# Control Barrier Functions and Application in Vehicle Control

Master Thesis Applied Mathematics

University of Groningen

Student: Siddhanta Mishra

First Supervisor: Prof. B. Besselink

Second Supervisor: Prof. S. Trenn

March 20, 2026

ABSTRACT

This thesis looks to understand the concept of Control Barrier Functions (CBFs) and the application of such functions in Adaptive Cruise Control (ACC) and Lane Keeping (LK). The thesis will begin with certain basic definitions, followed by a brief overview at Control Lyapunov Functions (CLFs). Then the concept of safety will be defined and formalised, followed by Nagumo's Theorem. We will then utilise Nagumo's theorem to prove how CBFs can ensure that the system remains safe. We will then use the CBFs in regard to the ACC and LK problems to achieve different goals with the assistance of different CLFs and additional CBFs. These goals will include the prevention of crashes, remaining within the lane, even under changing lane widths. The final goal would be to ensure a formation shape within the safe space.

CONTENTS

1 Introduction 4
2 Preliminaries 4
2.1 Basic Definitions . . . . . 4
2.2 Control Lyapunov Function . . . . . 5
3 Safety 7
3.1 Nagumo's Theorem . . . . . 7
3.2 Control Barrier Function . . . . . 9
3.3 Application . . . . . 10
4 Simulations 13
4.1 ACC . . . . . 13
4.2 Adaptive Cruise Control and Lane Keeping . . . . . 15
4.3 Fusing Barrier Functions . . . . . 21
4.4 Position and Time Varying Barrier Functions . . . . . 22
4.5 Network Model . . . . . 24
5 Conclusion & Further Ideas 27
6 Acknowledgment 29
7 Appendix 30

## 1 INTRODUCTION

As we have begun to see in the past few years, control systems have found homes in many aspects of life. This extends from home appliances to rockets. Hence, these control systems have an increasing demand to account for real-world dangers and ensure that they maintain a degree of safety. Although this could, in some systems, be achieved mechanically, having control designs that ensure safety would be more cost-efficient. Hence, we come to the concept of safety in control systems that we will explore in this thesis, both in theory and application.

Here, safety comes from the idea that there exist real-life constraints that could result in the system being harmed if breached, hence requiring it to limit itself to these constraints. One example of this would be in automobiles, with safe control designs like adaptive cruise control and lane keeping. The first ensures that when the vehicle has a set speed for cruising, it does not crash into the vehicle in front of it, which is slower than the set speed. The latter ensures that the vehicle does not veer into the other lane, hence reducing the possibility of a crash. Both of these can be achieved using Control Barrier Functions (CBF).

CBFs are functions that define safe spaces for the system. They serve as an analogue to the concept of safety, what Control Lyapunov Functions (CLF) are to the concept of stability in systems. The concept of CBF is founded on barrier certificates, which were first explored in Nagumo's Theorem. CBF mimic the constraints that we introduced earlier and defines a safe space for the system. If the system starts in the safe space, a safe control design then motivates the system to move away from the CBF, towards the interior of the safe space. This, as a result, ensures that the system does not breach the constraints, for example, ensuring that the vehicle does not crash due to cruise control or lane switching.

In Section 2, we will be forming the basis for approaching this topic, with basic definitions and a brief exploration of CLF. In Section 3, we will explore the concept of safety, Nagumo's Theorem, CBF, and the application of the barriers on general control systems. Based on these concepts, we will run simulations with various conditions imposed on the ACC and LK systems, in Section 4. Section 5 will end the thesis with a review of the points covered in the thesis and possible further topics of interest generated from this thesis. Finally, there will be the Acknowledgements, and the MATLAB codes in the Appendix.

## 2 PRELIMINARIES

Throughout the thesis, we will be dealing with systems of the following form:

$$\dot{x} = f(x) + g(x)u. \tag{1}$$

Here, $x \in \mathcal{X} \subseteq \mathbb{R}^n$ is the state, and $u \in \mathcal{U} \subseteq \mathbb{R}^m$ is the input. Furthermore, $f : \mathbb{R}^n \to \mathbb{R}^n$, and $g : \mathbb{R}^n \to \mathbb{R}^{n \times m}$ are locally Lipschitz continuous functions.

We will review certain basic definitions, followed by Control Lyapunov Functions (CLF), which should give us a basis when approaching Control Barrier Functions (CBF).

### 2.1 *Basic Definitions*

Firstly, we will look at a class of functions known as the extended class $\mathcal{K}$, which we will be using for both the CLF and CBF.

**Definition 2.1.** *A continuous function $\alpha : [0, a) \to [0, \infty)$, with $a > 0$ belongs to class $\mathcal{K}$ if it is strictly increasing and $\alpha(0) = 0$. Moreover, if $\alpha : [-b, a) \to [-\infty, \infty)$, $a > 0$, and $b > 0$ satisfies the same conditions, then it belongs to the extended class $\mathcal{K}$.*

An easy example for this definition would be $\alpha(x) = \gamma x$, where $\gamma > 0$. We can see a visual of it in Figure 1, where $\gamma = 0.5$. In the figure, we see that the function is strictly increasing and that $\alpha(0) = 0$.

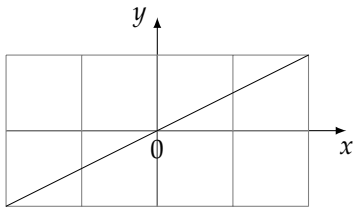


Figure 1: A function of Extended Class $\mathcal{K}$, given as $\alpha(x) = 0.5x$

We can now introduce the concept of forward invariance for a system. This concept is extremely important to the definition of a safe space.

**Definition 2.2.** *A set $\mathcal{C} \subseteq \mathcal{X}$ is forward invariant for the system of the form $\dot{x} = f(x)$, if its solution starting at any $x(0) \in \mathcal{C}$ satisfy $x(t) \in \mathcal{C}$, $\forall t \geq 0$*

This would imply that any solution starting in the forward invariant set would remain within

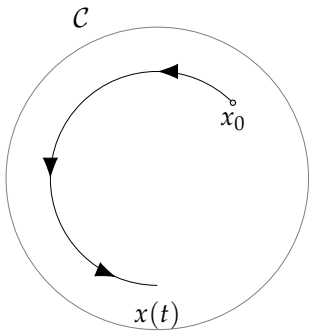


Figure 2: A Forward invariant set $\mathcal{C}$, where $x(t) \in \mathcal{C}$, $\forall t > 0$, if $x(0) \in \mathcal{C}$

the set itself [5]. A simple example of this has been visualised in Figure 2. The figure shows a very simplistic set defined as a sphere, with $x(t)$ travelling within it.

We need to define the Lie derivative, which gives us a derivative of a function with respect to the vector field. This can later allow us to derive a function along the system (1).

**Definition 2.3.** *Let $\mathcal{D} \subseteq \mathbb{R}^n$, then let $h : \mathcal{D} \to \mathbb{R}$ be a smooth function, and $f : \mathcal{D} \to \mathbb{R}^n$ be a smooth vector field on $\mathcal{D}$. The Lie derivative of $h$ with respect to $f$ is defined as follows:*

$$L_f h(x) = \frac{\partial h}{\partial x}(x) f(x),$$

## 2.2 *Control Lyapunov Function*

In the introduction, we stated that CBFs are comparable to CLF. Hence, we shall first explore stability and the purpose of CLF. Stabilisation aims to force the system to an equilibrium. One example of this would be the cruise control, where the system is required to maintain a certain speed.

A stabilization problem is defined as attempting to stabilize the system, by finding a $k : \mathbb{R}^n \to \mathbb{R}^m$, such that when replacing it in (1) as $u = k(x)$, we have that the origin of the closed loop system, of the form:

$$\dot{x} = f(x) + g(x)k(x), \tag{2}$$

is asymptotically stable.

Given the closed-loop system (2), we can employ the following theorem to prove that the feedback $k(x)$ solves the stabilisation problem.

**Theorem 2.4.** *If there exists a function $V : \mathbb{R}^n \to [0, \infty)$, such that the following hold:*

1. *$V(0) = 0$, and $V(x) > 0$, $\forall x \neq 0$ (positive definite)*
2. *there exists a class $\mathcal{K}$ function $\Gamma$ such that:*

$$L_f V(x) + L_g V(x)k(x) + \Gamma(V(x)) \leq 0, \quad \forall x \in \mathbb{R}^n \tag{3}$$

*Then the feedback origin of (2) is asymptotically stable.*

Theorem 2.4 relies on the fact that we have already chosen an appropriate $k(x)$. The following definition will allow us to bypass such a requirement.

**Definition 2.5.** *A continuously differentiable function $V : \mathbb{R}^n \to \mathbb{R}$ is a CLF for the system (1) if it is positive definite and the following holds:*

$$\inf_{u \in \mathcal{U}} [L_f V(x) + L_g V(x)u] \leq -\Gamma(V(x)),$$

It also allows us to construct the following set, which contains all $u$ that stabilise the system for point $x \in \mathcal{X}$:

$$K_{\text{clf}}(x) = \{u \in \mathcal{U} : L_f V(x) + L_g V(x)u + \Gamma(V(x)) \leq 0\}, \tag{4}$$

Then Theorem 2.4 and Definition 2.5 allow us to state the following:

**Theorem 2.6.** *For the nonlinear control system (1), if there exists a CLF $V : \mathcal{X} \to \mathbb{R}_{\geq 0}$, then any Lipschitz continuous feedback controller $k(x) \in K_{clf}(x)$ solves the stabilization problem.*

## 3 SAFETY

In the previous section, we looked at definitions and Control Lyapunov Functions (CLFs). Given this background, we can now begin to look towards safety. We saw that when attempting to stabilise a system, we try to drive it to a certain point, for example, in the case of CLFs to $V(x^*) = 0$. In the case of safety, we wish to keep the system within a certain region. We can define the safe set $\mathcal{C}$ as the superlevel set of a continuously differentiable function $h : \mathcal{X} \to \mathbb{R}$, that is

$$\mathcal{C} = \{x \in \mathcal{X} : h(x) \geq 0\}. \tag{5}$$

Given the definition of $\mathcal{C}$ as above, we can see that the the boundary $\partial\mathcal{C}$, and the interior $\mathbf{Int}(\mathcal{C})$ are as follows:

$$\begin{aligned} \partial\mathcal{C} &= \{x \in \mathcal{X} : h(x) = 0\}, \\ \mathbf{Int}(\mathcal{C}) &= \{x \in \mathcal{X} : h(x) > 0\}. \end{aligned} \tag{6}$$

Based on Definition 2.2, we can formalise the concept of safety as follows:

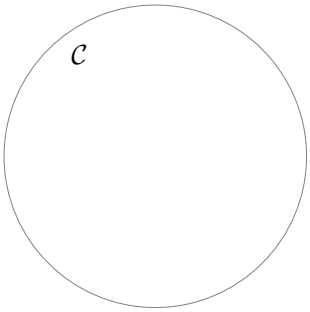


Figure 3: Safe set defined by $h(x, y) = 2 - \sqrt{x^2 + y^2}$

**Definition 3.1.** *If $\mathcal{C}$ is forward invariant with respect to the system of the form $\dot{x} = f(x)$, then the system is safe with respect to $\mathcal{C}$*

Having shifted the goal from a vague idea of the system being safe to requiring the set to be forward invariant will be very useful when going forward.

### 3.1 *Nagumo's Theorem*

Nagumo's Theorem allows us to define necessary and sufficient conditions for ensuring that a trajectory remains within a certain set. We will be using the Brezis version [2] of the proof.

Let $\mathcal{X} \subseteq \mathbb{R}^n$ be open, and $\mathcal{C} \subseteq \mathcal{X}$ is relatively closed in $\mathcal{X}$, which means there exists a closed set $\mathbf{C} \subseteq \mathbb{R}^n$, such that $\mathbf{C} \cap \mathcal{X} = \mathcal{C}$. Let $d(x, \mathcal{C}) = \min_{y \in \mathcal{C}} \|x - y\|$, which defines the distance between a point and the set. Finally, for $x_0 \in \mathcal{X}$, let $x(t)$ be the solution to system as follows:

$$\dot{x}(t) = f(x(t)), \quad x(0) = x_0. \tag{7}$$

**Theorem 3.2.** *The following are equivalent:*

1. *if $x(0) \in \mathcal{C}$ then $x(t) \in \mathcal{C}$, $\forall t \geq 0$.*
2. *$\lim_{h \to 0} \frac{d(x + hf(x), \mathcal{C})}{h} = 0$, $\forall x \in \mathcal{C}$.*

*Proof.* ($i \implies ii$) Set $x \in \mathcal{C}$, for a small $h$, we have that:

$$d(x + hf(x), \mathcal{C}) \leq \|x(h) - x - hf(x)\| = |h| \left\| \frac{x(h) - x}{h} - f(x) \right\|.$$

We then have the following:

$$\begin{aligned}\lim_{h\to 0}\frac{d(x+hf(x),\mathcal{C})}{h} &\leq \lim_{h\to 0}\frac{|h|\left\|\frac{x(h)-x}{h}-f(x)\right\|}{h}\\ &\leq \lim_{h\to 0}\left\|\frac{x(h)-x}{h}-f(x)\right\|\\ &\leq 0\end{aligned}$$

Hence, we can see that (i) is satisfied if (ii) holds.

($ii \implies i$) To prove this, we just need to show $x(t) \in \mathcal{C}$ for small $t$. Let $x = x(0) \in \mathcal{C}$, and then for an open ball $B(x,r)$, given that $f(x)$ is a locally Lipschitz function, we have the following:

$$\|f(x_1) - f(x_2)\| \leq K\|x_1 - x_2\|.$$

We can then use Grönwall's inequality to get the following:

$$\|x_1(t) - x_2(t)\| \leq e^{Kt}\|x_1 - x_2\|.$$

We can assume that $\|x(t) - x\| \leq r/2$. Let $\varphi(t) = d(x(t),\mathcal{C})$, and note that $\varphi(0) = 0$, as $x(0) \in \mathcal{C}$. Since $\mathcal{C}$ is relatively closed, we have that $d(x(t),\mathcal{C}) = \|x(t) - y_t\|$ for some $y_t \in \mathcal{C}$. As a result, we have that $\|y_t - x\| \leq r$ and for small $h$, $\|y_t(h) - x\| \leq r$. Taking all the above, we get the following:

$$\begin{aligned}\varphi(t+h) &= d(x(t+h),\mathcal{C})\\ &= \inf_{y\in\mathcal{C}}\|x(t+h) - y\|\\ &\leq \inf_{y\in\mathcal{C}}\{\|x(t+h) - y_t(h)\| + \|y_t(h) - y_t - hf(y_t)\| + \|y_t + hf(y_t) - y\|\}\\ &= \|x(t+h) - y_t(h)\| + \|y_t(h) - y_t - hf(y_t)\| + \inf_{y\in\mathcal{C}}\|y_t + hf(y_t) - y\|\\ &= \|x(t+h) - y_t(h)\| + \|y_t(h) - y_t - hf(y_t)\| + d(y_t + hf(y_t),\mathcal{C}).\end{aligned}$$

In the above calculations, we added 0 inside the norm, and then used the triangle inequality to get norms that are easier to work with. For the first norm, we can utilise the second inequality that we established earlier in the proof, to get the following:

$$\begin{aligned}\varphi(t+h) &\leq e^{Kh}\|y_t - x(t)\| + \|y_t(h) - y_t - hf(y_t)\| + d(y_t + hf(y_t),\mathcal{C})\\ &\leq e^{Kh}\varphi(t) + \|y_t(h) - y_t - hf(y_t)\| + d(y_t + hf(y_t),\mathcal{C}).\end{aligned}$$

At this point, we add $\varphi(t)$ on both sides of the inequality, and then divide them by $h$ to get the following:

$$\begin{aligned}\varphi(t+h) - \varphi(t) &\leq (e^{Kh} - 1)\varphi(t) + \|y_t(h) - y_t - hf(y_t)\| + d(y_t + hf(y_t),\mathcal{C}),\\ \frac{\varphi(t+h) - \varphi(t)}{h} &\leq \frac{e^{Kh} - 1}{h}\varphi(t) + \left\|\frac{y_t(h) - y_t}{h} - f(y_t)\right\| + \frac{1}{h}d(y_t + hf(y_t),\mathcal{C}).\end{aligned}$$

Firstly, given ii, we know that:

$$\limsup_{h\to 0}\frac{d(y_t + hf(y_t),\mathcal{C})}{h} = 0.$$

We also know that:

$$\limsup_{h\to 0}\left\|\frac{y_t(h) - y_t}{h} - f(y_t)\right\| = 0.$$

Finally, we have the following:

$$\limsup_{h\to 0} \frac{e^{Kh}-1}{h} = K.$$

As a result of these three, we have the following:

$$\limsup_{h\to 0} \frac{\varphi(t+h)-\varphi(t)}{h} \leq K\varphi(t).$$

Finally, using Grönwall's inequality, we get that

$$\varphi(t) \leq e^{Kt}\varphi(0).$$

As a result, we can see that $\varphi(t) = 0$, hence implying $\varphi(t) \in \mathcal{C}$ for small $t$. □

For $x \in \mathbf{Int}(\mathcal{C})$, we can see that for any choice of $f(x)$, we have that $\lim_{h\to 0} \frac{d(x+hf(x),\mathcal{C})}{h} = 0$. Hence, we are only interested in how $f(x)$ acts for $x \in \partial\mathcal{C}$. It is required that $f(x)$ must point inward or be tangent to $\mathcal{C}$, hence Nagumo's Theorem is referred to as the sub-tangentiality condition [1]. An example can be seen when Nagumo's Theorem is applied to a safe set in Figure 4. We can also characterise what happens at the border as follows:

$$L_f h(x) \geq 0, \quad \forall x \in \partial\mathcal{C}.$$

This is often called the barrier certificate [5].

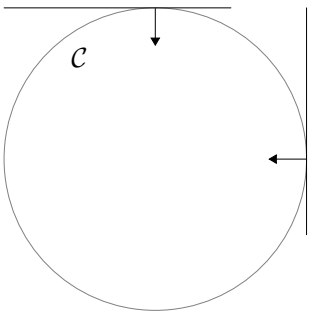


Figure 4: Nagumo's Theorem applied to a safe set defined by $h(x,y) = 2 - \sqrt{x^2+y^2}$

### 3.2 *Control Barrier Function*

Before we discuss Nagumo's theorem, to showcase how control barrier functions (CBFs) can be utilised to design controllers that ensure safety, we need to define what CBFs are.

**Definition 3.3.** *Let $\mathcal{C} \subseteq \mathcal{X}$ be a superlevel set of a continuously differentiable function $h : \mathcal{X} \to \mathbb{R}$, then $h$ is a control barrier function (CBF) if there exists an extended class $\mathcal{K}$ function $A$ such that for the system (1), the following holds true:*

$$\sup_{u\in\mathcal{U}} [L_f h(x) + L_g h(x)u] \geq -A(h(x)), \quad \forall x \in \mathcal{X}.$$

Here, we can see some similarity in how CBFs and CLFs are defined. We can utilise the above definition to define a set that contains all inputs that make $\mathcal{C}$ safe:

$$K_{\text{cbf}}(x) = \{u \in \mathcal{U} : L_f h(x) + L_g h(x)u + \alpha(h(x)) \geq 0\}. \tag{8}$$

Now, we can use Nagumo's Theorem and the definition of CBFs to get the following theorem.

