Augmenting Learning Components for Safety in Resource Constrained Autonomous Robots

Shreyas Ramakrishna*, Abhishek Dubey*, Matthew P Burruss*, Charles Hartsell*
Nagabhushan Mahadevan*, Saideep Nannapaneni†, Aron Laszka‡, Gabor Karsai*
*Vanderbilt University
†Wichita State University
‡University of Houston

Published in the proceedings of the 22nd IEEE International Symposium on Real-Time Computing (ISORC 2019).

***Abstract*—Learning enabled components (LECs) trained using data-driven algorithms are increasingly being used in autonomous robots commonly found in factories, hospitals, and educational laboratories. However, these LECs do not provide any safety guarantees, and testing them is challenging. In this paper, we introduce a framework that performs weighted simplex strategy based supervised safety control, resource management and confidence estimation of autonomous robots. Specifically, we describe two weighted simplex strategies: (a) simple weighted simplex strategy (SW-Simplex) that computes a weighted controller output by comparing the decisions between a safety supervisor and an LEC, and (b) a context-sensitive weighted simplex strategy (CSW-Simplex) that computes a context-aware weighted controller output. We use reinforcement learning to learn the contextual weights. We also introduce a system monitor that uses the current state information and a Bayesian network model learned from past data to estimate the probability of the robotic system staying in the safe working region. To aid resource constrained robots in performing complex computations of these weighted simplex strategies, we describe a resource manager that offloads tasks to an available fog nodes. The paper also describes a hardware testbed called DeepNNCar, which is a low cost resource-constrained RC car, built to perform autonomous driving. Using the hardware, we show that both SW-Simplex and CSW-Simplex have 40% and 60% fewer safety violations, while demonstrating higher optimized speed during indoor driving ($\sim 0.40\,m/s$) than the original system (using only LECs).**

***Index Terms*—Autonomous Robots, LEC, Convolutional Neural Networks, Simplex Architecture, Reinforcement Learning.**

I. Introduction

Autonomous systems are ubiquitously being used in transportation (self-driving cars [1], [2], buses), manufacturing (robotic arms, service robots), agriculture, social care, and search-and-rescue disaster management for their ability to accomplish tasks independently or with minimal human supervision. Techniques for developing autonomous systems include human encoded control and reinforcement learning. Reinforcement learning [3] is a powerful data-driven strategy in which the learning occurs in a closed loop agent-environment interactions whereas the other techniques require human involvement. In the presence of huge amounts of training data, some autonomous systems have proven to surpass human experts in performance, for example, Alpha Go Zero [4]. End-to-End (e2e) learning [5] is a key framework for realizing autonomy in robots, which makes use of deep learning models. For example, NVIDIA's DAVE-II [1] and ALVINN [2] use Convolutional Neural Networks (CNN) to design the controller for autonomous cars. A combination of reinforcement learning and deep learning approaches provide a framework to transition from model-based system components to data-driven Learning-Enabled Components (LECs).

While the use of data-driven LECs provides a paradigm shift in the ability to create adaptive systems, it also presents challenges in testing and assurance. For example, there are no established analogues to path coverage-based testing mechanisms for components designed with neural networks. There has been ongoing research in designing tools for automated testing [6], [7] of Deep Neural Network driven systems; however, they are limited by the exhaustive test case scenarios they support, and hence, may not be able to detect all the edge cases. In addition, existing verification tools [8] can only handle some types of activation functions, and Neural Network of limited complexity.

The key challenges in establishing confidence in data-driven LEC systems are: (1) Operating in unknown contexts [9] e.g., search-and-rescue robots, and (2) the limited availability of training data which reduces the confidence in the trained LECs.

Safety-critical Cyber Physical Systems (CPS) like aircraft (Boeing 777 [10]), unmanned aerial vehicles (UAV) [11], and mission critical ground rovers [12] are augmented with Simplex Architectures [10] to increase system assurance. This architectural pattern allows the integration of safety supervisors to aid the control decisions of the high performance unverified controller.

