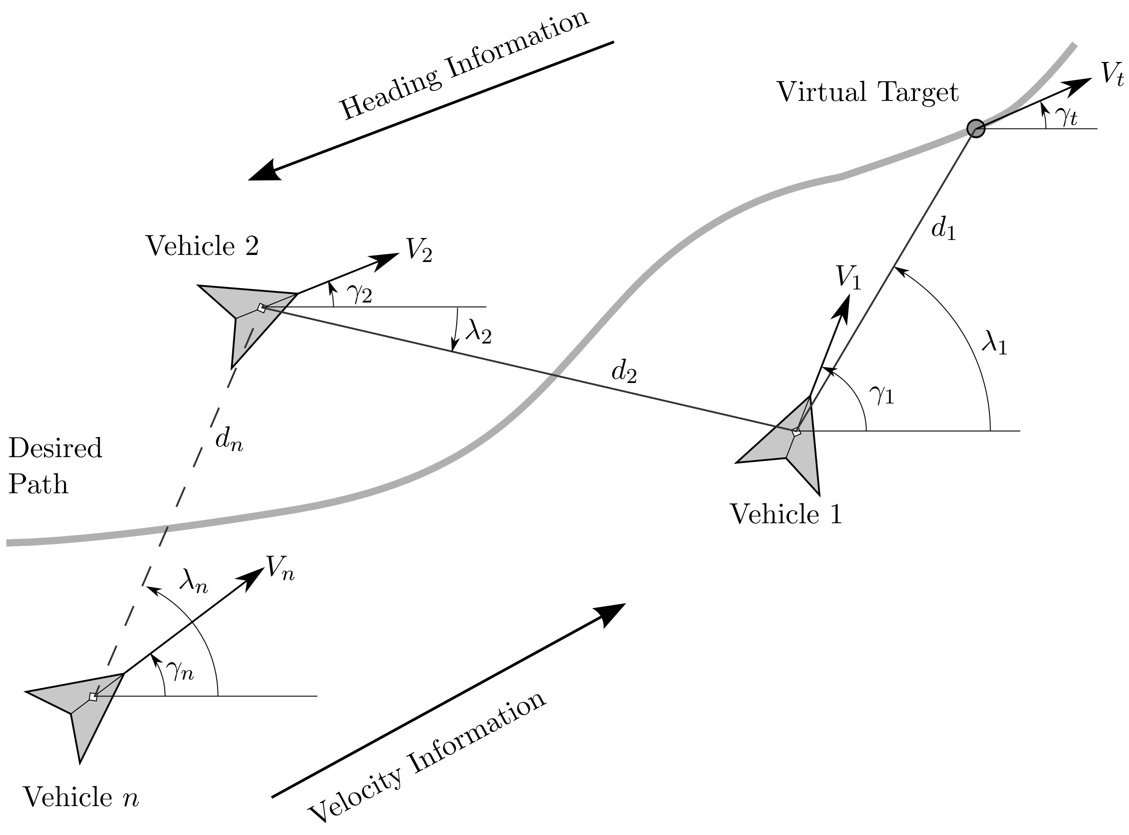


Fig. 3.1: Platooning Trajectory Shaping

The vehicles all use a north, west, up configuration. Thus all angles are positive in the counter-clockwise direction. Using these variables and sign conventions, equations of motion can be derived to define the system.

3.2 Formulation of Platooning Control

For the single vehicle case, since the target is chosen by the user, there are three degrees of freedom in this system, so there must be three states used to define the system. The purposes of this thesis don't require the states to be in global units, thus, to keep the system simple, relative units are used to define the system as shown in (3.1)-(3.3). The states chosen to define the system are d, α_t, and α_v. These states completely define the system. It is okay that the system states are all in relative terms because the virtual target position and heading are set by the user and thus allow relative states of the system to

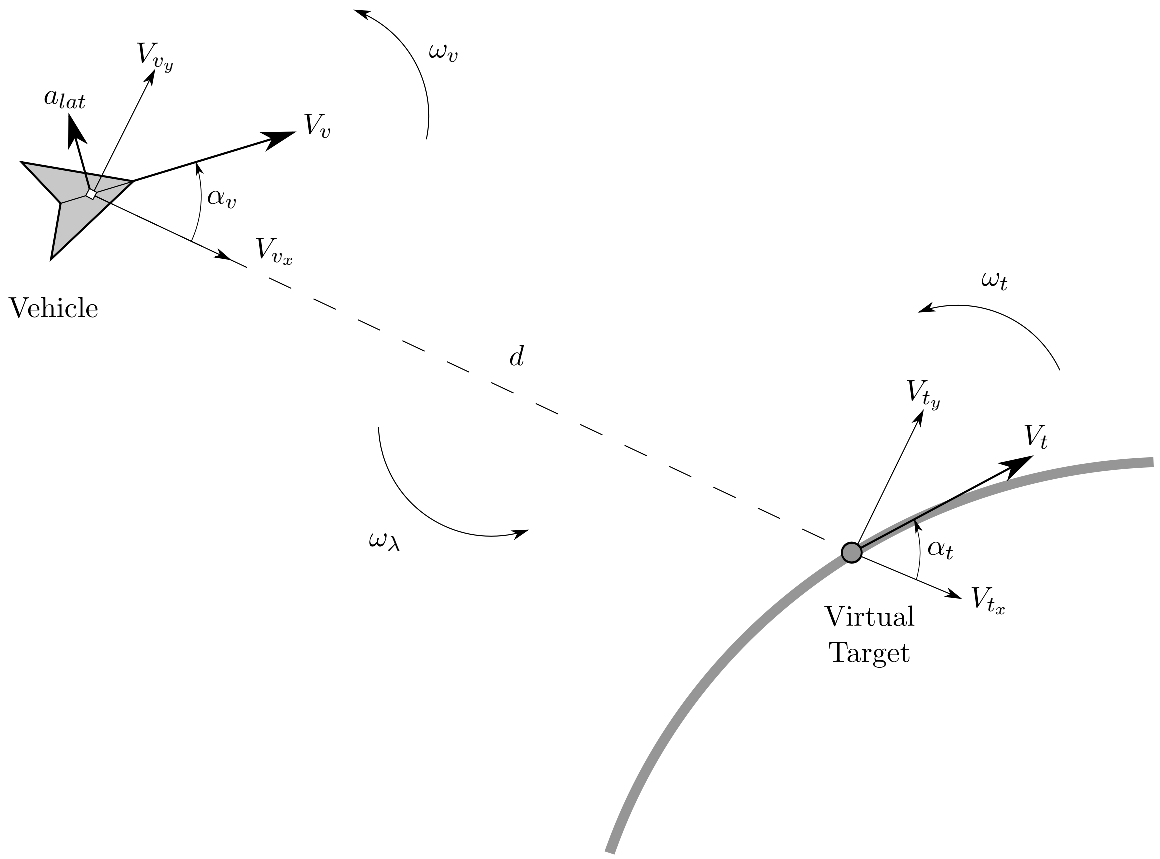


Fig. 3.2: Trajectory Shaping Variable Definitions

translate to controlling the vehicle to follow a path in global positions.

$$\dot{d} = V_{t_x} - V_{v_x} \tag{3.1}$$

$$\dot{\alpha}_t = \omega_t - \omega_\lambda \tag{3.2}$$

$$\dot{\alpha}_v = \omega_v - \omega_\lambda \tag{3.3}$$

In platooning, each vehicle adds three degrees of freedom to the system, thus three more states must be added. Equations (3.1)-(3.3) can be rewritten in general terms for a platoon of n vehicles.

$$\dot{d}_i = V_{t_{x_i}} - V_{v_{x_i}} \tag{3.4}$$

$$\dot{\alpha}_{t_i} = \omega_{t_i} - \omega_{\lambda_i} \tag{3.5}$$

$$\dot{\alpha}_{v_i} = \omega_{v_i} - \omega_{\lambda_i} \tag{3.6}$$

These equations of motion need to be put in terms of the states. Equations (3.5)-(3.6) can be rewritten by evaluating ω_{t_i}, ω_{v_i}, and ω_{λ_i}.

$$\dot{\alpha}_{t_i} = \frac{V_{i-1}}{R} - \frac{V_{i-1_y} - V_{i_y}}{d_i} \tag{3.7}$$

$$\dot{\alpha}_{v_i} = \frac{a_{lat_i}}{V_{v_i}} - \frac{V_{i-1_y} - V_{i_y}}{d_i} \tag{3.8}$$

The trajectory shaping guidance laws (2.1) and (2.3) can be updated for application in platooning control. This provides (3.9) for the regular method and (3.10) for the sine method. The velocity control (2.2) must also be updated for platooning and is shown in (3.11).

$$a_{lat_i^{reg}} = \frac{V_i^2}{d_i}\big(4(\lambda_i - \gamma_i) + 2(\lambda_i - \gamma_{i-1})\big) \tag{3.9}$$

$$a_{lat_i^{sine}} = \frac{V_i^2}{d_i}\big(4\sin(\lambda_i - \gamma_i) + 2\sin(\lambda_i - \gamma_{i-1})\big) \tag{3.10}$$

$$V_i = V_{i+1}\frac{d^*}{d_{i+1}} \tag{3.11}$$

These control equations show that the information needed for the lateral acceleration commands is passed backward through the platoon. The information needed for velocity control is passed forward through the platoon. This may seem counterintuitive, but can provide some benefits to the system when dealing with disturbances. For example, if a vehicle receives a lateral disturbance from the path, the vehicles in front of that vehicle won't be affected laterally; they will remain on the path. If a vehicle receives a longitudinal disturbance, the vehicles behind it won't be affected longitudinally.

Using trigonometry (3.4), (3.7), and (3.8) can be put in terms of V_t and V_v. The lateral acceleration (a_{lat}) is also replaced with the trajectory shaping guidance laws to produce

$$\dot{d}_i = V_{i-1}\cos(\alpha_{t_i}) - V_i\cos(\alpha_{v_i}) \tag{3.12}$$

$$\dot{\alpha}_{t_i} = \frac{V_{i-1}}{R} - \frac{V_{i-1}\sin(\alpha_{t_i}) - V_i \sin(\alpha_{v_i})}{d_i} \tag{3.13}$$

$$\dot{\alpha}_{v_i^{reg}} = \frac{V_i}{d_i}(4(-\alpha_{v_i}) + 2(-\alpha_{t_i})) - \frac{V_{i-1}\sin(\alpha_{t_i}) - V_i \sin(\alpha_{v_i})}{d_i} \tag{3.14}$$

for the regular method and

$$\dot{\alpha}_{v_i^{sine}} = \frac{V_i}{d_i}(4\sin(-\alpha_{v_i}) + 2\sin(-\alpha_{t_i})) - \frac{V_{i-1}\sin(\alpha_{t_i}) - V_i \sin(\alpha_{v_i})}{d_i} \tag{3.15}$$

for the sine method.

This thesis assumes constant path radius R, thus it can be kept in the equation as it is. The target velocity is given in (3.11), thus the following substitutions can be made.

$$\dot{d} = V_i \frac{d^*}{d_i}\cos(\alpha_{t_i}) - V_i \cos(\alpha_{v_i}) \tag{3.16}$$

$$\dot{\alpha}_{t_i} = \frac{V_i d^*}{R d_i} - \frac{V_i \frac{d^*}{d_i}\sin(\alpha_{t_i}) - V_i \sin(\alpha_{v_i})}{d_i} \tag{3.17}$$

$$\dot{\alpha}_{v_i^{reg}} = \frac{V_i}{d_i}(4(-\alpha_{v_i}) + 2(-\alpha_{t_i})) - \frac{V_i \frac{d^*}{d_i}\sin(\alpha_{t_i}) - V_i \sin(\alpha_{v_i})}{d_i} \tag{3.18}$$

$$\dot{\alpha}_{v_i^{sine}} = \frac{V_i}{d_i}(4\sin(-\alpha_{v_i}) + 2\sin(-\alpha_{t_i})) - \frac{V_i \frac{d^*}{d_i}\sin(\alpha_{t_i}) - V_i \sin(\alpha_{v_i})}{d_i} \tag{3.19}$$

These equations can be further simplified to the following equations of motion that will be used for analysis.

$$\dot{d}_i = V_i \left(\frac{d^*}{d_i}\cos(\alpha_{t_i}) - \cos(\alpha_{v_i}) \right) \tag{3.20}$$

$$\dot{\alpha}_{t_i} = \frac{V_i}{d_i} \left(\frac{d^*}{R} - \frac{d^*}{d_i}\sin(\alpha_{t_i}) + \sin(\alpha_{v_i}) \right) \tag{3.21}$$

$$\dot{\alpha}_{v_i^{reg}} = \frac{V_i}{d_i} \left(-4\alpha_{v_i} - 2\alpha_{t_i} - \frac{d^*}{d_i}\sin(\alpha_{t_i}) - \sin(\alpha_{v_i}) \right) \tag{3.22}$$

$$\dot{\alpha}_{v_i^{sine}} = \frac{V_i}{d_i} \left(-3\sin(\alpha_{v_i}) - \sin(\alpha_{t_i}) \left(2 + \frac{d^*}{d_i} \right) \right) \tag{3.23}$$

Each of these is given in terms of known states, other than V_i, which is different in all vehicles. However, in the single vehicle case, the vehicle's velocity is set to be constant and the virtual target will speed up if they are too close or slow down if they are too far apart. To apply this in platooning, since each vehicle is considered the target of the vehicle behind it, the last vehicle will be given a constant velocity (V_n). As can be seen in (3.11), the velocity of each vehicle is a function of the vehicle behind it and the distance between the two. Thus, using the following logic, the velocity of any vehicle in the platoon can be rewritten in terms of V_n.

$$\begin{array}{rcl}
V_t = V_1 \dfrac{d^*}{d_1} & \searrow & \\
V_1 = V_2 \dfrac{d^*}{d_2} & \searrow & V_t = V_2 \dfrac{{d^*}^2}{d_1 d_2} \\
V_2 = V_3 \dfrac{d^*}{d_3} & \searrow & V_1 = V_3 \dfrac{{d^*}^2}{d_2 d_3} \\
\vdots & \cdots & V_2 = V_4 \dfrac{{d^*}^2}{d_3 d_4} \\
V_{n-2} = V_{n-1} \dfrac{d^*}{d_{n-1}} & \searrow & \\
V_{n-1} = V_n \dfrac{d^*}{d_n} & \rightarrow & V_{n-2} = V_n \dfrac{{d^*}^2}{d_{n-1} d_n} \\
 & V_n = \text{constant} &
\end{array}$$

Continuing these equations through provides an alternate form of the velocity control that is only in terms of V_n, which is constant.

$$V_i = V_n \left(\frac{(d^*)^{n-i}}{\prod\limits_{j=(i+1)}^{n} d_j} \right) \forall i \in [0, n-1] \tag{3.24}$$

This form of the velocity control shows that the all velocities are in terms of the final vehicle velocity. Since the final vehicle velocity is considered constant, it is a parameter, and (3.20)-(3.23) are all in terms of the states and constant parameters.

3.3 Stability Analysis

The trajectory shaping guidance equations will be used to analyze the stability of the system. The object of this analysis is to show that by using the original trajectory shaping law there will be an offset of convergence, but when using the sine law there will not be an offset of convergence. This is first analyzed for a single vehicle in trajectory shaping guidance and will then be moved to the analysis of platooning control.

3.3.1 Single Vehicle

The equations of motion derived earlier will be used in the stability analysis of the system. The equations were derived for a platoon, but can easily be used for a single vehicle by assuming it to be a platoon with $n = 1$.

Sine Method

The sine method has been shown in Chapter 2 to accurately converge to a path of given radius. It will now be shown mathematically to converge to the desired final conditions.

It is the objective of this section to show that when assuming constant path radius (R) and vehicle velocity (V_v), a single vehicle will converge to the desired path at the desired spacing between vehicle and virtual target (d^*) when using the modified trajectory shaping guidance. This will occur for any chosen parameter values $V_v \in (0, \infty)$, $R \in [-\infty, 0) \cup (0, \infty]$, and $d^* \in (0, 2R)$.

If there is a solitary stable equilibrium point in the system space, than no matter the initial conditions within that space, they will converge to that equilibrium point. This section will show that this system has only one equilibrium point, that it is stable, and that the equilibrium point is on the desired path at the desired distance between the vehicle and virtual target.