**Theorem 3.4.** *Let $\mathcal{C}$ be defined for a continuously differentiable function $h$. If $h$ is a CBF on $\mathcal{X}$, then any locally Lipschitz continuous controller $k(x) \in K_{cbf}(x)$ for system (1) renders $\mathcal{C}$ forward invariant.*

*Proof.* For $k(x) \in K_{\text{cbf}}(x)$, we have the following:

$$L_f h(x) + L_g h(x) k(x) + \alpha(h(x)) \geq 0.$$

Here, given that we wish to ensure that the trajectory remains within the set, we only consider $x \in \partial\mathcal{C}$. Since we know that $h(x) = 0$ and $A(0) = 0$, we can see that the above inequality becomes the following:

$$L_f h(x) + L_g h(x) k(x) \geq 0.$$

From here, using Nagumo's Theorem, we know that $\mathcal{C}$ will be forward invariant, hence safe for this system. □

We can extend the idea of this to barrier functions that change over time using the following definition [3].

**Definition 3.5.** *Let $\mathcal{C}(t) \subseteq \mathcal{X}$ be a superlevel set of a continuously differentiable function $h : (\mathcal{X}, t) \to \mathbb{R}$, then $h$ is a control barrier function (CBF) if there exists an extended class $\mathcal{K}$ function $A$ such that for the system (1), the following holds true:*

$$\sup_{u \in \mathcal{U}} [L_f h(x,t) + L_g h(x,t) u + \frac{\partial h(x,t)}{\partial t}] \geq -A(h(x,t)), \quad \forall x \in \mathcal{X}.$$

We can use the above definition to define a set that contains all inputs that make $\mathcal{C}$ safe:

$$K_{\text{ctbf}}(x,t) = \{u \in \mathcal{U} : L_f h(x,t) + L_g h(x,t) u + \frac{\partial h(x,t)}{\partial t} + \alpha(h(x,t)) \geq 0\}. \tag{9}$$

With this set, we can get the following theorem:

**Theorem 3.6.** *Let $\mathcal{C}(t)$ be defined for a continuously differentiable function $h$. If $h$ is a CBF on $\mathcal{X}$, then any locally Lipschitz continuous controller $k(x,t) \in K_{ctbf}(x,t)$ for system (1) renders $\mathcal{C}(t)$ forward invariant.*

### 3.3 *Application*

During the following simulations, we will employ the unification of both CLF and CBF conditions under a Quadratic Programme (QP) based controller. The idea behind QP based controller is to optimise a quadratic multivariate function, with linear constraints applied to the variables.

For CLF and CBF, we can utilise the conditions defined in $K_{clf}$ and $K_{cbf}$ (or in the case of a time varying barrier, $K_{ctbf}$) as the constraints. However, since we wish to ensure safety over stability, we need to add a relaxation variable ($\delta > 0$), such that the QP is solvable if the two conditions conflict. An example of a CLF-CBF-QP-based controller is as follows:

$$\begin{aligned} u(\mathbf{x}) = & \quad \underset{\mathbf{u}=[u,\delta]^\top \in \mathbf{R}^{m+1}}{\arg\min} \quad \mathbf{u}^\top \mathbf{H}(\mathbf{x}) \mathbf{u}, \\ s.t. \quad & L_f V(\mathbf{x}) + L_g V(\mathbf{x}) u \leq -\gamma V(\mathbf{x}) + \delta, \\ & L_f(\mathbf{x}) h(\mathbf{x}) + L_g h(\mathbf{x}) u \geq -\alpha h(\mathbf{x}). \end{aligned} \tag{10}$$

Here we know that $\Gamma(x) = \gamma x$, and $A(x) = \alpha x$ are both extended class $\mathcal{K}$ functions, if the constants satisfy $\gamma > 0$ and $\alpha > 0$ respectively. We also have the following:

$$\mathbf{H}(\mathbf{x}) = \begin{bmatrix} H(\mathbf{x}) & 0 \\ 0 & p \end{bmatrix}.$$

Here, $H(\mathbf{x})$ is an appropriate positive definite matrix depending on the system. We can also determine how strongly we want any CLF to constrain the inputs by determining the penalty $p > 0$. A lower $p$ would give the programme more freedom when choosing the controller. If

there are multiple CLFs, the chosen penalty for each can assist in determining the respective importance.

In the case of time-varying barriers, we have the following QP:

$$\begin{aligned} u(\mathbf{x},t) = & \underset{\mathbf{u}=[u,\delta]^\top \in \mathbf{R}^{m+1}}{\arg\min} \ \mathbf{u}^\top \mathbf{H}(\mathbf{x})\mathbf{u}, \\ s.t. \quad & L_f V(\mathbf{x}) + L_g V(\mathbf{x}) u \leq -\gamma V(\mathbf{x}) + \delta, \\ & L_f(\mathbf{x})h(\mathbf{x},t) + L_g h(\mathbf{x},t)u + \frac{\partial h(\mathbf{x},t)}{\partial t} \geq -\alpha h(\mathbf{x},t). \end{aligned} \tag{11}$$

Given the above, we can see that we could apply multiple barriers to the system. When unifying the barriers, we will have two different conditions. The first condition is that at all times this new barrier function is less than or equal to the minimum of all barriers in question, as this would ensure that when this new barrier function is positive so are all the barrier functions in question. We need to also ensure that the new barrier function is differentiable. Hence, we can use the following lemma:

**Lemma 3.7.** *Let there be $p$ barrier functions $h_i(x,t)$. Then the following holds:*

$$-\ln\left(\sum_{i=1}^{p} e^{-h_i(x,t)}\right) \leq \min_{i\in\{1,\dots,p\}} h_i(x,t).$$

*Proof.* We have the following:

$$\max_{i\in\{1,\dots,p\}} \zeta_i \leq \ln\left(\sum_{i=1}^{p} e^{\zeta_i}\right), \qquad \zeta_1,\dots,\zeta_p \in \mathbb{R}$$

Let $\zeta_i = -h_i(x,t)$, then we have the following:

$$\max_{i\in\{1,\dots,p\}} -h_i(x,t) \leq \ln\left(\sum_{i=1}^{p} e^{-h_i(x,t)}\right).$$

Then given that $\max_i -c_i = -\min_i c_i$, we have the following:

$$-\min_{i\in\{1,\dots,p\}} h_i(x,t) \leq \ln\left(\sum_{i=1}^{p} e^{-h_i(x,t)}\right).$$

Finally, we can multiply both sides by $-1$ to get the following inequality:

$$\min_{i\in\{1,\dots,p\}} h_i(x,t) \geq -\ln\left(\sum_{i=1}^{p} e^{-h_i(x,t)}\right).$$

□

As a result of the lemma, we can use the following barrier:

$$h(x,t) = -\ln\left(\sum_{i=1}^{p} e^{-h_i(x,t)}\right).$$

To show that this can represent all barriers, we have the reasoning below. Firstly,

$$0 \leq -\ln\left(\sum_{i=1}^{p} e^{-h_i(x,t)}\right),$$

implies

$$0 \leq \min_{i\in\{1,\dots,p\}} h_i(x,t).$$

Hence, we know that the following relationship is true:

$$0 \leq -\ln\left(\sum_{i=1}^{p} e^{-h_i(x,t)}\right) \Rightarrow 0 \leq h_i(x,t), \quad \forall i \in \{1,...,p\}$$

## 4 SIMULATIONS

Having studied Control Barrier Functions (CBFs) in the last section from a theoretical point of view, we will apply them in the examples below. Here, we will attempt to understand how the two conditions of CLF and CBF interact when attempting to find the controller at each time step. We will be going through the construction of the models; hence, we will also be able to see issues that might arise when simulating the models in MATLAB, with the assistance of YALMIP[4]. The complete MATLAB code will be in the appendix.

During the following section, we will refer to conditions defined by CLFs and CBFs as soft conditions and barriers, respectively. This is because conditions defined by CBFs are more definite, which can be seen by the lack of a relaxation variable, in comparison to the conditions defined by CLFs. Firstly, we will focus on a simple example with one soft condition and one barrier. The second example will be more thorough with 3 soft conditions and 2 barriers. We will expand on the basis of this for the next models to test how the unification of two barriers affects the system.

### 4.1 *ACC*

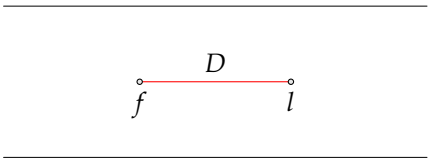


Figure 5: Basic ACC Problem

For a simple adaptive cruise control (ACC) model, we will only consider two conditions, while in the real world, there would be more factors to consider. One is the desired velocity of the vehicle, and the second is the distance from the vehicle in front of it. We can showcase the dynamics between the two vehicles, a simplification of the system in [6], as follows:

$$\mathbf{x} = \begin{bmatrix} v_f \\ v_l \\ D \end{bmatrix}. \tag{12}$$

Here $v_f$ and $v_l$ are the velocities of the follower and leader, with $D$ representing the current distance. The dynamics are given by:

$$\dot{\mathbf{x}} = \begin{bmatrix} -\frac{F_r}{m} \\ a_l \\ v_l - v_f \end{bmatrix} + \begin{bmatrix} \frac{1}{m} \\ 0 \\ 0 \end{bmatrix} u. \tag{13}$$

We will only be using an acceleration $a_l$ of our choosing on the leading vehicle, to ensure that we know the expected velocity of the leading vehicle. Hence, ensuring that the influence of the CBF on the system is clear. The acceleration of the follower is determined by $m$, and $F_r(v_f) = c_0 + c_1 v_f + c_2 v_f^2$, the mass and aerodynamic force acting against the vehicle, with the constants being defined empirically in [6]. We see that the input only affects the change of velocity of the follower and is divided by the mass of the car.

In this model, we will be attempting to fulfil two goals for the follower. The first goal would be to achive a desired speed, which can be formulated as a soft condition. The second goal would be to maintain enough distance from the leading vehicle, which will be formulated by a barrier.

Having defined the system, we can now begin to establish the conditions. Firstly, we can define the desired velocity in the Control Lyapunov function. Letting $v_d$ be defined as the desired velocity, we can construct a Lyapunov function as follows:

$$V_v(\mathbf{x}) = (v_f - v_d)^2. \tag{14}$$

The above Lyapunov function enforces a soft condition on the velocity of the following vehicle.

We can now define the barrier function; we have to consider both the minimum distance ($D_0$) as well as the minimum time-headway ($\tau$) we wish to maintain. As a result, we have the following barrier function:

$$h_{acc}(\mathbf{x}) = D - \tau v_f - D_0. \tag{15}$$

Given the two conditions, after unifying the two conditions, we have the following:

$$\begin{aligned} u(\mathbf{x}) = \quad & \underset{\mathbf{u}=[u,\delta]^\top \in \mathbb{R}^2}{\arg\min} \ \mathbf{u}^\top \mathbf{H}(\mathbf{x})\mathbf{u}, \\ s.t. \quad & L_f V_v(\mathbf{x}) + L_g V_v(\mathbf{x}) u \leq -\gamma(V_v(\mathbf{x})) + \delta, \\ & L_f(\mathbf{x}) h_{acc}(\mathbf{x}) + L_g h_{acc} u \geq -\alpha(h_{acc}(\mathbf{x})). \end{aligned} \tag{16}$$

Here $\mathbf{H}(\mathbf{x})$ is

$$\mathbf{H}(\mathbf{x}) = \begin{bmatrix} \frac{1}{2m^2} & 0 \\ 0 & p \end{bmatrix}.$$

Here we use $\frac{1}{2m^2}$, to account for the fact that the input is divided by the mass in the system.

We will be simulating the change of the state over time under three different conditions, by changing the initial velocity ($v_l$) and the acceleration over time $a_l(t)$ of the leading vehicle. In Table 1, we can see the values that remain constant throughout the three simulations of the model, while in Table 2, we can see the three initial $v_l$ and $a_l(t)$ for the three cases.

In the first case, we will see what happens when only one of the two conditions can be fulfilled. The second case represents an ideal condition where both can be fulfilled. The final case will showcase a barrier affected by change over time. These three simulations will allow us to see the interaction between the soft conditions defined in the CLF and the barriers defined in the CBF.

| Constant | Value | Constant | Value |
|---|---|---|---|
| $m$ | 1650 kg | $v_d$ | 3 m/s |
| $c_0$ | 51 N | $\tau$ | 0.1 s |
| $c_1$ | 1.26 Ns/m | $D_0$ | 1 m |
| $c_2$ | 0.4342 $Ns^2/m^2$ | Initial $D$ | 0.5 m |
| $p$ | 100 | Final Time | 100 s |

Table 1: Constants for simple ACC Model

| Case | Initial $v_l$ | $a_l(t)$ |
|---|---|---|
| 1 | 2.5 | 0 |
| 2 | 3.5 | 0 |
| 3 | 1.5 | $\sin(t)$ |

Table 2: Conditions on Velocity and Acceleration of the leading vehicle

For each of the simulations, we will be plotting the velocity of the two vehicles. The orange solid line belongs to the leading vehicle, and the dashed blue line belongs to the follower. We will also plot the distance between the two vehicles. We do so because we have conditions on the velocity of the follower and the distance between the leader and follower through the control and barrier functions.

In Figure 6, we see the velocity of the follower vehicle (Blue Dash line) increases initially to the desired velocity before the barrier causes it to slow down to maintain a minimum distance. This is further clarified in Figure 7, where we see the distance increase, while the velocity of the follower is less than the velocity of the leader. At around $t = 5$, we start to see a decrease in

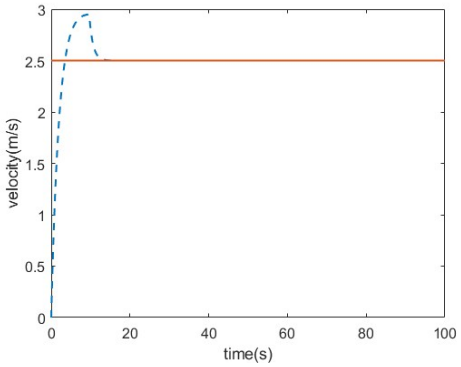


Figure 6: Velocity of the two vehicles (Case 1)

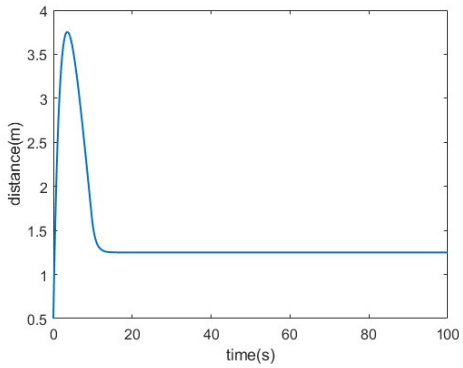


Figure 7: Distance between the two vehicles (Case 1)

distance, which corresponds with when the velocity of the follower is more than the velocity of the leader. Then, at around $t = 10$, we see that the distance has reached a plateau, which is maintained by forcing the velocity of the two to be the same. This is done even though the desired velocity of the follower is higher.

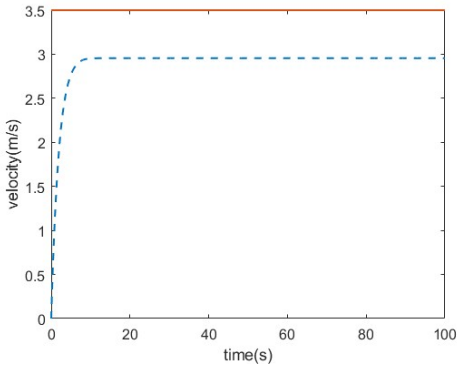


Figure 8: Velocity of the two vehicles (Case 2)

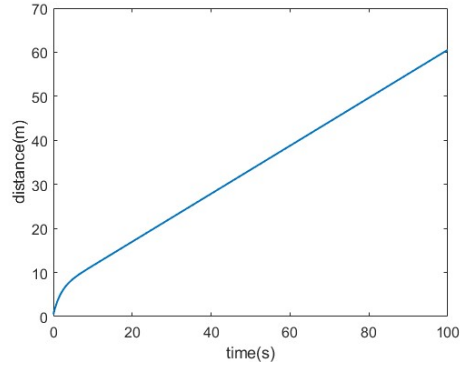


Figure 9: Distance between the two vehicles (Case 2)

Figure 8 shows that the follower nearly achieves the desired velocity, which is less than the velocity of the leading vehicle. In Figure 9, we see that as a result, the distance keeps increasing, hence always satisfying the barrier. Here, we can also see the importance of choosing the correct weight to use when penalising the relaxation variable, as the desired velocity was 3 m/s, yet it is not achieved when the velocity plateaus.

In Figure 10, we see that dynamic acceleration of the leading vehicle leads to a more dynamic relationship between the soft condition and the barrier, unlike the previous two cases. The figure also shows us a clearer visual of the differences between the CLF and CBF. This can be seen by the rise to the desired velocity being more gradual in comparison to the reduction of velocity required by the barrier, with the difference in duration being almost 2 seconds. We can see the effect of this on the distance between the two vehicles in Figure 11.

### 4.2 *Adaptive Cruise Control and Lane Keeping*

For this model, we will be looking at a different system with three soft conditions and two barriers. This particular model is in regard to the movement of a robot around an origin. It is

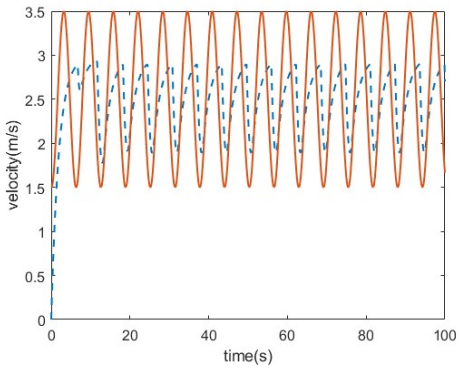


Figure 10: Velocity of the two vehicles (Case 3)

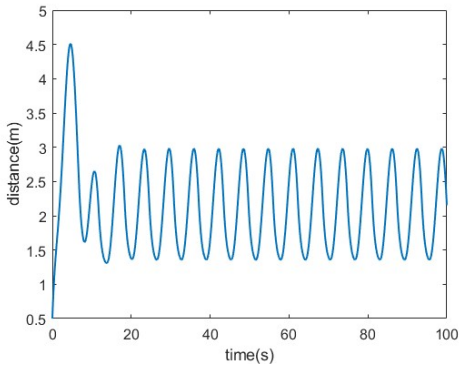


Figure 11: Distance between the two vehicles (Case 3)

based on [7]. Since we will be running two different robots of the same type, the models will be the same. Hence, we only need to define the following system once:

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ v \\ \psi \\ w \end{bmatrix}. \tag{17}$$

Here, $x$ and $y$ refer to the position of the robot in the Cartesian plane. $\psi$ refers to the orientation of the robot. Finally, $v$ and $w$ refer to the longitudinal and angular velocity. The dynamics are given as follows:

$$\dot{\mathbf{x}} = \begin{bmatrix} v\cos(\psi) - aw\sin(\psi) \\ v\sin(\psi) + aw\cos(\psi) \\ -aw^2 \\ w \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \frac{1}{m} & 0 \\ 0 & 0 \\ 0 & \frac{1}{I_z} \end{bmatrix} \begin{bmatrix} u_l \\ u_a \end{bmatrix}. \tag{18}$$

We can see that only $v$ and $w$ are influenced by inputs ($u_l$ and $u_a$), as well as the mass ($m$) and the moment of inertia ($I_z$), respectively. To allow us to more easily connect $x$ and $y$ with the inputs, we choose a point of interest located $a > 0$ in front of the wheel axis [7]. Finally, we can see that the orientation ($\psi$) is only affected by $w$.