However, Simplex Architectures do not provide a method to combine two unverified controllers. Applying a simplex strategy in such a scenario may not improve the systems safety (for such scenarios we introduce weighted simplex strategies). Additionally, it does not consider the different operational modes and contexts of the working environment in performing the arbitration, which could be crucial for the systems performance. Biswas, Gautam, et al. [13] have shown that mode detection is a crucial problem and data-driven anomaly detection methods should be context sensitive. They also do not provide any confidence metric that can be used to evaluate the decisions of the LEC if safety violations

Symbol	Description
S_L	Steering PWM value of DeepNNCar using LEC
S_S	Steering PWM value of DeepNNCar using Safety Supervisor
S_{SA}	Steering PWM value using Weighted Simplex Strategy
T_R	Inference pipeline time of DeepNNCar using CSW-Simplex
V	Current speed of DeepNNCar
V_{MAX}	Max Saturated speed during task offload
V_{SET}	Set speed computed by CSW-Simplex
W_L	Ensemble weight given to LEC.
W_S	Ensemble weight given to Safety Supervisor.
W_{SET}	Ensemble weights $\{W_L, W_S\}$ computed by CSW-Simplex
T_{SW}	Preset Threshold used by SW-Simplex
STOP	Command from Safety Supervisor during safety violations
$\hat{M}$	Estimated state of the track segment
$\hat{t}$	Deviation of car from the track center
I	Image captured by the front facing camera

TABLE I: List of Symbols

occur. Such diagnostic capabilities are crucial in safety-critical systems. We seek to address the following research questions:

1) Can we use an online simplex supervisor that can learn from past actions (experience) and augment the control actions taken by the LEC to improve safety?
2) Can we provide a confidence metric about the safety of current actions at system level in real-time, given all the past actions?

Our Contributions: To address the above questions, we describe: (**1**) an adaptive framework that allows the integration of safety supervisors and weighted simplex strategies, and performs active switching between them based on the performance of the system; (**2**) implement and evaluate two weighted simplex strategies that allow us to encode domain knowledge (e.g. the operating environment or actions to be taken in particular operating conditions). Using these strategies we show an improvement in the safety guarantees and performance of the system; (**3**) implement a system monitor which uses the current state information and a Bayesian network model to estimate a probability of the robotic system remaining in the safe working region; and (**4**) design a resource management and task offloading strategy to compensate for the increased computations of the weighted simplex strategies.

Outline: Section II describes the test environment, the controllers, and safety algorithms employed by the system. Section III introduces different weighted simplex strategies. Section IV describes a monitor that computes the probability of the robotic system to remain in the safe working region. Section V illustrates resource management and system integration, and Section VI evaluates the weighted simplex strategies and resource management. Section VII reviews related work, and finally, Section VIII presents our conclusion. The symbols used in the paper are described in Table I.

II. DeepNNCar: Testbed for Autonomous Driving

DeepNNCar[1] (in Figure 1) is built upon the chassis of Traxxas Slash 2WD 1/10 Scale RC car. The RC car has two

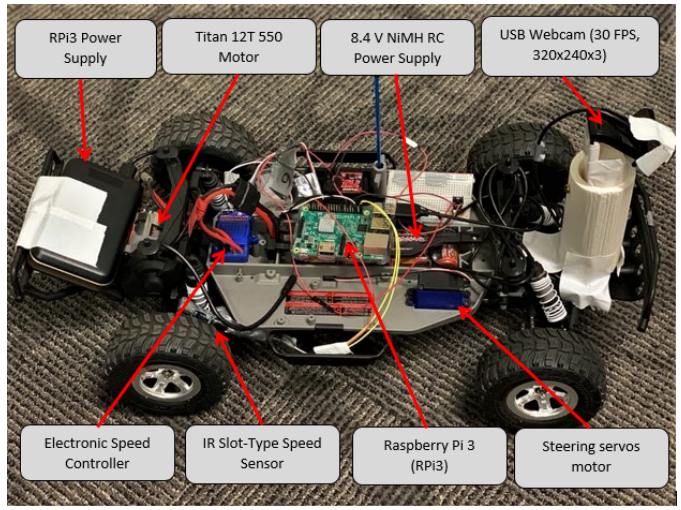


Fig. 1: DeepNNCar, a resource constrained autonomous robot used in our experiments.

on-board motors, a servomotor for steering control, and a Titan 12T 550 motor for motive force, which are powered by a 8.4volts NiMH battery. Raspberry Pi 3 (RPi3) is the onboard computing unit which performs all the required computations and interfaces with the sensors. RPi3 reserves two GPIO pins to generate Pulse Width Modulation (PWM) signals that are used to control the motors of the car. For the servomotor, a duty cycle range of (10, 20) corresponds to a continuous steering angle of (-30°, 30°), and for the Titan 12T 550 motor, we operate within the PWM range of (15, 15.8), which corresponds to a vehicle speed range of (0, 1) m/s.

