

# ACC verification using NNV toolbox

## Course Project CS659A

Himanshu Singh (180296)

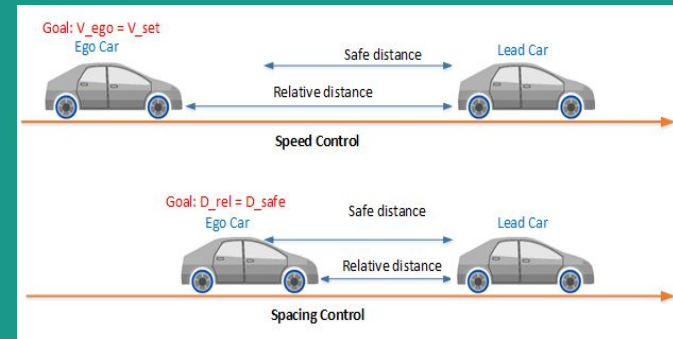
Deepanker Mishra (180225)

Areeb Ahmad (180135)

---

<https://github.com/deepankermishra225/NNV-Verification>

# What is ACC ?



Adaptive cruise control (ACC) is an intelligent form of cruise control that slows down and speeds up automatically to keep pace with the car in front of you. The driver sets the maximum speed — just as with cruise control — then a radar sensor watches for traffic ahead, locks on to the car in a lane, and instructs the car. Control is based on sensor information from on-board sensors ( such as radar or laser sensor or a camera setup )



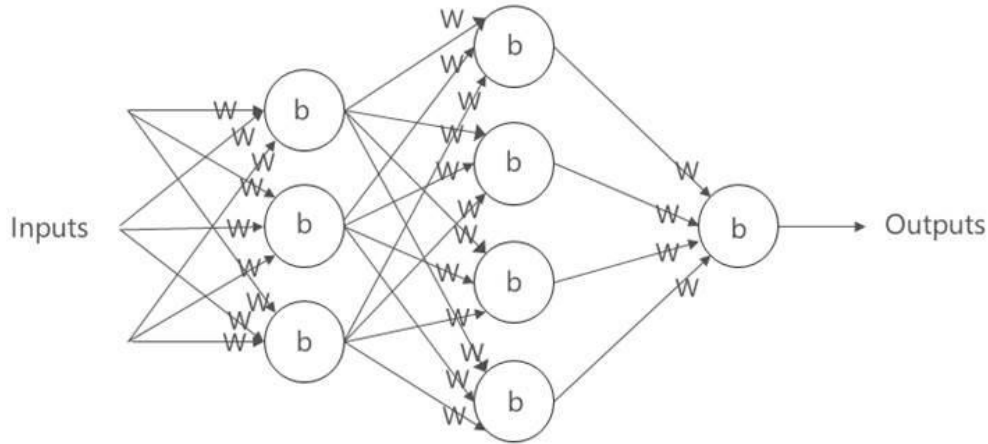
# What is NNV ?

The Neural Network Verification (NNV) software tool, a set-based verification framework for deep neural networks (DNNs) and learning-enabled cyber-physical systems (CPS)

The NNV toolbox contains two main modules: a computation engine and an analyzer

The computation engine consists of four sub-components: 1) the FFNN constructor, 2) the NNCS constructor, 3) the reachability solvers, and 4) the evaluator

## Feed-forward neural network (FNN)



$$y_i = f(\sum_j w_{ij} x_j + b_i)$$

*here  $f(x) = \text{ReLU}(x) = \max(x, 0)$*

A FNN consists of an input layer, an output layer, and multiple hidden layers in which each layer comprises of neurons that are connected to the neurons of preceding layer labeled using weights( $w$ ) .

# NNCS architecture

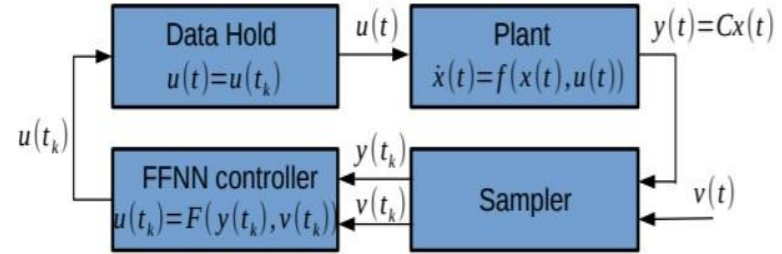


Figure IV.1: An architecture of NNCS supported in NNV.

NNV supports verification of closed loop control systems with an FFNN controller with piecewise linear activation function. The Verification of a neural network control system (NNCS) consists of six main steps: Constructing an NNCS object  $\Rightarrow$  Specifying a property of the system that we want to verify  $\Rightarrow$  Choosing a reachability analysis method  $\Rightarrow$  Constructing an initial set of states of the system  $\Rightarrow$  Choosing the number of cores utilized for computation  $\Rightarrow$  Verifying and visualizing the system

# Star based reachability methods



DNNs is a promising for fields like autonomous vehicles and air traffic collision avoidance systems .

But the safety of autonomous applications containing neural network components cannot be assured because DNNs usually have complex characteristics and behavior that are generally unpredictable.

It has been proved that well-trained DNNs may not be robust and are affected by slight changes in the input.



Several methods have been proposed for Safety verification and robustness certification of DNNs from different fields like machine learning , formal methods , and security, and a recent survey of the area is available.

Analyzing the behavior of a DNN can be categorized two ways

- (i) Exact analyses : eg. SMT-based and polyhedron-based approaches
- (ii) Over-approximate analyses : eg. The mixed integer linear program (MILP) , interval arithmetic, zonotope, input partition, linearization, and abstract-domain.

## The exact analysis



It is more time-consuming and less scalable. But it guarantees the soundness and completeness of the result.

## The approximate analysis:

It is faster and more scalable than the exact analysis, but only the soundness of the result can be guaranteed.

It is unclear how good the over-approximation is in terms of conservativeness since the exact result is not available for comparison.


Also, if an over-approximation approach is too conservative for neural networks with small or medium sizes, it will produce erroneous results for DNNs with a large number of layers and neurons because of accumulation of over-approximation error over layers.





By discussion in previous slide we can clearly see that a scalable, exact reachability analysis is needed on for formal verification of DNNs, and also for estimating the conservativeness of current and up-coming over-approximation methods.