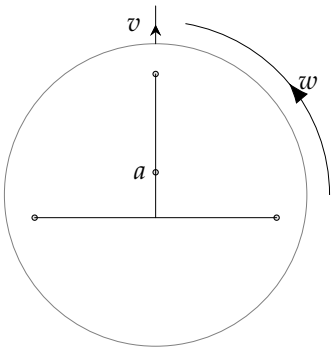


Figure 12: Robot Values

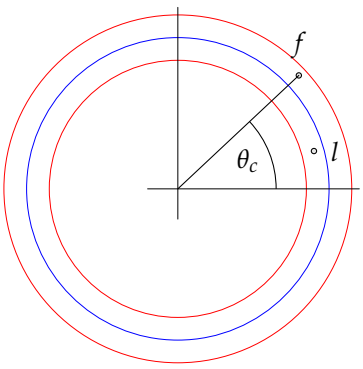


Figure 13: ACC and LK Problem

In Figure 13, we see that the two robots will be on a path, represented by the blue circle. We can define this path as follows:

$$R_p(\theta) = R + b\sin(n\theta). \tag{19}$$

Here $R$ is the average radius, $b$ is the amplitude, and $n$ is the number of periods in the path. One thing to notice is that the path is defined in the polar plane. We can find the corresponding $R_p(\theta)$ with respect to the current position of the robot as follows:

$$\theta_c = \arctan\left(\frac{y}{x}\right). \tag{20}$$

For convenience, we can already give the time derivation of $\theta_c$ given the system as follows:

$$\begin{aligned}\dot{\theta}_c &= \frac{\mathrm{d}}{\mathrm{d}t}\arctan\left(\frac{y}{x}\right)\\ &= \frac{x}{x^2+y^2}\dot{y} - \frac{y}{x^2+y^2}\dot{x}.\end{aligned} \tag{21}$$

Similarly, we can give the time derivative of $R_p(\theta_c)$ as follows:

$$\begin{aligned}\dot{R}_p(\theta_c) &= \frac{\mathrm{d}}{\mathrm{d}t}(R + b\sin(n\theta_c))\\ &= nb\cos(n\theta_c)\dot{\theta}_c.\end{aligned} \tag{22}$$

Besides the two objectives that we have already seen in the previous model, we have two additional conditions on the system. The first is to move along the path. The second is to keep within the lane, an example of which is defined by the red lines in Figure 13.

We can first work towards the first objective. To do so, we need to reduce the distance between the robot and the path and ensure the smoothness of the robot's path, both of which can easily be fulfilled with soft conditions. Firstly, to ensure that the path remains smooth, we require that the angular velocity go towards 0. We can do this using the following Lyapunov function:

$$V_s(\mathbf{x}) = w^2. \tag{23}$$

Following this, we can attempt to use the following definition of error as the Lyapunov function to reduce the distance between the robot and the path as follows:

$$\eta_e(\mathbf{x}) = x^2 + y^2 - R_p(\theta_c)^2. \tag{24}$$

Yet, if we were to take $V_r(\mathbf{x}) = \eta_e(\mathbf{x})^2$ as the Lyapunov function, as we have that $L_g V_r(\mathbf{x}) = 0$. This means that the resulting soft condition is of the form:

$$L_f V_r(\mathbf{x}) \leq -\gamma_r V_r(\mathbf{x}),$$

hence it does not optimise the inputs. To circumvent this, we can construct a relationship between the change of error and the error, to ensure that the error goes to 0, as follows:

$$\dot{\eta}_e(\mathbf{x}) = -\lambda\eta_e(\mathbf{x}). \tag{25}$$

We can first calculate the following derivative.

$$\begin{aligned}\dot{\eta}_e(\mathbf{x}) &= \dot{x}^2 + \dot{y}^2 - R_p(\dot{\theta}_c)^2\\ &= 2x\dot{x} + 2y\dot{y} - 2R_p(\theta_c)\dot{R}_p(\theta_c).\end{aligned} \tag{26}$$

Hence, we can use the following Lyapunov function as the tracker:

$$V_r(\mathbf{x}) = (\dot{\eta}_e(\mathbf{x}) + \lambda\eta_e(\mathbf{x}))^2 = \begin{bmatrix}\eta_e(\mathbf{x}) & \dot{\eta}_e(\mathbf{x})\end{bmatrix}\begin{bmatrix}\lambda^2 & \lambda\\ \lambda & 1\end{bmatrix}\begin{bmatrix}\eta_e(\mathbf{x})\\ \dot{\eta}_e(\mathbf{x})\end{bmatrix}. \tag{27}$$

Finally, we can construct the barrier for the barriers of the path. As a result, we have the basis for fulfilling the first objective.

To fulfil the second objective, we can construct a Lane Keeping (LK) barrier. We can use the following equation as defined in [7]:

$$h_{lk} = D_m - sign(\dot{d}_{lat})d_{lat} - \frac{1}{2}\frac{\dot{d}_{lat}^2}{a_{max}}. \tag{28}$$

Here $a_{max}$ is defined as the maximum acceleration allowed towards the path by the robot. Then $d_{lat}$ is the lateral distance between the robot and the path.

$$d_{lat} = \sqrt{x^2 + y^2} - R_p(\theta_c). \tag{29}$$

Similarly, we can also calculate the change in distance as follows:

$$\dot{d}_{lat} = \frac{x}{\sqrt{x^2+y^2}}\dot{x} + \frac{y}{\sqrt{x^2+y^2}}\dot{y} - \dot{R}_p(\theta_c). \tag{30}$$

Hence, we have a barrier to fulfilling the second objective.

Having established the three new conditions, we can also address the only difference between the two conditions we had set in the last model, which is the calculation of the distance. In the last model, the distance between the two vehicles was part of the state, which is not the case here. As a result, we must now define the distance. In this case, as we have the positions of the two robots in terms of x and y coordinates, we can use the Euclidean distance.

Given the five conditions, we have the following two optimisations, for the leading robot and the follower, respectively:

$$\begin{aligned} u_{lead}(\mathbf{x}_{lead}) = & \underset{\mathbf{u}=[u_l,u_a,\delta_v,\delta_s,\delta_r]^\top \in \mathbb{R}^5}{\arg\min} \quad \mathbf{u}^\top \mathbf{H}(\mathbf{x}_{lead})\mathbf{u} \\ & s.t. \quad L_f V_i(\mathbf{x}_{lead}) + L_g V_i(\mathbf{x}_{lead}) \begin{bmatrix} u_l & u_a \end{bmatrix}^\top \leq -\gamma_i V_i(\mathbf{x}_{lead}) + \delta_i, \quad \forall i \in \{v,s,r\}, \\ & \quad L_f h_{lk}(\mathbf{x}_{lead}) + L_g h_{lk}(\mathbf{x}_{lead}) \begin{bmatrix} u_l & u_a \end{bmatrix}^\top \geq -\alpha_j h_{lk}(\mathbf{x}_{lead}), \end{aligned} \tag{31}$$

$$\begin{aligned} u_{foll}(\mathbf{x}_{foll}) = & \underset{\mathbf{u}=[u_l,u_a,\delta_v,\delta_s,\delta_r]^\top \in \mathbb{R}^5}{\arg\min} \quad \mathbf{u}^\top \mathbf{H}(\mathbf{x}_{foll})\mathbf{u} \\ & s.t. \quad L_f V_i(\mathbf{x}_{foll}) + L_g V_i(\mathbf{x}_{foll}) \begin{bmatrix} u_l & u_a \end{bmatrix}^\top \leq -\gamma_i V_i(\mathbf{x}_{foll}) + \delta_i, \quad \forall i \in \{v,s,r\}, \\ & \quad L_f h_{lk}(\mathbf{x}_{foll}) + L_g h_{lk}(\mathbf{x}_{foll}) \begin{bmatrix} u_l & u_a \end{bmatrix}^\top \geq -\alpha_j h_{lk}(\mathbf{x}_{foll}), \\ & \quad L_f h_{acc}(\mathbf{x}_{foll}, \mathbf{x}_{lead}) + L_g h_{acc}(\mathbf{x}_{foll}, \mathbf{x}_{lead}) \begin{bmatrix} u_l & u_a \end{bmatrix}^\top \geq -\alpha_j h_{acc}(\mathbf{x}_{foll}, \mathbf{x}_{lead}). \end{aligned} \tag{32}$$

Here $H$ is as follows:

$$\mathbf{H}(\mathbf{x}) = \begin{bmatrix} \frac{1}{m^2} & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{I_z^2} & 0 & 0 & 0 \\ 0 & 0 & p_v & 0 & 0 \\ 0 & 0 & 0 & p_s & 0 \\ 0 & 0 & 0 & 0 & p_r \end{bmatrix}. \tag{33}$$

Similar to the previous model, we penalise the inputs with $1/m^2$ and $1/I_z^2$. Then the remaining three soft conditions are penalised using $p_v, p_s, p_r > 0$.

We will be running the model through simulations under three different conditions to showcase the interaction between the two different barriers. The first case will be done on a simple circular path. The next two cases will be over more triangular paths. The third case will have a higher

| Constant | Value | Constant | Value |
|---|---|---|---|
| $m$ | 0.08 Kg | $a_{max}$ | 1 $m/s^2$ |
| $a$ | 0.03 m | $D_m$ | 0.3 m |
| $r$ | 0.06 m | $\tau$ | 0.1 s |
| $T_{off}$ | 25s | $T_{on}$ | 0.5 |
| $I_z$ | $0.5mr^2$ | $p_v$ | $10^5$ |
| $R$ | 3 m | $p_s$ | 1 |
| $v_l$ | 0.2 m/s | $p_r$ | $10^3$ |
| $\gamma_v, \gamma_s, \gamma_r$ | 2 | $\alpha_{acc}, \alpha_{lk}$ | 1 |

Table 3: Constants for the ACC and LK Model

| Variables | Case 1 | Case 2 | Case 3 |
|---|---|---|---|
| b | 0 | 0.2 | 0.2 |
| n | 0 | 3 | 3 |
| $v_f$ | 0.25 | 0.25 | 0.3 |
| $D_0$ | 0.6 | 0.6 | 0.1 |

Table 4: Variables of the path and follower for the ACC and LK Model

desired velocity for the follower robot and a lowered minimum distance.

In these simulations, we will be seeing four different plots. Similar to the simulation results for the earlier model, we will plot the velocity and the distance between the two robots over time. We will also be plotting the angular velocity to see the effect of the control function on the smoothness of the trajectory. Finally, we will plot the trajectory of the two robots to visualise their movement both when the tracker, as defined earlier, is active and when only the boundary function is active.

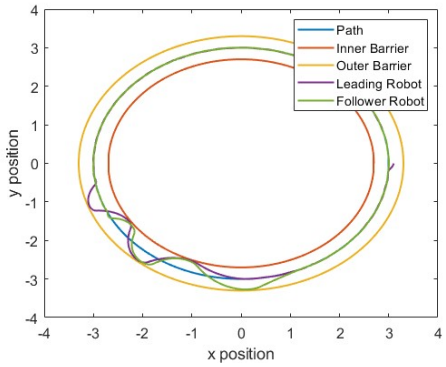


Figure 14: Trajectories of the two vehicles (Case 1)

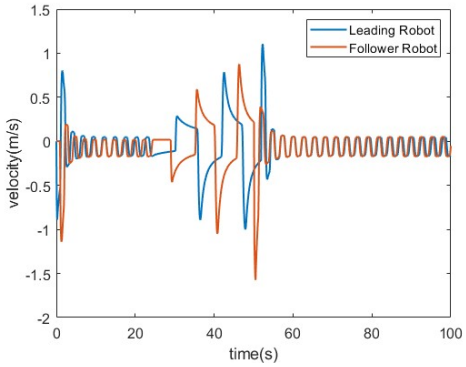


Figure 15: Angular velocity of the two vehicles (Case 1)

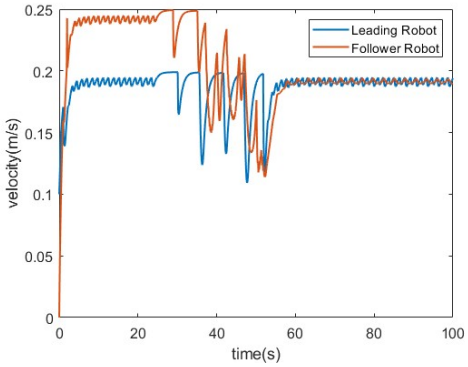


Figure 16: Velocity of the two vehicles (Case 1)

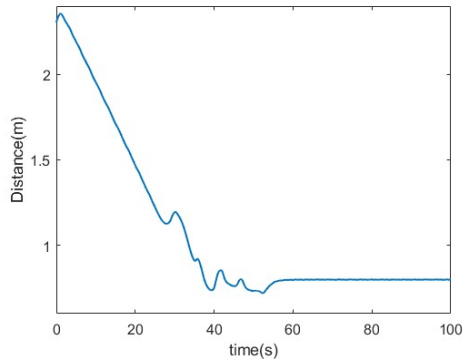


Figure 17: Distance between the two vehicles (Case 1)

Figures for Cases 1 and 2 showcase similar changes over time, hence we will discuss them at the

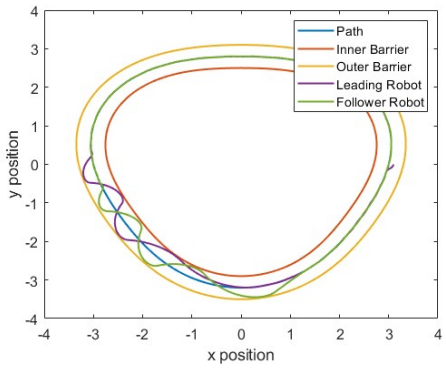


Figure 18: Trajectories of the two vehicles (Case 2)

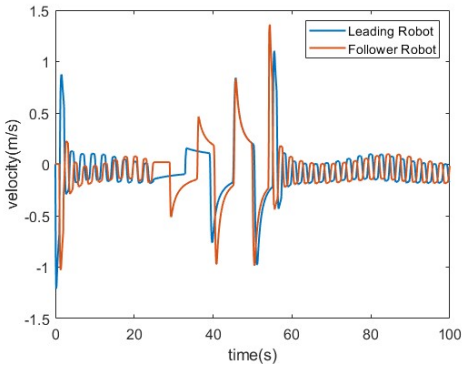


Figure 19: Angular velocity of the two vehicles (Case 2)

same time. In Figures 14 and 18, we see that the robots remain close to the path while the tracker condition is on, and then swerve within the lane when not due to the lane keeping condition. Figures 16 and 20, showcase that the follower robot does slow down to match the velocity of the leading robot. We see a correlation with the stabilisation of the distance between the two, as seen in Figures 17 and 21. Finally, Figures 15, and 19 showcase the effect of all these on the angular velocity. When the tracker is off, we see that the change in angular velocity is great when hitting the lane barriers.

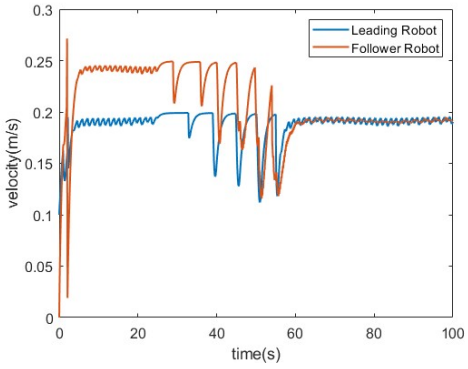


Figure 20: Velocity of the two vehicles (Case 2)

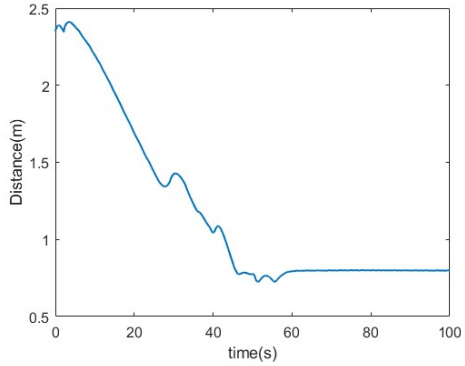


Figure 21: Distance between the two vehicles (Case 2)

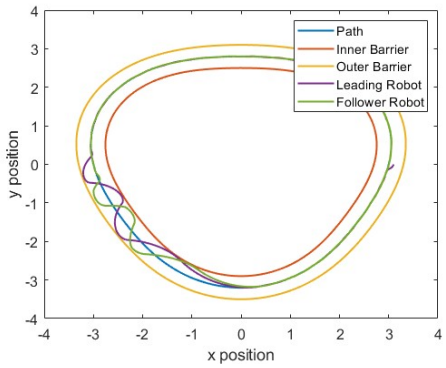


Figure 22: Trajectories of the two vehicles (Case 3)

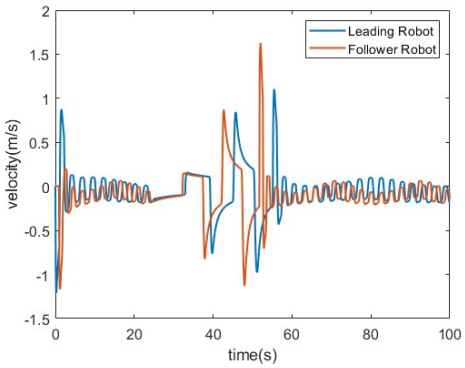


Figure 23: Angular velocity of the two vehicles (Case 3)

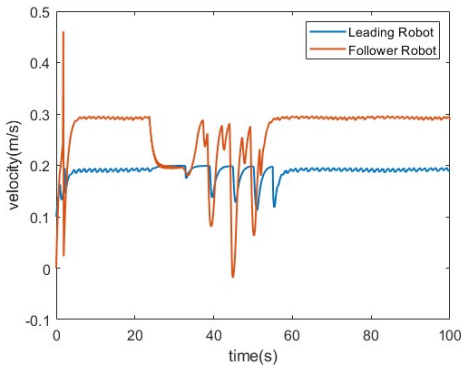


Figure 24: Velocity of the two vehicles (Case 3)

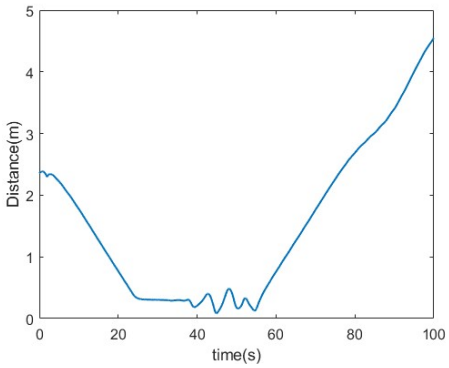


Figure 25: Distance between the two vehicles (Case 3)

For Case 3, Figures 22 and 23 we do not see any significant differences from the previous cases. It is in the longitudinal velocity and distance of case 3, that we see a different result from the previous cases. Figure 24 shows that the follower robot slows down for a small period of time before increasing again. Similarly, the minimum distance is maintained for that duration before increasing. Both of these suggest that the follower robot slipped past the lead, at which point the ACC turned off, hence allowing the follower robot to reach its own desired velocity. This is an understandable result as the minimum distance was set to 0.1 m, which is the width of the lane is 0.6 m, as the maximum distance from the path on either side is 0.3 m. This results in the barrier conditions on the follower robot not conflicting, despite not adhering to the idea behind the two barriers.