A. Sensors

A USB webcam is attached to the RPi3 to capture images at 30 FPS with a resolution of $320 \times 240 \times 3$ (320x240 RGB pixels). During autonomous driving, these images are used by the onboard controllers to compute desired steering angle. A slot-type IR opto-coupler speed sensor[2] is attached to the chassis near the rear wheel and counts revolutions of the wheel. The speed of the car is calculated based on the frequency of revolutions and is used to estimate the 2D relative position of the car (shown in Figure 2). During data collection camera images, vehicle speed, and steering angle are stored on a USB drive.

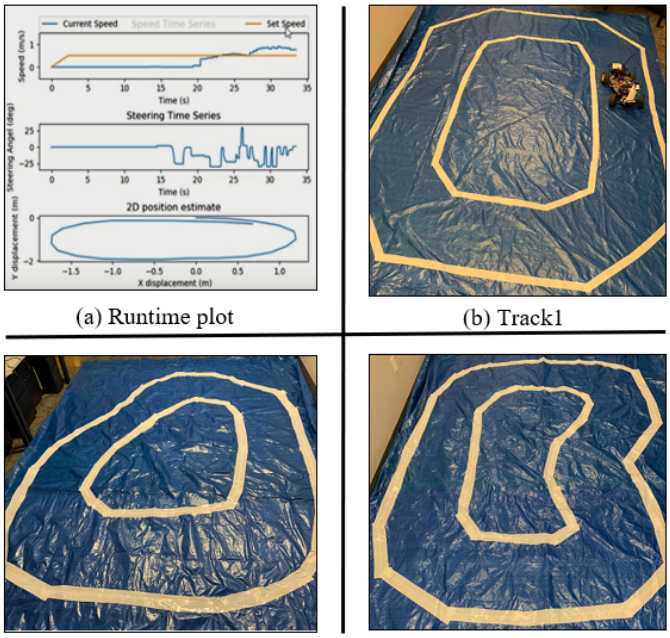
B. LEC in DeepNNCar

End-to-End learning is a perception based control approach that uses supervised learning to directly compute the control action. Is widely used because of its conceptual simplicity and computationally efficient approach. The e2e learning approach in fully autonomous cars was first demonstrated by ALVINN [2] in 1989, and was recently extended by NVIDIA through their self-driving car, DAVE-II [1].

In the current implementation, the hardware uses e2e learning which implements a modified version of DAVE-II to predict steering (S_L). The original DAVE-II CNN [1] takes

[1]Build instructions, source code, datasets, bill of materials, and videos of DeepNNCar can be found at: https://github.com/scope-lab-vu/deep-nn-car

[2]We refer to this sensor as opto-coupler in the rest of the paper



(a) Runtime plot (b) Track1

(c) Track2 (d) Track3

Fig. 2: (a) Runtime plot: shows runtime speed, steering, and position on track of DeepNNCar as displayed on a fog node, (b) Track1: on which different weighted simplex strategies were trained and tested; (c) and (d) Other tracks: used to test the trained controllers, and strategies. The tracks were built indoor in our laboratory using 10' x 12' blue tarps.

an image (I) as input and predicts S_L as the output without considering the impact of speed (V) on S_L.

Modified DAVE-II CNN has five convolutional layers and seven fully connected layers. It takes an image (I) with resolution (66x200 RGB pixels) and vehicle speed (V) as inputs and predicts S_L as the output. The model is trained with 6000 images collected from Track1 and Track2 (see Figure 2).

The modification of the CNN is required for two reasons. First, the steering and speed are coupled, thus any change in the speed will impact the steering performance. We also observed that the modified CNN takes wider trajectories at turns compared to the original one. Second, since the quality of the captured image deteriorates as speed increases, additional information is required for the CNN to predict correct steering values.