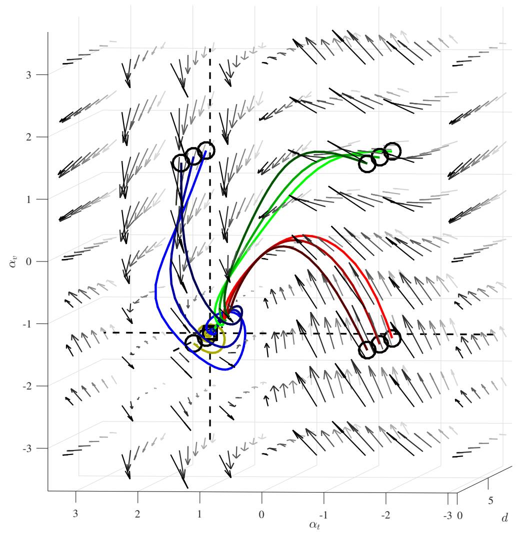
Using basic geometry from Fig. 3.2, it is observed that if $d = d^*$, than the following condition must be met to be on the desired path: $\alpha_t = -\alpha_v = \sin^{-1}(\frac{d^*}{2R})$. Using the equations of motion for the sine method (3.20), (3.21), and (3.23), it is seen that there is an equilibrium point at $d = d^*$, $\alpha_t = \sin^{-1}(\frac{d^*}{2R})$, and $\alpha_v = -\alpha_t$. This is found by plugging these values in to our equations of motion and seeing that the state derivatives go to zero.

$$\dot{d} = V_v \left(\frac{d^*}{d^*} \cos(\alpha_t) - \cos(-\alpha_t) \right) = 0 \tag{3.25}$$

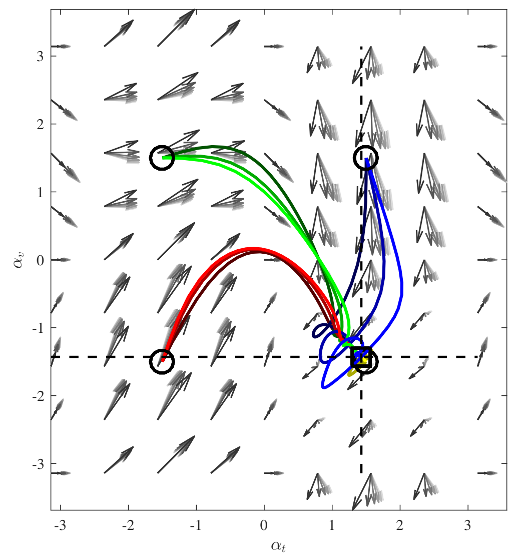
$$\dot{\alpha}_t = \frac{V_v}{d^*} \left(\frac{d^*}{R} - \frac{d^*}{d^*}\frac{d^*}{2R} - \frac{d^*}{2R} \right) = 0 \tag{3.26}$$

$$\dot{\alpha}_v = \frac{V_v}{d^*} \left(-3(-\frac{d^*}{2R}) - (\frac{d^*}{2R}(2 + \frac{d^*}{d^*}) \right) = \frac{V_v}{d^*} \left(3\frac{d^*}{2R} - \frac{d^*}{2R}(2+1) \right) = 0 \tag{3.27}$$

The constraints on this system are that all angles are mapped to be from $-\pi$ to π. Using Matlab's nonlinear solver, it was found that $d = d^*$, $\alpha_t = \sin^{-1}(\frac{d^*}{2R})$, and $\alpha_v = -\alpha_t$ is the only equilibrium point and that no matter the initial condition, the vehicle will converge to that equilibrium point and, thus, converge to the path at the desired distance. This is shown through Fig. 3.3. These phase portraits show the existence of the single equilibrium point and that it is found at the desired location. Initial conditions are shown by circles and final conditions are shown as a square. The dotted lines represent the desired equilibrium point.



(a) 3D Phase Plot



(b) α_t-α_v View of Phase Plot

Fig. 3.3: Phase Plot for Sine Method

The nonlinear solver in Matlab requires numerical values for R, V_v, and d^*. For the plots shown, the following parameter values were used: $R = 3$m, $V_v = 1$m/s, and $d^* = 5.94$m. Although these numerical values were used, the solution can be generalized do be in terms of d^* and R. As is seen in (3.20)-(3.23), the vehicle velocity does not affect the phase portrait other than to scale how quickly the values converge. Changing the value of R also has no effect on the phase portrait as long as d^* remains in scale with the change. For the phase plots shown, d^* was set to be 0.99 times the diameter of the path.

This shows that for all possible values of V_v, R, and d^*, the vehicle will converge to the desired path at the desired distance from the virtual target from any initial condition.

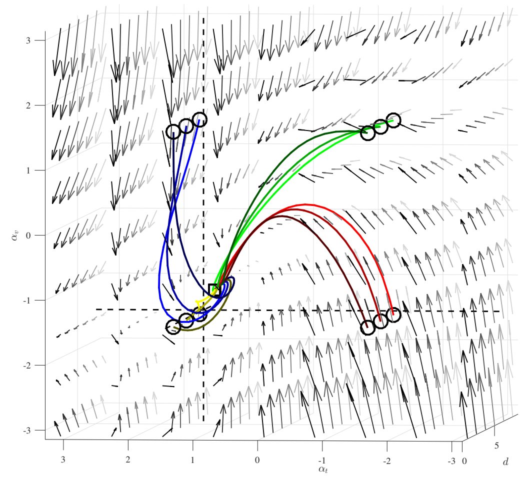
Regular Method

The regular method has been shown to converge to an offset of a path of given radius. It will now be shown mathematically that this is the case.

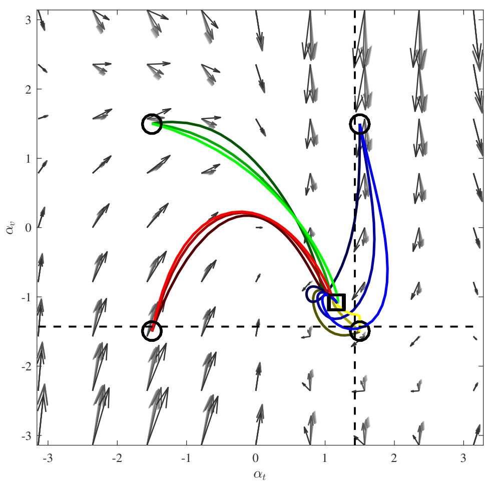
It is the objective of this section to show that when assuming constant path radius (R) and vehicle velocity (V_v), a single vehicle will converge to an offset of the desired path at an offset to the desired spacing between vehicle and virtual target (d^*) when using the original trajectory shaping guidance. This will occur for any chosen parameter values $V_v \in (0, \infty)$, $R \in [-\infty, 0) \cup (0, \infty]$, and $d^* \in (0, 2R)$.

As shown previously, an equilibrium point must be found. This is no longer a simple task since the $\dot{\alpha_v}$ equation no longer cancels easily. Therefore, the equilibrium point is found purely using Matlab's nonlinear solver. The phase plots created through this analysis are shown in Fig. 3.4.

This analysis shows there is a single equilibrium point and that all initial conditions in the space will converge to the equilibrium point. It also shows that $d \neq d^*$, $\alpha_t \neq \sin^{-1}(\frac{d^*}{2R})$, and $\alpha_v \neq -\alpha_t$. This means that in the regular method, the vehicle will converge to a point that is offset from the desired equilibrium point. Chapter 2 shows this principle through simulation and hardware results. The vehicle converges to an offset of the desired distance as well as to a radius smaller than the desired path radius. Again, as seen in (3.20)-(3.22), the vehicle velocity does not affect the phase portrait other than to scale how quickly the



(a) 3D Phase Plot



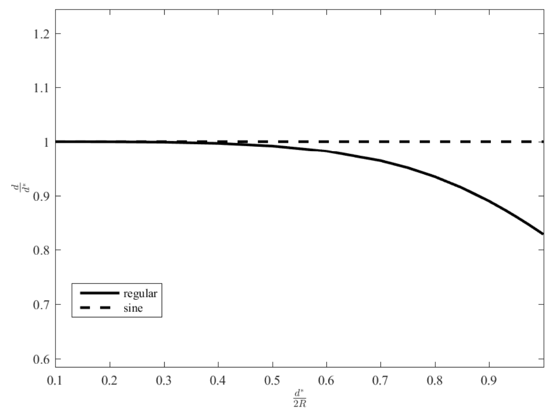
(b) α_t-α_v View of Phase Plot

Fig. 3.4: Phase Plot for Regular Method

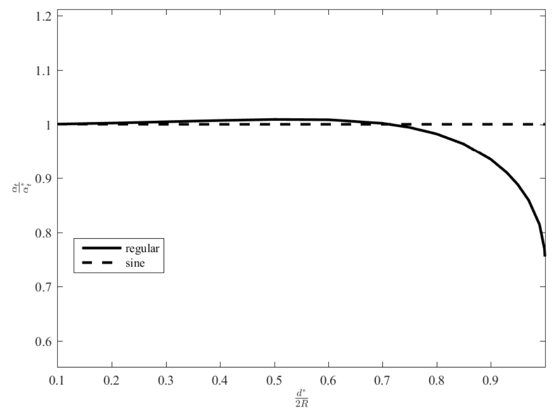
values converge. Changing the $\frac{d^*}{2R}$ ratio changes the location of the equilibrium point, but it only converges to the desired equilibrium point when the path is a straight line.

Comparison

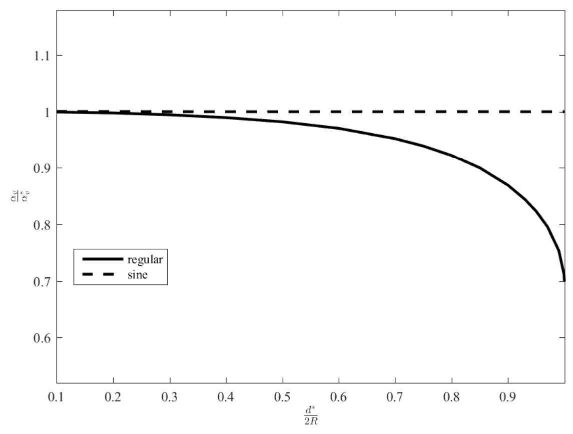
In both the sine and regular method, as long as the ratio of d^* to R is maintained, the equilibrium point will be at the same location. However, to further simplify the controller, it would be advantageous to allow d^* to be chosen to any value regardless of the path radius. Obviously, there will always be the constraint that d^* must be less than $2R$. However, as discussed in Chapter 2, it can be advantageous to have a larger ratio of d^* to R. The nonlinear solver in Matlab was used to test the equilibrium point at different ratios $\frac{d^*}{2R}$. These equilibrium points were then compared to the desired equilibrium point that will ensure the vehicle is on the path at the desired distance. These results are shown in Fig. 3.5. Fig. 3.5 shows that as d^* approaches the diameter of the path, the regular method converges to a point that is not at the desired distance on the path. Fig. 3.5(b) and Fig. 3.5(c) show that α_t will converge to a point that has a slightly greater magnitude than the desired equilibrium α_v. Along with the fact that d converges to something less than d^*, this means



(a) Distance Equilibrium



(b) Target Heading Equilibrium



(c) Vehicle Heading Equilibrium

Fig. 3.5: Ratios of Desired Equilibrium Values to Actual Equilibrium Values

that the regular method converges to an offset on the inside of the path radius and that this offset increases as d approaches $2R$. Fig. 3.5 also shows that as d^* approaches the diameter of the path, when using the sine method, the equilibrium point is unchanged and the system will converge to be on the desired path at the desired separation distance.

3.3.2 Multiple Vehicle

For the multiple vehicle platoon, the system will first be studied for three vehicles. Since the regular method has been shown to be undesirable, the remainder of this chapter will only focus on the sine method.

It is the objective of this section to show that when assuming constant path radius

(R) and vehicle velocity (V_v), a platoon of vehicles in the neighborhood of the equilibrium point will all converge to the desired path at the desired spacing between vehicle and virtual target (d^*) when using the modified trajectory shaping guidance. This will occur for any chosen parameter values $V_n \in (0, \infty)$, $R \in [-\infty, 0) \cup (0, \infty]$, and $d^* \in (0, 2R)$, regardless of the number of vehicles in the platoon.

By using each vehicle as the virtual target of the vehicle behind it, the equations of motion are derived as follows:

$$\dot{d} = \begin{bmatrix} \dot{d}_1 \\ \dot{d}_2 \\ \dot{d}_3 \end{bmatrix} = \begin{bmatrix} V_1 \left(\frac{d^*}{d_1} \cos(\alpha_{t_1}) - \cos(\alpha_{v_1}) \right) \\ V_2 \left(\frac{d^*}{d_2} \cos(\alpha_{t_2}) - \cos(\alpha_{v_2}) \right) \\ V_3 \left(\frac{d^*}{d_3} \cos(\alpha_{t_3}) - \cos(\alpha_{v_3}) \right) \end{bmatrix} \tag{3.28}$$

$$\dot{\alpha}_t = \begin{bmatrix} \dot{\alpha}_{t_1} \\ \dot{\alpha}_{t_2} \\ \dot{\alpha}_{t_3} \end{bmatrix} = \begin{bmatrix} \frac{V_1}{d_1} \left(\frac{d^*}{R} - \frac{d^*}{d_1} \sin(\alpha_{t_1}) + \sin(\alpha_{v_1}) \right) \\ \frac{V_1}{d_2} \left(\frac{d^*}{R} - \frac{d^*}{d_2} \sin(\alpha_{t_2}) + \sin(\alpha_{v_2}) \right) \\ \frac{V_3}{d_3} \left(\frac{d^*}{R} - \frac{d^*}{d_3} \sin(\alpha_{t_3}) + \sin(\alpha_{v_3}) \right) \end{bmatrix} \tag{3.29}$$

$$\dot{\alpha}_v = \begin{bmatrix} \dot{\alpha}_{v_1} \\ \dot{\alpha}_{v_2} \\ \dot{\alpha}_{v_3} \end{bmatrix} = \begin{bmatrix} \frac{V_1}{d_1} \left(-3 \sin(\alpha_{v_1}) - \sin(\alpha_{t_1}) \left(2 + \frac{d^*}{d_1} \right) \right) \\ \frac{V_2}{d_2} \left(-3 \sin(\alpha_{v_2}) - \sin(\alpha_{t_2}) \left(2 + \frac{d^*}{d_2} \right) \right) \\ \frac{V_3}{d_3} \left(-3 \sin(\alpha_{v_3}) - \sin(\alpha_{t_3}) \left(2 + \frac{d^*}{d_3} \right) \right) \end{bmatrix} \tag{3.30}$$

The Jacobian of this matrix is taken to find the local stability. The velocity is known for each vehicle from (3.11). Since each vehicle's velocity is a function of the velocity of the vehicle behind it, all the velocities can be written in terms of the final vehicle's velocity which is assumed constant. Thus, the Jacobian is evaluated to be the following:

$$J = \left[\begin{array}{c:c:c} J_{d/d} & J_{d/\alpha_t} & J_{d/\alpha_v} \\ \hdashline J_{\alpha_t/d} & J_{\alpha_t/\alpha_t} & J_{\alpha_t/\alpha_v} \\ \hdashline J_{\alpha_v/d} & J_{\alpha_v/\alpha_t} & J_{\alpha_v/\alpha_v} \end{array} \right] \tag{3.31}$$

$$J_{d/d} = \begin{bmatrix} -V_3\frac{d^{*3}}{d_1^2 d_2 d_3}\cos(\alpha_{t_1}) & V_3\frac{d^{*2}}{d_2^2 d_3}(\cos(\alpha_{v_1}) - \frac{d^*}{d_1}\cos(\alpha_{t_1})) & V_3\frac{d^{*2}}{d_2 d_3^2}(\cos(\alpha_{v_1}) - \frac{d^*}{d_1}\cos(\alpha_{t_1})) \\ 0 & -V_3\frac{d^{*2}}{d_2^2 d_3}\cos(\alpha_{t_2}) & V_3\frac{d^*}{d_3^2}(\cos(\alpha_{v_2}) - \frac{d^*}{d_2}\cos(\alpha_{t_2})) \\ 0 & 0 & -V_3\frac{d^*}{d_3^2}\cos(\alpha_{t_3}) \end{bmatrix} \tag{3.32}$$

$$J_{d/\alpha_t} = \begin{bmatrix} -V_3\frac{d^{*3}}{d_1 d_2 d_3}\sin(\alpha_{t_1}) & 0 & 0 \\ 0 & -V_3\frac{d^{*2}}{d_2 d_3}\sin(\alpha_{t_2}) & 0 \\ 0 & 0 & -V_3\frac{d^*}{d_3}\sin(\alpha_{t_3}) \end{bmatrix} \tag{3.33}$$

$$J_{d/\alpha_v} = \begin{bmatrix} V_3\frac{d^{*2}}{d_2 d_3}\sin(\alpha_{v_1}) & 0 & 0 \\ 0 & V_3\frac{d^*}{d_3}\sin(\alpha_{v_2}) & 0 \\ 0 & 0 & V_3\sin(\alpha_{v_3}) \end{bmatrix} \tag{3.34}$$

$$J_{\alpha_t/d} = \begin{bmatrix} -V_3\frac{d^{*2}}{d_1^2 d_2 d_3}(\frac{d^*}{R} + \sin(\alpha_{v_1}) - 2\frac{d^*}{d_1}\sin(\alpha_{t_1})) & -V_3\frac{d^{*2}}{d_1 d_2^2 d_3}(\frac{d^*}{R} + \sin(\alpha_{v_1}) - \frac{d^*}{d_1}\sin(\alpha_{t_1})) & -V_3\frac{d^{*2}}{d_1 d_2 d_3^2}(\frac{d^*}{R} + \sin(\alpha_{v_1}) - \frac{d^*}{d_1}\sin(\alpha_{t_1})) \\ 0 & -V_3\frac{d^*}{d_2^2 d_3}(\frac{d^*}{R} + \sin(\alpha_{v_2}) - 2\frac{d^*}{d_2} * \sin(\alpha_{t_2})) & -V_3\frac{d^*}{d_2 d_3^2}(\frac{d^*}{R} + \sin(\alpha_{v_2}) - \frac{d^*}{d_2}\sin(\alpha_{t_2})) \\ 0 & 0 & -\frac{V_3}{d_3^2}(\frac{d^*}{R} + \sin(\alpha_{v_3}) - 2\frac{d^*}{d_3}\sin(\alpha_{t_3})) \end{bmatrix} \tag{3.35}$$

$$J_{\alpha_t/\alpha_t} = \begin{bmatrix} -V_3\frac{d^{*3}}{d_1^2 d_2 d_3}\cos(\alpha_{t_1}) & 0 & 0 \\ 0 & -V_3\frac{d^{*2}}{d_2^2 d_3}\cos(\alpha_{t_2}) & 0 \\ 0 & 0 & -V_3\frac{d^*}{d_3^2}\cos(\alpha_{t_3}) \end{bmatrix} \tag{3.36}$$

$$J_{\alpha_t/\alpha_v} = \begin{bmatrix} V_3\frac{d^{*2}}{d_1 d_2 d_3}\cos(\alpha_{v_1}) & 0 & 0 \\ 0 & V_3\frac{d^*}{d_2 d_3}\cos(\alpha_{v_2}) & 0 \\ 0 & 0 & \frac{V_3}{d_3}\cos(\alpha_{v_3}) \end{bmatrix} \tag{3.37}$$

$$J_{\alpha_v/d} = \begin{bmatrix} V_3\frac{d^{*2}}{d_1^2 d_2 d_3}(3\sin(\alpha_{v_1}) + 2\sin(\alpha_{t_1})(\frac{d^*}{d_1} + 1)) & V_3\frac{d^{*2}}{d_1 d_2^2 d_3}(3\sin(\alpha_{v_1}) + \sin(\alpha_{t_1})(\frac{d^*}{d_1} + 2)) & V_3\frac{d^{*2}}{d_1 d_2 d_3^2}(3\sin(\alpha_{v_1}) + \sin(\alpha_{t_1})(\frac{d^*}{d_1} + 2)) \\ 0 & V_3\frac{d^*}{d_2^2 d_3}(3\sin(\alpha_{v_2}) + 2\sin(\alpha_{t_2})(\frac{d^*}{d_2} + 1)) & V_3\frac{d^*}{d_2 d_3^2}(3\sin(\alpha_{v_2}) + \sin(\alpha_{t_2})(\frac{d^*}{d_2} + 2)) \\ 0 & 0 & \frac{V_3}{d_3^2}(3\sin(\alpha_{v_3}) + 2\sin(\alpha_{t_3})(\frac{d^*}{d_3} + 1)) \end{bmatrix} \tag{3.38}$$

$$J_{\alpha_v/\alpha_t} = \begin{bmatrix} -V_3\frac{d^{*2}}{d_1 d_2 d_3}\cos(\alpha_{t_1})(2 + \frac{d^*}{d_1}) & 0 & 0 \\ 0 & -V_3\frac{d^*}{d_2 d_3}\cos(\alpha_{t_2})(2 + \frac{d^*}{d_2}) & 0 \\ 0 & 0 & -\frac{V_3}{d_3}\cos(\alpha_{t_3})(\frac{d^*}{d_3} + 2) \end{bmatrix} \tag{3.39}$$

$$J_{\alpha_v/\alpha_v} = \begin{bmatrix} -3V_3\frac{d^{*2}}{d_1 d_2 d_3}\cos(\alpha_{v_1}) & 0 & 0 \\ 0 & -3V_3\frac{d^*}{d_2 d_3}\cos(\alpha_{v_2}) & 0 \\ 0 & 0 & -3\frac{V_3}{d_3}\cos(\alpha_{v_3}) \end{bmatrix} \tag{3.40}$$