For the following model simulations, we will be using the conditions set in Case 2 as the basis.

### 4.3 *Fusing Barrier Functions*

While the quadratic programming allows us to easily implement two or more barriers into a optimization problem, it raises the question whether unifying the barrier conditions may be better. Hence, our objective in this model would be to see what would occur were we to combine multiple barriers. We can then construct a case where using the lemma 3.7, we can combine the barrier functions for the follower robot. We can improve the approximation using the following:

$$\begin{aligned} -\frac{1}{\beta}\ln\left(\sum_{i=1}^{p} e^{-\beta h_i(x,t)}\right) &\leq \frac{1}{\beta}\min_{i\in\{1,\dots,p\},} \beta h_i(x,t), \\ -\frac{1}{\beta}\ln\left(\sum_{i=1}^{p} e^{-\beta h_i(x,t)}\right) &\leq \min_{i\in\{1,\dots,p\},} h_i(x,t). \end{aligned} \tag{34}$$

Hence, given that we have two barriers, LK and ACC, our barrier function becomes the following:

$$h_{tot} = -\frac{1}{\beta}\ln\left(e^{-\beta h_{lk}} + e^{-\beta h_{acc}}\right). \tag{35}$$

We will also be removing the tracking control from both robots, to be able see the effect of the combined barrier function. Hence, the optimisation problem for the follower vehicle becomes the following:

$$\begin{aligned} u_{foll}(\mathbf{x}_{foll}) = & \underset{\mathbf{u}=[u_l,u_a,\delta_v,\delta_s,\delta_r]^\top \in \mathbb{R}^5}{\arg\min} \quad \mathbf{u}^\top \mathbf{H}(\mathbf{x}_{foll})\mathbf{u}, \\ s.t. \quad & L_f V_i(\mathbf{x}_{foll}) + L_g V_i(\mathbf{x}_{foll}) \begin{bmatrix} u_l & u_a \end{bmatrix}^\top \leq -\gamma_i V_i(\mathbf{x}_{foll}) + \delta_i, \quad \forall i \in \{v, s\}, \\ & L_f h_{tot}(\mathbf{x}_{foll}, \mathbf{x}_{lead}) + L_g h_{tot}(\mathbf{x}_{foll}, \mathbf{x}_{lead}) \begin{bmatrix} u_l & u_a \end{bmatrix}^\top \geq -\alpha_j h_{tot}(\mathbf{x}_{foll}, \mathbf{x}_{lead}). \end{aligned} \tag{36}$$

For this simulation, we will not be plotting the distance between the leading and follower robots. Instead, we will plot the values of the three barriers.

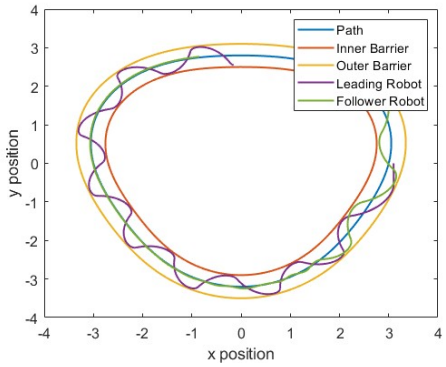


Figure 26: Trajectories of the two vehicles

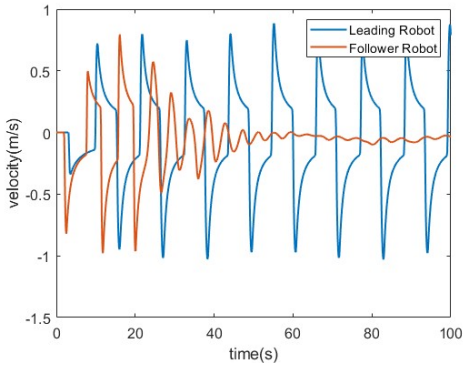


Figure 27: Angular velocity of the two vehicles

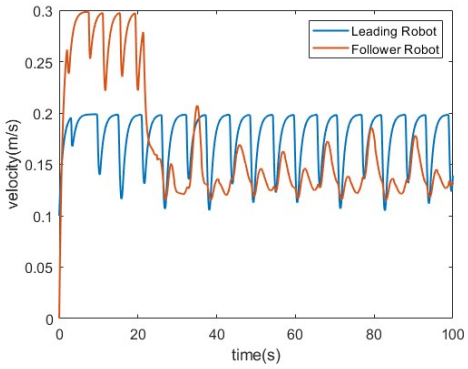


Figure 28: Velocity of the two vehicles

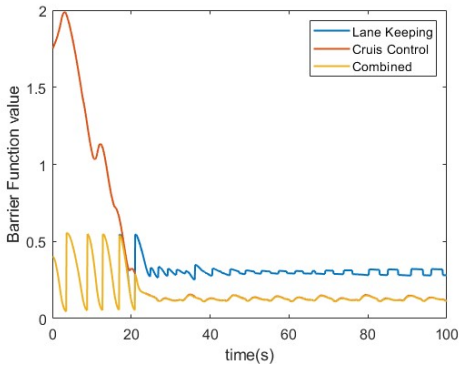


Figure 29: Barrier Function values over time

Using Figure 29, we can see that the barrier function that affects the follower robot switches from the LK barrier to the ACC barrier at around $t = 20$, when the value for the ACC barrier becomes the minimum. However, in Figure 26, we can also see that the unification of the two barriers results in the LK barrier becoming conservative. Hence, it restricts the system to a far greater degree than required.

### 4.4 *Position and Time Varying Barrier Functions*

Given that not all lanes are subject to the same length, we need to ensure that the barriers and conditions of the second model will hold when the lane is subject to change. Hence, we will be defining the maximum distance from the route as:

$$D_m(\theta) = 0.3 - k\sin(\theta). \tag{37}$$

For the time-dependent component of this model, we will be using a very simple barrier function that will construct a time-dependent lower bound for the velocity of the leading robot. We use the following barrier function:

$$h_{vlb} = v_l - 1.75\frac{t}{100}. \tag{38}$$

We do not apply the lower bound to the follower robot, as that would cause it to clash with the barrier enforcing the ACC. Again, we will also be removing the tracking control from both robots. Hence, the optimisation problem for the leading robot becomes the following:

$$\begin{aligned} u_{lead}(\mathbf{x}_{lead}) = & \underset{\mathbf{u}=[u_l,u_a,\delta_v,\delta_s,\delta_r]^\top \in \mathbb{R}^5}{\arg\min} \quad \mathbf{u}^\top \mathbf{H}(\mathbf{x}_{lead})\mathbf{u} \\ s.t. \quad & L_f V_i(\mathbf{x}_{lead}) + L_g V_i(\mathbf{x}_{lead}) \begin{bmatrix} u_l & u_a \end{bmatrix}^\top \leq -\gamma_i V_i(\mathbf{x}_{lead}) + \delta_i, \quad \forall i \in \{v, s\}, \\ & L_f h_{lk}(\mathbf{x}_{lead}) + L_g h_{lk}(\mathbf{x}_{lead}) \begin{bmatrix} u_l & u_a \end{bmatrix}^\top \geq -\alpha_j h_{lk}(\mathbf{x}_{lead}), \\ & L_f h_{vlb}(\mathbf{x}_{lead}, t) + L_g h_{vlb}(\mathbf{x}_{lead}, t) \begin{bmatrix} u_l & u_a \end{bmatrix}^\top + \frac{\partial h_{vlb}(\mathbf{x}_{lead}, t)}{\partial t} \geq -\alpha_j h_{vlb}(\mathbf{x}_{lead}, t), \end{aligned} \tag{39}$$

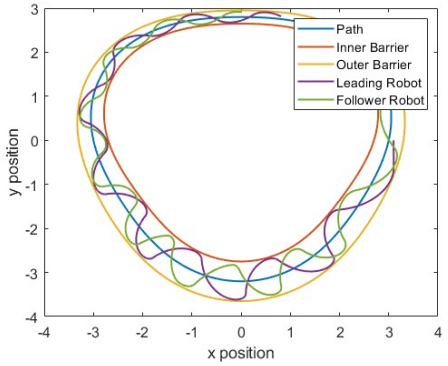


Figure 30: Trajectories of the two vehicles

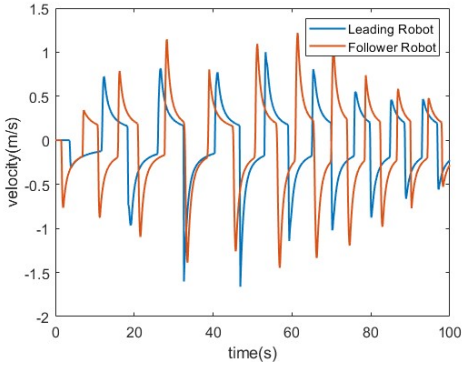


Figure 31: Angular velocity of the two vehicles

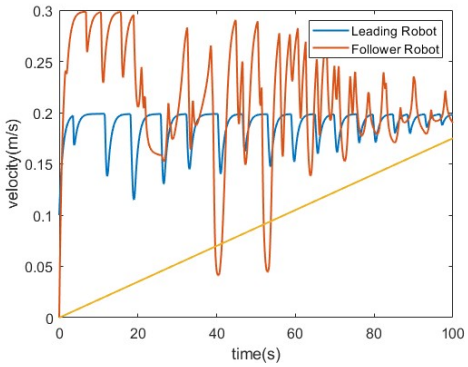


Figure 32: Velocity of the two vehicles

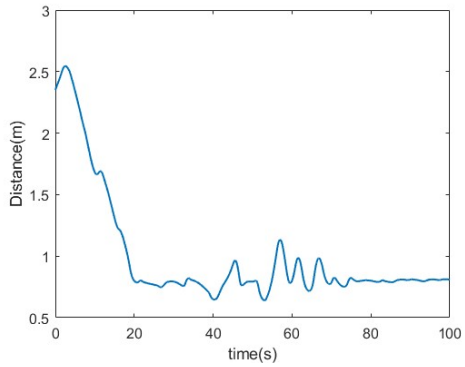


Figure 33: Distance between the two vehicles

Using Figure 30, we can easily see that a change in the width of the lane does not cause any stress on the system. This is primarily due to the fact that the barrier function is still continuous. In Figure 32, we can see that the leading robot slows down by a smaller margin when hitting the barrier.

### 4.5 *Network Model*

Sometimes, we would like the group of robots to maintain a certain formation, yet real-life constraints would still exist. Hence, we would like to see what would occur in such a situation. Having seen multiple different aspects regarding the applications of the CBFs, we can begin to bring them together in the following model of such a formation. We will be setting up a network as follows:

$$\begin{bmatrix} \dot{\mathbf{x}}_{l_1} \\ \dot{\mathbf{x}}_{l_2} \\ \dot{\mathbf{x}}_f \end{bmatrix} = \begin{bmatrix} F_{l_1} \\ F_{l_2} \\ F_f \end{bmatrix} + \begin{bmatrix} G_{l_1} & \mathbf{0}_{3\times 2} & \mathbf{0}_{3\times 2} \\ \mathbf{0}_{3\times 2} & G_{l_2} & \mathbf{0}_{3\times 2} \\ \mathbf{0}_{3\times 2} & \mathbf{0}_{3\times 2} & G_f \end{bmatrix} \begin{bmatrix} \mathbf{u}_{l_1} \\ \mathbf{u}_{l_2} \\ \mathbf{u}_f \end{bmatrix}. \tag{40}$$

Here, using the earlier definitions for the state, we have that:

$$\mathbf{x}_i = \begin{bmatrix} x_i \\ y_i \\ v_i \\ \psi_i \\ w_i \end{bmatrix}. \tag{41}$$

We also define the change in each of the states as defined earlier. We define each of the inputs as follows:

$$\mathbf{u}_i = \begin{bmatrix} u_{l_i} \\ u_{a_i} \end{bmatrix}. \tag{42}$$

For this model, what we wish to investigate is whether we can utilise a set of CLFs to uniformly enforce the CBFs to all parts of the network. For example, should one of the robots in the formation run into the lane barrier we would like it to also be enforced on the parts of the formation not close to the barrier. We will be keeping the soft conditions and barriers that we have imposed on the leading and follower robots in the last section. The only addition would be an ACC barrier between the second leading robot and the follower.

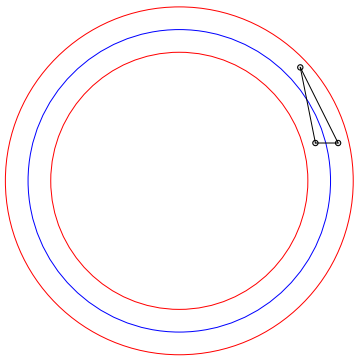

Figure 34: Network Problem

We will only be adding three soft conditions to ensure that the three robots maintain a triangular formation. The first step would be to ensure that the two leading robots run parallel at a certain distance from each other. To ensure this, we require two soft conditions.

The first condition is to maintain a certain distance between the robots. To do this, we can look back at how we constructed the soft condition for the tracker. We first define an error:

$$\eta_d = \sqrt{(x_{l_1} - x_{l_2})^2 + (y_{l_1} + y_{l_2})^2} - d_d. \tag{43}$$

Here, we are using the Euclidean distance between the two leading robots, and $d_d$ represents the desired distance between the two. For this simulation, we will be using $d_d = 0.4$. We can derive $\eta_d$ in terms of $t$ to get the following:

$$\dot{\eta}_d = \frac{(x_{l_1} - x_{l_2})}{\sqrt{(x_{l_1} - x_{l_2})^2 + (y_{l_1} + y_{l_2})^2}}(\dot{x}_{l_1} - \dot{x}_{l_2}) + \frac{(y_{l_1} - y_{l_2})}{\sqrt{(x_{l_1} - x_{l_2})^2 + (y_{l_1} + y_{l_2})^2}}(\dot{y}_{l_1} - \dot{y}_{l_2}). \tag{44}$$

We can then combine the two into a Lyapunov function as follows:

$$V_d = (\dot{\eta}_d + \lambda_d \eta_d)^2 = \begin{bmatrix} \eta_d & \dot{\eta}_d \end{bmatrix} \begin{bmatrix} \lambda_d^2 & \lambda_d \\ \lambda_d & 1 \end{bmatrix} \begin{bmatrix} \eta_d \\ \dot{\eta}_d \end{bmatrix}. \tag{45}$$

This soft condition will ensure that the two leading robots do not crash into each other. We do have to ensure that there exists a soft condition that forces the robots to move in parallel, rather than in succession. To do so, we ensure that the angles of the two robots in polar coordinates are close. To do so, we have the following:

$$\eta_a = \theta_{c_{l_1}}^2 - \theta_{c_{l_2}}^2. \tag{46}$$

Taking a derivative in terms of time, we get the following:

$$\dot{\eta}_a = 2\theta_{c_{l_1}} \dot{\theta}_{c_{l_1}} - 2\theta_{c_{l_2}} \dot{\theta}_{c_{l_2}}. \tag{47}$$

Finally, we enforce the condition using the following Lyapunov function.

$$V_a = (\dot{\eta}_a + \lambda_a \eta_a)^2 = \begin{bmatrix} \eta_a & \dot{\eta}_a \end{bmatrix} \begin{bmatrix} \lambda_a^2 & \lambda_a \\ \lambda_a & 1 \end{bmatrix} \begin{bmatrix} \eta_a \\ \dot{\eta}_a \end{bmatrix}. \tag{48}$$

Finally, we need to ensure the follower follows the midpoint between the two leading robots. To do so, we utilise the radius of the three robots in polar coordinates. As a result, we have the following error:

$$\eta_m = 0.5\sqrt{x_{l_1}^2 + y_{l_1}^2} + 0.5\sqrt{x_{l_2}^2 + y_{l_2}^2} - \sqrt{x_f^2 + y_f^2}. \tag{49}$$

Deriving in terms of time, gives us the following:

$$\dot{\eta}_m = 0.5\frac{x_{l_1}\dot{x}_{l_1} + y_{l_1}\dot{y}_{l_1}}{\sqrt{x_{l_1}^2 + y_{l_1}^2}} + 0.5\frac{x_{l_2}\dot{x}_{l_2} + y_{l_2}\dot{y}_{l_2}}{\sqrt{x_{l_2}^2 + y_{l_2}^2}} - \frac{x_f\dot{x}_f + y_f\dot{y}_f}{\sqrt{x_f^2 + y_f^2}}. \tag{50}$$

We have the final Lyapunov function to enforce the network formation.

$$V_m = (\dot{\eta}_m + \lambda_m \eta_m)^2 = \begin{bmatrix} \eta_m & \dot{\eta}_m \end{bmatrix} \begin{bmatrix} \lambda_m^2 & \lambda_m \\ \lambda_m & 1 \end{bmatrix} \begin{bmatrix} \eta_m \\ \dot{\eta}_m \end{bmatrix}. \tag{51}$$

With these three conditions in place, when optimizing we have the following:

$$\mathbf{u} = \begin{bmatrix} \begin{bmatrix} \mathbf{u}_{l_1} & \mathbf{u}_{l_2} & \mathbf{u}_f & \delta_{v_{l_1}} & \delta_{s_{l_1}} & \delta_{v_{l_2}} \end{bmatrix}^\top \\ \begin{bmatrix} \delta_{s_{l_2}} & \delta_{v_f} & \delta_{s_f} & \delta_d & \delta_a & \delta_m \end{bmatrix}^\top \end{bmatrix} \in \mathbb{R}^{15}.$$

Given this, we have the following optimisation problem:

$$\begin{aligned} u_{net}(\mathbf{x}) = \underset{\mathbf{u}\in\mathbb{R}^{15}}{\arg\min}\, & \mathbf{u}^\top \mathbf{H}(\mathbf{x})\mathbf{u} \\ s.t. \quad & L_f V_i(\mathbf{x}) + L_g V_i(\mathbf{x}) \begin{bmatrix} \mathbf{u}_{l_1} & \mathbf{u}_{l_2} & \mathbf{u}_f \end{bmatrix}^\top \leq -\gamma_i V_i(\mathbf{x}) + \delta_i, \\ & \qquad \forall i \in \{v_{l_1}, s_{l_1}, v_{l_2}, s_{l_2}, v_f, s_f, d, a, m\}, \\ & L_f h_j(\mathbf{x}) + L_g h_j(\mathbf{x}) \begin{bmatrix} \mathbf{u}_{l_1} & \mathbf{u}_{l_2} & \mathbf{u}_f \end{bmatrix}^\top \geq -\alpha_j h_j(\mathbf{x}), \\ & \qquad \forall j \in \{lk_{l_1}, lk_{l_2}, lk_f, acc_{l_1}, acc_{l_2}\}, \\ & L_f h_j(\mathbf{x}, t) + L_g h_j(\mathbf{x}, t) \begin{bmatrix} \mathbf{u}_{l_1} & \mathbf{u}_{l_2} & \mathbf{u}_f \end{bmatrix}^\top + \frac{\partial h_j(\mathbf{x}, t)}{\partial t} \geq -\alpha_j h_j(\mathbf{x}, t), \\ & \qquad \forall j \in \{vlb_{l_1}, vlb_{l_2}\}. \end{aligned} \tag{52}$$

Here, the $\mathbf{H}(\mathbf{x})$ takes the following form:

$$\mathbf{H}(\mathbf{x}) = \text{diag}(\frac{1}{m^2}, \frac{1}{I_z^2}, \frac{1}{m^2}, \frac{1}{I_z^2}, \frac{1}{m^2}, \frac{1}{I_z^2}, p_{v_{l_1}}, p_{s_{l_1}}, p_{v_{l_2}}, p_{s_{l_2}}, p_{v_f}, p_{s_f}, p_d, p_a, p_m). \tag{53}$$

Before we can simulate the model, we have to define a few more constants:

| Constant | Value | Constant | Value |
|---|---|---|---|
| $\gamma_i$ | 1 | $p_{v_i}$ | $10^7$ |
| $\alpha_j$ | 1 | $p_{s_i}$ | 10 |
| $\lambda_d$,$\lambda_a$,$\lambda_m$ | 2 | $p_d$,$p_a$,$p_m$ | $10^5$ |

Table 5: Constants for the Network model

As can be seen, we penalise the relaxation to the soft conditions for the longitudinal and angular velocity to a greater extent than we did for the ACC and LK model. We also penalise the soft condition in regard to the network to the same degree.