C. Safety Supervisor

The Safety Supervisor (SS) is designed using classical image processing algorithms. It performs lane detection (LD). The LD algorithm is implemented in OpenCV and provides labeling information (straight, right, left, or out of track) for the track segment ($\hat{M}$) in which the car resides. The LD algorithm was tested using a dataset of 3000 images and it correctly labeled the track segments with an accuracy of 89.6%.

Lane Detection: The LD algorithm performs the following operations sequentially on a 200x66 gray scale image.

- Gaussian blur and white masking: A 3x3 Gaussian kernel is convolved across the image to reduce the noise. Next, all pixels except those within a specified range (e.g., [215, 255]) are masked, thus differentiating the track lanes from the foreground.
- Canny edge detection: The algorithm first computes a gradient of pixel intensities. An upper and lower threshold of these gradients is defined at compile time. A comparison of the pixel gradients to these thresholds in addition to hysteresis (suppress all weak and unconnected edges) can determine if a pixel is an edge or not. The edges reveal the boundary of the lanes.
- Region of interest (ROI) selection: The image is divided into two similar 30x66 regions of interest to capture the left and right lane respectively.
- Hough line transform: A Hough line transform is applied to each ROI to detect the existence of a lane based on the results of the canny edge detection algorithm. Using this information we determine a label for the track segment.

For every estimate of the track segment ($\hat{M}$) we associate a discrete steering S_S: if LD detects the car in the straight segment, $S_S = 0°$; if LD detects the car in the left segment, $S_S = -30°$; and if LD detects the car in the right segment, $S_S = 30°$.

Speed control in LEC and SS is initially set by a human supervisor, it can be varied or controlled using either the constant throttle mode or the PID controller provided by the DeepNNCar.

Goals of DeepNNCar: (1) Minimize the soft safety violations, i.e. the number of times the car crosses the track boundary (safety requirement), and (2) optimize the speed. Below we discuss the simplex strategies which uses the LEC and SS, to achieve the two goals.

III. Weighted Simplex Strategies for DeepNNCar

Simplex architectures [10] have been used before to ensure the safe operation of a high performance but unverified controller. It works by integrating a high assurance controller in the system, which activates the high assurance (SS) controller whenever the high performance controller is on the verge of jeopardizing the safety of the system.

However, as shown by our experiments (see Figure 3) the SS does not always perform in a high assurance manner. The SS controller is safer in curved regions while the LEC is safer in the straight region. It is hard to design a single controller that is safe across all modes of the track. In such scenarios applying the classical Simplex Architectures may not guarantee safety. However, it might be possible to improve the safe operation if we take an ensemble approach [14] and utilize the weighted output from the two controllers. We call this approach as the "Weighted Simplex Strategy". The weighted controller output is shown in Equation 1.

$$S_{SA} = W_L \times S_L + W_S \times S_S \tag{1}$$

In order to optimize for the speed (V) along with the steering, we update the current speed of the system. The V can be incremented or decremented by (δV) based on the systems current state, and position on the track.

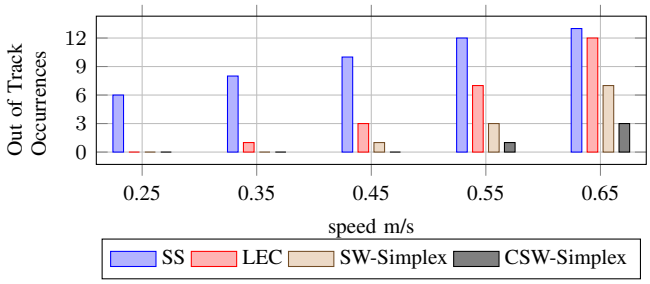


Fig. 3: The DeepNNCar performs fewer safety violations when combined with the CSW-Simplex strategy. (a) SS: driving only with the safety supervisor, (b) LEC: driving only with the modified Dave-II model, (c) SW-Simplex, and (d) CSW-Simplex. The horizontal axis shows the different speeds of the car during the experiment.

$$V = V \pm \delta V \tag{2}$$

Applying the weighted simplex strategy to our system could (1) improves the safety of the system, while optimizing for speed (see Figure 3), and (2) allows us to integrate context-sensitive weights to compute the systems output. Biswas et al [13] have addressed the importance of mode and context-sensitive information in data-driven anomaly detection methods. Also, context-aware machine learning approaches have been found effective in different applications of face recognition [15], speech recognition, and query classification [16].