If all the real parts of the eigenvalues of the Jacobian evaluated at the equilibrium point are negative, the system is exponentially stable. Plugging in the equilibrium value $d_i = d^*$

provides the following:

$$J\big|_{eq} = \begin{bmatrix} J_{d/d}\Big|_{eq} & J_{d/\alpha_t}\Big|_{eq} & J_{d/\alpha_v}\Big|_{eq} \\ J_{\alpha_t/d}\Big|_{eq} & J_{\alpha_t/\alpha_t}\Big|_{eq} & J_{\alpha_t/\alpha_v}\Big|_{eq} \\ J_{\alpha_v/d}\Big|_{eq} & J_{\alpha_v/\alpha_t}\Big|_{eq} & J_{\alpha_v/\alpha_v}\Big|_{eq} \end{bmatrix} \tag{3.41}$$

$$J_{d/d}\Big|_{eq} = \begin{bmatrix} -\frac{V_3}{d^*}\cos(\alpha_{t_1}) & \frac{V_3}{d^*}(\cos(\alpha_{v_1}) - \cos(\alpha_{t_1})) & \frac{V_3}{d^*}(\cos(\alpha_{v_1}) - \cos(\alpha_{t_1})) \\ 0 & -\frac{V_3}{d^*}\cos(\alpha_{t_2}) & \frac{V_3}{d^*}(\cos(\alpha_{v_2}) - \cos(\alpha_{t_2})) \\ 0 & 0 & -\frac{V_3}{d^*}\cos(\alpha_{t_3}) \end{bmatrix} \tag{3.42}$$

$$J_{d/\alpha_t}\Big|_{eq} = \begin{bmatrix} -V_3\sin(\alpha_{t_1}) & 0 & 0 \\ 0 & -V_3\sin(\alpha_{t_2}) & 0 \\ 0 & 0 & -V_3\sin(\alpha_{t_3}) \end{bmatrix} \tag{3.43}$$

$$J_{d/\alpha_v}\Big|_{eq} = \begin{bmatrix} V_3\sin(\alpha_{v_1}) & 0 & 0 \\ 0 & V_3\sin(\alpha_{v_2}) & 0 \\ 0 & 0 & V_3\sin(\alpha_{v_3}) \end{bmatrix} \tag{3.44}$$

$$J_{\alpha_t/d}\Big|_{eq} = \begin{bmatrix} -\frac{V_3}{d^{*2}}(\frac{d^*}{R} + \sin(\alpha_{v_1}) - 2\sin(\alpha_{t_1})) & -\frac{V_3}{d^{*2}}(\frac{d^*}{R} + \sin(\alpha_{v_1}) - \sin(\alpha_{t_1})) & -\frac{V_3}{d^{*2}}(\frac{d^*}{R} + \sin(\alpha_{v_1}) - \sin(\alpha_{t_1})) \\ 0 & -\frac{V_3}{d^{*2}}(\frac{d^*}{R} + \sin(\alpha_{v_2}) - 2\sin(\alpha_{t_2})) & -\frac{V_3}{d^{*2}}(\frac{d^*}{R} + \sin(\alpha_{v_2}) - \sin(\alpha_{t_2})) \\ 0 & 0 & -\frac{V_3}{d_3^2}(\frac{d^*}{R} + \sin(\alpha_{v_3}) - 2\sin(\alpha_{t_3})) \end{bmatrix} \tag{3.45}$$

$$J_{\alpha_t/\alpha_t}\Big|_{eq} = \begin{bmatrix} -\frac{V_3}{d^*}\cos(\alpha_{t_1}) & 0 & 0 \\ 0 & -\frac{V_3}{d^*}\cos(\alpha_{t_2}) & 0 \\ 0 & 0 & -\frac{V_3}{d^*}\cos(\alpha_{t_3}) \end{bmatrix} \tag{3.46}$$

$$J_{\alpha_t/\alpha_v}\Big|_{eq} = \begin{bmatrix} \frac{V_3}{d^*}\cos(\alpha_{v_1}) & 0 & 0 \\ 0 & \frac{V_3}{d^*}\cos(\alpha_{v_2}) & 0 \\ 0 & 0 & \frac{V_3}{d^*}\cos(\alpha_{v_3}) \end{bmatrix} \tag{3.47}$$

$$J_{\alpha_v/d}\Big|_{eq} = \begin{bmatrix} \frac{V_3}{d^{*2}}(3\sin(\alpha_{v_1}) + 2\sin(\alpha_{t_1})(1+1)) & \frac{V_3}{d^{*2}}(3\sin(\alpha_{v_1}) + \sin(\alpha_{t_1})(1+2)) & \frac{V_3}{d^{*2}}(3\sin(\alpha_{v_1}) + \sin(\alpha_{t_1})(1+2)) \\ 0 & \frac{V_3}{d^{*2}}(3\sin(\alpha_{v_2}) + 2\sin(\alpha_{t_2})(1+1)) & \frac{V_3}{d^{*2}}(3\sin(\alpha_{v_2}) + \sin(\alpha_{t_2})(1+2)) \\ 0 & 0 & \frac{V_3}{d^{*2}}(3\sin(\alpha_{v_3}) + 2\sin(\alpha_{t_3})(1+1)) \end{bmatrix} \tag{3.48}$$

$$J_{\alpha_v/\alpha_t}\Big|_{eq} = \begin{bmatrix} -\frac{V_3}{d^*}\cos(\alpha_{t_1})(2+1) & 0 & 0 \\ 0 & -V_3\frac{V_3}{d^*}\cos(\alpha_{t_2})(2+1) & 0 \\ 0 & 0 & -\frac{V_3}{d^*}\cos(\alpha_{t_3})(1+2) \end{bmatrix} \tag{3.49}$$

$$J_{\alpha_v/\alpha_v}\Big|_{eq} = \begin{bmatrix} -3\frac{V_3}{d^*}\cos(\alpha_{v_1}) & 0 & 0 \\ 0 & -3\frac{V_3}{d^*}\cos(\alpha_{v_2}) & 0 \\ 0 & 0 & -3\frac{V_3}{d^*}\cos(\alpha_{v_3}) \end{bmatrix} \tag{3.50}$$

Plugging in the equilibrium condition, $\alpha_{t_i} = -\alpha_{v_i} = \alpha_{nom_i}$.

$$J_{d/d}\Big|_{eq} = \begin{bmatrix} -\frac{V_3}{d^*}\cos(\alpha_{nom_1}) & 0 & 0 \\ 0 & -\frac{V_3}{d^*}\cos(\alpha_{nom_2}) & 0 \\ 0 & 0 & -\frac{V_3}{d^*}\cos(\alpha_{nom_3}) \end{bmatrix} \tag{3.51}$$

$$J_{d/\alpha_t}\Big|_{eq} = \begin{bmatrix} -V_3\sin(\alpha_{nom_1}) & 0 & 0 \\ 0 & -V_3\sin(\alpha_{nom_2}) & 0 \\ 0 & 0 & -V_3\sin(\alpha_{nom_3}) \end{bmatrix} \tag{3.52}$$

$$J_{d/\alpha_v}\Big|_{eq} = \begin{bmatrix} V_3\sin(-\alpha_{nom_1}) & 0 & 0 \\ 0 & V_3\sin(-\alpha_{nom_2}) & 0 \\ 0 & 0 & V_3\sin(-\alpha_{nom_3}) \end{bmatrix} \tag{3.53}$$

$$J_{\alpha_t/d}\Big|_{eq} = \begin{bmatrix} -\frac{V_3}{d^{*2}}(\frac{d^*}{R} - 3\sin(\alpha_{nom_1})) & -\frac{V_3}{d^{*2}}(\frac{d^*}{R} - 2\sin(\alpha_{nom_1})) & -\frac{V_3}{d^{*2}}(\frac{d^*}{R} - 2\sin(\alpha_{nom_1})) \\ 0 & -\frac{V_3}{d^{*2}}(\frac{d^*}{R} - 3\sin(\alpha_{nom_2})) & -\frac{V_3}{d^{*2}}(\frac{d^*}{R} - 2\sin(\alpha_{nom_2})) \\ 0 & 0 & -\frac{V_3}{d_3^2}(\frac{d^*}{R} + 3\sin(\alpha_{nom_3})) \end{bmatrix} \tag{3.54}$$

$$J_{\alpha_t/\alpha_t}\Big|_{eq} = \begin{bmatrix} -\frac{V_3}{d^*}\cos(\alpha_{nom_1}) & 0 & 0 \\ 0 & -\frac{V_3}{d^*}\cos(\alpha_{nom_2}) & 0 \\ 0 & 0 & -\frac{V_3}{d^*}\cos(\alpha_{nom_3}) \end{bmatrix} \tag{3.55}$$

$$J_{\alpha_t/\alpha_v}\Big|_{eq} = \begin{bmatrix} \frac{V_3}{d^*}\cos(\alpha_{nom_1}) & 0 & 0 \\ 0 & \frac{V_3}{d^*}\cos(\alpha_{nom_2}) & 0 \\ 0 & 0 & \frac{V_3}{d^*}\cos(\alpha_{nom_3}) \end{bmatrix} \tag{3.56}$$

$$J_{\alpha_v/d}\Big|_{eq} = \begin{bmatrix} \frac{V_3}{d^{*2}}\sin(\alpha_{nom_1}) & 0 & 0 \\ 0 & \frac{V_3}{d^{*2}}\sin(\alpha_{nom_2}) & 0 \\ 0 & 0 & \frac{V_3}{d^{*2}}\sin(\alpha_{nom_3}) \end{bmatrix} \tag{3.57}$$

$$J_{\alpha_v/\alpha_t}\Big|_{eq} = \begin{bmatrix} -3\frac{V_3}{d^*}\cos(\alpha_{nom_1}) & 0 & 0 \\ 0 & -3\frac{V_3}{d^*}\cos(\alpha_{nom_2}) & 0 \\ 0 & 0 & -3\frac{V_3}{d^*}\cos(\alpha_{nom_3}) \end{bmatrix} \tag{3.58}$$

$$J_{\alpha_v/\alpha_v}\Big|_{eq} = \begin{bmatrix} -3\frac{V_3}{d^*}\cos(\alpha_{nom_1}) & 0 & 0 \\ 0 & -3\frac{V_3}{d^*}\cos(\alpha_{nom_2}) & 0 \\ 0 & 0 & -3\frac{V_3}{d^*}\cos(\alpha_{nom_3}) \end{bmatrix} \tag{3.59}$$

Since α_{nom_i} at equilibrium is $\sin^{-1}(\frac{d^*}{2R})$, the equations simplify even further.

$$J_{d/d}\Big|_{eq} = \begin{bmatrix} -\frac{V_3}{d^*}\sqrt{1-\frac{d^*}{2R}} & 0 & 0 \\ 0 & -\frac{V_3}{d^*}\sqrt{1-\frac{d^*}{2R}} & 0 \\ 0 & 0 & -\frac{V_3}{d^*}\sqrt{1-\frac{d^*}{2R}} \end{bmatrix} \tag{3.60}$$

$$J_{d/\alpha_t}\Big|_{eq} = \begin{bmatrix} -V_3\frac{d^*}{2R} & 0 & 0 \\ 0 & -V_3\frac{d^*}{2R} & 0 \\ 0 & 0 & -V_3\frac{d^*}{2R} \end{bmatrix} \tag{3.61}$$

$$J_{d/\alpha_v}\Big|_{eq} = \begin{bmatrix} -V_3\frac{d^*}{2R} & 0 & 0 \\ 0 & -V_3\frac{d^*}{2R} & 0 \\ 0 & 0 & -V_3\frac{d^*}{2R} \end{bmatrix} \tag{3.62}$$

$$J_{\alpha_v/d}\Big|_{eq} = \begin{bmatrix} \frac{V_3}{2Rd^*} & 0 & 0 \\ 0 & \frac{V_3}{2Rd^*} & 0 \\ 0 & 0 & \frac{V_3}{2Rd^*} \end{bmatrix} \tag{3.63}$$

$$J_{\alpha_t/\alpha_t}\Big|_{eq} = \begin{bmatrix} -\frac{V_3}{d^*}\sqrt{1-\frac{d^*}{2R}} & 0 & 0 \\ 0 & -\frac{V_3}{d^*}\sqrt{1-\frac{d^*}{2R}} & 0 \\ 0 & 0 & -\frac{V_3}{d^*}\sqrt{1-\frac{d^*}{2R}} \end{bmatrix} \tag{3.64}$$

$$J_{\alpha_t/\alpha_v}\Big|_{eq} = \begin{bmatrix} \frac{V_3}{d^*}\sqrt{1-\frac{d^*}{2R}} & 0 & 0 \\ 0 & \frac{V_3}{d^*}\sqrt{1-\frac{d^*}{2R}} & 0 \\ 0 & 0 & \frac{V_3}{d^*}\sqrt{1-\frac{d^*}{2R}} \end{bmatrix} \tag{3.65}$$

$$J_{\alpha_v/d}\Big|_{eq} = \begin{bmatrix} \frac{V_3}{2Rd^*} & 0 & 0 \\ 0 & \frac{V_3}{2Rd^*} & 0 \\ 0 & 0 & \frac{V_3}{2Rd^*} \end{bmatrix} \tag{3.66}$$

$$J_{\alpha_v/\alpha_t}\Big|_{eq} = \begin{bmatrix} -3\frac{V_3}{d^*}\sqrt{1-\frac{d^*}{2R}} & 0 & 0 \\ 0 & -3\frac{V_3}{d^*}\sqrt{1-\frac{d^*}{2R}} & 0 \\ 0 & 0 & -3\frac{V_3}{d^*}\sqrt{1-\frac{d^*}{2R}} \end{bmatrix} \tag{3.67}$$

$$J_{\alpha_v/\alpha_v}\Big|_{eq} = \begin{bmatrix} -3\frac{V_3}{d^*}\sqrt{1-\frac{d^*}{2R}} & 0 & 0 \\ 0 & -3\frac{V_3}{d^*}\sqrt{1-\frac{d^*}{2R}} & 0 \\ 0 & 0 & -3\frac{V_3}{d^*}\sqrt{1-\frac{d^*}{2R}} \end{bmatrix} \tag{3.68}$$

This provides the following total Jacobian:

$$J|_{eq} = \left[\begin{array}{ccc|ccc|ccc} -\frac{V_3}{d^*}\sqrt{1-\left(\frac{d^*}{2R}\right)^2} & 0 & 0 & -V_3\frac{d^*}{2R} & 0 & 0 & -V_3\frac{d^*}{2R} & 0 & 0 \\ 0 & -\frac{V_3}{d^*}\sqrt{1-\left(\frac{d^*}{2R}\right)^2} & 0 & 0 & -V_3\frac{d^*}{2R} & 0 & 0 & -V_3\frac{d^*}{2R} & 0 \\ 0 & 0 & -\frac{V_3}{d^*}\sqrt{1-\left(\frac{d^*}{2R}\right)^2} & 0 & 0 & -V_3\frac{d^*}{2R} & 0 & 0 & -V_3\frac{d^*}{2R} \\ \hline \frac{V_3}{2Rd^*} & 0 & 0 & -\frac{V_3}{d^*}\sqrt{1-\left(\frac{d^*}{2R}\right)^2} & 0 & 0 & \frac{V_3}{d^*}\sqrt{1-\left(\frac{d^*}{2R}\right)^2} & 0 & 0 \\ 0 & \frac{V_3}{2Rd^*} & 0 & 0 & -\frac{V_3}{d^*}\sqrt{1-\left(\frac{d^*}{2R}\right)^2} & 0 & 0 & \frac{V_3}{d^*}\sqrt{1-\left(\frac{d^*}{2R}\right)^2} & 0 \\ 0 & 0 & \frac{V_3}{2Rd^*} & 0 & 0 & -\frac{V_3}{d^*}\sqrt{1-\left(\frac{d^*}{2R}\right)^2} & 0 & 0 & \frac{V_3}{d^*}\sqrt{1-\left(\frac{d^*}{2R}\right)^2} \\ \hline \frac{V_3}{2Rd^*} & 0 & 0 & -3\frac{V_3}{d^*}\sqrt{1-\left(\frac{d^*}{2R}\right)^2} & 0 & 0 & -3\frac{V_3}{d^*}\sqrt{1-\left(\frac{d^*}{2R}\right)^2} & 0 & 0 \\ 0 & \frac{V_3}{2Rd^*} & 0 & 0 & -3\frac{V_3}{d^*}\sqrt{1-\left(\frac{d^*}{2R}\right)^2} & 0 & 0 & -3\frac{V_3}{d^*}\sqrt{1-\left(\frac{d^*}{2R}\right)^2} & 0 \\ 0 & 0 & \frac{V_3}{2Rd^*} & 0 & 0 & -3\frac{V_3}{d^*}\sqrt{1-\left(\frac{d^*}{2R}\right)^2} & 0 & 0 & -3\frac{V_3}{d^*}\sqrt{1-\left(\frac{d^*}{2R}\right)^2} \end{array}\right] \tag{3.69}$$