For this model, we will be plotting six different plots. The first three will be the same as the previous simulations, the trajectories, angular velocity, and longitudinal velocity of the three robots in the network. Despite there being two ACC barriers, we will only be plotting the distance between the midpoint of the two leading robots and the follower robot. Finally, we will be plotting the distance between the leading robots and between the midpoint of the radii of the leading robots and the radius of the follower.

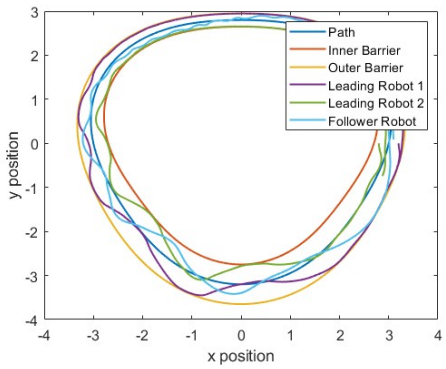


Figure 35: Trajectories of the three robots in the Network

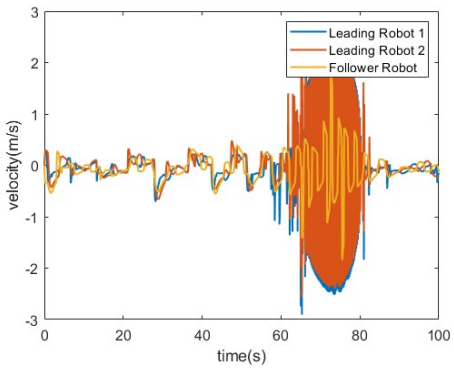


Figure 36: Angular Velocity of the three robots in the Network

Figures 37 and 38, showcase a similar output to earlier results, where the follower slows down as it gets closer the leader robots. Figure 35 shows us the two leading robots mirroring each other, while Figure 40 helps us see more clearly that the follower is maintaining the formation. When one of the leading robots hits the barrier and has to move away, the other two robots follow. We also see, using Figure 39, that the only time the two leading robots become much closer than the desired distance is when the lane narrows to a radius of 0.3. We can also see that being forced into the barrier has an extreme affect of the the angular velocity in Figure 36.

Given this set of results, we can see that a formation can be maintained within constrained environment. However, we can see that this has an adverse effect the angular velocity of the robots in question, especially when the two leading robots are forced into the lane barriers. Hence the formation should be implemented differently to ensure that the smoothness is not so heavily compromised.

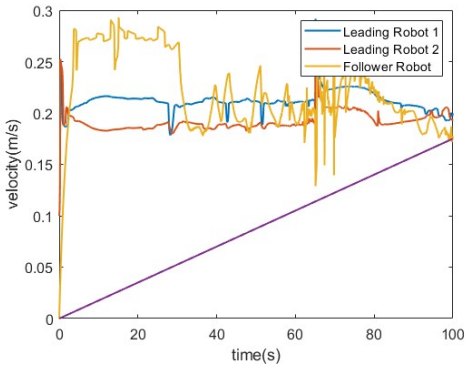


Figure 37: Longitudinal Velocity of the three robots in the Network

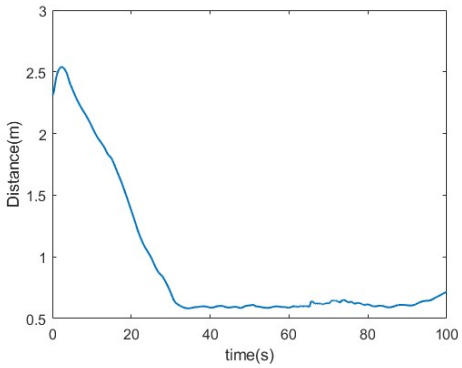


Figure 38: Distance between the midpoint of the leading robots and the follower

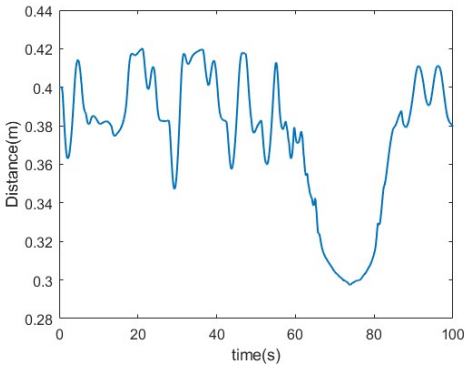


Figure 39: Distance between the Leading robots

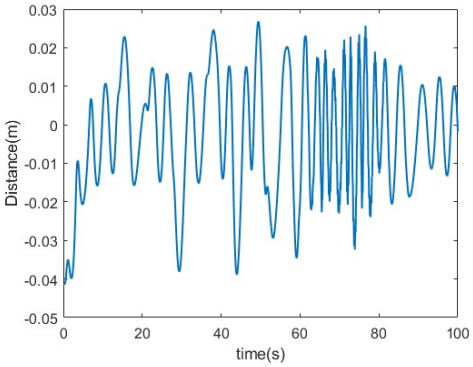


Figure 40: Distance between the midpoint of the radii of the leading robots and the radius of the follower

## 5 CONCLUSION & FURTHER IDEAS

In this thesis, we have studied Control Barrier Functions (CBFs) and their application in vehicle control. We began with three basic definitions, and a brief look at the stabilisation problem and Control Lyapunov Functions (CLFs). We have defined what safety means mathematically, as well as how a set can be considered safe. We considered Nagumo's Theorem, along with a proof by Brezis. We then defined CBFs and how they can be used to design safe controllers.

We then had 2 model simulations under 3 different conditions each. The first was only used to study the ACC problem, while the second was used to study an ACC and LK problem. These allowed us to see how CLFs and CBFs interact. We then simulated the second model under fused barriers to see if it would be better than the individual barriers. The results showcased that while the resulting barrier would remain safe with both barriers, it would be extremely conservative. We then simulated the model under time varying lower bound barrier for the velocity, along with a change in lane width. Finally, we simulated a network of robots tasked with maintaining a triangular shape. We saw that due to the barriers and soft conditions with respect to the shape of the network, the soft conditions on the individual robots suffered.

The research for the thesis could be extended in two directions, one in theory and another in application. In the thesis, we looked at barriers where $L_g h(x) \neq 0$, which may not always be the case. In that case, we could look to Higher Order Control Barrier Functions, where the barrier function has a relative degree greater than 1. In terms of application, we could look at how CBFs

can be employed to ensure that crashes do not occur at intersections, merging, or roundabouts among autonomous vehicles.

## 6 ACKNOWLEDGMENT

I would like to thank Prof. Bart Besselink for his constant support throughout the duration of this thesis. It was his constant feedback and assistance that allowed me to achieve my initial goals and beyond.

## 7 APPENDIX

```
%addpath(genpath([pwd filesep 'yalmip']));

clear all

%System Entries
Time_step    = 0.1; %Time Step
Final_T      = 100; %Final Time
m            = 1650; %Car Mass
c_0          = 51; %Constant
c_1          = 1.26; %Constant
c_2          = 0.4342; %Constant
a_1          = @(x)sin(x); %Acceleration of Leading vehicle
j            = 1; %Count start
v_d          = 3; %Desired velocity
t_s          = 0.1; %Time Headway
D_0          = 1; %Minimum Distance
DH         = [-t_s,0,1]; %Derivative of barrier function
F_r          = @(v_f)c_0+c_1*v_f+ c_2*v_f^2; %Aerodynamic Force
p            = 100; %Relaxing variable penalizing weight

%System and Entry setup
State            = zeros(3,Final_T/Time_step); %State Collection
State(2,1)       = 1.5; %Inital Velocity of lead
State(3,1)       = 0.5; %Initial Difference in velocity
state_change     = zeros(3,1); %State Change
F                = zeros(3,1); %F(x)
G                = zeros(3,1); %G(x)
G(1,1)           = 1/m;
H                = [0.5*(1/m)^2,0;0,p];

%System Run
for t=Time_step:Time_step:Final_T

    %Current State
    v_f = State(1,j); %Current Velocity of follower
    v_l = State(2,j); %Current Velocity of lead
    D   = State(3,j); %Current Difference in velocity

    %Current System
    F(1,1)  = -F_r(v_f)/m;
    F(2,1)  = a_1(t-Time_step);
    F(3,1)  = v_l-v_f;

    %Current Constraints
    DV_1    = [2*v_f-2*v_d,0,0]; %Derivative of Lyapunov function
    V_1     = (v_f-v_d)^2; %Lyapunov function
    h       = D-t_s*v_f-D_0; %Barrier function

    %Optimization
    u_star  = sdpvar(2,1); %Controller and relaxation variable
    Obj     = u_star'*H*u_star; %Objective
    D_v_1   = [DV_1*G,-1];
    D_h     = [DH*G,0];
    Con     = [DV_1*F+D_v_1*u_star+V_1<=0,DH*F+D_h*u_star>=-h]; %Constraints
              optimize(Con,Obj);
    U_c     = value(u_star); %Optimized Controller and relaxation variable

    %System
    state_change    = F+G*U_c(1,1); %Current State change
    j               = j+1; %New Count
    State(:,j)      = State(:,j-1)+Time_step*state_change; %New State
end

%Plot of Velocities for the two vehicles
figure (1)
```

```
plot(0:Time_step:Final_T,State(1,:),'--',0:Time_step:Final_T,State(2,:),'LineWidth',1.5)
    ;
set(gca,'fontsize', 12)
xlabel('time(s)')
ylabel('velocity(m/s)')

%Plot of the distance between the two vehicles
figure (2)
plot(0:Time_step:Final_T,State(3,:),'LineWidth',1.5);
set(gca,'fontsize', 12)
xlabel('time(s)')
ylabel('distance(m)')
```

Listing 1: Simple ACC Model

```
%addpath(genpath([pwd filesep 'yalmip']));
clear all
yalmip('clear')

%System Entries
Time_step    = 0.1; %Time Step
Final_T      = 100; %Final Time
m            = 0.08; %Robot Mass
a            = 0.03; %Distance between Point of Intrest and axis
I_z          = 0.5*m*0.06^2; %Moment of inertia
v_d_l        = 0.2; %Desired velocity for leading vehicle
v_d_f        = 0.3; %Desired velocity for following vehicle
R            = 3; %Mean Radius of path
b            = 0.2; %Amplitude
n            = 3; %Periods
T_s          = 0.05; %Time difference
d_min        = 0.1; %Desired minimum distance
d_max        = 0.3; %max distance from the path allowed
a_max        = 1;
weights      = [1/(m^2);1/(I_z^2);10^5;1;10^3];
j            = 1;
c_1          = 2;
c_2          = 2;
c_3          = 2;
c_lk         = 1;
c_acc        = 1;

%Path Functions
Path             = @(x)R+b*sin(n*x);
Theta_func       = @(x,y)atan2(y,x); %Theta
DTheta_func      = @(ang,v,w,x,y)(x/(x^2+y^2))*(v*sin(ang)+ a*w*cos(ang)) - (y/(x^2+y^2))
    *(v*cos(ang)- a*w*sin(ang)); %dTheta/dt
R_path           = @(x,y)R+b.*sin(n.*Theta_func(x,y)); %Path
DR_path          = @(ang,v,w,x,y)n*b*DTheta_func(ang,v,w,x,y)*cos(n*Theta_func(x,y)); %
    dPath/dt

%Lyapunov Function
V_rou        = @(x,y)x.^2+y.^2-R_path(x,y).^2;
V_cha        = @(ang,v,w,x,y)2*(x*(v*cos(ang)- a*w*sin(ang))+y*(v*sin(ang)+ a*w*cos(ang))
     -R_path(x,y)*DR_path(ang,v,w,x,y));
P            = [4,2;2,1];
V_3_func     = @(ang,v,w,x,y)[V_rou(x,y),V_cha(ang,v,w,x,y)]*P*[V_rou(x,y);V_cha(ang,v,w,
    x,y)];

%Lane Keeping Barrier
y_lat = @(x,y) sqrt(x.^2+y.^2)-R_path(x,y);
v_lat = @(ang,v,w,x,y) (x/sqrt(x.^2+y.^2))*( v*cos(ang)- a*w*sin(ang))+(y/sqrt(x.^2+y
    .^2))*(v*sin(ang)+ a*w*cos(ang))-DR_path(ang,v,w,x,y);
h_lk_func = @(ang,v,w,x,y)d_max-sign(v_lat(ang,v,w,x,y))*(y_lat(x,y)) -0.5*((v_lat(ang,v,
    w,x,y))^2/a_max);
%h_lk_func = @(ang,v,w,x,y)1- y_lat(x,y)^2/d_max^2-0.5*v_lat(ang,v,w,x,y)^2;