With the goal of finding context-sensitive weights and optimal speed, we describe two different weighted simplex strategies: (1) simple weighted simplex, and (2) context-sensitive weighted simplex.

A. Simple Weighted Simplex Strategy (SW-Simplex)

To integrate the concept of weighted sum into our system we extend the concept introduced by Fridman et. al. called the "Arguing Machines" [17] . In our approach, we use the LEC and SS as the two controllers, and if the difference between their predicted steering values is higher than a predefined threshold (T_{SW}), then the steering is computed by Equation 1, with the weights being W_L=0.8, W_S=0.2 (shown in Equation 3). However, if the differences between the predicted steering values are lower than the T_{SW}, then the LEC action is chosen to drive the system.

$$S_{SA} = 0.8 \times S_L + 0.2 \times S_S \tag{3}$$

The speed of the system speed (V) is also computed based on the argument between the controllers. If there is a disagreement among the controllers predictions, then V is decremented using Equation 2. However, if there is no disagreement, then V is incremented. Disagreement among the controllers is an important factor to vary the speed, as it indicate if the two controllers are in agreement.

The chosen ensemble weights (W_L, W_S), threshold (T_{SW}), and change in speed (δV) are track specific and were found through trial and error experimental runs on Track1 (in Figure 2). The fixed weights are trying to capture the performance of the controller for the specific shape of track. To start, a reasonable number was chosen for each parameter and was later tuned according to the speed and safety performance on the track. The tuning of weights was stopped when the safety violations did not reduce any further, and this was chosen as the optimal weights.

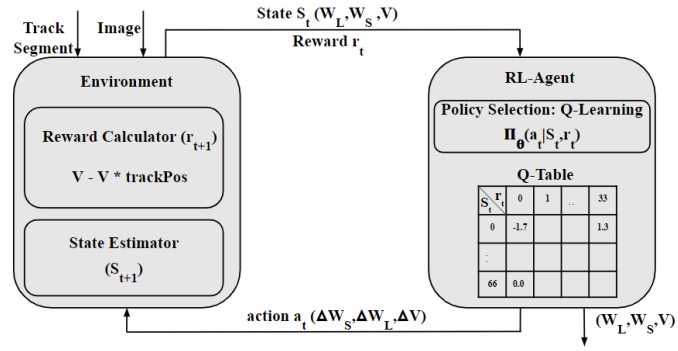


Fig. 4: Agent-Environment interactions in the RL-Actor (Figure 9) of DeepNNCar.

B. Context-Sensitive Weighted Simplex Strategy (CSW-Simplex)

The SW-Simplex used a weight tuning method based on trial and error experimental runs. After the tuning, the optimal weights were fixed, even though it captured the controller's performance specific to the Track1(in Figure 2). However, if we could involve some more context information like track segments (straight, left, right, and out), then the weights could probably be dynamically adjusted. In order to find optimal weights (W_L, W_S) and speed (V), that involves contextual information, we use Reinforcement Learning (RL). In this work, we use the Q-learning [18] approach to compute the optimal simplex weights and speed. The RL setup for our problem is shown in Figure 4.

The **RL-Actor** involves different components of DeepNNCar which are responsible for the Q-learning process. Figure 4 shows the interactions between the RL-Agent, and the Environment.

The **Environment** component within the RL-Actor tracks and estimates the state (S) of the system based on an internal estimate of the Markov Decision Process (MDP) and computes a reward (r) for an action (a).

S represents the current state of the RL-Agent in the environment. These states continuously change as the agent interacts with the environment, and the state information is used by the agent to continuously learn the optimal action. The weights (W_L, W_S) and speed (V) are encoded as the state information S(W_L, W_S, V). It is important to have discretized states to build the MDP and perform the necessary action. For each of the state variables $W_L, W_S \in (0, 1)$ and $V \in (15.58, 15.62)(PWM)$, we define a set of equidistant points of separation $\delta W_L, \delta W_S = 0.05$ and $\delta V = 0.01$ to get a vector of ensemble weights containing 21 elements and a vector of V containing 41 elements.