As can be seen in (3.69), there is a pattern in the Jacobian that will not change, other than to increase in size as more vehicles are added. This Jacobian (3.69), can be written generally as follows where $I_{n\times n}$ is an identity matrix with n columns and n rows.

$$J|_{eq} = \left[\begin{array}{c|c|c} -\frac{V_n}{d^*}\sqrt{1-\left(\frac{d^*}{2R}\right)^2}I_{n\times n} & -V_n\frac{d^*}{2R}I_{n\times n} & -V_n\frac{d^*}{2R}I_{n\times n} \\ \hline \frac{V_n}{2Rd^*}I_{n\times n} & -\frac{V_n}{d^*}\sqrt{1-\left(\frac{d^*}{2R}\right)^2}I_{n\times n} & \frac{V_n}{d^*}\sqrt{1-\left(\frac{d^*}{2R}\right)^2}I_{n\times n} \\ \hline \frac{V_n}{2Rd^*}I_{n\times n} & -3\frac{V_n}{d^*}\sqrt{1-\left(\frac{d^*}{2R}\right)^2}I_{n\times n} & -3\frac{V_n}{d^*}\sqrt{1-\left(\frac{d^*}{2R}\right)^2}I_{n\times n} \end{array}\right] \tag{3.70}$$

The Jacobian in (3.69) is a very complex matrix to solve symbolically for the eigenvalues. However, in the case of a straight line path $R = \infty$. This greatly simplifies the matrix (see (3.71)) and the Jacobian becomes simple enough to solve for exact eigenvalues.

$$J_{straight}|_{eq} = \left[\begin{array}{ccc|ccc|ccc} -\frac{V_3}{d^*} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -\frac{V_3}{d^*} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{V_3}{d^*} & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & -\frac{V_3}{d^*} & 0 & 0 & \frac{V_3}{d^*} & 0 & 0 \\ 0 & 0 & 0 & 0 & -\frac{V_3}{d^*} & 0 & 0 & \frac{V_3}{d^*} & 0 \\ 0 & 0 & 0 & 0 & 0 & -\frac{V_3}{d^*} & 0 & 0 & \frac{V_3}{d^*} \\ \hline 0 & 0 & 0 & -3\frac{V_3}{d^*} & 0 & 0 & -3\frac{V_3}{d^*} & 0 & 0 \\ 0 & 0 & 0 & 0 & -3\frac{V_3}{d^*} & 0 & 0 & -3\frac{V_3}{d^*} & 0 \\ 0 & 0 & 0 & 0 & 0 & -3\frac{V_3}{d^*} & 0 & 0 & -3\frac{V_3}{d^*} \end{array}\right] \tag{3.71}$$

The straight line path produces three eigenvalues at $(-2+\sqrt{2}i)\frac{V_3}{d^*}$, three eigenvalues at $(-2-\sqrt{2}i)\frac{V_3}{d^*}$, and three eigenvalues at $-\frac{V_n}{d^*}$. These all have negative real values and are thus exponentially stable around the equilibrium point. The straight path Jacobian (3.71) can also be written for the general case of n vehicles as follows.

$$J_{straight}\Big|_{eq} = \left[\begin{array}{c:c:c} -\frac{V_n}{d^*}I_{n\times n} & 0_{n\times n} & 0_{n\times n} \\ \hdashline 0_{n\times n} & -\frac{V_n}{d^*}I_{n\times n} & \frac{V_n}{d^*}I_{n\times n} \\ \hdashline 0_{n\times n} & -3\frac{V_n}{d^*}I_{n\times n} & -3\frac{V_n}{d^*}I_{n\times n} \end{array}\right] \tag{3.72}$$

To generalize the straight path case, with n number of vehicles, there will be n eigenvalues at $(-2+\sqrt{2}i)\frac{V_n}{d^*}$, n eigenvalues at $(-2-\sqrt{2}i)\frac{V_n}{d^*}$, and n eigenvalues at $-\frac{V_n}{d^*}$. No matter the amount of vehicles, the system will converge to a straight path at the desired distance.

For the case of a curved path, the analysis was done numerically. There is a V_3 variable in every term of the $J|_{eq}$ matrix (3.69). This means that changing V_3 will only scale the eigenvalues, but no matter what V_3 is set to, it will not affect the point at which an eigenvalue would go unstable. The ratio $\frac{d^*}{2R}$ is bounded between -1 (right turn with d^* equal to the path diameter) and 1 (left turn with d^* equal to the path diameter). When this ratio equals 0, it is a straight line path since d^* should never be 0. Due to the fact that a right turn will be a mirror image of a left turn, for the numerical analysis, the eigenvalues will be found over the range $\frac{d^*}{2R} \in (0,1)$. There are nine eigenvalues in this system, so the maximum eigenvalue of the system is plotted as shown in Fig. 3.6.

Fig. 3.6 shows that under all possible conditions of R, the system remains exponentially stable around the equilibrium point. This shows that for three vehicles the system is stable. As shown through the pattern in (3.69), the number of vehicles doesn't change the eigenvalues, only the number of eigenvalues at each value. To prove this point, the same algorithm used to get the results in Fig. 3.6 was done again for fifty vehicles and the results are shown in Fig. 3.7. This shows no difference due to the number of vehicles.

This chapter has shown that by using the modified trajectory shaping guidance, each

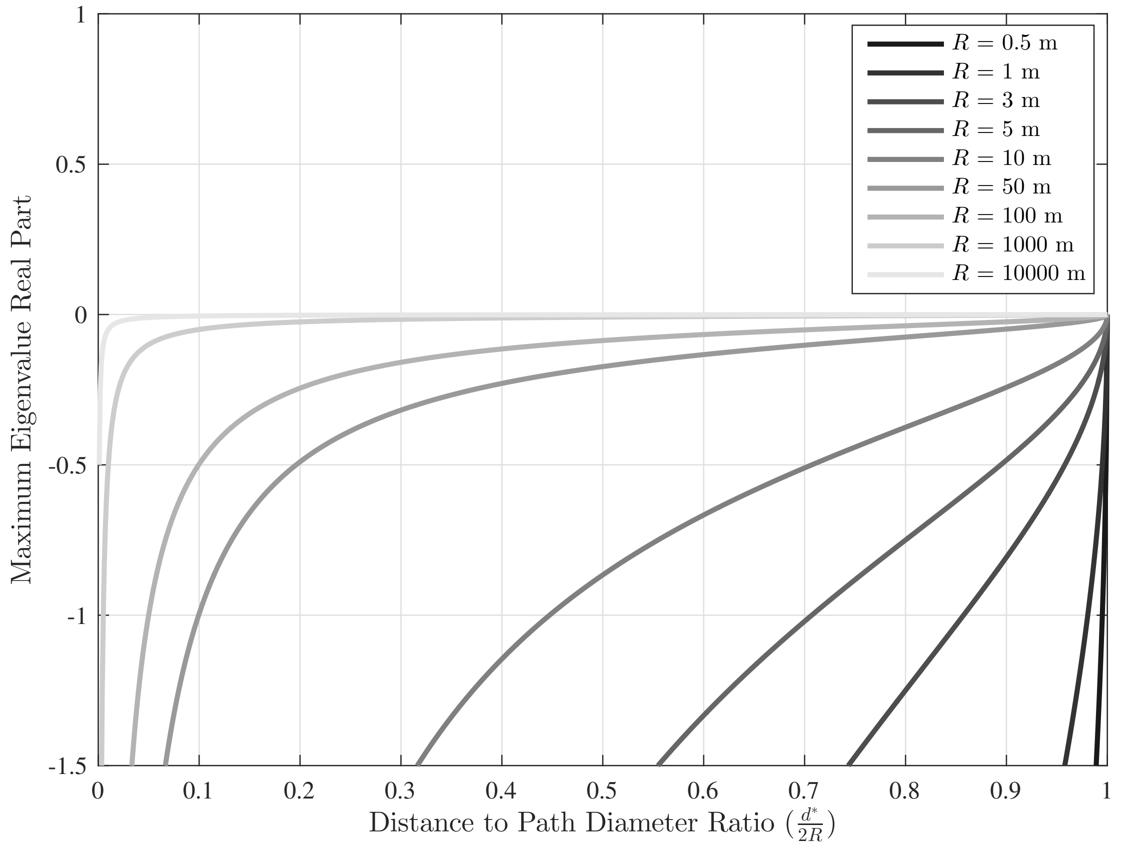


Fig. 3.6: Maximum Possible Real Eigenvalues for Three Vehicles

vehicle in a platoon will converge to the desired path at the desired distance from each other. This is under the original assumptions that the path is of constant radius and that the velocity of the last vehicle in the platoon has a constant velocity. This convergence will happen at any of the parameter values $V_n \in (0, \infty)$, $R \in [-\infty, 0) \cup (0, \infty]$, and $d^* \in (0, 2R)$. This is a powerful benefit in using this simple guidance law for platooning control.

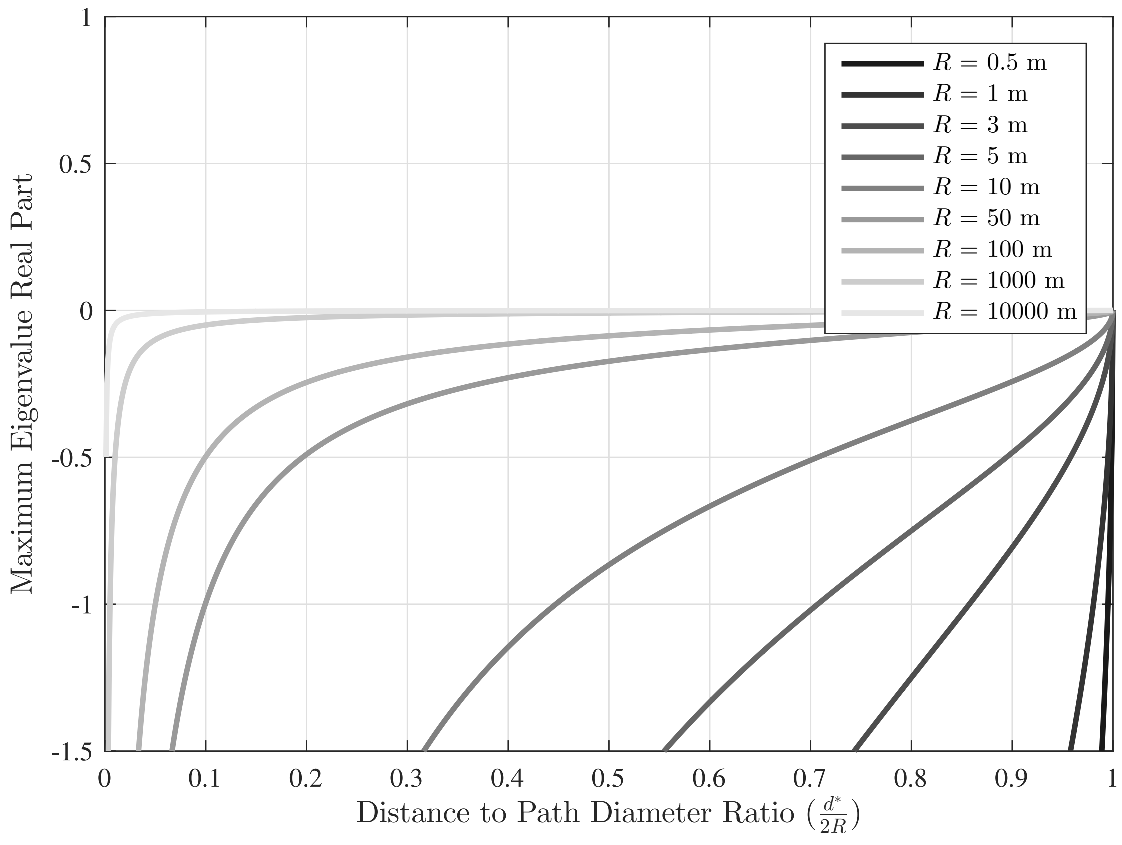


Fig. 3.7: Maximum Possible Real Eigenvalues for Fifty Vehicles

CHAPTER 4
PLATOONING RESULTS

This chapter shows and discusses the results obtained through the simulations and hardware of applying the modified trajectory shaping guidance law to platooning control. Simulation results will first be analyzed to show path convergence of every vehicle in the platoon. The same tests conducted in simulation were performed on Pololu m3pi robots and also show path convergence of all vehicles.

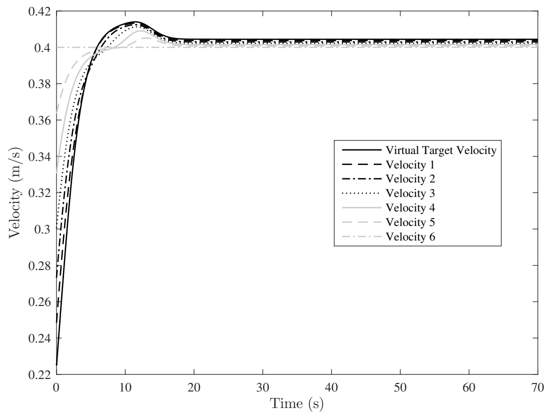
4.1 Simulation

The simulations were created using the same method in Simulink as used for the simulations in Chapter 2, however with the additional capability of having multiple vehicles in the platoon. Each vehicle is treated as the target of the vehicle behind it. The simulations run for this section were performed with the following parameters: $R = 1$m, $d^* = 0.7$m (unless otherwise specified), and $V_n = 0.4$m/s.

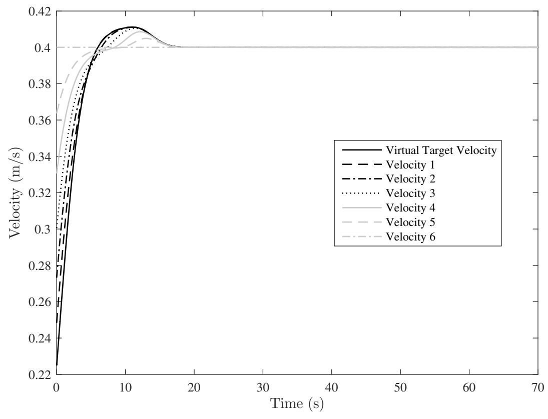
4.1.1 Regular Trajectory Shaping to Sine Modification Comparison

For platooning, convergence was shown analytically to occur for the sine method of trajectory shaping. The regular method has been shown to converge to an offset and each vehicle will converge to an offset of the circle converged to by the vehicle in front of it. Therefore the offset adds with each vehicle in the platoon. This was tested in simulation to give the results in Fig. 4.1.

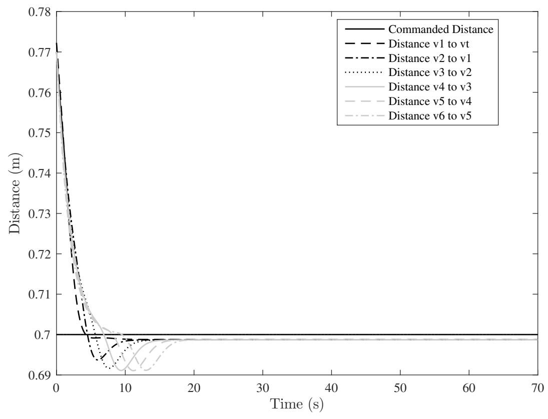
While the results look very similar, they are not equal. The sine method had every vehicle converge to the desired path at the desired spacing. The regular method converged to a slight offset that can be seen in the velocity and distance error plots, but is more difficult to see on the path error plot. The last vehicle in the regular method converged to a circle that was offset 9.1 mm toward the center. This is not an excessive offset, but is



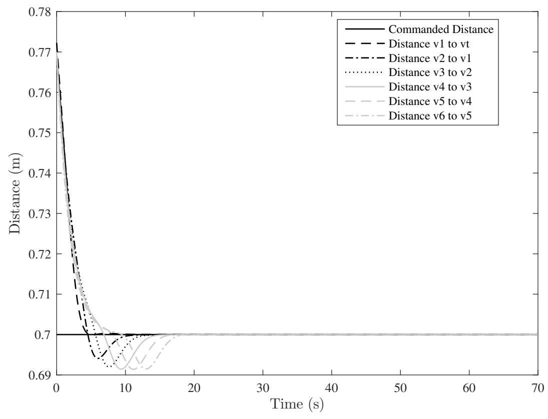
(a) Regular Method Velocities



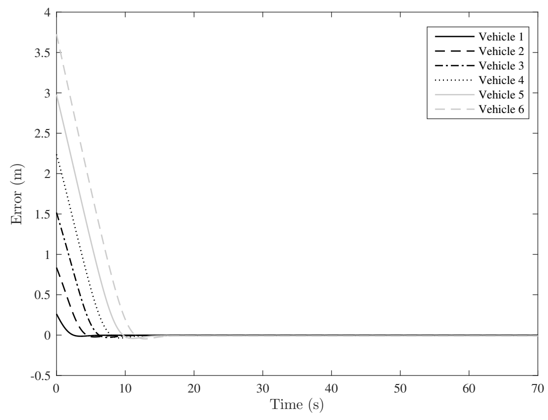
(b) Sine Method Velocities



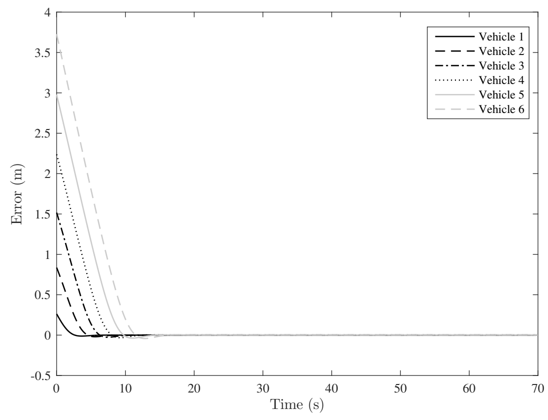
(c) Regular Method Distances



(d) Sine Method Distances



(e) Regular Method Path Errors



(f) Sine Method Path Errors

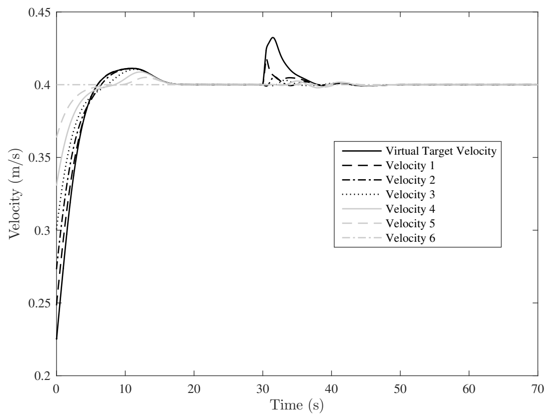
Fig. 4.1: Platooning Simulation Results: Vehicle Errors at $d^* = 0.7$ m

worth getting rid of through the sine method.