```

```
%Derivative
syms x_l y_l v_l ang_l w_l real
DV_3_l        = [diff(V_3_func(ang_l,v_l,w_l,x_l,y_l), x_l)
              diff(V_3_func(ang_l,v_l,w_l,x_l,y_l), y_l)
              diff(V_3_func(ang_l,v_l,w_l,x_l,y_l), v_l)
              diff(V_3_func(ang_l,v_l,w_l,x_l,y_l), ang_l)
              diff(V_3_func(ang_l,v_l,w_l,x_l,y_l), w_l)];
DV_3_func    = matlabFunction(DV_3_l);
DH_lk_l       = [diff(h_lk_func(ang_l,v_l,w_l,x_l,y_l), x_l)
              diff(h_lk_func(ang_l,v_l,w_l,x_l,y_l), y_l)
              diff(h_lk_func(ang_l,v_l,w_l,x_l,y_l), v_l)
              diff(h_lk_func(ang_l,v_l,w_l,x_l,y_l), ang_l)
              diff(h_lk_func(ang_l,v_l,w_l,x_l,y_l), w_l)];
DH_lk_func = matlabFunction(DH_lk_l);

%System
F_l              = zeros(5,1); %Frame for F for the leading robot
F_f              = zeros(5,1); %Frame for F for the leading robot
G                = zeros(5,2); %Frame for G
G(3,1)           = 1/m; %value for G
G(5,2)           = 1/I_z; %value for G
state_change_l    = zeros(5,1);
State_l          = zeros(5,Final_T/Time_step);
state_change_f    = zeros(5,1);
State_f          = zeros(5,Final_T/Time_step);
%H_arr            = zeros(1,Final_T/Time_step);
H_ly             = diag(weights);
%H_ba             = diag(weights(1:2));

%Initial State for lead
[State_l(1,1),State_l(2,1)] = pol2cart(0,Path(0)+0.1);
State_l(3,1) = 0.1;
State_l(4,1) = -pi/2;
State_l(5,1) = 0.001;

%Initial State for follower
[State_f(1,1),State_f(2,1)] = pol2cart(pi/4,Path(pi/4)-0.1);
State_f(3,1) = 0;
State_f(4,1) = 0;
State_f(5,1) = 0.001;

tic
%System run
for t=Time_step:Time_step:Final_T

    %Current State for lead
    x_l   = State_l(1,j); %Horizontal position
    y_l   = State_l(2,j); %Vertical position
    v_l   = State_l(3,j); %Longitudinal velocity
    ang_l = State_l(4,j); %Orientation
    w_l   = State_l(5,j); %Angular velocity

    %Current State for follower
    x_f   = State_f(1,j); %Horizontal position
    y_f   = State_f(2,j); %Vertical position
    v_f   = State_f(3,j); %Longitudinal velocity
    ang_f = State_f(4,j); %Orientation
    w_f   = State_f(5,j); %Angular velocity

    %Current System for lead
    F_l(1,1) = v_l*cos(ang_l)- a*w_l*sin(ang_l);
    F_l(2,1) = v_l*sin(ang_l)+ a*w_l*cos(ang_l);
    F_l(3,1) = -a*w_l^2;
    F_l(4,1) = w_l;

    %Current System for lead
    F_f(1,1) = v_f*cos(ang_f)- a*w_f*sin(ang_f);
```

```
    F_f(2,1) = v_f*sin(ang_f)+ a*w_f*cos(ang_f);
    F_f(3,1) = -a*w_f^2;
    F_f(4,1) = w_f;

    %Lyapunov Constraints for lead
    V_1_l    = (v_l-v_d_l)^2;
    V_2_l    = w_l^2;
    V_3_l    = V_3_func(ang_l,v_l,w_l,x_l,y_l);
    DV_1_l   = [0,0,2*v_l-2*v_d_l,0,0];
    DV_2_l   = [0,0,0,0,2*w_l];
    DV_3_l   = DV_3_func(ang_l,v_l,w_l,x_l,y_l)';

    %Lyapunov Constraints for follow
    V_1_f    = (v_f-v_d_f)^2;
    V_2_f    = w_f^2;
    V_3_f    = V_3_func(ang_f,v_f,w_f,x_f,y_f);
    DV_1_f   = [0,0,2*v_f-2*v_d_f,0,0];
    DV_2_f   = [0,0,0,0,2*w_f];
    DV_3_f   = DV_3_func(ang_f,v_f,w_f,x_f,y_f)';

    %Boundaries for lead
    H_lk_l = h_lk_func(ang_l,v_l,w_l,x_l,y_l);
    DH_lk_l = DH_lk_func(ang_l,v_l,w_l,x_l,y_l)';

    %Boundaries for follower
    Dist =(x_l-x_f)^2+(y_l-y_f)^2;
    H_lk_f = h_lk_func(ang_f,v_f,w_f,x_f,y_f);
    DH_lk_f = DH_lk_func(ang_f,v_f,w_f,x_f,y_f)';
    H_acc = sqrt(Dist)-d_min -T_s*v_f;
    Dx = (x_f-x_l)/sqrt(Dist);
    Dy = (y_f-y_l)/sqrt(Dist);
    DH_acc =[Dx,Dy,-T_s,0,0];

    %Optimization for lead
    u_star   = sdpvar(5,1);
    Obj      = u_star'*H_ly*u_star;
    D_v_1    = [DV_1_l*G,-1,0,0];
    D_v_2    = [DV_2_l*G,0,-1,0];
    D_v_3    = [DV_3_l*G,0,0,-1];
    D_h_1    = [DH_lk_l*G,0,0,0];
    if (t>=25 && t<= 50)|(abs(V_rou(x_l,y_l))>=0.2 && t>= 50)
        Con      = [DV_1_l*F_l+D_v_1*u_star+c_1*V_1_l<=0,...
                    DV_2_l*F_l+D_v_2*u_star+c_2*V_2_l<=0,...
                    DH_lk_l*F_l+D_h_1*u_star+c_lk*H_lk_l>= 0];
    else
        Con      = [DV_1_l*F_l+D_v_1*u_star+c_1*V_1_l<=0,...
                    DV_2_l*F_l+D_v_2*u_star+c_2*V_2_l<=0,...
                    DV_3_l*F_l+D_v_3*u_star+c_3*V_3_l<=0,...
                    DH_lk_l*F_l+D_h_1*u_star+c_lk*H_lk_l>= 0];
    end
    optimize(Con,Obj);
    U_l      = value(u_star);

    %Optimization
    u_star   = sdpvar(5,1);
    Obj      = u_star'*H_ly*u_star;
    D_v_1    = [DV_1_f*G,-1,0,0];
    D_v_2    = [DV_2_f*G,0,-1,0];
    D_v_3    = [DV_3_f*G,0,0,-1];
    D_h_1    = [DH_lk_f*G,0,0,0];
    D_h_2    = [DH_acc*G,0,0,0];
    if (t>=25 && t<= 50)|(abs(V_rou(x_f,y_f))>=0.2 && t>= 50)
        Con      = [DV_1_f*F_f+D_v_1*u_star+c_1*V_1_f<=0,...
                    DV_2_f*F_f+D_v_2*u_star+c_2*V_2_f<=0,...
                    DH_lk_f*F_f+D_h_1*u_star+c_lk*H_lk_f>= 0,...
                    DH_acc*F_f+D_h_2*u_star+c_acc*H_acc>=0];
    else
```

```
        Con      = [DV_1_f*F_f+D_v_1*u_star+c_1*V_1_f<=0,...
                    DV_2_f*F_f+D_v_2*u_star+c_2*V_2_f<=0,...
                    DV_3_f*F_f+D_v_3*u_star+c_3*V_3_f<=0,...
                    DH_lk_f*F_f+D_h_1*u_star+c_lk*H_lk_f>= 0,...
                    DH_acc*F_f+D_h_2*u_star+c_acc*H_acc>=0];
    end
    optimize(Con,Obj);
    U_f      = value(u_star);

    %System
    state_change_l   = F_l+G*U_l(1:2,1); %Current State change
    state_change_f   = F_f+G*U_f(1:2,1); %Current State change
    %H_arr(1,j) = DH_lk*F+D_h_1*u_star+c_lk*H_lk;
    j                = j+1; %New Count
    State_l(:,j)       = State_l(:,j-1)+Time_step*state_change_l; %New State
    State_f(:,j)       = State_f(:,j-1)+Time_step*state_change_f; %New State
end
toc
%Plot of robot path against true path
theta=0:0.00001:100;
D = sqrt((State_l(1,:)-State_f(1,:)).^2 + (State_l(2,:)-State_f(2,:)).^2);
[x_path,y_path] = pol2cart(theta,Path(theta));
[x_path_i,y_path_i] = pol2cart(theta,Path(theta)-d_max);
[x_path_o,y_path_o] = pol2cart(theta,Path(theta)+d_max);
figure(1)
plot(x_path,y_path,x_path_i,y_path_i,x_path_o,y_path_o,State_l(1,:),State_l(2,:),State_f
    (1,:),State_f(2,:),'LineWidth',1.5)
legend('Path','Inner Barrier','Outer Barrier','Leading Robot','Follower Robot')
set(gca,'fontsize', 12)
xlabel('x position')
ylabel('y position')

figure(2)
plot(0:Time_step:Final_T,State_l(3,:),0:Time_step:Final_T,State_f(3,:),'LineWidth',1.5)
legend('Leading Robot','Follower Robot')
set(gca,'fontsize', 12)
xlabel('time(s)')
ylabel('velocity(m/s)')
figure(3)
plot(0:Time_step:Final_T,State_l(5,:),0:Time_step:Final_T,State_f(5,:),'LineWidth',1.5)
legend('Leading Robot','Follower Robot')
set(gca,'fontsize', 12)
xlabel('time(s)')
ylabel('velocity(m/s)')
figure(4)
plot(0:Time_step:Final_T,D,'LineWidth',1.5)
set(gca,'fontsize', 12)
xlabel('time(s)')
ylabel('Distance(m)')
```

Listing 2: ACC and LK Model

```
%addpath(genpath([pwd filesep 'yalmip']));
clear all
yalmip('clear')

%System Entries
Time_step    = 0.1; %Time Step
Final_T      = 100; %Final Time
m            = 0.08; %Robot Mass
a            = 0.03; %Distance between Point of Intrest and axis
I_z          = 0.5*m*0.06^2; %Moment of inertia
v_d_l        = 0.2; %Desired velocity for leading vehicle
v_d_f        = 0.3; %Desired velocity for following vehicle
R            = 3; %Mean Radius of path
b            = 0.2; %Amplitude
n            = 3; %Periods
```

```
T_s          = 0.1; %Time difference
d_min        = 0.6; %Desired minimum distance
d_max        = 0.3; %max distance from the path allowed
a_max        = 1;
weights      = [1/(m^2);1/(I_z^2);10^5;1;10^3];
j            = 1;
c_1          = 2;
c_2          = 2;
c_3          = 2;
c_lk         = 1;
c_acc        = 1;
alpha        = 20;

%Path Functions
Path             = @(x)R+b*sin(n*x);
Theta_func       = @(x,y)atan2(y,x); %Theta
DTheta_func      = @(ang,v,w,x,y)(x/(x^2+y^2))*(v*sin(ang)+ a*w*cos(ang)) - (y/(x^2+y^2))
    *(v*cos(ang)- a*w*sin(ang)); %dTheta/dt
R_path           = @(x,y)R+b.*sin(n.*Theta_func(x,y)); %Path
DR_path          = @(ang,v,w,x,y)n*b*DTheta_func(ang,v,w,x,y)*cos(n*Theta_func(x,y)); %
    dPath/dt

%Lyapunov Function
V_rou        = @(x,y)x.^2+y.^2-R_path(x,y).^2;
V_cha        = @(ang,v,w,x,y)2*(x*(v*cos(ang)- a*w*sin(ang))+y*(v*sin(ang)+ a*w*cos(ang))
     -R_path(x,y)*DR_path(ang,v,w,x,y));
P            = [4,2;2,1];
V_3_func     = @(ang,v,w,x,y)[V_rou(x,y),V_cha(ang,v,w,x,y)]*P*[V_rou(x,y);V_cha(ang,v,w,
    x,y)];

%Lane Keeping Barrier
y_lat = @(x,y) sqrt(x.^2+y.^2)-R_path(x,y);
v_lat = @(ang,v,w,x,y) (x/sqrt(x.^2+y.^2))*( v*cos(ang)- a*w*sin(ang))+(y/sqrt(x.^2+y
    .^2))*(v*sin(ang)+ a*w*cos(ang))-DR_path(ang,v,w,x,y);
h_lk_func = @(ang,v,w,x,y)d_max-sign(v_lat(ang,v,w,x,y))*(y_lat(x,y)) -0.5*((v_lat(ang,v,
    w,x,y))^2/a_max);
%h_lk_func = @(ang,v,w,x,y)1- y_lat(x,y)^2/d_max^2-0.5*v_lat(ang,v,w,x,y)^2;

%Derivative
syms x_l y_l v_l ang_l w_l real
DV_3_l         = [diff(V_3_func(ang_l,v_l,w_l,x_l,y_l), x_l)
             diff(V_3_func(ang_l,v_l,w_l,x_l,y_l), y_l)
             diff(V_3_func(ang_l,v_l,w_l,x_l,y_l), v_l)
             diff(V_3_func(ang_l,v_l,w_l,x_l,y_l), ang_l)
             diff(V_3_func(ang_l,v_l,w_l,x_l,y_l), w_l)];
DV_3_func    = matlabFunction(DV_3_l);
DH_lk_l        = [diff(h_lk_func(ang_l,v_l,w_l,x_l,y_l), x_l)
             diff(h_lk_func(ang_l,v_l,w_l,x_l,y_l), y_l)
             diff(h_lk_func(ang_l,v_l,w_l,x_l,y_l), v_l)
             diff(h_lk_func(ang_l,v_l,w_l,x_l,y_l), ang_l)
             diff(h_lk_func(ang_l,v_l,w_l,x_l,y_l), w_l)];
DH_lk_func = matlabFunction(DH_lk_l);

%System
F_l              = zeros(5,1); %Frame for F for the leading robot
F_f              = zeros(5,1); %Frame for F for the leading robot
G                = zeros(5,2); %Frame for G
G(3,1)           = 1/m; %value for G
G(5,2)           = 1/I_z; %value for G
state_change_l     = zeros(5,1);
State_l          = zeros(5,Final_T/Time_step);
state_change_f     = zeros(5,1);
State_f          = zeros(5,Final_T/Time_step);
H_arr            = zeros(3,Final_T/Time_step);
H_ly             = diag(weights);
%H_ba              = diag(weights(1:2));

```

```
%Initial State for lead
[State_l(1,1),State_l(2,1)] = pol2cart(0,Path(0)+0.1);
State_l(3,1) = 0.1;
State_l(4,1) = -pi/2;
State_l(5,1) = 0.001;

%Initial State for follower
[State_f(1,1),State_f(2,1)] = pol2cart(pi/4,Path(pi/4)-0.1);
State_f(3,1) = 0;
State_f(4,1) = 0;
State_f(5,1) = 0.001;

tic
%System run
for t=Time_step:Time_step:Final_T

    %Current State for lead
    x_l   = State_l(1,j); %Horizontal position
    y_l   = State_l(2,j); %Vertical position
    v_l   = State_l(3,j); %Longitudinal velocity
    ang_l = State_l(4,j); %Orientation
    w_l   = State_l(5,j); %Angular velocity

    %Current State for follower
    x_f   = State_f(1,j); %Horizontal position
    y_f   = State_f(2,j); %Vertical position
    v_f   = State_f(3,j); %Longitudinal velocity
    ang_f = State_f(4,j); %Orientation
    w_f   = State_f(5,j); %Angular velocity

    %Current System for lead
    F_l(1,1) = v_l*cos(ang_l)- a*w_l*sin(ang_l);
    F_l(2,1) = v_l*sin(ang_l)+ a*w_l*cos(ang_l);
    F_l(3,1) = -a*w_l^2;
    F_l(4,1) = w_l;

    %Current System for lead
    F_f(1,1) = v_f*cos(ang_f)- a*w_f*sin(ang_f);
    F_f(2,1) = v_f*sin(ang_f)+ a*w_f*cos(ang_f);
    F_f(3,1) = -a*w_f^2;
    F_f(4,1) = w_f;

    %Lyapunov Constraints for lead
    V_1_l     = (v_l-v_d_l)^2;
    V_2_l     = w_l^2;
    V_3_l     = V_3_func(ang_l,v_l,w_l,x_l,y_l);
    DV_1_l    = [0,0,2*v_l-2*v_d_l,0,0];
    DV_2_l    = [0,0,0,0,2*w_l];
    DV_3_l    = DV_3_func(ang_l,v_l,w_l,x_l,y_l)';

    %Lyapunov Constraints for follow
    V_1_f     = (v_f-v_d_f)^2;
    V_2_f     = w_f^2;
    V_3_f     = V_3_func(ang_f,v_f,w_f,x_f,y_f);
    DV_1_f    = [0,0,2*v_f-2*v_d_f,0,0];
    DV_2_f    = [0,0,0,0,2*w_f];
    DV_3_f    = DV_3_func(ang_f,v_f,w_f,x_f,y_f)';

    %Boundaries for lead
    H_lk_l = h_lk_func(ang_l,v_l,w_l,x_l,y_l);
    DH_lk_l = DH_lk_func(ang_l,v_l,w_l,x_l,y_l)';

    %Boundaries for follower
    Dist =(x_l-x_f)^2+(y_l-y_f)^2;
    H_lk_f = h_lk_func(ang_f,v_f,w_f,x_f,y_f);
    DH_lk_f = DH_lk_func(ang_f,v_f,w_f,x_f,y_f)';
    H_acc = sqrt(Dist)-d_min -T_s*v_f;
```

```
    Dx = (x_f-x_l)/sqrt(Dist);
    Dy = (y_f-y_l)/sqrt(Dist);
    DH_acc =[Dx,Dy,-T_s,0,0];
    H_tot = -(1/alpha)*log(exp(-alpha*H_lk_f)+exp(-alpha*H_acc));
    DH_tot = (exp(alpha*-H_lk_f)/(exp(-alpha*H_lk_f)+exp(-alpha*H_acc)))*DH_lk_f+(exp(-alpha*H_acc)/(exp(-alpha*H_lk_f)+exp(-alpha*H_acc)))*DH_acc;
    H_arr(:,j) = [H_lk_f;H_acc;H_tot];

    %Optimization for lead
    u_star  = sdpvar(5,1);
    Obj     = u_star'*H_ly*u_star;
    D_v_1   = [DV_1_l*G,-1,0,0];
    D_v_2   = [DV_2_l*G,0,-1,0];
    D_v_3   = [DV_3_l*G,0,0,-1];
    D_h_1   = [DH_lk_l*G,0,0,0];
     Con     = [DV_1_l*F_l+D_v_1*u_star+c_1*V_1_l<=0,...
                DV_2_l*F_l+D_v_2*u_star+c_2*V_2_l<=0,...
                DH_lk_l*F_l+D_h_1*u_star+c_lk*H_lk_l>=0];
    optimize(Con,Obj);
    U_l     = value(u_star);

    %Optimization
    u_star  = sdpvar(5,1);
    Obj     = u_star'*H_ly*u_star;
    D_v_1   = [DV_1_f*G,-1,0,0];
    D_v_2   = [DV_2_f*G,0,-1,0];
    D_v_3   = [DV_3_f*G,0,0,-1];
    D_h_1   = [DH_tot*G,0,0,0];
    Con     = [DV_1_f*F_f+D_v_1*u_star+c_1*V_1_f<=0,...
                DV_2_f*F_f+D_v_2*u_star+c_2*V_2_f<=0,...
                DH_tot*F_f+D_h_1*u_star+c_lk*H_tot>=0];
    optimize(Con,Obj);
    U_f     = value(u_star);

    %System
    state_change_l   = F_l+G*U_l(1:2,1); %Current State change
    state_change_f   = F_f+G*U_f(1:2,1); %Current State change
    j               = j+1; %New Count
    State_l(:,j)      = State_l(:,j-1)+Time_step*state_change_l; %New State
    State_f(:,j)      = State_f(:,j-1)+Time_step*state_change_f; %New State
end
toc
%Plot of robot path against true path
theta=0:0.00001:100;
D = sqrt((State_l(1,:)-State_f(1,:)).^2 + (State_l(2,:)-State_f(2,:)).^2);
[x_path,y_path] = pol2cart(theta,Path(theta));
[x_path_i,y_path_i] = pol2cart(theta,Path(theta)-d_max);
[x_path_o,y_path_o] = pol2cart(theta,Path(theta)+d_max);
figure(1)
plot(x_path,y_path,x_path_i,y_path_i,x_path_o,y_path_o,State_l(1,:),State_l(2,:),State_f(1,:),State_f(2,:),'LineWidth',1.5)
legend('Path','Inner Barrier','Outer Barrier','Leading Robot','Follower Robot')
set(gca,'fontsize', 12)
xlabel('x position')
ylabel('y position')

figure(2)
plot(0:Time_step:Final_T,State_l(3,:),0:Time_step:Final_T,State_f(3,:),'LineWidth',1.5)
legend('Leading Robot','Follower Robot')
set(gca,'fontsize', 12)
xlabel('time(s)')
ylabel('velocity(m/s)')
figure(3)
plot(0:Time_step:Final_T,State_l(5,:),0:Time_step:Final_T,State_f(5,:),'LineWidth',1.5)
legend('Leading Robot','Follower Robot')
set(gca,'fontsize', 12)
xlabel('time(s)')
```

```
ylabel('velocity(m/s)')
figure(4)
plot(Time_step:Time_step:Final_T,H_arr(1,:),Time_step:Time_step:Final_T,H_arr(2,:),
    Time_step:Time_step:Final_T,H_arr(3,:),'LineWidth',1.5)
legend('Lane Keeping','Cruis Control','Combined')
set(gca,'fontsize', 12)
xlabel('time(s)')
ylabel('Barrier Function value')
```

Listing 3: Combined Barrier Function

```
%addpath(genpath([pwd filesep 'yalmip']));
clear all
yalmip('clear')

%System Entries
Time_step    = 0.1; %Time Step
Final_T      = 100; %Final Time
m            = 0.08; %Robot Mass
a            = 0.03; %Distance between Point of Intrest and axis
I_z          = 0.5*m*0.06^2; %Moment of inertia
v_d_l        = 0.2; %Desired velocity for leading vehicle
v_d_f        = 0.3; %Desired velocity for following vehicle
R            = 3; %Mean Radius of path
b            = 0.2; %Amplitude
n            = 3; %Periods
T_s          = 0.1; %Time difference
d_min        = 0.6; %Desired minimum distance
d_max        = 0.3; %max distance from the path allowed
a_max        = 1;
weights      = [1/(m^2);1/(I_z^2);10^5;1;10^3];
j            = 1;
c_1          = 2;
c_2          = 2;
c_3          = 2;
c_lk         = 1;
c_acc        = 1;
v_min_l      = 0.175;
alpha        = 100;
k            = 0.15;

%Path Functions
Path             = @(x)R+b*sin(n*x);
Theta_func       = @(x,y)atan2(y,x); %Theta
DTheta_func      = @(ang,v,w,x,y)(x/(x^2+y^2))*(v*sin(ang)+ a*w*cos(ang)) - (y/(x^2+y^2))
    *(v*cos(ang)- a*w*sin(ang)); %dTheta/dt
R_path           = @(x,y)R+b.*sin(n.*Theta_func(x,y)); %Path
DR_path          = @(ang,v,w,x,y)n*b*DTheta_func(ang,v,w,x,y)*cos(n*Theta_func(x,y)); %
    dPath/dt

%Lyapunov Function
V_rou        = @(x,y)x.^2+y.^2-R_path(x,y).^2;
V_cha        = @(ang,v,w,x,y)2*(x*(v*cos(ang)- a*w*sin(ang))+y*(v*sin(ang)+ a*w*cos(ang))
    -R_path(x,y)*DR_path(ang,v,w,x,y));
P            = [4,2;2,1];
V_3_func     = @(ang,v,w,x,y)[V_rou(x,y),V_cha(ang,v,w,x,y)]*P*[V_rou(x,y);V_cha(ang,v,w,
    x,y)];

%Lane Keeping Barrier
y_lat = @(x,y) sqrt(x.^2+y.^2)-R_path(x,y);
v_lat = @(ang,v,w,x,y) (x/sqrt(x.^2+y.^2))*( v*cos(ang)- a*w*sin(ang))+(y/sqrt(x.^2+y
    .^2))*(v*sin(ang)+ a*w*cos(ang))-DR_path(ang,v,w,x,y);
h_lk_func = @(ang,v,w,x,y)d_max-sign(v_lat(ang,v,w,x,y))*(y_lat(x,y)) -0.5*((v_lat(ang,v,
    w,x,y))^2/a_max)-k*sin(Theta_func(x,y));
%h_lk_func = @(ang,v,w,x,y)1- y_lat(x,y)^2/d_max^2-0.5*v_lat(ang,v,w,x,y)^2;