Reward: The environment also computes a reward for the previous action performed by the agent. The reward in Equation 4 is formulated to account for V and $\hat{M}$:

$$r(s_t, a_t) = V_t - V_t \cdot \hat{t} \tag{4}$$

action space	$\uparrow V$ *by* 0.01	$\downarrow V$ *by* 0.01	NOP
$\uparrow W_L$ *by* 0.05	(0.55,0.45,15.86)	(0.55,0.45,15.84)	(0.55,0.45,15.85)
$\downarrow W_L$ *by* 0.05	(0.45,0.55,15.86)	(0.45,0.55,15.84)	(0.45,0.55,15.85)
NOP	(0.50,0.50,15.86)	(0.50,0.50,15.84)	(0.50,0.50,15.85)

TABLE II: Action space for a given state ($W_L = 0.5$, $W_S = 0.5$, $V = 15.85$). Similar action combinations are generated for other states. NOP:means no operation

where V_t is the speed of the car in the current state, and $\hat{t}$ is a scalar quantity computed based on the deviation of the car from the center of the track. The measure $\hat{t}$ is decided by the LD algorithm of the SS: if the algorithm detects both lanes of the track in the captured image, then the car is at the center of the track ($\hat{t} = 0$); if the algorithm detects only one lane, then the car has deviated from the center ($\hat{t} = 1/2$); and if it detects no lanes then the car is out of track ($\hat{t} = 10$). The system may initially stray away from the center of the track or choose to remain at low speeds; however as it learns to optimize its speed while also trying to keep the center of the track in an effort to receive the highest reward.

RL-Agent selects optimal actions for the state information provided by the environment. For each state $s \in S$, the agent performs an action $a \in A$, which results in a reward, $r : S \times A \rightarrow \mathbb{R}$, as the agent transitions to state $s' \in S$. The possible action space for a state with ($W_L = 0.5$, $W_S = 0.5$, $V = 15.85$) is shown in Table II, a similar action space is created for all the different combinations of (W_L, W_S, V). Thus, there are 9 possible actions that can be performed from any state.

For each state-action pair (s, a), the agent learns the "quality" $Q(s, a)$ of taking action a in state s. The Q value is updated using the Bellman equation [18], which takes the current state and action as inputs along with the parameters learning rate $\alpha \in [0, 1]$, and discount factor $\gamma \in [0, 1]$. These parameters control the amount of learning done by the RL-Agent. For our experiments we used $\alpha = 0.1$, and discount factor $\gamma = 0.4$, and number of training steps = 1000 obtained by tuning through various episodes.

During training (exploration), the computed Q values for each state-action pair is stored in a lookup table called the Q-Table. During testing, the RL-Agent uses the Q-Table to select actions at each state.

IV. System Level Confidence Estimation

For autonomous systems, it is necessary to develop a mechanism that can monitor the system operation and provide a level of confidence as to how safe the system will be in different operating scenarios. This is difficult as it requires an effective knowledge of the distribution of the environment in which the system operates. In a limited setting where the systems operation and the environment modes can be characterized, we can use a Bayesian network to learn the probability distribution. We then can use this distribution to estimate the probability that the system will remain safe given a particular control action. The evidence for the confidence increases over time as we collect more data from safe operation trajectories.

In case of the DeepNNCar, we use a Bayesian Network model (Figure 5) to estimate the probability that the car will remain on track, given its current state and control actions. Data collected during training and evaluation, is used to build a model to estimate the current position of the car, using the output of the opto-coupler and the steering commands issued to the car. The data is also used to identify safe-turn regions (in different segments of the track) and the ranges for commands that keep the car within the track (at different speeds of operation). The nodes *current-position*, *current-velocity* and *current-steering* capture the current state of the car. The node *SafeTurnRegion* captures the likelihood of the car being in the safe-turn region when a control-cycle update is triggered. *SafeTurnRegion* and the steering command *CmdSteeringOnTurn* issued when the car is in the safe turn region influence the likelihood of the car remaining *InTrack*.