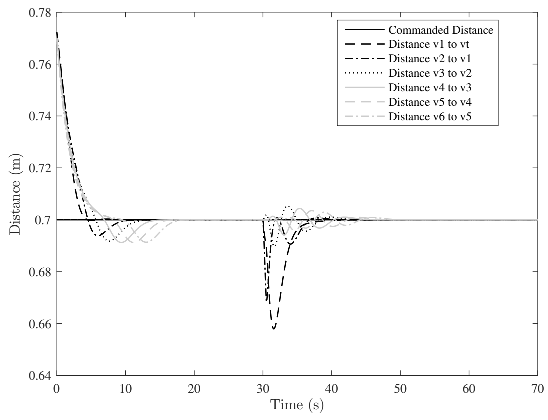
4.1.2 Trajectory Shaping with Disturbance

It was also desired to see how the system reacts to a disturbance. The disturbance was created by adding 1 m/s^2 of acceleration to a vehicle for 0.5 seconds when the simulation reaches 35 seconds. This was first tested by applying it on the front vehicle. The results found in Fig. 4.2 show that the vehicles will converge again back to the path.

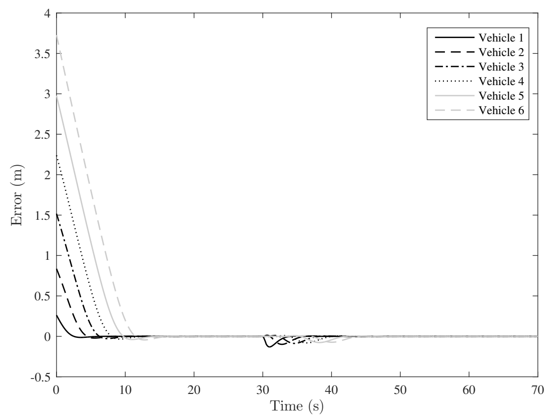
This also shows that the disturbance dissipates with each progressing vehicle in the platoon. This would lead one to believe that the system may also be laterally string stable,



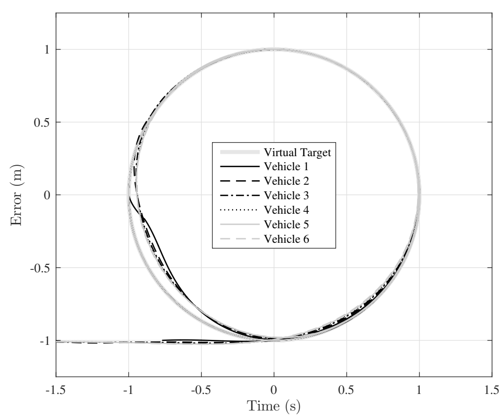
(a) Vehicle Velocities



(b) Vehicle Distances



(c) Vehicle to Path Errors



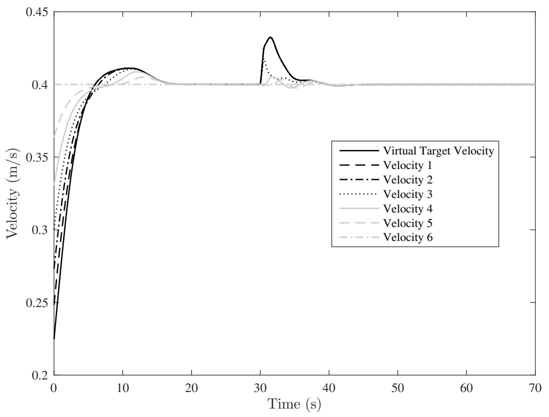
(d) X-Y Positions

Fig. 4.2: Platooning Simulation Results: Platoon Behavior with Front Vehicle Disturbance at $d^* = 0.7$ m

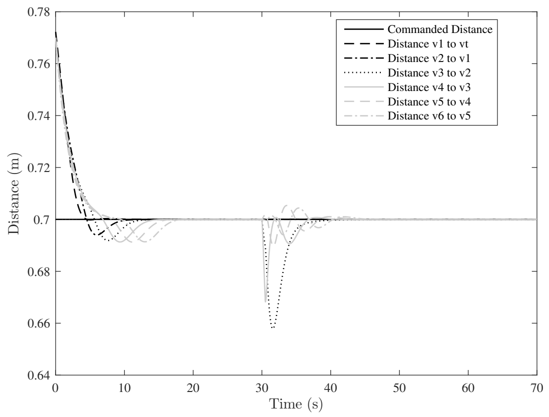
however, proving string stability is not within the scope of this thesis, but could be future work.

The next test case was to do the same disturbance, but this time on Vehicle 3 instead of Vehicle 1. The results shown in Fig. 4.3 show that the path errors of Vehicle 1 and Vehicle 2 are unaffected by the disturbance. However they do slow down to allow the others cars to catch up again. This makes sense because the lateral acceleration of each vehicle is only a function of the vehicle in front of it, so any lateral disturbance in the vehicles behind a vehicle would not affect its lateral control.

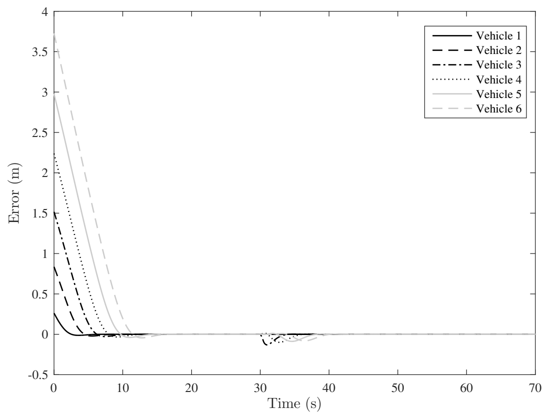
These simulation results validate the analysis from Chapter 3 that using the sine



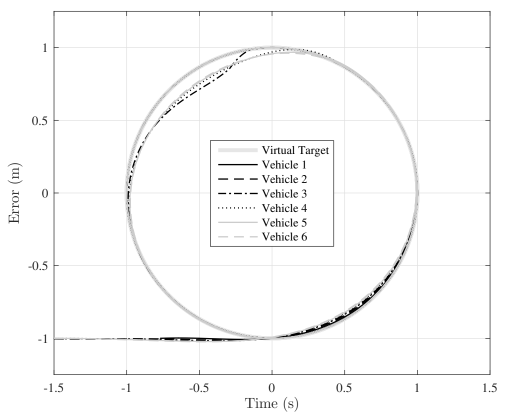
(a) Vehicle Velocities



(b) Vehicle Distances



(c) Vehicle to Path Errors



(d) X-Y Positions

Fig. 4.3: Platooning Simulation Results: Platoon Behavior with Middle Vehicle Disturbance at $d^* = 0.7$ m

method will result in every vehicle in the platoon converging to the desired path at the desired vehicle spacing. The last step of validation in this thesis was to test this on hardware.

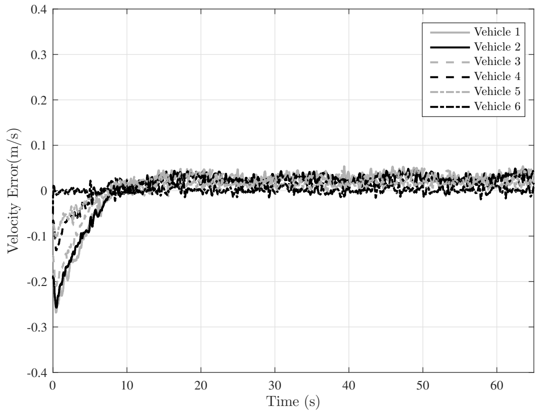
4.2 Hardware

The same system described in Section 2.4, was used to implement the trajectory shaping guidance on a platoon of vehicles. Each m3pi robot had to be equipped with a unique reflective marker template for the Cortex software to recognize in the motion capture system.

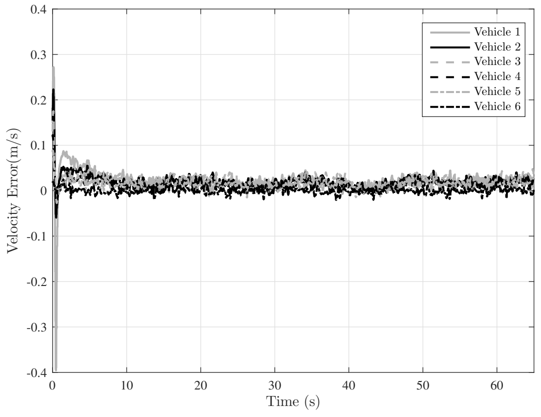
4.2.1 Regular Trajectory to Sine Modification Comparison

One of the main purposes of using trajectory shaping as the control method for a platoon of vehicles in this thesis is its simplicity. It is simple to derive and simple to implement. It was shown in Chapter 3 that as d^* approaches $2R$ the regular method converges to an offset. However, in platooning there are many vehicles, thus if d^* is close to $2R$ there can only be two vehicles on the track. If there are more than two they will start running in to each other. Thus, the more vehicles there are in the platoon, the smaller the ratio $\frac{d^*}{2R}$ must be to fit the vehicles on the track. This then begs the question as to whether it is even worth doing the sine method over the regular method. If one of the goals of this thesis is to keep the controller simple, keeping the sine function out of the controller would keep it that much simpler. However, the main purpose is to achieve accurate path convergence of the entire platoon. Although the path errors may be small they build with each vehicle, thus by the end of the platoon, the error can be substantial. This was tested and the results are shown in Fig. 4.4. To better quantify what is being shown in the graphs, the RMS was taken of these errors from 20s to 65s to see if there was significant improvement in using the sine method. The RMS values are shown in Table 4.1.

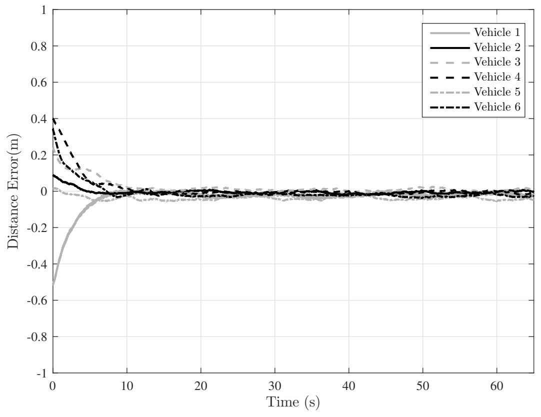
Table 4.1 shows Vehicle 1 did not do as well at keeping its proper distance from the virtual target as the regular method and that Vehicle 2 did slightly worse at converging to the path. However, these are small enough discrepancies that it could very well be from noise, communication latency, and uneven road surface. The strangest anomaly is that



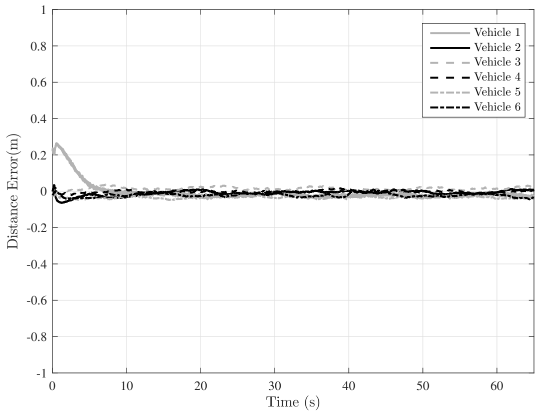
(a) Regular Method Velocity Errors



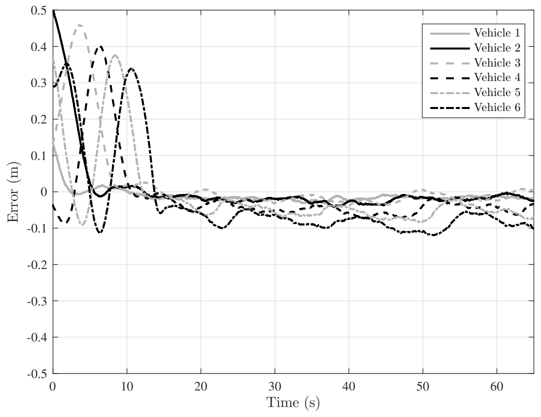
(b) Sine Method Velocity Errors



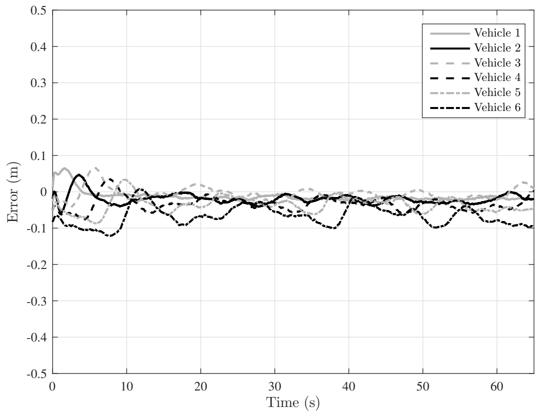
(c) Regular Method Distance Errors



(d) Sine Method Distance Errors



(e) Regular Method Path Errors



(f) Sine Method Path Errors

Fig. 4.4: Platooning Hardware Results: Vehicle Errors at $d^* = 0.9$ m

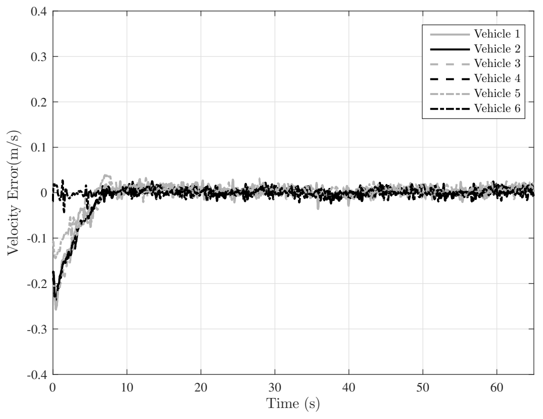
Table 4.1: RMS of Error in Platoon at $d^* = 0.9$m

	Control Method		
	Regular	Sine	Improvement
Velocity Errors (m/s)			
Vehicle 1	0.0313	0.0208	33.5%
Vehicle 2	0.0295	0.0192	34.9%
Vehicle 3	0.0323	0.0252	21.9%
Vehicle 4	0.0189	0.0149	21.1%
Vehicle 5	0.0155	0.0150	3.18%
Vehicle 6	Constant Command Velocity		
Distance Errors (m)			
Vehicle 1	0.0213	0.0234	-9.79%
Vehicle 2	0.0098	0.0091	7.36%
Vehicle 3	0.0111	0.0165	-47.8%
Vehicle 4	0.0097	0.0073	25.1%
Vehicle 5	0.0401	0.0317	21.0%
Vehicle 6	0.0241	0.0239	0.81%
Path Errors (m)			
Vehicle 1	0.0188	0.0187	0.38%
Vehicle 2	0.0236	0.0244	-3.36%
Vehicle 3	0.0198	0.0145	26.4%
Vehicle 4	0.0481	0.0354	26.4%
Vehicle 5	0.0551	0.0355	35.5%
Vehicle 6	0.0862	0.0674	21.8%

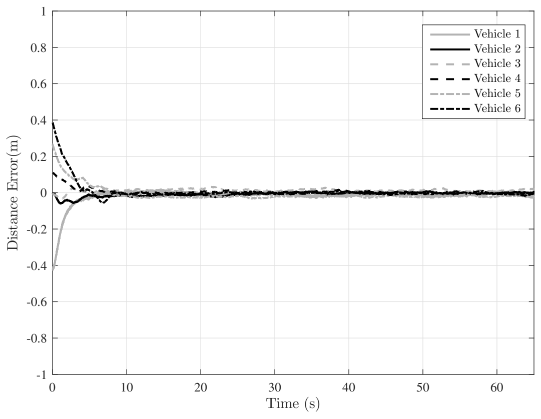
Vehicle 3 shows that it was nearly 50% worse at keeping the desired distance from Vehicle 2 using the sine method. This may also be a strange anomaly due to noise, communication latency or dropout, and uneven road surface. Overall however, this shows that even in the case of noise and disturbance, the sine method tends to produce more accurate results.

The same test cases that were done in simulation were carried out on hardware at the same parameter values. These results are shown in Fig. 4.5 - 4.7. Pictures are also provided in Fig. 4.8 and 4.9 to show the vehicles after they have originally converged to the desired path, the time of the disturbance of the vehicle, and end of the test run showing that the vehicles converge to the path after the disturbance.