In a paper by ,a fast and scalable approach for the exact and over-approximate reachability analysis of DNN with ReLU activation functions is proposed ,using the concept of 'star sets' .




**Reachable Set of FNN:** Given a bounded convex polyhedron input set defined as  $I, \{x \mid Ax \leq b, x \in \mathbb{R}^n\}$ , and an  $k$ -layers feed-forward neural network  $F, \{L_1, \dots, L_k\}$ , the reachable set  $F(I) = R_{L_k}$  of the neural network  $F$  corresponding to the input set  $I$  is defined as:

$$R_{L(1)} = \{y_1 \mid y_1 = \text{ReLU}(W_1 x + b_1); x \in I\}$$

$$R_{L(k)} = \{y_k \mid y_k = \text{ReLU}(W_k y_{k-1} + b_k) \mid y_{k-1} \in R_{L(k-1)}\}$$

$W_k$  = weight matrix of  $k$ th layer ;  $b_k$  = bias vector of  $k$ th layer



**Safety verification:** Given a safety specification  $S$  defined as a set of linear constraints on the neural network outputs  $S = \{y_k \mid Cy_k \leq d\}$ . A FNN  $F$  will be called safe i.e  $F(I) \models S$  iff, for input set  $I$

$$R_{L(k)} \cap \sim S = \emptyset$$

**Star set :** A star set is defined as a tuple  $\langle c, V, P \rangle$ , where  $P(\text{predicate}): \mathbb{R}^m = \{T (\text{True}), \perp (\text{false})\}$ ,  $c \in \mathbb{R}^n$  is the center.  $V = \{v_1, v_2, \dots, v_m\}$  is a set of vectors where  $v_i \in \mathbb{R}^n$ .

The basis vectors are arranged to form the star's  $n \times m$  basis matrix. The set of states represented by the star is given as:

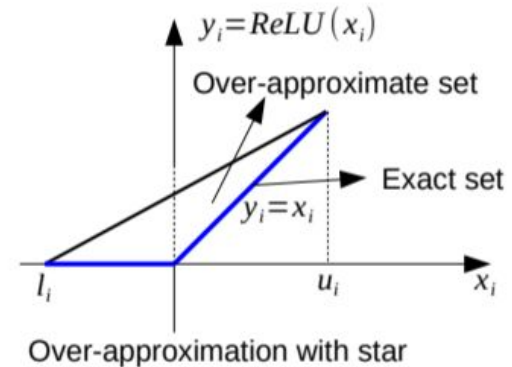
$$\|\Theta\| = \{x \mid x = c + \sum (\alpha_i v_i) \mid P(\alpha) = C\alpha \leq d \text{ where } d \in \mathbb{R}^{p \times 1}, C \in \mathbb{R}^{p \times m}\}$$



The worst case complexity of number of stars in exact star method is  $O(2^N)$ . Thus it is computationally very heavy. Hence we will not use 'exact star' method which computes exact value at every layer of FNN.

We will use 'approx star method'. The experiments show that approximate star-based approach can obtain tight reachable sets for many networks compared to the exact sets.

$l_i$  = lower bound ,  $u_i$  = upper bound



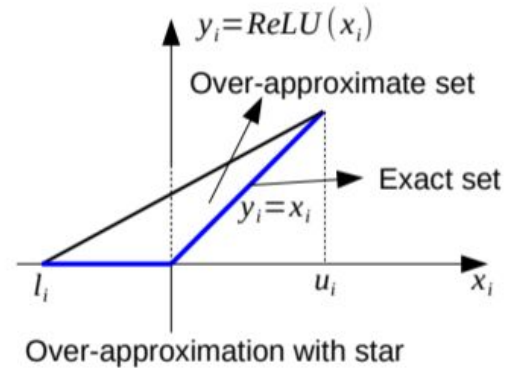
$l_i$  = lower bound ,  $u_i$  = upper bound

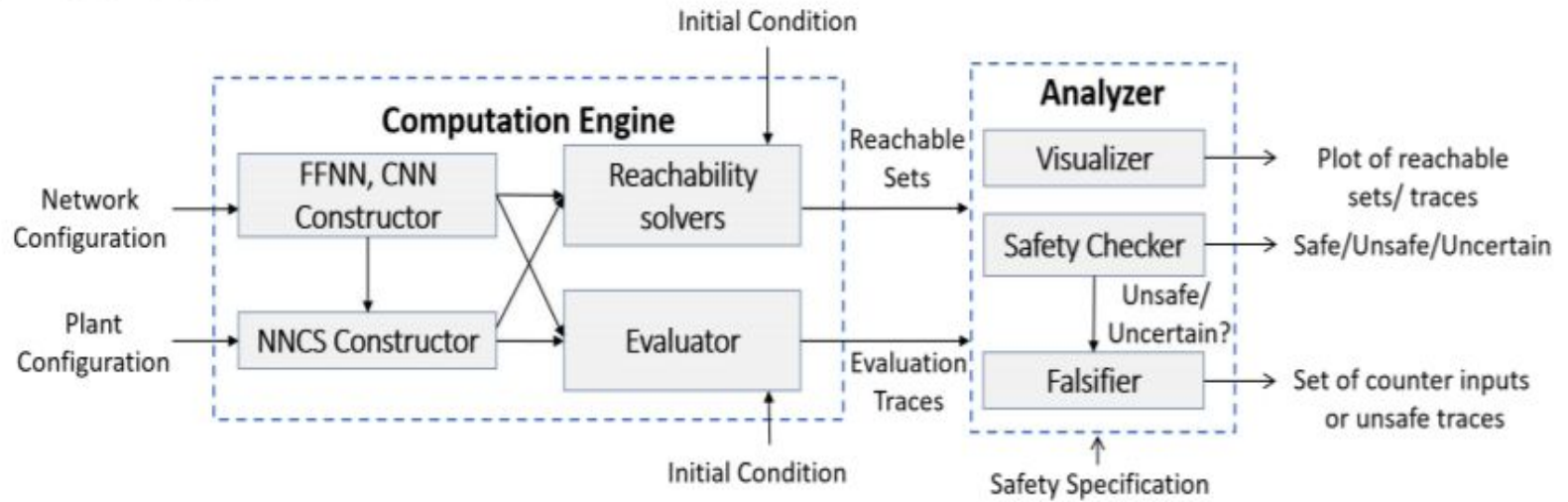
Approximation rule for a single neuron is given as follows:

$$y_i = x_i \quad \text{if } l_i \geq 0$$

$$y_i = 0 \quad \text{if } u_i \leq 0$$

$$y_i = u_i(x_i - l_i)/(u_i - l_i) \quad \text{if } l_i < 0 \text{ and } u_i > 0$$