```

```
%Derivative
syms x_l y_l v_l ang_l w_l real
DV_3_l        = [diff(V_3_func(ang_l,v_l,w_l,x_l,y_l), x_l)
              diff(V_3_func(ang_l,v_l,w_l,x_l,y_l), y_l)
              diff(V_3_func(ang_l,v_l,w_l,x_l,y_l), v_l)
              diff(V_3_func(ang_l,v_l,w_l,x_l,y_l), ang_l)
              diff(V_3_func(ang_l,v_l,w_l,x_l,y_l), w_l)];
DV_3_func    = matlabFunction(DV_3_l);
DH_lk_l       = [diff(h_lk_func(ang_l,v_l,w_l,x_l,y_l), x_l)
              diff(h_lk_func(ang_l,v_l,w_l,x_l,y_l), y_l)
              diff(h_lk_func(ang_l,v_l,w_l,x_l,y_l), v_l)
              diff(h_lk_func(ang_l,v_l,w_l,x_l,y_l), ang_l)
              diff(h_lk_func(ang_l,v_l,w_l,x_l,y_l), w_l)];
DH_lk_func = matlabFunction(DH_lk_l);

%System
F_l              = zeros(5,1); %Frame for F for the leading robot
F_f              = zeros(5,1); %Frame for F for the leading robot
G                = zeros(5,2); %Frame for G
G(3,1)           = 1/m; %value for G
G(5,2)           = 1/I_z; %value for G
state_change_l    = zeros(5,1);
State_l          = zeros(5,Final_T/Time_step);
state_change_f    = zeros(5,1);
State_f          = zeros(5,Final_T/Time_step);
%H_arr            = zeros(1,Final_T/Time_step);
H_ly             = diag(weights);
%H_ba             = diag(weights(1:2));

%Initial State for lead
[State_l(1,1),State_l(2,1)] = pol2cart(0,Path(0)+0.1);
State_l(3,1) = 0.1;
State_l(4,1) = -pi/2;
State_l(5,1) = 0.001;

%Initial State for follower
[State_f(1,1),State_f(2,1)] = pol2cart(pi/4,Path(pi/4)-0.1);
State_f(3,1) = 0;
State_f(4,1) = 0;
State_f(5,1) = 0.001;

tic
%System run
for t=Time_step:Time_step:Final_T

    %Current State for lead
    x_l   = State_l(1,j); %Horizontal position
    y_l   = State_l(2,j); %Vertical position
    v_l   = State_l(3,j); %Longitudinal velocity
    ang_l = State_l(4,j); %Orientation
    w_l   = State_l(5,j); %Angular velocity

    %Current State for follower
    x_f   = State_f(1,j); %Horizontal position
    y_f   = State_f(2,j); %Vertical position
    v_f   = State_f(3,j); %Longitudinal velocity
    ang_f = State_f(4,j); %Orientation
    w_f   = State_f(5,j); %Angular velocity

    %Current System for lead
    F_l(1,1) = v_l*cos(ang_l)- a*w_l*sin(ang_l);
    F_l(2,1) = v_l*sin(ang_l)+ a*w_l*cos(ang_l);
    F_l(3,1) = -a*w_l^2;
    F_l(4,1) = w_l;

    %Current System for lead
    F_f(1,1) = v_f*cos(ang_f)- a*w_f*sin(ang_f);
```

```
118     F_f(2,1) = v_f*sin(ang_f)+ a*w_f*cos(ang_f);
119     F_f(3,1) = -a*w_f^2;
120     F_f(4,1) = w_f;
121
122     %Lyapunov Constraints for lead
123     V_1_l      = (v_l-v_d_l)^2;
124     V_2_l      = w_l^2;
125     V_3_l      = V_3_func(ang_l,v_l,w_l,x_l,y_l);
126     DV_1_l     = [0,0,2*v_l-2*v_d_l,0,0];
127     DV_2_l     = [0,0,0,0,2*w_l];
128     DV_3_l     = DV_3_func(ang_l,v_l,w_l,x_l,y_l)';
129
130     %Lyapunov Constraints for follow
131     V_1_f      = (v_f-v_d_f)^2;
132     V_2_f      = w_f^2;
133     V_3_f      = V_3_func(ang_f,v_f,w_f,x_f,y_f);
134     DV_1_f     = [0,0,2*v_f-2*v_d_f,0,0];
135     DV_2_f     = [0,0,0,0,2*w_f];
136     DV_3_f     = DV_3_func(ang_f,v_f,w_f,x_f,y_f)';
137
138     %Boundaries for lead
139     H_lk_l = h_lk_func(ang_l,v_l,w_l,x_l,y_l);
140     DH_lk_l = DH_lk_func(ang_l,v_l,w_l,x_l,y_l)';
141     H_v_l = v_l-(v_min_l/100)*t;
142     DH_v_l = [0,0,1,0,0];
143
144     %Boundaries for follower
145     Dist =(x_l-x_f)^2+(y_l-y_f)^2;
146     H_lk_f = h_lk_func(ang_f,v_f,w_f,x_f,y_f);
147     DH_lk_f = DH_lk_func(ang_f,v_f,w_f,x_f,y_f)';
148     H_acc = sqrt(Dist)-d_min -T_s*v_f;
149     Dx = (x_f-x_l)/sqrt(Dist);
150     Dy = (y_f-y_l)/sqrt(Dist);
151     DH_acc =[Dx,Dy,-T_s,0,0];
152     %Optimization for lead
153     u_star   = sdpvar(5,1);
154     Obj      = u_star'*H_ly*u_star;
155     D_v_1    = [DV_1_l*G,-1,0,0];
156     D_v_2    = [DV_2_l*G,0,-1,0];
157     D_v_3    = [DV_3_l*G,0,0,-1];
158     D_h_1    = [DH_lk_l*G,0,0,0];
159     D_h_2    = [DH_v_l*G,0,0,0];
160     Con      = [DV_1_l*F_l+D_v_1*u_star+c_1*V_1_l<=0,...
161               DV_2_l*F_l+D_v_2*u_star+c_2*V_2_l<=0,...
162               DH_lk_l*F_l+D_h_1*u_star+c_lk*H_lk_l>= 0,...
163               DH_v_l*F_l+D_h_2*u_star+c_lk*H_v_l>= 0];
164     optimize(Con,Obj);
165     U_l      = value(u_star);
166
167     %Optimization
168     u_star   = sdpvar(5,1);
169     Obj      = u_star'*H_ly*u_star;
170     D_v_1    = [DV_1_f*G,-1,0,0];
171     D_v_2    = [DV_2_f*G,0,-1,0];
172     D_v_3    = [DV_3_f*G,0,0,-1];
173     D_h_1    = [DH_lk_f*G,0,0,0];
174     D_h_2    = [DH_acc*G,0,0,0];
175     Con      = [DV_1_f*F_f+D_v_1*u_star+c_1*V_1_f<=0,...
176                 DV_2_f*F_f+D_v_2*u_star+c_2*V_2_f<=0,...
177                 DH_lk_f*F_f+D_h_1*u_star+c_lk*H_lk_f>= 0,...
178                 DH_acc*F_f+D_h_2*u_star+c_acc*H_acc>=0];
179     optimize(Con,Obj);
180     U_f      = value(u_star);
181
182     %System
183     state_change_l    = F_l+G*U_l(1:2,1); %Current State change
184     state_change_f    = F_f+G*U_f(1:2,1); %Current State change
```

```
    j                = j+1; %New Count
    State_l(:,j)       = State_l(:,j-1)+Time_step*state_change_l; %New State
    State_f(:,j)       = State_f(:,j-1)+Time_step*state_change_f; %New State
end
toc
%Plot of robot path against true path
theta=0:0.00001:100;
D = sqrt((State_l(1,:)-State_f(1,:)).^2 + (State_l(2,:)-State_f(2,:)).^2);
[x_path,y_path] = pol2cart(theta,Path(theta));
[x_path_i,y_path_i] = pol2cart(theta,Path(theta)-(d_max-k*sin(theta)));
[x_path_o,y_path_o] = pol2cart(theta,Path(theta)+(d_max-k*sin(theta)));
figure(1)
plot(x_path,y_path,x_path_i,y_path_i,x_path_o,y_path_o,State_l(1,:),State_l(2,:),State_f
    (1,:),State_f(2,:),'LineWidth',1.5)
legend('Path','Inner Barrier','Outer Barrier','Leading Robot','Follower Robot')
set(gca,'fontsize', 12)
xlabel('x position')
ylabel('y position')

figure(2)
plot(0:Time_step:Final_T,State_l(3,:),0:Time_step:Final_T,State_f(3,:),0:Time_step:
    Final_T,(v_min_l/100)*[0:Time_step:Final_T],'LineWidth',1.5)
legend('Leading Robot','Follower Robot')
set(gca,'fontsize', 12)
xlabel('time(s)')
ylabel('velocity(m/s)')
figure(3)
plot(0:Time_step:Final_T,State_l(5,:),0:Time_step:Final_T,State_f(5,:),'LineWidth',1.5)
legend('Leading Robot','Follower Robot')
set(gca,'fontsize', 12)
xlabel('time(s)')
ylabel('velocity(m/s)')
figure(4)
plot(0:Time_step:Final_T,D,'LineWidth',1.5)
set(gca,'fontsize', 12)
xlabel('time(s)')
ylabel('Distance(m)')
```