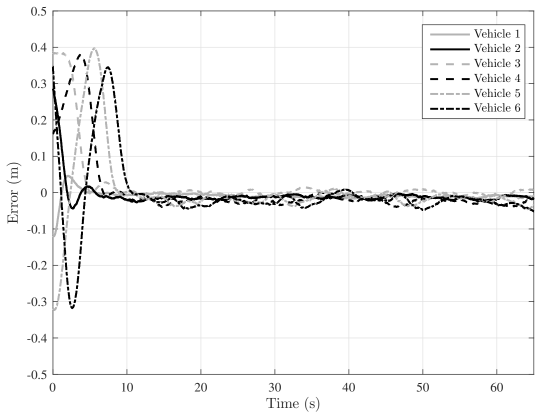
The results found through simulation and hardware testing, confirm the analysis performed in Chapter 3, that using the modified trajectory shaping guidance will provide path



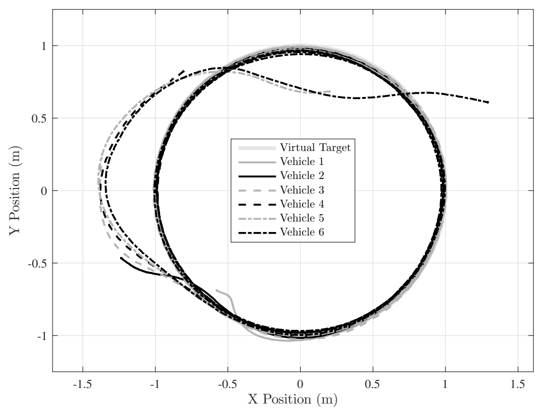
(a) Vehicle Velocity Errors



(b) Vehicle Distance Errors



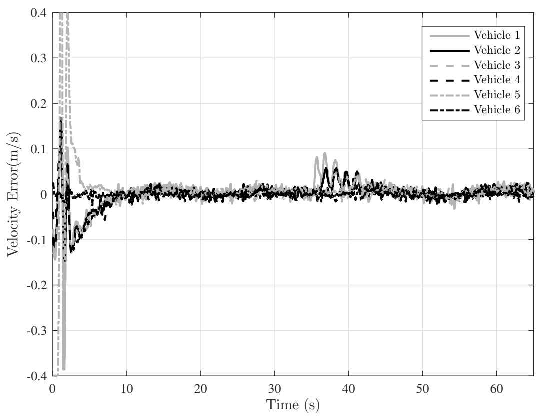
(c) Vehicle to Path Errors



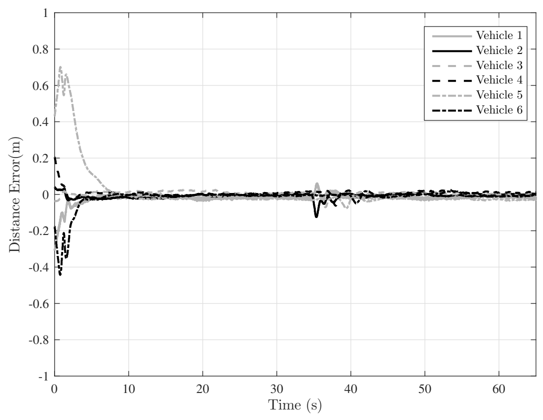
(d) X-Y Positions

Fig. 4.5: Platooning Hardware Results: Platoon Behavior at $d^* = 0.7$ m

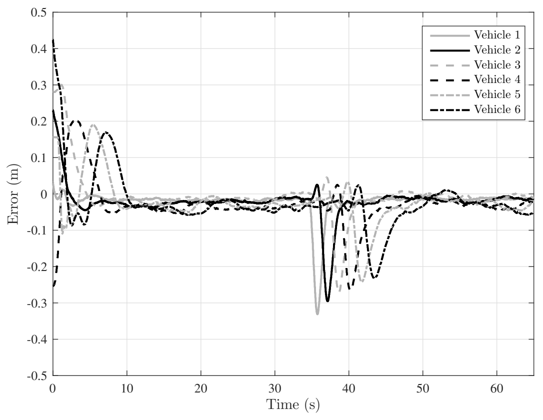
convergence to each vehicle in a platoon of vehicles. This guidance law provides a simple solution to stabilize a platoon of vehicles and get them to follow a desired path.



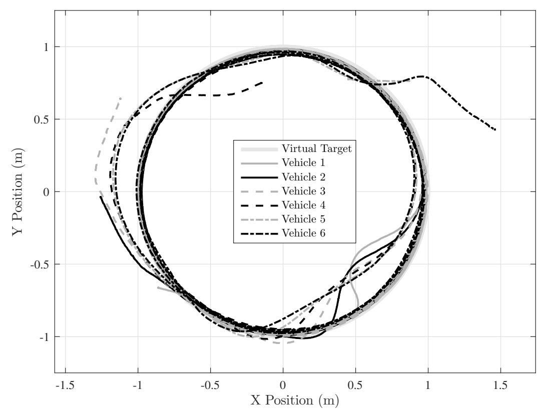
(a) Vehicle Velocity Errors



(b) Vehicle Distance Errors

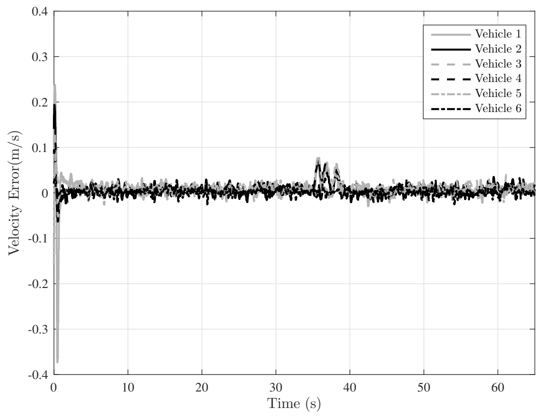


(c) Vehicle to Path Errors

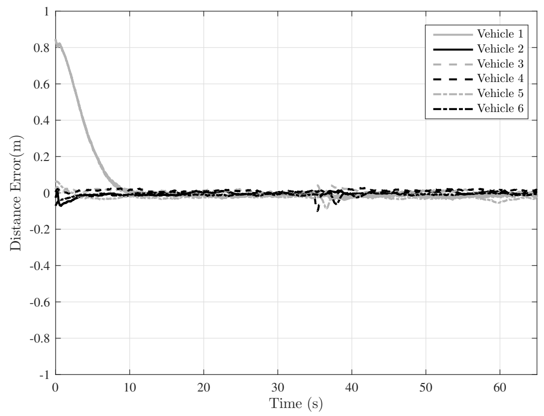


(d) X-Y Positions

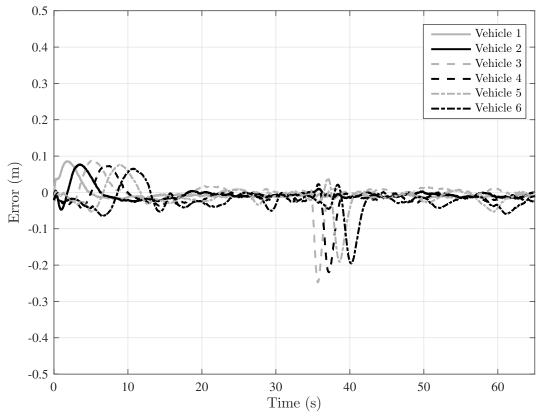
Fig. 4.6: Platooning Hardware Results: Platoon Behavior with Front Vehicle Disturbance at $d^* = 0.7$ m



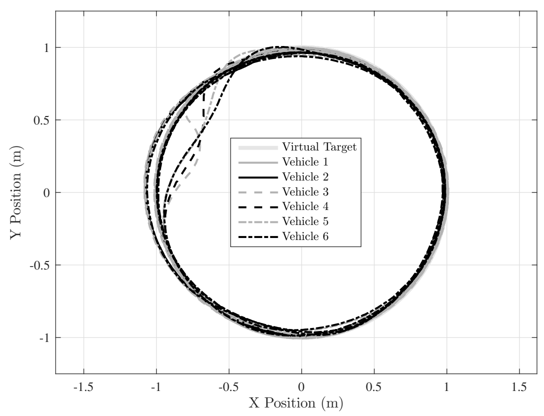
(a) Vehicle Velocity Errors



(b) Vehicle Distance Errors

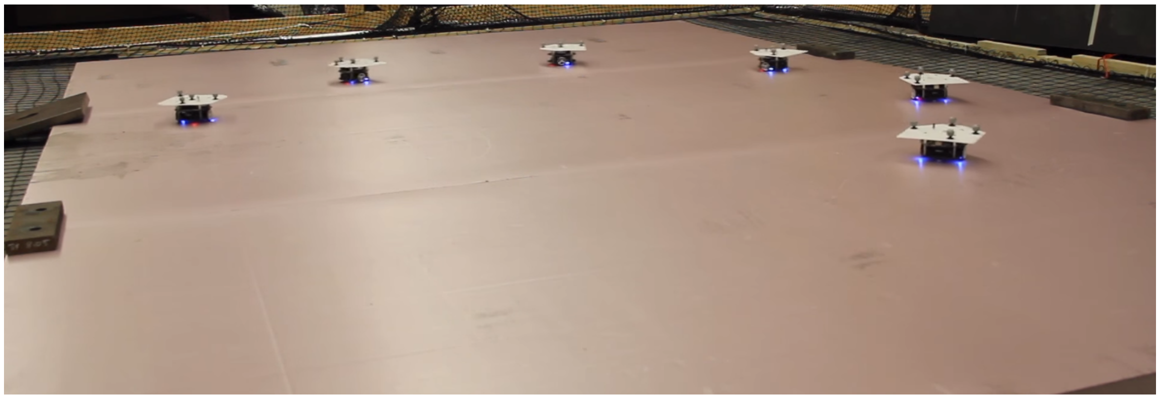


(c) Vehicle to Path Errors

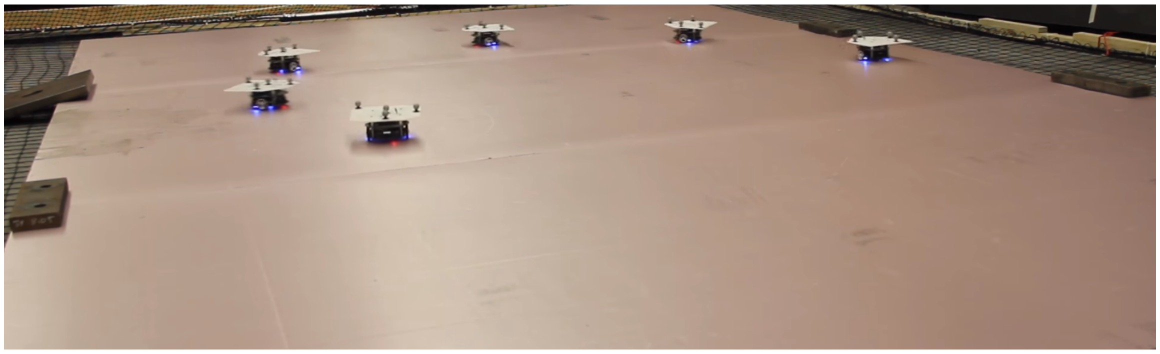


(d) X-Y Positions

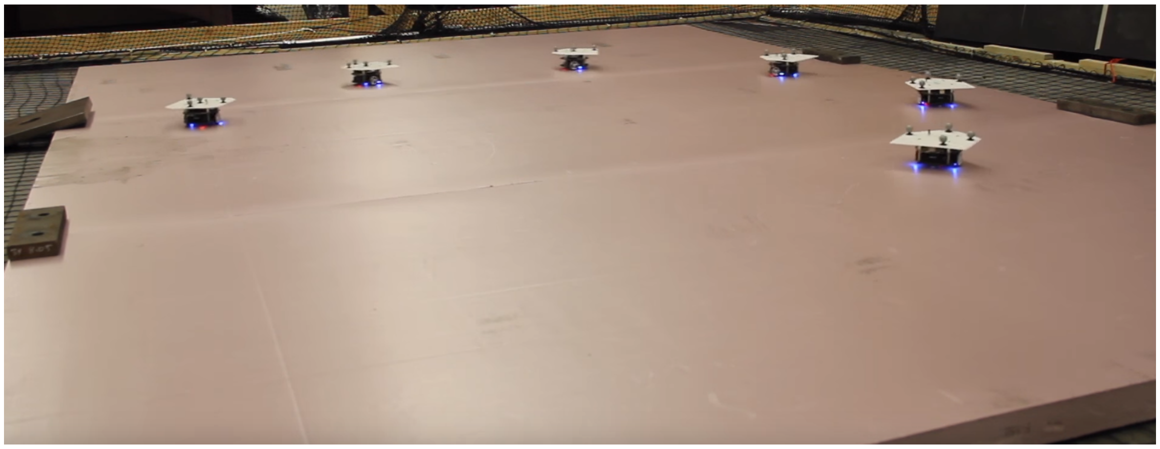
Fig. 4.7: Platooning Hardware Results: Platoon Behavior with Middle Vehicle Disturbance at $d^* = 0.7$ m



(a) Vehicles Have Converged to Path

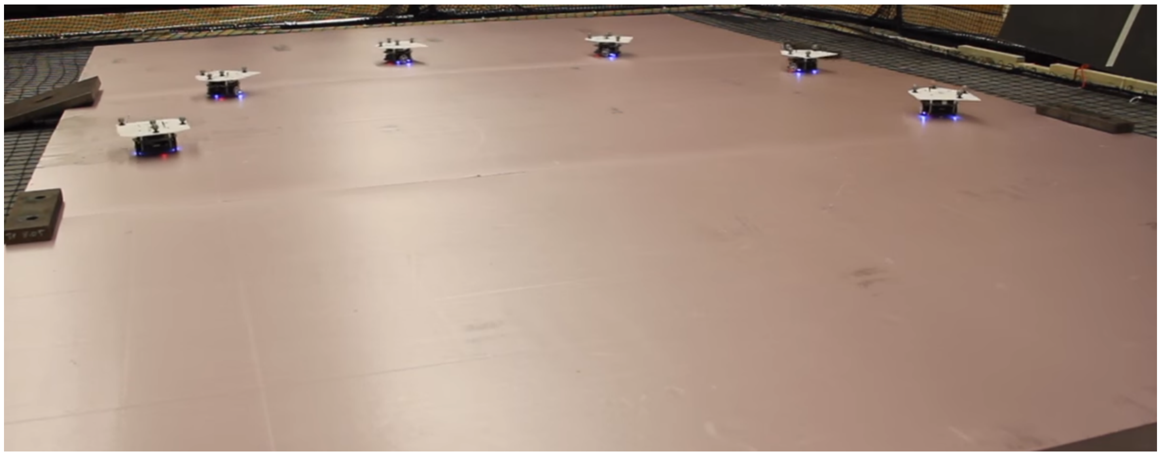


(b) Vehicle One is Given a Lateral Acceleration Disturbance at 30 Seconds

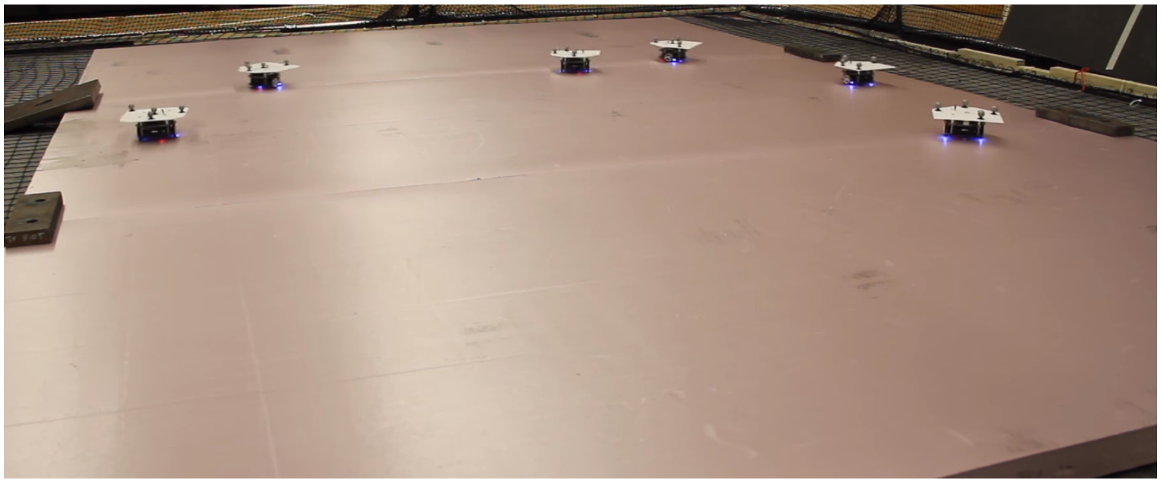


(c) Vehicles Have Converged Again After Disturbance

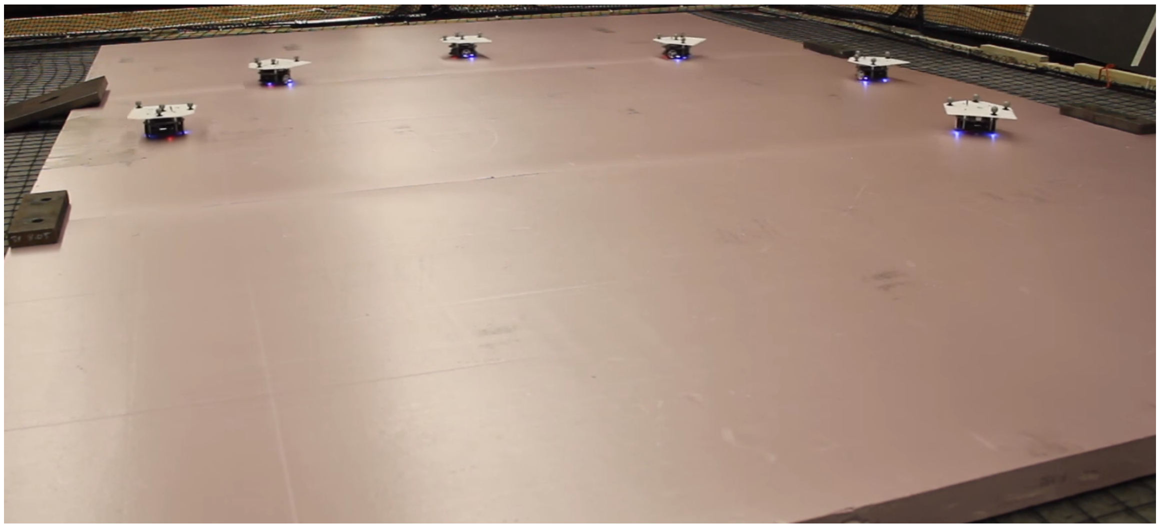
Fig. 4.8: Hardware Test Images: Disturbance to Front Vehicle



(a) Vehicles Have Converged to Path



(b) Vehicle One is Given a Lateral Acceleration Disturbance at 30 Seconds



(c) Vehicles Have Converged Again After Disturbance

Fig. 4.9: Hardware Test Images: Disturbance to Middle Vehicle

CHAPTER 5
CONCLUSION

This thesis proposed the modification of trajectory shaping guidance to achieve more accurate path convergence. This new sine method was developed and shown to converge to a desired path of constant radius with the vehicle at a desired distance from the virtual target. This method of control was then applied to platooning control to ensure each vehicle in the platoon would converge to the path at the desired vehicle separation distance. This was shown mathematically and confirmed through simulation and hardware results. Using simple control laws like missile guidance laws can be a simple solution for path-following platooning control.