# SYSTEM DYNAMICS

---

The ego car is set to travel at a set speed  $V_{set} = 30$  and maintains a safe distance  $D_{safe}$  from the lead car. The car's dynamics are described as follows:

$$\dot{x}_{lead}(t) = v_{lead}(t)$$

$$\dot{v}_{lead}(t) = \gamma_{lead}(t)$$

$$\gamma_{lead}(t) = -2\gamma_{lead}(t) + a_{lead} - \mu v_{lead}^2(t)$$

$$\dot{x}_{ego}(t) = v_{ego}(t)$$

$$\dot{v}_{ego}(t) = \gamma_{ego}(t)$$

$$\gamma_{ego}(t) = -2\gamma_{ego}(t) + a_{ego} - \mu v_{ego}^2(t)$$

where  $x_i$ ,  $v_i$ ,  $\gamma_i$ ,  $a_i$  and  $\mu = 0.0001$  are the position, velocity, acceleration, acceleration control input applied and friction control respectively and  $i \in \{\text{ego}, \text{lead}\}$ . For this benchmark we have used four neural network controllers with 3, 5, 7, and 10 hidden layers of 20 neurons each

# Safety Specification

The verification objective of this system is that given a scenario where both cars are driving safely, the lead car suddenly slows down with  $a_{\text{lead}} = -2$ . We want to check whether there is a collision in the following 5 seconds. Formally, this safety specification of the system can be expressed as

$$D_{\text{rel}} = x_{\text{lead}} - x_{\text{ego}} \geq D_{\text{safe}}, \text{ where } D_{\text{safe}} = D_{\text{default}} + T_{\text{gap}} \times v_{\text{ego}}, \text{ and } T_{\text{gap}} = 1.4 \text{ seconds and } D_{\text{default}} = 10.$$

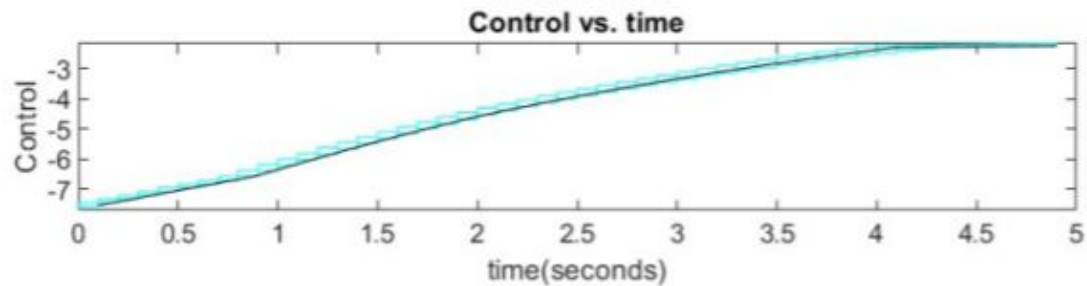
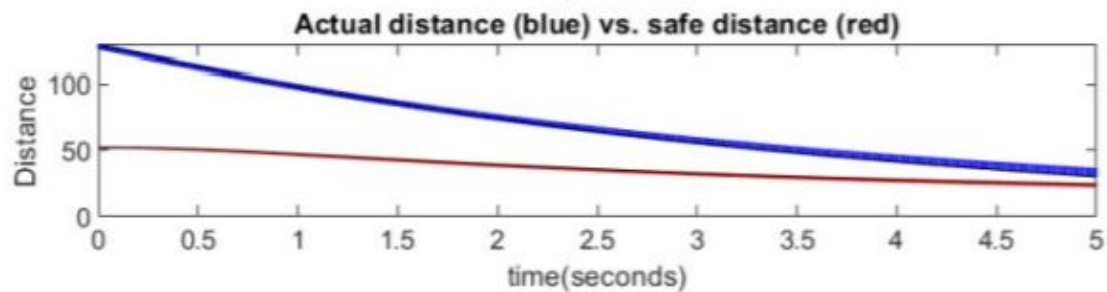
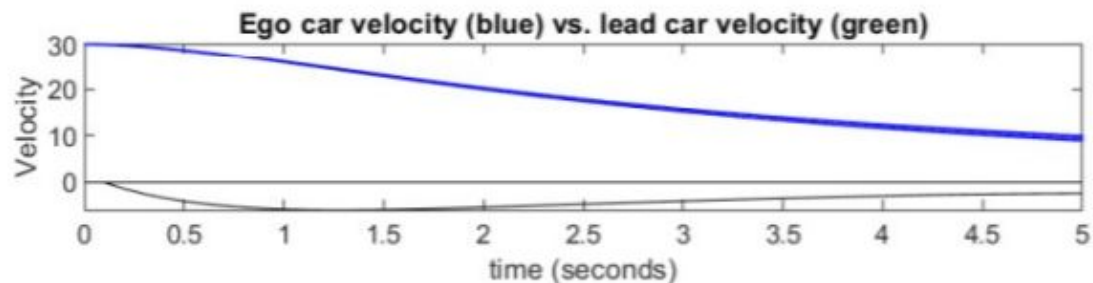