Listing 4: Lane Change and Time based Barrier Model

```
%addpath(genpath([pwd filesep 'yalmip']));
clear all
yalmip('clear')

%System Entries
Time_step    = 0.1; %Time Step
Final_T      = 100; %Final Time
m            = 0.08; %Robot Mass
a            = 0.03; %Distance between Point of Intrest and axis
I_z          = 0.5*m*0.06^2; %Moment of inertia
R            = 3; %Mean Radius of path
b            = 0.2; %Amplitude
n            = 3; %Periods
T_s          = 0.05; %Time difference
d_min        = 0.6; %Desired minimum distance
d_max        = 0.3; %max distance from the path allowed
v_d_l        = 0.2; %Desired velocity for leading vehicle
v_d_f        = 0.275; %Desired velocity for following vehicle
a_max        = 1;
weights      = [1/(m^2);1/(I_z^2);1/(m^2);1/(I_z^2);1/(m^2);1/(I_z^2)
    ;10^7;10;10^7;10;10^5;10^5;1;10^5];
j            = 1;
v_min        = 0.175;
alpha        = 100;
k            = 0.15;
theta        = 2;
dd_ab        = 0.4;
```

```
dd_c          = 0.5;

%Time Change of position
F_x = @(ang,v,w,x,y)v*cos(ang)- a*w*sin(ang);
F_y = @(ang,v,w,x,y)v*sin(ang)+ a*w*cos(ang);

%Path Functions
Path              = @(x)R+b*sin(n*x);
Thetaunc        = @(x,y)atan2(y,x); %Theta
DThetaunc       = @(ang,v,w,x,y)(x/(x^2+y^2))*( F_y(ang,v,w,x,y)) - (y/(x^2+y^2))*( F_x(
    ang,v,w,x,y)); %dTheta/dt
R_path            = @(x,y)R+b.*sin(n.*Thetaunc(x,y)); %Path
DR_path           = @(ang,v,w,x,y)n*b*DThetaunc(ang,v,w,x,y)*cos(n*Thetaunc(x,y)); %dPath/
    dt

%Lane Keeping Barrier
y_hat = @(x,y) sqrt(x.^2+y.^2)-R_path(x,y);
v_at = @(ang,v,w,x,y) (x/sqrt(x.^2+y.^2))*( F_x(ang,v,w,x,y))+(y/sqrt(x.^2+y.^2))*(F_y(
    ang,v,w,x,y))-DR_path(ang,v,w,x,y);
hk_func = @(ang,v,w,x,y)d_max-sign(v_at(ang,v,w,x,y))*(y_hat(x,y)) -0.5*((v_at(ang,v,w,x,
    y))^2/a_max)-k*sin(Thetaunc(x,y));

%Par Distance based condition
Dist = @(a_x,a_y,b_x,b_y)sqrt((a_x-b_x)^2+(a_y-b_y)^2);
Dist_x = @(a_x,a_y,b_x,b_y) (a_x-b_x)/Dist(a_x,a_y,b_x,b_y);
Dist_y = @(a_x,a_y,b_x,b_y) (a_y-b_y)/Dist(a_x,a_y,b_x,b_y);
Err_d = @(a_x,a_y,b_x,b_y,dd)Dist(a_x,a_y,b_x,b_y)-dd;
DErr_d = @(a_ang,a_v,a_w,a_x,a_y,b_ang,b_v,b_w,b_x,b_y)Dist_x(a_x,a_y,b_x,b_y)*F_x(a_ang
    ,a_v,a_w,a_x,a_y)+Dist_y(a_x,a_y,b_x,b_y)*F_y(a_ang,a_v,a_w,a_x,a_y)-(Dist_x(a_x,a_y
    ,b_x,b_y)*F_x(b_ang,b_v,b_w,b_x,b_y)+Dist_y(a_x,a_y,b_x,b_y)*F_y(b_ang,b_v,b_w,b_x,
    b_y));
Dist_ly = @(a_ang,a_v,a_w,a_x,a_y,b_ang,b_v,b_w,b_x,b_y,dd)[Err_d(a_x,a_y,b_x,b_y,dd),
    DErr_d(a_ang,a_v,a_w,a_x,a_y,b_ang,b_v,b_w,b_x,b_y)]*[theta^2,theta;theta,1]*[Err_d(
    a_x,a_y,b_x,b_y,dd);DErr_d(a_ang,a_v,a_w,a_x,a_y,b_ang,b_v,b_w,b_x,b_y)];

%Angle based condition
Err_a = @(a_x,a_y,b_x,b_y)Thetaunc(a_x,a_y)^2-Thetaunc(b_x,b_y)^2;
DErr_a = @(a_ang,a_v,a_w,a_x,a_y,b_ang,b_v,b_w,b_x,b_y)2*(Thetaunc(a_x,a_y)*DThetaunc(
    a_ang,a_v,a_w,a_x,a_y)-Thetaunc(b_x,b_y)*DThetaunc(b_ang,b_v,b_w,b_x,b_y));
Ang_ly = @(a_ang,a_v,a_w,a_x,a_y,b_ang,b_v,b_w,b_x,b_y)[Err_a(a_x,a_y,b_x,b_y),DErr_a(
    a_ang,a_v,a_w,a_x,a_y,b_ang,b_v,b_w,b_x,b_y)]*[theta^2,theta;theta,1]*[Err_a(a_x,a_y
    ,b_x,b_y);DErr_a(a_ang,a_v,a_w,a_x,a_y,b_ang,b_v,b_w,b_x,b_y)];

%Follower Distance based condition
Err_Mid = @(a_x,a_y,b_x,b_y,c_x,c_y) 0.5*sqrt(a_x^2+a_y^2)+ 0.5*sqrt(b_x^2+b_y^2) - sqrt
    (c_x^2+c_y^2);
Dr_pol = @(ang,v,w,x,y) (x/sqrt(x.^2+y.^2))*( F_x(ang,v,w,x,y))+(y/sqrt(x.^2+y.^2))*(F_y
    (ang,v,w,x,y));
DErr_Mid =@(a_ang,a_v,a_w,a_x,a_y,b_ang,b_v,b_w,b_x,b_y,c_ang,c_v,c_w,c_x,c_y)0.5*Dr_pol
    (a_ang,a_v,a_w,a_x,a_y)+0.5*Dr_pol(b_ang,b_v,b_w,b_x,b_y)-Dr_pol(c_ang,c_v,c_w,c_x,
    c_y);
Mid_ly   =@(a_ang,a_v,a_w,a_x,a_y,b_ang,b_v,b_w,b_x,b_y,c_ang,c_v,c_w,c_x,c_y)[Err_Mid(
    a_x,a_y,b_x,b_y,c_x,c_y),DErr_Mid(a_ang,a_v,a_w,a_x,a_y,b_ang,b_v,b_w,b_x,b_y,c_ang,
    c_v,c_w,c_x,c_y)]*[theta^2,theta;theta,1]*[Err_Mid(a_x,a_y,b_x,b_y,c_x,c_y);DErr_Mid
    (a_ang,a_v,a_w,a_x,a_y,b_ang,b_v,b_w,b_x,b_y,c_ang,c_v,c_w,c_x,c_y)];

%Derivative
syms a_ang a_v a_w a_x a_y b_ang b_v b_w b_x b_y dd c_ang c_v c_w c_x c_y
DHk       = [diff(hk_func(a_ang,a_v,a_w,a_x,a_y), a_x)
             diff(hk_func(a_ang,a_v,a_w,a_x,a_y), a_y)
             diff(hk_func(a_ang,a_v,a_w,a_x,a_y), a_v)
             diff(hk_func(a_ang,a_v,a_w,a_x,a_y), a_ang)
             diff(hk_func(a_ang,a_v,a_w,a_x,a_y), a_w)];
DHk_func = matlabFunction(DHk);

DDist_ly_a = [diff(Dist_ly(a_ang,a_v,a_w,a_x,a_y,b_ang,b_v,b_w,b_x,b_y,dd),a_x)
              diff(Dist_ly(a_ang,a_v,a_w,a_x,a_y,b_ang,b_v,b_w,b_x,b_y,dd),a_y)
```

```
75              diff(Dist_ly(a_ang,a_v,a_w,a_x,a_y,b_ang,b_v,b_w,b_x,b_y,dd),a_v)
76              diff(Dist_ly(a_ang,a_v,a_w,a_x,a_y,b_ang,b_v,b_w,b_x,b_y,dd),a_ang)
77              diff(Dist_ly(a_ang,a_v,a_w,a_x,a_y,b_ang,b_v,b_w,b_x,b_y,dd),a_w)];
78 DDist_ly_a_func = matlabFunction(DDist_ly_a);
79
80 DDist_ly_b = [diff(Dist_ly(a_ang,a_v,a_w,a_x,a_y,b_ang,b_v,b_w,b_x,b_y,dd),b_x)
81              diff(Dist_ly(a_ang,a_v,a_w,a_x,a_y,b_ang,b_v,b_w,b_x,b_y,dd),b_y)
82              diff(Dist_ly(a_ang,a_v,a_w,a_x,a_y,b_ang,b_v,b_w,b_x,b_y,dd),b_v)
83              diff(Dist_ly(a_ang,a_v,a_w,a_x,a_y,b_ang,b_v,b_w,b_x,b_y,dd),b_ang)
84              diff(Dist_ly(a_ang,a_v,a_w,a_x,a_y,b_ang,b_v,b_w,b_x,b_y,dd),b_w)];
85 DDist_ly_b_func = matlabFunction(DDist_ly_b);
86
87 DAng_ly_a = [diff(Ang_ly(a_ang,a_v,a_w,a_x,a_y,b_ang,b_v,b_w,b_x,b_y),a_x)
88              diff(Ang_ly(a_ang,a_v,a_w,a_x,a_y,b_ang,b_v,b_w,b_x,b_y),a_y)
89              diff(Ang_ly(a_ang,a_v,a_w,a_x,a_y,b_ang,b_v,b_w,b_x,b_y),a_v)
90              diff(Ang_ly(a_ang,a_v,a_w,a_x,a_y,b_ang,b_v,b_w,b_x,b_y),a_ang)
91              diff(Ang_ly(a_ang,a_v,a_w,a_x,a_y,b_ang,b_v,b_w,b_x,b_y),a_w)];
92 DAng_ly_a_func = matlabFunction(DAng_ly_a);
93
94 DAng_ly_b = [diff(Ang_ly(a_ang,a_v,a_w,a_x,a_y,b_ang,b_v,b_w,b_x,b_y),b_x)
95              diff(Ang_ly(a_ang,a_v,a_w,a_x,a_y,b_ang,b_v,b_w,b_x,b_y),b_y)
96              diff(Ang_ly(a_ang,a_v,a_w,a_x,a_y,b_ang,b_v,b_w,b_x,b_y),b_v)
97              diff(Ang_ly(a_ang,a_v,a_w,a_x,a_y,b_ang,b_v,b_w,b_x,b_y),b_ang)
98              diff(Ang_ly(a_ang,a_v,a_w,a_x,a_y,b_ang,b_v,b_w,b_x,b_y),b_w)];
99 DAng_ly_b_func = matlabFunction(DAng_ly_b);
100
101 DMid_ly = [diff(Mid_ly(a_ang,a_v,a_w,a_x,a_y,b_ang,b_v,b_w,b_x,b_y,c_ang,c_v,c_w,c_x,c_y
    ),a_x)
102              diff(Mid_ly(a_ang,a_v,a_w,a_x,a_y,b_ang,b_v,b_w,b_x,b_y,c_ang,c_v,c_w,c_x,
    c_y),a_y)
103              diff(Mid_ly(a_ang,a_v,a_w,a_x,a_y,b_ang,b_v,b_w,b_x,b_y,c_ang,c_v,c_w,c_x,
    c_y),a_v)
104              diff(Mid_ly(a_ang,a_v,a_w,a_x,a_y,b_ang,b_v,b_w,b_x,b_y,c_ang,c_v,c_w,c_x,
    c_y),a_ang)
105              diff(Mid_ly(a_ang,a_v,a_w,a_x,a_y,b_ang,b_v,b_w,b_x,b_y,c_ang,c_v,c_w,c_x,
    c_y),a_w)
106              diff(Mid_ly(a_ang,a_v,a_w,a_x,a_y,b_ang,b_v,b_w,b_x,b_y,c_ang,c_v,c_w,c_x,
    c_y),b_x)
107              diff(Mid_ly(a_ang,a_v,a_w,a_x,a_y,b_ang,b_v,b_w,b_x,b_y,c_ang,c_v,c_w,c_x,
    c_y),b_y)
108              diff(Mid_ly(a_ang,a_v,a_w,a_x,a_y,b_ang,b_v,b_w,b_x,b_y,c_ang,c_v,c_w,c_x,
    c_y),b_v)
109              diff(Mid_ly(a_ang,a_v,a_w,a_x,a_y,b_ang,b_v,b_w,b_x,b_y,c_ang,c_v,c_w,c_x,
    c_y),b_ang)
110              diff(Mid_ly(a_ang,a_v,a_w,a_x,a_y,b_ang,b_v,b_w,b_x,b_y,c_ang,c_v,c_w,c_x,
    c_y),b_w)
111              diff(Mid_ly(a_ang,a_v,a_w,a_x,a_y,b_ang,b_v,b_w,b_x,b_y,c_ang,c_v,c_w,c_x,
    c_y),c_x)
112              diff(Mid_ly(a_ang,a_v,a_w,a_x,a_y,b_ang,b_v,b_w,b_x,b_y,c_ang,c_v,c_w,c_x,
    c_y),c_y)
113              diff(Mid_ly(a_ang,a_v,a_w,a_x,a_y,b_ang,b_v,b_w,b_x,b_y,c_ang,c_v,c_w,c_x,
    c_y),c_v)
114              diff(Mid_ly(a_ang,a_v,a_w,a_x,a_y,b_ang,b_v,b_w,b_x,b_y,c_ang,c_v,c_w,c_x,
    c_y),c_ang)
115              diff(Mid_ly(a_ang,a_v,a_w,a_x,a_y,b_ang,b_v,b_w,b_x,b_y,c_ang,c_v,c_w,c_x,
    c_y),c_w)];
116 DMid_ly_func = matlabFunction(DMid_ly);
117 clear a_ang a_v a_w a_x a_y b_ang b_v b_w b_x b_y dd c_ang c_v c_w c_x c_y
118
119
120 %System Setup
121 G_i             = zeros(5,2); %Frame for G
122 G_i(3,1)        = 1/m; %value for G
123 G_i(5,2)        = 1/I_z; %value for G
124 G               = [G_i,zeros(5,2),zeros(5,2);zeros(5,2),G_i,zeros(5,2);zeros(5,2),zeros
    (5,2),G_i];
125 F_func          = @(ang,v,w,x,y)[ F_x(ang,v,w,x,y); F_y(ang,v,w,x,y);-a*w^2;w;0];
```

```
State_change     = zeros(15,1);
State            = zeros(15,Final_T/Time_step);
H                = diag(weights);

%Initial Points
[State(1,1),State(2,1)] = pol2cart(0,Path(0)+0.2);
State(3,1) = 0.1;
State(4,1) = -pi/2;
State(5,1) = 0.001;
[State(6,1),State(7,1)] = pol2cart(0,Path(0)-0.2);
State(8,1) = 0.1;
State(9,1) = -pi/2;
State(10,1) = 0.001;
[State(11,1),State(12,1)] =  pol2cart(pi/4,Path(pi/4)-0.1);
State(13,1) = 0;
State(14,1) = 0;
State(15,1) = 0.001;

tic
%System run
for t=Time_step:Time_step:Final_T

    %Current State
    x_l_1   = State(1,j); %Horizontal position
    y_l_1   = State(2,j); %Vertical position
    v_l_1   = State(3,j); %Longitudinal velocity
    ang_l_1 = State(4,j); %Orientation
    w_l_1   = State(5,j); %Angular velocity
    x_l_2   = State(6,j); %Horizontal position
    y_l_2   = State(7,j); %Vertical position
    v_l_2   = State(8,j); %Longitudinal velocity
    ang_l_2 = State(9,j); %Orientation
    w_l_2   = State(10,j); %Angular velocity
    x_f   = State(11,j); %Horizontal position
    y_f   = State(12,j); %Vertical position
    v_f   = State(13,j); %Longitudinal velocity
    ang_f = State(14,j); %Orientation
    w_f   = State(15,j); %Angular velocity

    %System
    F_l_1 = F_func(ang_l_1,v_l_1,w_l_1,x_l_1,y_l_1);
    F_l_2 = F_func(ang_l_2,v_l_2,w_l_2,x_l_2,y_l_2);
    F_f = F_func(ang_f,v_f,w_f,x_f,y_f);
    F = [F_l_1;F_l_2;F_f];

    Err_1(j) = sqrt((0.5*x_l_1+0.5*x_l_2-x_f)^2+(0.5*y_l_1+0.5*y_l_2-y_f)^2);
    Err_2(j) = Err_Mid(x_l_1,y_l_1,x_l_2,y_l_2,x_f,y_f);

    %Lyapunov Constraints
    V_1 = (v_l_1-v_d_l)^2;
    V_2 = w_l_1^2;
    V_3 = (v_l_2-v_d_l)^2;
    V_4 = w_l_2^2;
    V_5 = (v_f-v_d_f)^2;
    V_6 = w_f^2;
    V_7 = Dist_ly(ang_l_1,v_l_1,w_l_1,x_l_1,y_l_1,ang_l_2,v_l_2,w_l_2,x_l_2,y_l_2,dd_ab)
    ;
    V_8 = Mid_ly(ang_l_1,v_l_1,w_l_1,x_l_1,y_l_1,ang_l_2,v_l_2,w_l_2,x_l_2,y_l_2,ang_f,
    v_f,w_f,x_f,y_f);
    V_10 = Ang_ly(ang_l_1,v_l_1,w_l_1,x_l_1,y_l_1,ang_l_2,v_l_2,w_l_2,x_l_2,y_l_2);

    %Lyapunov Derivatives
    DV_1 = [0,0,2*v_l_1-2*v_d_l,0,0,zeros(1,5),zeros(1,5)];
    DV_2 = [0,0,0,0,2*w_l_1,zeros(1,5),zeros(1,5)];
    DV_3 = [zeros(1,5),0,0,2*v_l_2-2*v_d_l,0,0,zeros(1,5)];
    DV_4 = [zeros(1,5),0,0,0,0,2*w_l_2,zeros(1,5)];
    DV_5 = [zeros(1,5),zeros(1,5),0,0,2*v_f-2*v_d_f,0,0];
```

```
DV_6 = [zeros(1,5),zeros(1,5),0,0,0,0,2*w_f];
dv_7_1 = DDist_ly_a_func(ang_l_1,v_l_1,w_l_1,x_l_1,y_l_1,ang_l_2,v_l_2,w_l_2,x_l_2,y_l_2,dd_ab);
dv_7_2 = DDist_ly_b_func(ang_l_1,v_l_1,w_l_1,x_l_1,y_l_1,ang_l_2,v_l_2,w_l_2,x_l_2,y_l_2,dd_ab);
DV_7 = [dv_7_1',dv_7_2',zeros(1,5)];
DV_8 = DMid_ly_func(ang_l_1,v_l_1,w_l_1,x_l_1,y_l_1,ang_l_2,v_l_2,w_l_2,x_l_2,y_l_2,ang_f,v_f,w_f,x_f,y_f)';
dv_10_1 = DAng_ly_a_func(ang_l_1,v_l_1,w_l_1,x_l_1,y_l_1,ang_l_2,v_l_2,w_l_2,x_l_2,y_l_2);
dv_10_2 = DAng_ly_b_func(ang_l_1,v_l_1,w_l_1,x_l_1,y_l_1,ang_l_2,v_l_2,w_l_2,x_l_2,y_l_2);
DV_10 = [dv_10_1',dv_10_2',zeros(1,5)];

%Boundaries
H_1 = hk_func(ang_l_1,v_l_1,w_l_1,x_l_1,y_l_1);
H_2 = hk_func(ang_l_2,v_l_2,w_l_2,x_l_2,y_l_2);
H_3 = hk_func(ang_f,v_f,w_f,x_f,y_f);
H_4 = Dist(x_l_1,y_l_1,x_f,y_f)-d_min -T_s*v_f;
H_5 = Dist(x_l_2,y_l_2,x_f,y_f)-d_min -T_s*v_f;
H_6 = v_l_1-(v_min/100)*t;
H_7 = v_l_2-(v_min/100)*t;

%Boundary Derivatives
DH_1 = [DHk_func(ang_l_1,v_l_1,w_l_1,x_l_1,y_l_1)',zeros(1,5),zeros(1,5)];
DH_2 = [zeros(1,5),DHk_func(ang_l_2,v_l_2,w_l_2,x_l_2,y_l_2)',zeros(1,5)];
DH_3 = [zeros(1,5),zeros(1,5),DHk_func(ang_f,v_f,w_f,x_f,y_f)'];
DH_4 = [Dist_x(x_l_1,y_l_1,x_f,y_f),Dist_y(x_l_1,y_l_1,x_f,y_f),zeros(1,8),-Dist_x(x_l_1,y_l_1,x_f,y_f),-Dist_y(x_l_1,y_l_1,x_f,y_f),-T_s,0,0];
DH_5 = [zeros(1,5),Dist_x(x_l_2,y_l_2,x_f,y_f),Dist_y(x_l_2,y_l_2,x_f,y_f),0,0,0,-Dist_x(x_l_2,y_l_2,x_f,y_f),-Dist_y(x_l_2,y_l_2,x_f,y_f),-T_s,0,0];
DH_6 = [0,0,1,0,0,zeros(1,5),zeros(1,5)];
DH_7 = [zeros(1,5),0,0,1,0,0,zeros(1,5)];

%Optimization
u_star = sdpvar(16,1);
Obj    = u_star'*H*u_star;
D_1    = [DV_1*G,-1,0,0,0,0,0,0,0,0,0];
D_2    = [DV_2*G,0,-1,0,0,0,0,0,0,0,0];
D_3    = [DV_3*G,0,0,-1,0,0,0,0,0,0,0];
D_4    = [DV_4*G,0,0,0,-1,0,0,0,0,0,0];
D_5    = [DV_5*G,0,0,0,0,-1,0,0,0,0,0];
D_6    = [DV_6*G,0,0,0,0,0,-1,0,0,0,0];
D_7    = [DV_7*G,0,0,0,0,0,0,-1,0,0,0];
D_8    = [DV_8*G,0,0,0,0,0,0,0,-1,0,0];
D_10   = [DV_10*G,0,0,0,0,0,0,0,0,0,-1];
L_1    = [DH_1*G,zeros(1,10)];
L_2    = [DH_2*G,zeros(1,10)];
L_3    = [DH_3*G,zeros(1,10)];
L_4    = [DH_4*G,zeros(1,10)];
L_5    = [DH_5*G,zeros(1,10)];
L_6    = [DH_6*G,zeros(1,10)];
L_7    = [DH_7*G,zeros(1,10)];
Con    = [DV_1*F+D_1*u_star+V_1<=0,...
          DV_2*F+D_2*u_star+V_2<=0,...
          DV_3*F+D_3*u_star+V_3<=0,...
          DV_4*F+D_4*u_star+V_4<=0,...
          DV_5*F+D_5*u_star+V_5<=0,...
          DV_6*F+D_6*u_star+V_6<=0,...
          DV_7*F+D_7*u_star+V_7<=0,...
          DV_8*F+D_8*u_star+V_8<=0,...
          DV_10*F+D_10*u_star+V_10<=0,...
          DH_1*F+L_1*u_star+H_1>=0,...
          DH_2*F+L_2*u_star+H_2>=0,...
          DH_3*F+L_3*u_star+H_3>=0,...
          DH_4*F+L_4*u_star+H_4>=0,...
          DH_5*F+L_5*u_star+H_5>=0,...
```

```
                DH_6*F+L_6*u_star-(v_min/100)+H_6>=0,...
                DH_7*F+L_7*u_star-(v_min/100)+H_7>=0];

    optimize(Con,Obj);
    U      = value(u_star);
    Err(j) = U(15);
    %System Change
    State_change = F+G*U(1:6,1);
    j            = j+1; %New Count
    State(:,j)   = State(:,j-1)+Time_step*State_change; %New State

end


toc
theta=0:0.00001:100;
[x_path,y_path] = pol2cart(theta,Path(theta));
[x_path_i,y_path_i] = pol2cart(theta,Path(theta)-(d_max-k*sin(theta)));
[x_path_o,y_path_o] = pol2cart(theta,Path(theta)+(d_max-k*sin(theta)));
figure(1)
plot(x_path,y_path,x_path_i,y_path_i,x_path_o,y_path_o,State(1,:),State(2,:),State(6,:),
    State(7,:),State(11,:),State(12,:),'LineWidth',1.5)
legend('Path','Inner Barrier','Outer Barrier','Leading Robot 1','Leading Robot 2','
    Follower Robot')
set(gca,'fontsize', 12)
xlabel('x position')
ylabel('y position')

figure(2)
plot(0:Time_step:Final_T,State(3,:),0:Time_step:Final_T,State(8,:),0:Time_step:Final_T,
    State(13,:),0:Time_step:Final_T,((0:Time_step:Final_T)/100)*0.175,'LineWidth',1.5)
legend('Leading Robot 1','Leading Robot 2','Follower Robot')
set(gca,'fontsize', 12)
xlabel('time(s)')
ylabel('velocity(m/s)')

figure(3)
plot(0:Time_step:Final_T,State(5,:),0:Time_step:Final_T,State(10,:),0:Time_step:Final_T,
    State(15,:),'LineWidth',1.5)
legend('Leading Robot 1','Leading Robot 2','Follower Robot')
set(gca,'fontsize', 12)
xlabel('time(s)')
ylabel('velocity(m/s)')

D = arrayfun(Dist,State(1,:),State(2,:),State(6,:),State(7,:));
figure(4)
plot(0:Time_step:Final_T,D,'LineWidth',1.5)
set(gca,'fontsize', 12)
xlabel('time(s)')
ylabel('Distance(m)')

figure(5)
plot(Time_step:Time_step:Final_T,Err_1,'LineWidth',1.5)
set(gca,'fontsize', 12)
xlabel('time(s)')
ylabel('Distance(m)')

figure(6)
plot(Time_step:Time_step:Final_T,Err_2,'LineWidth',1.5)
set(gca,'fontsize', 12)
xlabel('time(s)')
ylabel('Distance(m)')
```

## REFERENCES


1. Blanchini, F., Miani, S., *et al. Set-theoretic methods in control* (Springer, 2008).
2. Brezis, H. On a characterization of flow-invariant sets. *Communications on Pure and Applied Mathematics* **23,** 261–263 (1970).
3. Lindemann, L. & Dimarogonas, D. V. Control barrier functions for signal temporal logic tasks. *IEEE control systems letters* **3,** 96–101 (2018).
4. Löfberg, J. *YALMIP : A Toolbox for Modeling and Optimization in MATLAB* in *In Proceedings of the CACSD Conference* (Taipei, Taiwan, 2004).
5. Xiao, W., Cassandras, C. G. & Belta, C. *Safe autonomy with control barrier functions: theory and applications* (Springer, 2023).
6. Xu, X., Grizzle, J. W., Tabuada, P. & Ames, A. D. Correctness guarantees for the composition of lane keeping and adaptive cruise control. *IEEE Transactions on Automation Science and Engineering* **15,** 1216–1229 (2017).
7. Xu, X. *et al. Realizing simultaneous lane keeping and adaptive speed regulation on accessible mobile robot testbeds* in *2017 IEEE conference on control technology and applications (CCTA)* (2017), 1769–1775.