Future work on this subject could include: testing this modified trajectory shaping law's ability in paths that don't have constant radius, testing it in applications of variable end vehicle velocity, or performing robustness analysis. It would also be of benefit to study the lateral and longitudinal string stability of this system. The results of the tests in this thesis point to the system being laterally string stable, but a more rigorous study of its string stability would help in understanding its viability in real world applications. Other valuable future work would include the application of this control law using on board sensors to measure the relative distance and angles of the vehicles instead of the motion capture system. This would provide a more realizable system for future autonomous highways.

REFERENCES

[1] R. C. Coulter, “Implementation of the Pure Pursuit Path Tracking Algorithm,” Carnegi Mellon University, Pittsburgh, Tech. Rep. January, 1992.

[2] E. J. Rossetter and J. C. Gerdes, “A Study of Lateral Vehicle Control Under a ‘Virtual’ Force Framework,” *AVEC Conference*, pp. 1–19, 2002. [Online]. Available: http://ddl.stanford.edu/files/ITSjournal{_}v4.pdf

[3] S. Park, J. Deyst, and J. P. How, “A New Nonlinear Guidance Logic for Trajectory Tracking,” in *Proceedings of the AIAA Guidance Navigation and Control Conference and Exhibit.* Providence: AIAA, 2004, pp. 1–16. [Online]. Available: http://acl.mit.edu/papers/gnc{_}park{_}deyst{_}how.pdf

[4] A. Ratnoo, S. Y. Hayoun, A. Granot, and T. Shima, “Path Following using Trajectory Shaping Guidance,” *Journal of Guidance, Control, and Dynamics*, vol. 38, no. 1, pp. 106–116, 2014.

[5] N. Congress, “The Automated Highway System: An Idea Whose Time Has Come,” *Public Roads*, vol. 58, no. 1, 1994.

[6] R. M. Gerdes, C. Winstead, and K. Heaslip, “CPS: An Efficiency-motivated Attack Against Autonomous Vehicular Transportation,” in *Proceedings of the 29th Annual Computer Security Applications Conference on - ACSAC ’13*, 2013. [Online]. Available: http://dx.doi.org/10.1145/2523649.2523658$\delimiter”026E30F$npapers3://publication/doi/10.1145/2523649.2523658

[7] S. Dadras, R. M. Gerdes, and R. Sharma, “Vehicular Platooning in an Adversarial Environment Categories and Subject Descriptors,” in *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, 2015, pp. 167–178.

[8] P. Barooah, P. G. Mehta, and J. P. Hespanha, “Control of large vehicular platoons: Improving closed loop stability by mistuning,” in *Proceedings of the American Control Conference*, 2007, pp. 4666–4671.

[9] M. R. Jovanović and B. Bamieh, “On the ill-posedness of certain vehicular platoon control problems,” *IEEE Transactions on Automatic Control*, vol. 50, no. 9, pp. 1307–1321, 2005.

[10] H. Fritz, “Longitudinal and lateral control of heavy duty trucks for automated vehicle following in mixed traffic: experimental results from the CHAUFFEUR project,” in *Proceedings of the 1999 IEEE International Conference on Control Applications (Cat. No.99CH36328)*, vol. 2, 1999, pp. 1348–1352. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=801168

[11] G. Lee, S. Kim, Y. Yim, J. Jung, S. Oh, and B. Kim, "Longitudinal and lateral control system development for a platoon of vehicles," in *Proceedings 1999 IEEE/IEEJ/JSAI International Conference on Intelligent Transportation Systems (Cat. No.99TH8383)*, 1999, pp. 605–610.

[12] R. Rajamani, H. S. Tan, B. K. Law, and W. B. Zhang, "Demonstration of integrated longitudinal and lateral control for the operation of automated vehicles in platoons," *IEEE Transactions on Control Systems Technology*, vol. 8, no. 4, pp. 695–708, 2000.

[13] J. Ferrin, "CONTROLS OF A TETHER-BASED ROBOTIC CONVOY," Master's thesis, Utah State University, 2007.

[14] C.-K. Ryoo, H. Cho, and M.-J. Tahk, "Optimal Guidance Laws with Terminal Impact Angle Constraint," *Journal of Guidance, Control, and Dynamics*, vol. 28, no. 4, pp. 724–732, 2005.

[15] R. Rajamani, *Vehicle Dynamics and Control (Mechanical Engineering Series)*, 2nd ed., F. F. Ling, Ed. Springer, 2012. [Online]. Available: http://books.google.com/books?hl=en{&}lr={&}id=fRVcCtQ1vzYC{&}oi=fnd{&}pg=PR7{&}dq=Structural+sensitivity+analysis+and+optimization+2{&}ots=FMdU0yb9Ug{&}sig=7yL9DxDFI47HKEGO{_}rZ3v5IUCK0

[16] D. S. Maughan, "Robust Intelligent Sensing and Control Multi-Agent Analysis Platform," Master's thesis, Utah State University, 2016.

APPENDICES

Appendix A
Pololu m3pi Robots

A.1 Robot Overview

The Pololu m3pi Robot is a very simple pre-made robot package. It programs like an Arduino and can even be programmed using the Arduino IDE. This robot was chosen for this thesis due to its cost, power, accuracy and ease of use. It comes with a port to easily plug in an XBee RF module for wireless communication. The robots will only be receiving motor commands from ROS, thus the Arduino code used is listed below, but with each robot set up to take the motor command number appropriate for the vehicle. For example this is the code for Vehicle 1 because it uses `mot_com1`.

```
#include <ros.h>
#include <three_pi_control/motor_command.h>
#include <OrangutanMotors.h>
OrangutanMotors motors;

ros::NodeHandle nh;

int left1 = 0;
int right1 = 0;

void messageCb(const three_pi_control::motor_command& msg){
  left1 = msg.left;
  right1 = msg.right;
}

ros::Subscriber<three_pi_control::motor_command> s("mot_com1", &messageCb);

void setup()
{
  nh.initNode();
  nh.subscribe(s);
}

void loop()
{
  motors.setSpeeds(left1,right1);
  delay(1);
  nh.spinOnce();
}
```

The robots only take a motor command for the left and right wheel. This required preliminary velocity testing to map the motor command inputs to velocity output. This ended up being very linear and quite accurate. There were also slight tuning issues between the left and right wheel. This meant the left wheel would actually go slightly faster or slower than the left wheel for a given motor command and this also needed to be mapped for each vehicle. These mappings are included in the Trajectory Shaping C++ code that will be discussed later.

A.2 XBee Communication

XBees are wireless RF modules that can be used to set up wireless communication with other XBees. There are pre-coded packages in ROS to set up and run networks with XBees. The XBees used were the Series 1 model. Each XBee first had to be plugged in to a Windows machine with XCTU software. This software is used to change the XBee baud rate to communicate with the ROS package. There needs to be at least one XBee set up as the coordinator, and the XBees on the m3pi robots need to be set up to communicate with the coordinator. In the experiments done for platooning in this thesis, it was found that when six vehicles were in the system it was difficult for only one coordinator XBee to keep up with the information throughput. To solve this problem, two coordinator XBees were created. Up to four m3pi robots could receive commands per coordinator XBee. The items needed to get the XBees ready for the experiments in this thesis are listed below.

Coordinator XBes

1. Plug into Windows computer with XCTU and change baud rate to 57600 bps
2. Plug first coordinator XBee into a computer with ROS and ensure that the rosserial_xbee package is installed
3. Run `rosrun rosserial_xbee setup_xbee.py -C -P 1331 /dev/ttyUSB0 0` in the command line

This will make the the XBee a coordinator and allow it to send and receive data from any XBee with the 1331 pan_id

4. Mark the XBee to remember its pan_id number

5. Plug second coordinator XBee into a computer with ROS

6. Run `rosrun rosserial_xbee setup_xbee.py -C -P 1332 /dev/ttyUSB0 0` in the command line

   This will make the the XBee a coordinator and allow it to send and receive data from any XBee with the 1332 pan_id

7. Mark the XBee to remember its pan_id number

m3pi XBees

1. Plug into Windows computer with XCTU and change baud rate to 57600 bps

2. Plug first XBee into a computer with ROS

3. Run the following command in the command line

   ```
   rosrun rosserial_xbee setup_xbee.py -P 1331 /dev/ttyUSB0 1
   ```

   This makes the XBee node 1 and allows it to communicate back and forth only with the coordinator with pan_id 1331

4. Mark the XBee with a 1

5. Plug second XBee into a computer with ROS

6. Run the following command in the command line

   ```
   rosrun rosserial_xbee setup_xbee.py -P 1331 /dev/ttyUSB0 2
   ```

   This makes the XBee node 2 and allows it to communicate back and forth only with the coordinator with pan_id 1331

7. Mark the XBee with a 2
8. Plug third XBee into a computer with ROS
9. Run the following command in the command line

   ```
   rosrun rosserial_xbee setup_xbee.py -P 1331 /dev/ttyUSB0 3
   ```

   This makes the XBee node 3 and allows it to communicate back and forth only with the coordinator with pan_id 1331
10. Mark the XBee with a 3
11. Plug fourth XBee into a computer with ROS
12. Run the following command in the command line

    ```
    rosrun rosserial_xbee setup_xbee.py -P 1331 /dev/ttyUSB0 4
    ```

    This makes the XBee node 4 and allows it to communicate back and forth only with the coordinator with pan_id 1331
13. Mark the XBee with a 4
14. Plug fifth XBee into a computer with ROS
15. Run the following command in the command line

    ```
    rosrun rosserial_xbee setup_xbee.py -P 1332 /dev/ttyUSB0 5
    ```

    This makes the XBee node 5 and allows it to communicate back and forth only with the coordinator with pan_id 1332
16. Mark the XBee with a 5
17. Plug sixth XBee into a computer with ROS
18. Run the following command in the command line

    ```
    rosrun rosserial_xbee setup_xbee.py -P 1332 /dev/ttyUSB0 6
    ```

This makes the XBee node 6 and allows it to communicate back and forth only with the coordinator with pan_id 1332

19. Mark the XBee with a 6

20. Repeat the necessary steps to set up as many XBees as needed

Running in ROS

1. Plug the XBees into the corresponding m3pi robot and power them up

2. Plug Coordinator XBees into a computer on the ROS network

3. Run `roscore`

4. On the computer with the coordinator XBee with pan_id 1331, in the command line run `rosrun rosserial_xbee xbee_network.py /dev/ttyUSB0 1 2 3 4`

5. On the computer with the coordinator XBee with pan_id 1332, in the command line run `rosrun rosserial_xbee xbee_network.py /dev/ttyUSB0 5 6` along with any other XBees set up for that pan_id. If both XBees are on the same computer you might need to use `/dev/ttyUSB1`

6. The XBees should be communicating over the ROS network

A.3 Reflector Templates

Each vehicle needed a unique reflector template so that Cortex could recognize each vehicle. One of the difficult issues is that the markers should not be closer than about 1.5in to each other. There is not much room on the platforms for these robots to make a lot of unique templates. There were even a few attempts at templates that seemed unique, but were actually the mirror image of another template, thus Cortex could recognize it as a different template upside down. These templates are made of 1/8in rigid HDPE polyethylene. Holes had to be accurately drilled for mounting to the m3pi as well as for mounting the markers. Four markers were used in each template and placed accurately so that the RISC-MAAP system could be used to give accurate vehicle positions and headings. The RISC-MAAP system had already been set up to accept other vehicle templates. The file had to be slightly modified to allow it to recognize all the ground vehicle templates.

Appendix B
ROS and Code

B.1 Use of RISC-MAAP System

The RISC-MAAP System as developed at USU by Spencer Maughan [16] was crucial to the success of the hardware testing of this thesis. It already had the infrastructure set up for ROS communication with the motion capture system. A few minor modifications had to be made to include the ability of the system to recognize the ground robot reflector templates. These modifications were made in the risc_estimation package.

B.2 Nodes and Messages

Fig. 2.6 shows the general flow of information in the RISC MAAP System. To be a little more specific to the platooning system, the messages and topics used are discussed in this section. The risc_estimation package has node written called states_estimation.py. This publishes the position, velocity, heading, and angular rates to a topic called /cortex_raw. A node was written in C++ to subscribe to this information and publish motor commands that reflect the modified trajectory shaping guidance. This file is called TrajShap.cpp. The motor commands are a published in a custom message to help keep the data flow as small as possible. This custom message is called motor_command and is defined as follows:

```
int16 left
int16 right
```

These motor commands are subscribed to by the m3pi robots and are translated to actual motor commands of the left and right wheel of the robot.

Another custom message was created for publishing the necessary data to show the results in this thesis. This was not necessary, but was helpful in keeping the data organized. This message is called data_log and is defined as follows:

```
float64[] distance
```

```
float64[] distance_error
float64[] velocity
float64[] velocity_error
float64[] path_error
```

This message is used by a custom written node that subscribes to the cortex data and publishes the important data from the data run. These values could be found post-processing from the cortex data, but sometimes the timestamps didn't match properly with the data so I just created a node to publish the data I wanted to a topic called /data. This is run in the three_pi_control package in the dataTest file.

The only other new topic to this system in the RISC_MAAP platform is the /virt_t topic which uses the std_msgs:Float32 to publish the virtual target x and y positions and velocity. This is also done in the TrajShap file.

B.3 Running an Experiment in ROS

The following commands should be run in order to run experiments on the m3pi multi-robot platform for platooning.

1. Place the vehicles you wish to be in the platoon in their initial conditions and turn them on

2. Run Cortex on the Windows machine and add the vehicles you wish to be in the platoon

3. Open a terminal and run Terminator to allow multiple terminals to be open simultaneously

4. Run the following command: `roslaunch three_pi_control TrajShap.launch rad:=1 vel:=0.4 dis:=0.7 meth:=sine dv:=1 dstb:=1.5`

   This will set the path radius to 1m, the commanded velocity of the last vehicle in the platoon to 0.4m/s, the desired vehicle separation distance to 0.7m, the method to the sine method, the disturbance will occur to vehicle 1, the disturbance will add 1.5 m/s^2 to the commanded lateral acceleration for 0.5 seconds.

5. In a new terminal run the necessary commands to set up the network (see A.2)

6. Ensure all robots are communicating properly with their coordinator XBee

   This can be done by running the command `rostopic list`. If all /mot_com topics are shown, they are communicating properly.