# Verification Result



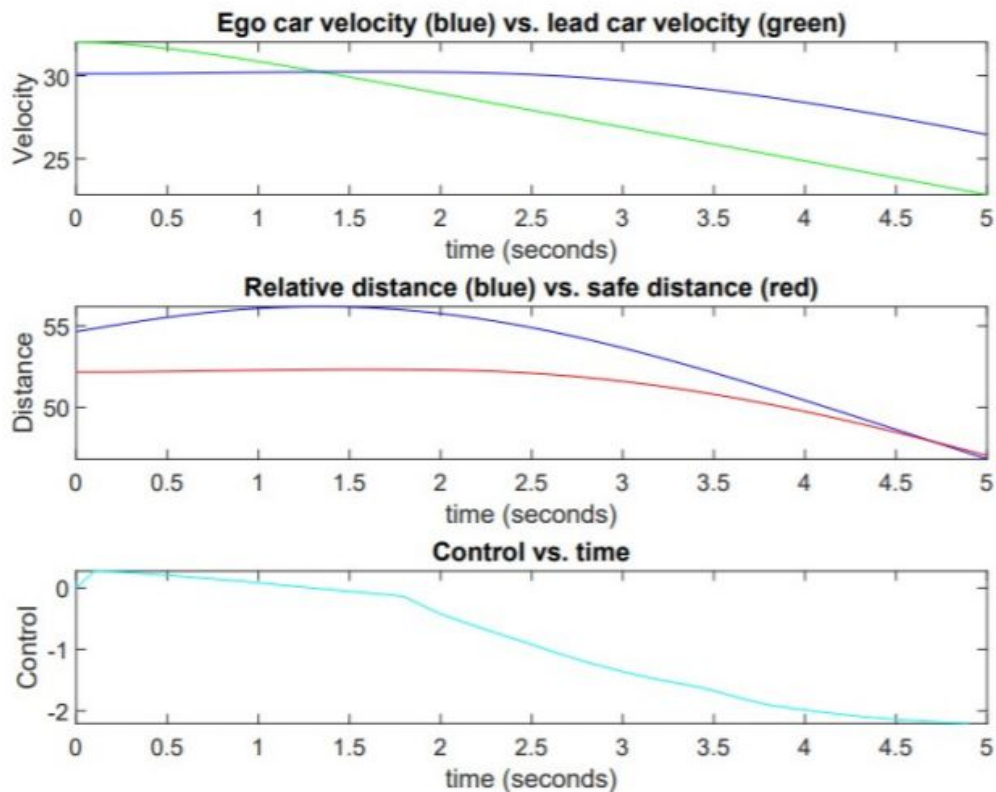
**Table 3.1.** Verification Result for fnns with 3 hidden layers of 20 neurons each

Distance lower bound	Distance upper bound	Safety Result	Verification Time
108	110	Safe	130.1396
106	108	Safe	32.2183
104	106	Safe	31.1764
102	104	Safe	36.1196
100	102	Unsafe	35.7430
98	100	Unsafe	34.5732
96	98	Unsafe	36.9429
94	96	Unsafe	38.2245
92	94	Unsafe	50.3784



Falsification Plots for  $x_{\text{lead}} \in [65.0 \ 70.0]$ ,  $v_{\text{lead}} \in [32.0 \ 32.2]$ ,  $a_{\text{internal\_lead}} = 0$ ,

$x_{\text{lead}} \in [10 \ 11]$ ,  $v_{\text{lead}} \in [30 \ 30.2]$ ,  $a_{\text{internal\_lead}} = 0$



# Emergency Breaking Verification



$$\dot{x}_{\text{ego}}(t) = v_{\text{ego}}(t)$$

$$\dot{v}_{\text{ego}}(t) = \gamma_{\text{ego}}(t)$$

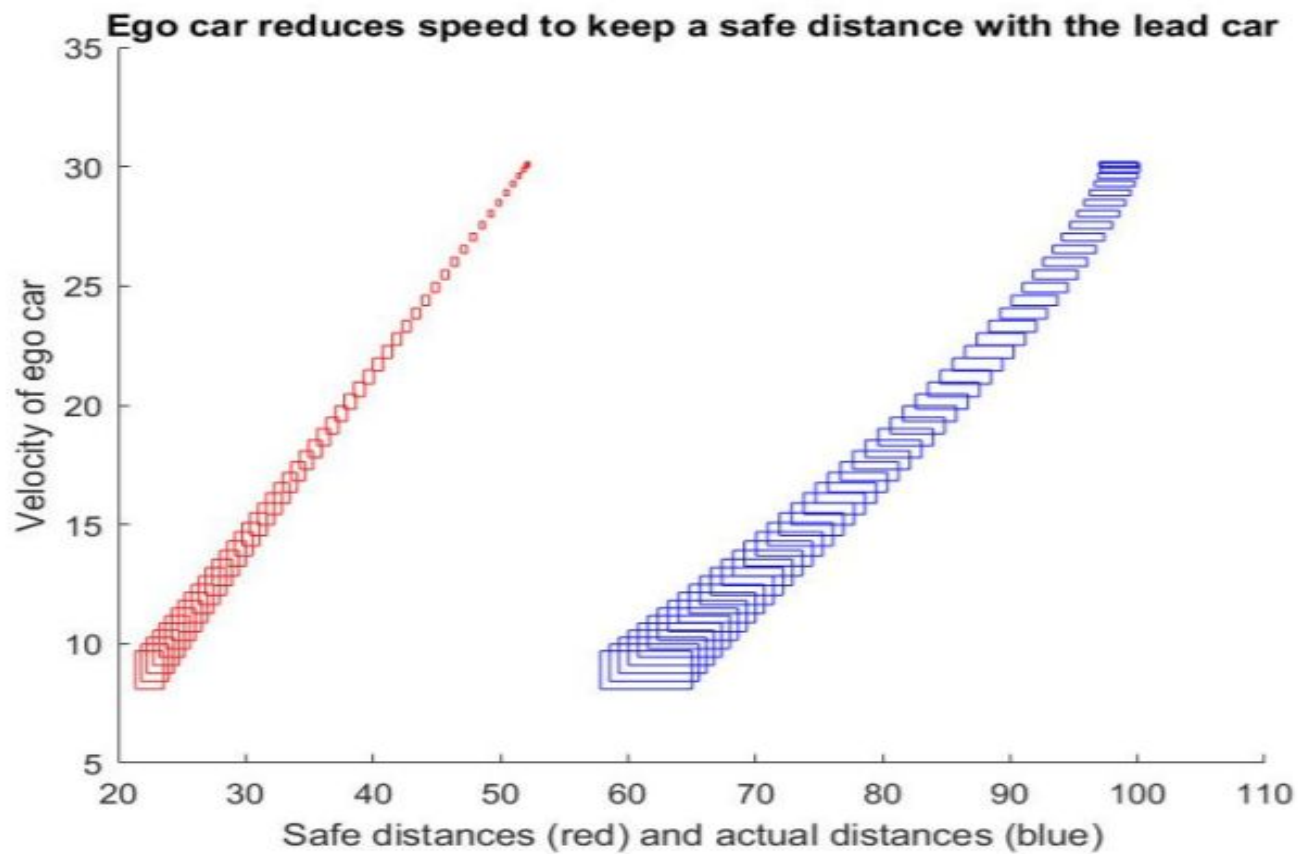
$$\dot{\gamma}_{\text{ego}}(t) = -2\gamma_{\text{ego}}(t) + a_{\text{ego}} - \mu v_{\text{ego}}^2(t)$$

where  $x_{\text{ego}}$  is the position,  $v_{\text{ego}}$  is the velocity,  $\gamma_{\text{ego}}$  is the acceleration of the car,  $a_{\text{ego}}$  is the acceleration control input applied to the car, and  $\mu = 0.0001$  is the friction control. **The lead car is at rest.**