7. After communication is working with all robots, run the following command in a new terminal: `rosrun three_pi_control TrajShap`

8. If you desire to save data it is necessary to rosbag the topics in which you are interested

B.4 Code

TrajShap.launch

```
<launch>
    <arg name="rad" default="1"/>
    <arg name="vel" default="0.4"/>
    <arg name="dis" default="0.7"/>
    <arg name="meth" default="sine"/>
    <arg name="dv" default="1"/>
    <arg name="dstb" default="0"/>

    <node pkg="cortex_ros" name="Cortex_Node" type="stream_markers" />
    <param name="RADIUS" type="double" value="$(arg rad)" />
    <param name="VEL_DES" type="double" value="$(arg vel)" />
    <param name="DIS_DES" type="double" value="$(arg dis)" />
    <param name="Method" type="str" value="$(arg meth)" />
    <param name="DSTB_VEH" type="int" value="$(arg dv)" />
    <param name="DSTB" type="double" value="$(arg dstb)" />
    <node pkg="risc_estimation" name="Estimator_Node" type="ground_states_estimation.py"/>
    <node pkg="three_pi_control" name="Data_logger" type="dataTest" output = "screen"/>
    <node pkg="rosserial_xbee" name="Network1" type="xbee_network.py" args="/dev/ttyUSB0 1 2 3 4"/>
    <!-- node pkg="rvis_node" name="Visualization" type="pi_robot"/ -->
    <!-- node pkg="rviz" name="rviz_node" type="rviz"/ -->

</launch>
```

TrajShap.cpp

```
#include "ros/ros.h"
#include "math.h"
#include "three_pi_control/motor_command.h"
#include "geometry_msgs/Point32.h"
#include <risc_msgs/Cortex.h>
```

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

/*==================================\\
||    Initialize Global Variables    ||
||    Set up for Max of 8 vehicles   ||
\\==================================*/

int s;              // Number of Vehicles detected by Cortex
float x[8];         // X position in meters
float y[8];         // Y position in meters
float u[8];         // X velocity in m/s
float v[8];         // Y velocity in m/s
float gamma_v[8];   // Vehicle Heading in radians
float span = 0.082; // Distance between m3pi wheels
double pi  = 3.14159265359;


/*=================================\\
||    Cortex Subscriber Function    ||
\\=================================*/

void getdata(const risc_msgs::Cortex& msg)
{
  // Define the number of vehicles recognized by Cortex
  s = msg.Obj.size();

  // Define the position, heading and velocity array values for each vehicle
  for(int i = 0; i<s; i = i+1)
  {
    // Position and Heading information is only used if Cortex gives allowable data
    if (fabs(msg.Obj[i].x) < 5 and fabs(msg.Obj[i].y) < 5 and fabs(msg.Obj[i].z) < 1)
    {
       x[i]       = msg.Obj[i].x;
       y[i]       = msg.Obj[i].y;
       gamma_v[i] = - msg.Obj[i].psi; // Convert NED from Cortex to NWU
    }

    // Velocity information is only used if Cortex gives allowable data
    if (fabs(msg.Obj[i].u) < 3 and fabs(msg.Obj[i].v) < 3)
    {
       u[i] = msg.Obj[i].u;
       v[i] = msg.Obj[i].v;
    }
  }
}


/*==========================\\
||    Define a -pi to pi     ||
||  Angle Wrapping Equation  ||
\\==========================*/

double pi2pi(double ang)
```

```
{
  while (ang > pi)
  {
    ang = ang - 2*pi;
  }
  while (ang < -pi)
  {
    ang = ang + 2*pi;
  }
  return ang;
}


/*=================================\\
||  Define the Trajectory Shaping  ||
||  Lateral Acceleration Equation  ||
\\=================================*/

double trsh(const char *meth, double lam, double gam_v, double gam_t, double d, double v, double dcom)
{
  // Check the user input conditions of which version of Trajectory Shaping to use
  bool req = (std::strcmp(meth,"regular") == 0);
  double a_lat = 0;

  if (req)  // Regular Method Trajectory Shaping
  {
    a_lat = v*v/d*(4*pi2pi(lam - gam_v) + 2*pi2pi(lam - gam_t));
  }
  else  // Sine Method Trajectory Shaping
  {
    a_lat = v*v/d*(4*sin(lam - gam_v) + 2*sin(lam - gam_t));
  }
  return a_lat;
}


/*==============================\\
||  Define Function to Compute  ||
||     Line of Sight Angle      ||
\\==============================*/

double LAM(double x,double y,double xt,double yt)
{
  double lambda = atan2((yt-y),(xt-x));
  return lambda;
}


/*==============================\\
||  Define Function to Compute  ||
||  Distance Between Vehicles   ||
\\==============================*/

double distance(double x,double y,double xt,double yt)
{
  double dist = sqrt((xt - x)*(xt - x) + (yt - y)*(yt - y));
  return dist;
}


/*=================\\
||  MAIN FUNCTION  ||
\\=================*/

int main(int argc, char **argv)
{
  // Initialize ROS Node for Trajectory_Shaping
  ros::init(argc, argv, "Trajectory_Shaping");
  ros::NodeHandle n;
  ros::Time::init();

  // Initialize User Specified Parameter Values
  double rad;       // Path Radius (m)  If rad = 0, this is considered a straight line
  double vel_d;     // Commanded Velocity of last vehicle (m/s)
  double dist_com;  // Desired Vehicle Spacing Distance (m)
  int dv;           // Vehicle to recieve disturbance
  double dstb;      // Disturbanc amount (m/s^2)
  std::string meth; // Method: Sine or Regular

  // Set Parameter Values to User Input
  n.getParam("DIS_DES",dist_com);
  n.getParam("VEL_DES",vel_d);
  n.getParam("RADIUS",rad);
  n.getParam("Method",meth);
  n.getParam("DSTB_VEH",dv);
  n.getParam("DSTB",dstb);


  /*============================\\
  || Initialize Other Variables ||
  \\============================*/

  // Virtual Target
  double xt;         // Virtual Target X position (m)
  double yt;         // Virtual Target Y position (m)
  double vt = 0;     // Virtual Target Velocity (m/s)
  double gamma_t;    // Virtual Target Heading
  double ang_v = 0;  // Virtual Target Angular Velocity (only used for curved paths)
  double ang_p = 0;  // Virtual Target Angular Position (only used for curved paths)

  // Set Initial Virtual Target Postion and Heading Based on User Path Input
  if (fabs(rad) < 0.000001) // Straight Path
  {
    xt      = -0.1;
    yt      = -0.1;
    gamma_t = pi/4;
  }
  else // Curved Path
  {
    xt      = 0;
    yt      = -rad;
    gamma_t = 0;
  }

  // Vehicle Variables
  double d[8]  = {1}; // Distance Between Vehicles (m)
  double vel_com[8];  // Commanded Vehicle Velocities (m/s)
  double vel[8];      // Actual Vehicle Velocities (m/s)
  double lamb[8];     // Line Of Sight Angle (rad)
  double alat  = 0;   // Commanded Lateral Acceleration (m/s^2) + = left turn    - = right turn
  double rturn = 0;   // Turn Radius Calculated from velocity and lateral acceleration

  // Robot Variables
  double vl = 0;   // Commanded Velocity of Left Wheel (m/s)
  double vr = 0;   // Commanded Velocity of Right Wheel (m/s)
  double left[8];  // Left Motor Command (pwm signal)
  double right[8]; // Right Motor Command (pwm signal)

  // Other Variables
  double dt    = 1./50.; // One over the loop rate (s)
  double begin = 0;      // Time holder to be used later
  double later = 0;      // Time holder to be used later
  int count2   = 0;      // Counter to be used later

  // Set ROS Loop Rate
  ros::Rate loop_rate(50);  // Set to run at 50 Hz

  // Subscribe to /cortex_raw topic
  ros::Subscriber sub = n.subscribe("cortex_raw", 1, getdata);

  // Set up publishers to 8 mot_com topics
  ros::Publisher motor_com1 = n.advertise<three_pi_control::motor_command>("mot_com1", 1);
  ros::Publisher motor_com2 = n.advertise<three_pi_control::motor_command>("mot_com2", 1);
  ros::Publisher motor_com3 = n.advertise<three_pi_control::motor_command>("mot_com3", 1);
  ros::Publisher motor_com4 = n.advertise<three_pi_control::motor_command>("mot_com4", 1);
  ros::Publisher motor_com5 = n.advertise<three_pi_control::motor_command>("mot_com5", 1);
  ros::Publisher motor_com6 = n.advertise<three_pi_control::motor_command>("mot_com6", 1);
  ros::Publisher motor_com7 = n.advertise<three_pi_control::motor_command>("mot_com7", 1);
  ros::Publisher motor_com8 = n.advertise<three_pi_control::motor_command>("mot_com8", 1);

  // Set up Virtual Target Publisher
  ros::Publisher virtual_target = n.advertise<geometry_msgs::Point32>("virt_t", 1);

  // Pause the node for 10 seconds before running any code to allow other nodes to set up correctly
  begin = ros::Time::now().toSec();  // Get time now
  later = ros::Time::now().toSec();  // Get time later to compare
  while (later - begin < 10)
  {
      later = ros::Time::now().toSec();
  }

  // Set begin to start counting when code starts
  begin = ros::Time::now().toSec();
  int count = 0;
  while (ros::ok())
```

```
{
  /*===========================\\
  || Set Virtual Target States ||
  \\===========================*/

  vt = vel_com[0]*dist_com/d[0];
  if (fabs(rad) < 0.000001) // Straight Path
  {
    xt = xt + vt*dt*cos(pi/4);
    yt = yt + vt*dt*sin(pi/4);
    gamma_t = pi/4;
  }
  else // Curved Path
  {
    ang_v = vt/rad;
    ang_p = atan2(yt,xt)+ang_v*dt;
    gamma_t = (ang_p+pi/2);
    xt = rad*cos(ang_p);
    yt = rad*sin(ang_p);
  }



  /*======================\\
  || Set Vehicle Commands ||
  \\======================*/

  later = ros::Time::now().toSec();
  three_pi_control::motor_command mot[8]; // Initialize motor command message to publish

  // Run through all recognized vehicles and provide commands
  for(int i = 0; i<s; i = i+1)
  {
    /*============================================================\\
    ||         Calculate Vehicle Velocities, distances,           ||
    ||  line of sight angles, and commanded lateral accelerations  ||
    \\============================================================*/

    vel[i] = sqrt(u[i]*u[i] + v[i]*v[i]);  // Total Velocity (m/s)
    if (i == 0)  // For Vehicle 1
    {
       d[i] = distance(x[i],y[i],xt,yt);
       lamb[i] = LAM(x[i],y[i],xt,yt);
       alat = trsh(meth.c_str(), lamb[i], gamma_v[i], gamma_t, d[i], vel[i], dist_com);

    }
    else  // For all vehicle besides Vehicle 1
    {
       d[i] = distance(x[i],y[i],x[i-1],y[i-1]);
       lamb[i] = LAM(x[i],y[i],x[i-1],y[i-1]);
       alat = trsh(meth.c_str(), lamb[i], gamma_v[i], gamma_v[i-1], d[i], vel[i], dist_com);
    }



    /*============================================================\\
    ||  Add user specified disturbance to user specified vehicle  ||
\\==========================================================*/

if ((later - begin) > 55 and (later - begin) < 55.5 and i == dv-1)
{
   alat = alat + dstb;
}


/*==================================================================\\
||                       Give commands from 20s - 93s                   ||
||  This gives a little more time for communication and other nodes to ||
||  set up properly and turns of motors after 83 seconds of running    ||
\\==================================================================*/

if ((later - begin) > 20 and (later - begin) < 93)
{
   // Print to screen that the commands should be publishing correct motor commands
   if(count2 == 0)
   {
      printf("Should Be Publishing . . . \n");
      count2 = 1;
   }

   // Set the commanded vehicle velocities
   if (i == s-1)  // Last Vehicle
   {
      vel_com[i] = vel_d;
   }
   else  // All vehicles except the last vehicle
   {
      vel_com[i] = vel_com[i+1]*dist_com/d[i+1];
   }

   // Include small proportional controller to the Velocity Command
   vel_com[i] = vel_com[i] + 0.05*(vel_com[i] - vel[i]);
}
else
{
   vel_com[i] = 0;

   // Print to the Screen that the commanded velocities are no longer published
   // The motors should all stop
   if(count2 == 1)
   {
      printf("Finished Publishing . . . \n");
      count2 = 2;
   }
}


/*============================================\\
||  Set the Left and Right Vehicle Velocities  ||
\\============================================*/

if (alat == 0)
```

```
{
  vl = vel_com[i];
  vr = vel_com[i];
}
else
{
  rturn = vel_com[i]*vel_com[i]/alat;
  vl = vel_com[i]*(1 - span/(2*rturn));
  vr = vel_com[i]*(1 + span/(2*rturn));
}


/*======================================================\\
||  Velocity to Motor Command Mapping for Each Vehicle  ||
\\======================================================*/

if (i == 0)
{
  right[i] = (vr + 0.039889654831088)/0.004545032044941;
  left[i]  = (vl + 0.039889654831088)/0.004545032044941;
  right[i] = 0.949*right[i];
}
else if (i == 1)
{
  right[i] = (vr + 0.048971242120369)/0.004640938536453;
  left[i]  = (vl + 0.048971242120369)/0.004640938536453;
  right[i] = 0.99*right[i];
}
else if (i == 2)
{
  right[i] = (vr + 0.041594957446630)/0.004566105857106;
  left[i]  = (vl + 0.041594957446630)/0.004566105857106;
  right[i] = 0.94*right[i]+4;
}
else if (i == 3 or i == 4)
{
  right[i] = (vr + 0.041594957446630)/0.004566105857106;
  left[i]  = (vl + 0.041594957446630)/0.004566105857106;
  left[i] = 0.98*left[i] + 2;
}
else
{
  right[i] = (vr + 0.041594957446630)/0.004566105857106;
  left[i]  = (vl + 0.041594957446630)/0.004566105857106;
  left[i] = 0.9995*left[i];
}

if(vel_com[i] == 0)
{
    left[i] = 0;
    right[i] = 0;
}

// Set message for publishing motor commands
mot[i].left = int(left[i]);
```

```
      mot[i].right = int(right[i]);
    }


    /*==========================\\
    ||  Publish Motor Commands  ||
    \\==========================*/

    motor_com1.publish(mot[0]);
    motor_com2.publish(mot[1]);
    motor_com3.publish(mot[2]);
    motor_com4.publish(mot[3]);
    motor_com5.publish(mot[4]);
    motor_com6.publish(mot[5]);
    motor_com7.publish(mot[6]);
    motor_com8.publish(mot[7]);


    /*===============================\\
    ||  Publish Virtual Target Data  ||
    \\===============================*/

    geometry_msgs::Point32 virt;
    virt.x = xt;
    virt.y = yt;
    virt.z = vt;

    virtual_target.publish(virt);

    /*==============================\\
    ||  Required ROS Loop Commands  ||
    \\==============================*/

    ros::spinOnce();
    loop_rate.sleep();
    ++count;
  }

  ros::spin();
  return 0;
}
```

dataTest.cpp

```
#include "ros/ros.h"
#include "math.h"
#include "three_pi_control/data_log.h"
#include "geometry_msgs/Point32.h"
#include <risc_msgs/Cortex.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
```

```
/*===================================\\
||    Initialize Global Variables    ||
||    Set up for Max of 8 vehicles   ||
\\===================================*/

int s;      // Number of Vehicles detected by Cortex
float x[8]; // X position in meters
float y[8]; // Y position in meters
float u[8]; // X velocity in m/s
float v[8]; // Y velocity in m/s
float xt;   // Virtual Target X Position (m)
float yt;   // Virtual Target Y Position (m)
float vt;   // Virtual Target Velocity (m/s)
double pi = 3.14159265359;


/*==================================\\
||    Cortex Subscriber Function    ||
\\==================================*/

void getdata(const risc_msgs::Cortex& msg)
{
  // Define the number of vehicles recognized by Cortex
  s = msg.Obj.size();

  // Define the position and velocity array values for each vehicle
  for(int i = 0; i<s; i = i+1)
  {
    // Position and Heading information is only used if Cortex gives allowable data
    if (fabs(msg.Obj[i].x) < 5 and fabs(msg.Obj[i].y) < 5 and fabs(msg.Obj[i].z) < 1)
    {
       x[i]       = msg.Obj[i].x;
       y[i]       = msg.Obj[i].y;
    }

    // Velocity information is only used if Cortex gives allowable data
    if (fabs(msg.Obj[i].u) < 3 and fabs(msg.Obj[i].v) < 3)
    {
       u[i] = msg.Obj[i].u;
       v[i] = msg.Obj[i].v;
    }
  }
}


/*====================================\\
|| Virtual Target Subscriber Function ||
\\====================================*/

void virt(const geometry_msgs::Point32& msg)
{
  xt = msg.x;
  yt = msg.y;
  vt = msg.z;
}
```

```
/*==============================\\
||  Define Function to Compute  ||
||  Distance Between Vehicles   ||
\\==============================*/

double distance(double x,double y,double xt,double yt)
{
  double dist = sqrt((xt - x)*(xt - x) + (yt - y)*(yt - y));
  return dist;
}


/*=================\\
||  MAIN FUNCTION  ||
\\=================*/

int main(int argc, char **argv)
{
  // Initialize ROS Node for Data_Logging
  ros::init(argc, argv, "Data_Logging");
  ros::NodeHandle n;

  // Initialize User Specified Parameter Values
  double rad;       // Path Radius (m)  If rad = 0, this is considered a straight line
  double vel_d;     // Commanded Velocity of last vehicle (m/s)
  double dist_com;  // Desired Vehicle Spacing Distance (m)

  // Set Parameter Values to User Input
  n.getParam("DIS_DES",dist_com);
  n.getParam("VEL_DES",vel_d);
  n.getParam("RADIUS",rad);

  // Initialize Other Variables
  double vel[8];     // Actual Vehicle Velocities (m/s)
  double d[8]  = {1}; // Distance Between Vehicles (m)

  // Subscribe to /cortex_raw topic
  ros::Subscriber sub = n.subscribe("cortex_raw", 1, getdata);

  // Subscribe to /virt_t topic
  ros::Subscriber sub2 = n.subscribe("virt_t", 1, virt);

  // Set up publisher to /data topic
  ros::Publisher dat = n.advertise<three_pi_control::data_log>("data", 1);

  // Set ROS Loop Rate
  ros::Rate loop_rate(100);  // Set to run at 50 Hz

  // Rund main Code
  int count = 0;
  while (ros::ok())
  {
    three_pi_control::data_log logger; // Initialize data logging message to publish
    /*================\\
    ||  Collect Data  ||
    \\================*/
    for(int i = 0; i<s; i = i+1)
    {
      vel[i] = sqrt(u[i]*u[i] + v[i]*v[i]);  // Total Velocity (m/s)

      // Calculate Distance Data
      if (i == 0) // For Vehicle 1
      {
         d[i] = distance(x[i],y[i],xt,yt);
      }
      else  // For all other vehicles beside Vehicle 1
      {
         d[i] = distance(x[i],y[i],x[i-1],y[i-1]);
      }

      // Define Logger Message to publish
      logger.distance.resize(s);
      logger.distance[i] = d[i];
      logger.distance_error.resize(s);
      logger.distance_error[i] = d[i] - dist_com;
      logger.velocity.resize(s);
      logger.velocity[i] = vel[i];
      logger.velocity_error.resize(s);
      logger.velocity_error[i] = vel[i] - vel_d;
      logger.path_error.resize(s);
      logger.path_error[i] = sqrt(x[i]*x[i] + y[i]*y[i]) - rad;
    }
    // Define Logger Message for Virtual Target Velocities
    logger.velocity.resize(s+1);
    logger.velocity[s] = vt;
    logger.velocity_error.resize(s+1);
    logger.velocity_error[s] = vt - vel_d;

    // Publish Data to /data topic
    dat.publish(logger);


    /*==============================\\
    ||  Required ROS Loop Commands  ||
    \\==============================*/

    ros::spinOnce();

    loop_rate.sleep();
    ++count;
  }

  ros::spin();
  return 0;
}
```