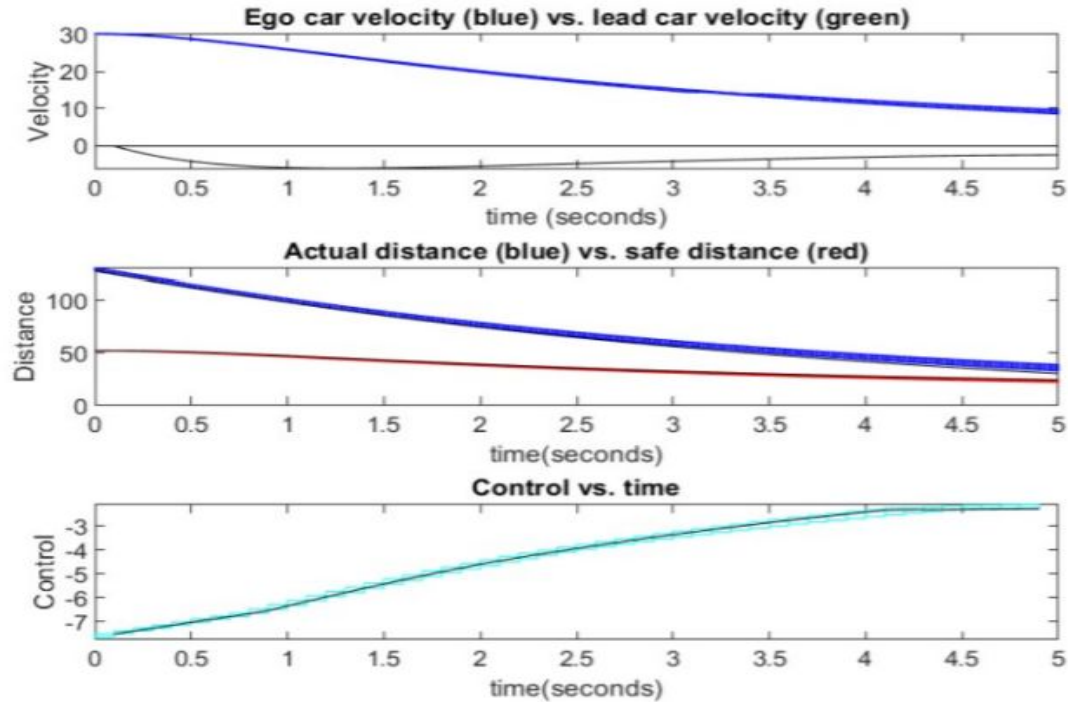
# Verification Result

**Table 3.2.** Verification Result for ffnns with 5 hidden layers of 20 neurons each

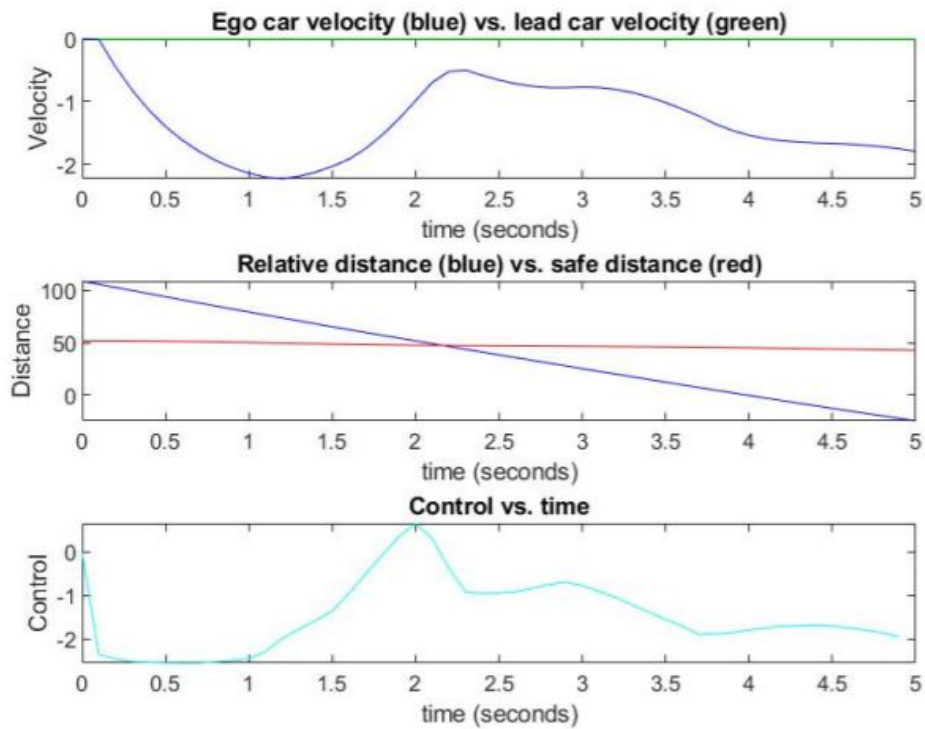
Distance lower bound	Distance upper bound	Safety Result	Verification Time
138	140	Safe	54.4681
136	138	Safe	64.1555
134	136	Safe	68.3523
132	134	Unsafe	76.8436
130	132	Unsafe	55.1441



# Reachability Analysis for $x_{\text{lead}} \in [138 \ 140]$



# Falsification Trace for $x_{\text{lead}} \in [118 \ 120]$





# Discrete Linear Plant with Discrete Controller Dynamics



In the above two scenarios we have considered the plant model to be non linear. In this case we consider cars with linear dynamics given as follows:

$$\dot{x}_{\text{lead}}(t) = v_{\text{lead}}(t)$$

$$\dot{v}_{\text{lead}}(t) = -0.5 * \gamma_{\text{lead}}(t)$$

$$\dot{x}_{\text{ego}}(t) = v_{\text{ego}}(t)$$

$$\dot{v}_{\text{ego}}(t) = -0.5 * \gamma_{\text{ego}}(t) + a_{\text{ego}}$$

where  $x_i$  is the position,  $v_i$  is the velocity,  $\gamma_i$  is the acceleration of the car,  $a_i$  is the acceleration control input applied to the car, and  $\mu = 0$  (for linear dynamics) where  $i \in \{\text{ego}, \text{lead}\}$ .

# Verification Result



**Table 3.3.** Verification Result for ffnnns with 5 hidden layers of 20 neurons each

Distance lower bound	Distance upper bound	Safety Result	Verification Time
53	55	Unsafe	36.8356
63	65	Unsafe	38.1781
73	75	Unsafe	21.0776
83	85	Unsafe	24.2650
93	95	Unsafe	30.4207
103	105	Unsafe	33.0474
113	115	Unsafe	28.8478
123	125	Unsafe	40.5837
133	135	Unsafe	28.8478
143	145	Unsafe	27.0596
153	155	Unsafe	40.1446

# Observations



⇒ The performance is poor for the chosen linear control dynamics as indicated by the verification result

⇒ The following verification times for ffns with 5 hidden layers of 20 neurons each

Average verification time for Discrete Linear NNCS = 31.54s approx

Average verification time for NNCS with nonlinear dynamics = 295.25s approx.

We observe that Nonlinear NNCS is **almost 10 times slower** than linear NNCS for ffns with **5 hidden layers of 20 neurons each**

# Different Car Dynamics



Verification analysis when car

dynamics for lead car and ego

are different for the case of

non-linear dynamics.

The car dynamics are described

as follows:

$$\dot{x}_{\text{lead}}(t) = v_{\text{lead}}(t)$$

$$\dot{v}_{\text{lead}}(t) = \gamma_{\text{lead}}(t)$$

$$\gamma_{\text{lead}}(t) = -2\gamma_{\text{lead}}(t) + a_{\text{lead}} - \mu v_{\text{lead}}^2(t)$$

$$\dot{x}_{\text{ego}}(t) = v_{\text{ego}}(t)$$

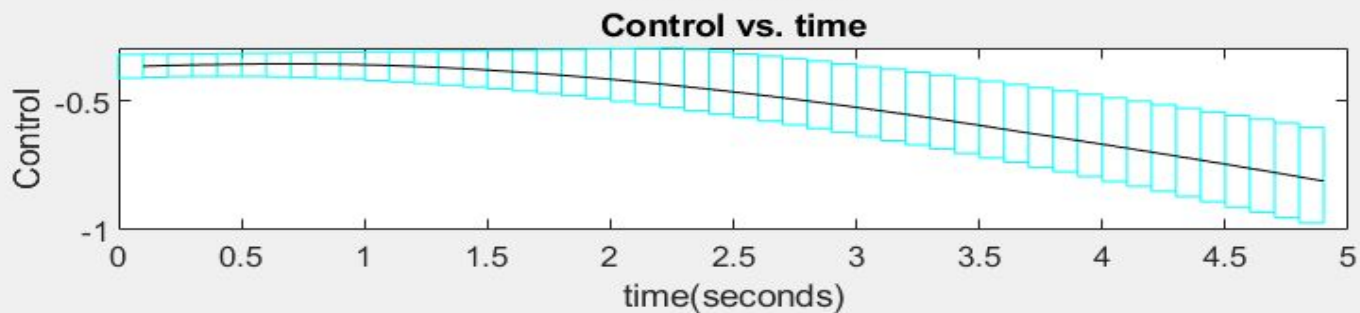
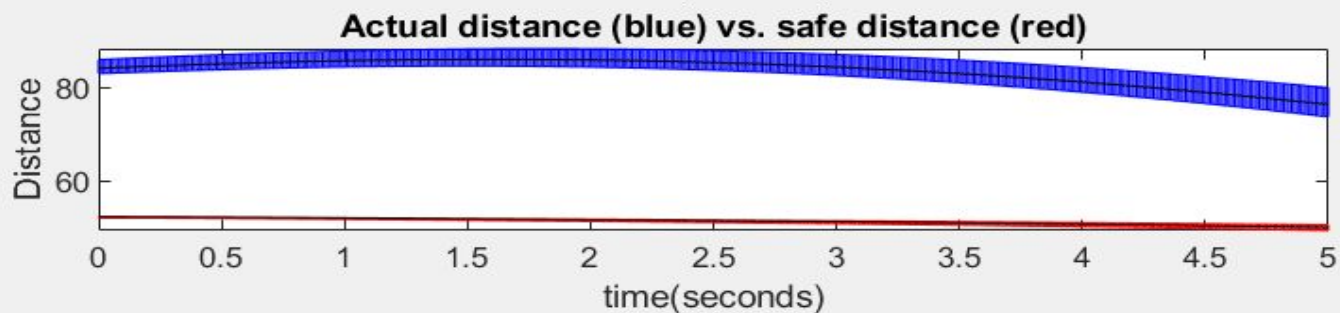
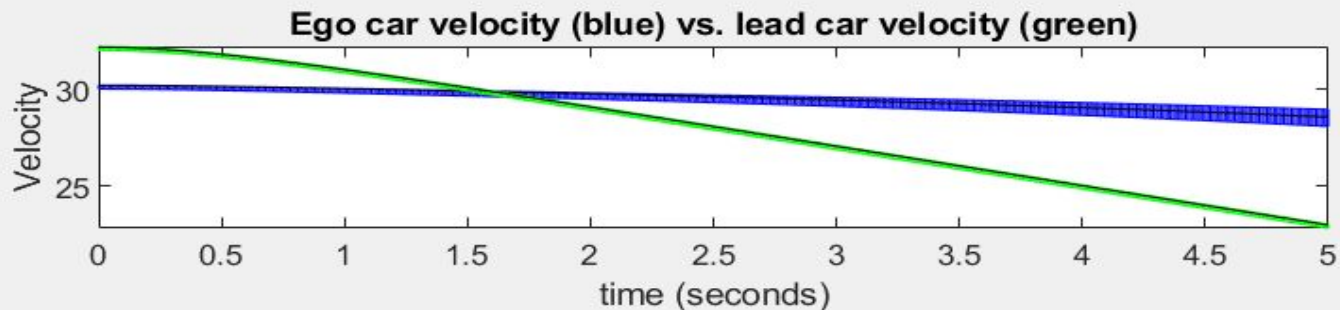
$$\dot{v}_{\text{ego}}(t) = \gamma_{\text{ego}}(t)$$

$$\gamma_{\text{ego}}(t) = -3\gamma_{\text{ego}}(t) + a_{\text{ego}} - \mu v_{\text{ego}}^2(t)$$

# Verification Result

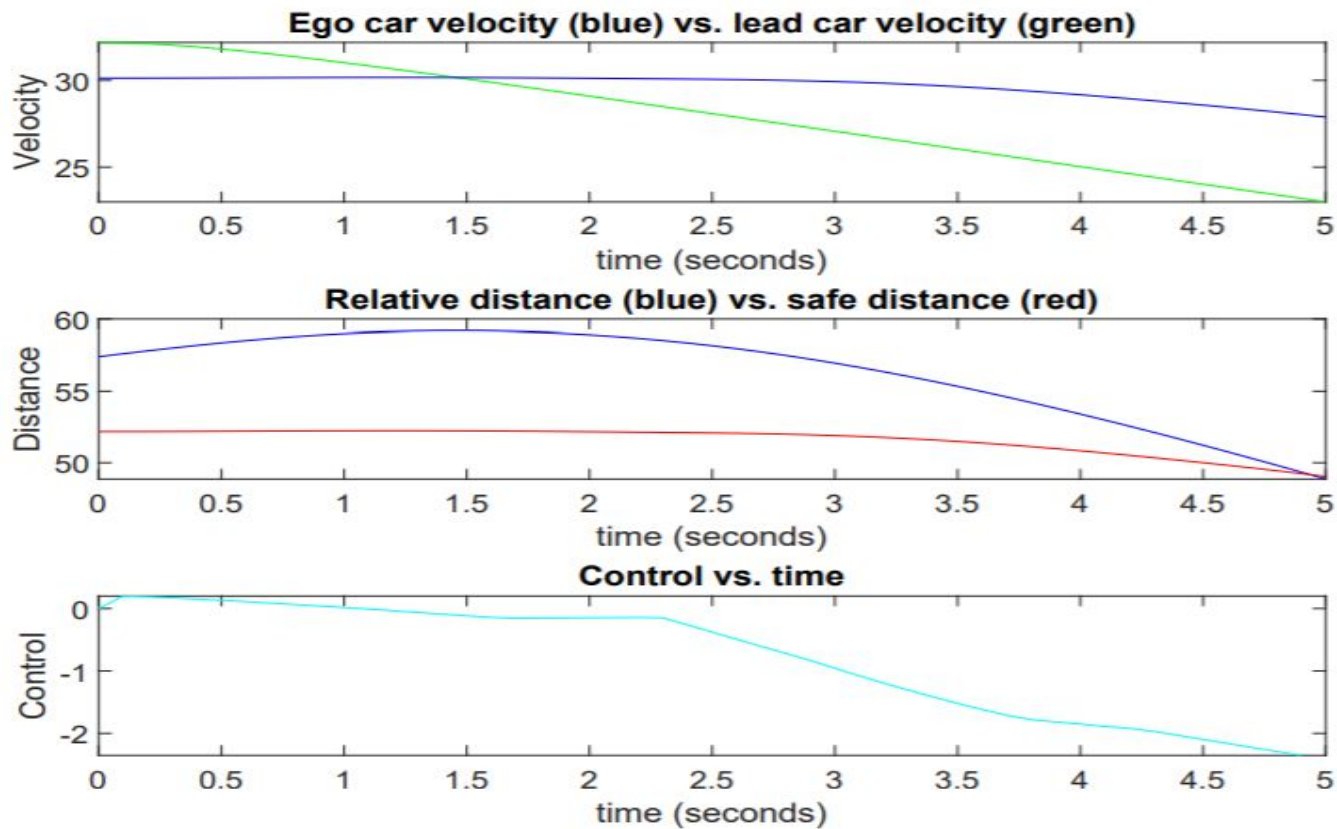
**Table 3.4.** Verification Result for ffnnns with 3 hidden layers of 20 neurons each

Distance lower bound	Distance upper bound	Safety Result	Verification Time
108	110	Safe	31.8169
106	108	Safe	32.6047
104	106	Safe	37.5159
102	104	Safe	43.5759
100	102	safe	42.0104
98	100	safe	38.8890
96	98	safe	40.2513
94	96	safe	41.8990
92	94	safe	52.4296



Falsification Plots for  $x_{\text{lead}} \in [65.0 \ 70.0]$ ,  $v_{\text{lead}} \in [32.0 \ 32.2]$ ,  $a_{\text{internal\_lead}} = 0$ ,

$x_{\text{lead}} \in [10 \ 11]$ ,  $v_{\text{lead}} \in [30 \ 30.2]$ ,  $a_{\text{internal\_lead}} = 0$



# Observations



- ⇒ Performance of the acc model improves due to the faster dynamics of the ego car used
- ⇒ avg verification time for both dynamics is approximately same
- ⇒ Control plots indicate that the control value improves in the case of faster dynamics for the ego car
- ⇒ Reachable States for the ego car in case of faster dynamics is much larger as compared to that in case of slower dynamics



# Performance of Different neural networks

**Table 3.4.** Verification Time for different NNVS

Distance	3 Hidden Layers	5 Hidden Layers	7 Hidden Layers	10 Hidden Layers
108-110	116.0203	292.6686	398.4133	1762.0
106-108	35.3254	288.8284	393.3460	2270.3
104-106	37.0537	302.3103	412.8096	2674.5
102-104	38.6174	292.9358	446.4725	2863.8
100-102	41.8711	294.8067	440.9385	2606.0
98-100	39.8067	302.7365	491.2920	2855.0
96-98	41.1680	289.8745	515.4267	3249.9
94-96	44.0950	301.9848	571.7489	3220.2

# Performance of Different neural networks



**Table 3.5.** Verification Result for different NNVS

Distance	3 Hidden Layers	5 Hidden Layers	7 Hidden Layers	10 Hidden Layers
108-110	1	1	1	1
106-108	1	1	1	1
104-106	1	1	1	1
102-104	1	1	1	1
100-102	1	1	1	1
98-100	1	1	1	0
96-98	1	1	0	0
94-96	1	1	0	0
92-94	1	1	0	0

# Observations



- ⇒ There is almost a 10 fold increase in verification time for addition of 2 hidden layers
- ⇒ Performance does not necessarily increase with increase in the number of hidden layers
- ⇒ Verification of system with complex dynamics may not be possible by approximation based reachability schemes because of the conservativeness of the over-approximation reachable sets. Hence it can be called a limitation of NNV
- ⇒ Since nnv toolbox performance reachability analysis layer by layer, more the number of layers , more will be the time complexity of the reachability analysis and therefore verification.

# Additional possibilities



(i) While Approximating value at each neuron(slide 13), we can interpolate it with concave curve by using some known numerical techniques . This will help us to get better approximation of values with less errors. So we propose a new idea:

**\*Lemma :** if we have a  $\lambda \in \mathbb{R}$ , such that  $\lambda$  satisfies the following conditions:

(a)  $0 < \lambda < |l_i u_i| / |u_i - l_i|$

(b)  $\lambda \geq (1 - u_i) / |u_i + l_i - 1|$

(c)  $\lambda > (2 |l_i - 2| + u_i) l_i / (u_i - l_i)(2l_i - 1)$

Then we can modify the approximation function such that it will give less error than the proposed method in slide 13 which is defined as

$$y_i = x_i \quad \text{if } l_i \geq 0$$

$$y_i = 0 \quad \text{if } u_i \leq 0$$

$$y_i = u_i(x_i - l_i)/(u_i - l_i) \quad \text{if } l_i < 0 \text{ and } u_i > 0$$

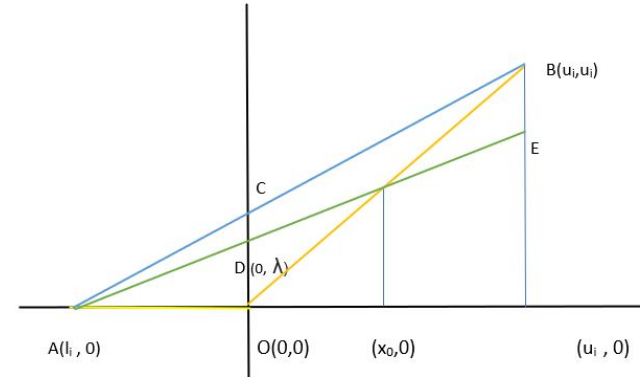
**Proof of Lemma:** Exact value function is AOB, approx value function is ACB.

Now, we will introduce a new function ADE having less error than ACB

$$\text{i.e } E_{ADE} < E_{ACB}$$

Choose a value  $\lambda$  such that  $OD < OC$  which will lead to condition

$$(a) 0 < \lambda < |l_i u_i| / |u_i - l_i|$$



**Exact** value function AOB meets our modified approx function ADE at point  $x_0 = l_i \lambda / (\lambda + l_i)$ .

As we can see from figure that  $E_{ADE} < E_{ACB}$  when  $x < x_0$

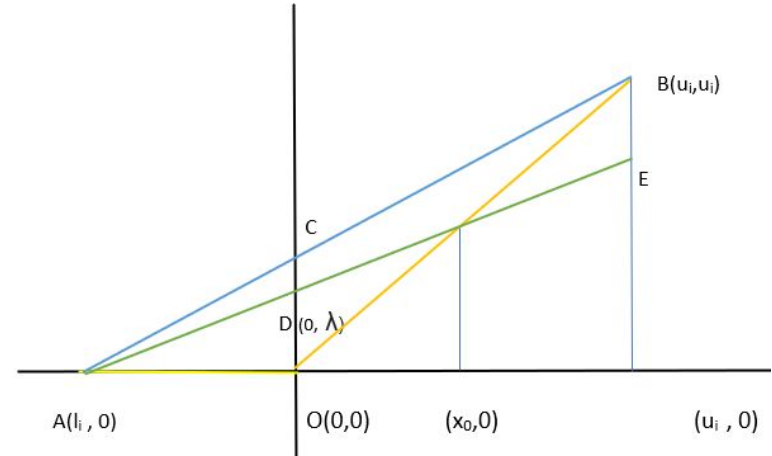
So we only have to prove  $E_{ADE} < E_{ACB}$  when  $x > x_0$

$$\begin{aligned} E_{ACB}(x > x_0) &= \\ &= \int_{x_0}^{u_i} \left( \frac{u_i(x_i - l_i)}{u_i - l_i} - x \right) dx \\ &= \frac{1}{2} \left( \frac{l_i}{u_i - l_i} - \frac{l_i + \lambda}{l_i} \right) (u_i^2 - x_0^2) \end{aligned}$$

$$\begin{aligned} E_{ADE}(x > x_0) &= \\ &= \int_{x_0}^{u_i} x - \left( \frac{-\lambda}{l_i} x + \lambda \right) dx \end{aligned}$$

=

$$\left( \frac{u_i l_i}{u_i - l_i} - \lambda \right) (u_i - x_0)$$



Now we had to get this condition given below ( $E_{ADE} < E_{ACB}$  when  $x > x_o$ )

$$\frac{1}{2} \left( \frac{l_i}{u_i - l_i} - \frac{l_i + \lambda}{l_i} \right) (u_i^2 - x_o^2) > \left( \frac{u_i l_i}{u_i - l_i} - \lambda \right) (u_i - x_o)$$

For simplification assume  $u_i + x_o \geq 1$  which will lead to condition (b)

And by solving remaining equation we will get condition (c)

Hence we can see that by synchronizing parameters we can decrease the error. May be in future we can

Come up with more cleaner conditions

# Bibliography



- <https://github.com/verivital/nnv/blob/master/docs/manual.pdf>
- <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7363192/>
- <https://github.com/ezapridou/carla-acc>
- Verification of Closed-loop Systems with Neural Network Controllers (Benchmark Proposal) Diego Manzananas Lopez, Patrick Musau, Hoang-Dung Tran, and Taylor T. Johnson
- <https://macsphere.mcmaster.ca/bitstream/11375/16611/2/thesis.pdf>
- [https://en.wikipedia.org/wiki/Adaptive\\_cruise\\_control](https://en.wikipedia.org/wiki/Adaptive_cruise_control)





Work done since the presentation on 13/04/2021

- (i) Modified the approximation function such that we can get less error the previous version.
- (ii) Compared performance of different neural networks using their verification time and verification result



# Work Distribution

Areeb Ahmad	(180135)	35%
Deepanker Mishra	(180225)	35%
Himanshu Singh	(180296)	30%