The *current-position* node corresponds to the car distance from the next safe turn region. The discrete states based on the distance include *On*, *Near* and, *Far*. The discrete states for *current-velocity* are *Slow* (less than 0.25 m/s), *Medium* (between 0.25 and 0.6 m/s) and *Fast* (greater than 0.6 m/s). The *current-steering* node corresponds to the current steering angle of the car relative to its desired direction and includes the states *left* (less than -10°), *straight* (between -10°and +10°) and *right* (greater than 10°). The node *CmdSteeringOnTurn* corresponds to the steering command issued when the car is on the safe turn region and includes the states (*Left*, *Right* and *Straight*) which are based on the range of the steering command values. The node *InTrack* indicates if the car will remain on track when the turn is executed. This node include two states *yes* and *no*.

The priors and the conditional probability tables have been filled based on our understanding of the system during experimentation. The bar graph in each of the root nodes in Figure 5 shows the node probabilities based on logical inferences using the priors and conditional probabilities. Tables III and IV capture the conditional probability tables for the nodes *SafeTurnRegion* and *InTrack* respectively.

The prior probabilities on the *Current-Position* node shows that there is an equal chance of the car being in the three position states. With regards to *Current-Steering*, the *Left* state has a low probability due to the shape of the track and the nature of the mission. The *Left* state is observed only when there is an error or there is a course correction for an error. The *Current-Velocity* has been observed to be in the medium range most of the time giving the *Medium* state a higher prior probability. The lower velocity states are observed in the beginning, while the fast speeds are not common. Given the shape of the track, the steering command during turns is mostly right as seen in the priors for *CmdSteeringOnTurn*.

Based on the priors for the root (observation) nodes and the likelihoods captured in the conditional probability tables (Tables III, IV), the priors for the assurance nodes (*SafeTurnRegion* and *InTrack*) can be inferred. The prior probability of being on track (0.7) reflects our experimental evidence with CSW-Simplex architecture. The Bayesian network model was

TABLE III: Conditional Probability Table for *SafeTurnRegion* node.

Current Position	Near									On									Far								
Current Velocity	Slow			Medium			Fast			Slow			Medium			Fast			Slow			Medium			Fast		
Current Steering*	S	L	R	S	L	R	S	L	R	S	L	R	S	L	R	S	L	R	S	L	R	S	L	R	S	L	R
SafeTurnRegion =Yes	1	0.8	0.8	0.9	0.6	0.6	0.2	0.1	0.1	1	1	1	1	1	1	1	1	1	0.9	0.9	0.9	0.8	0.7	0.7	0.5	0.2	0.2
SafeTurnRegion =No	0	0.2	0.2	0.1	0.4	0.4	0.8	0.9	0.9	0	0	0	0	0	0	0	0	0	0.1	0.1	0.1	0.2	0.3	0.3	0.5	0.8	0.8

* - Current Steering states : S=Straight, L=Left, R=Right.

TABLE IV: Conditional Probability Table for *InTrack* node

SafeTurnRegion	Yes									No								
CmdSteering	Left			Straight			Right			Left			Straight			Right		
Velocity	Slow	Med.	Fast	Slow	Med.	Fast	Slow	Med.	Fast	Slow	Med.	Fast	Slow	Med.	Fast	Slow	Med.	Fast
InTrack=Y	0.6	0.2	0	0.7	0.5	0	1	0.9	0	0.2	0.1	0	0.3	0.2	0	0.5	0.4	0
InTrack=N	0.4	0.8	1	0.3	0.5	1	0	0.1	1	0.8	0.9	1	0.7	0.8	1	0.5	0.6	1

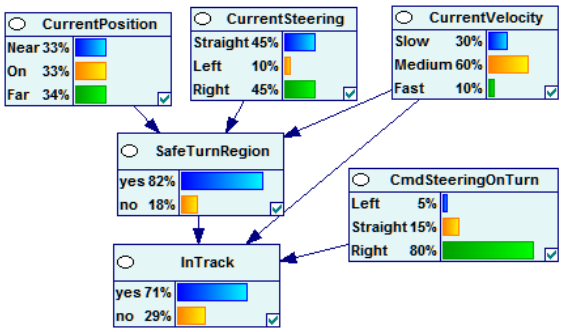


Fig. 5: Bayesian Network model for Safety Assurance

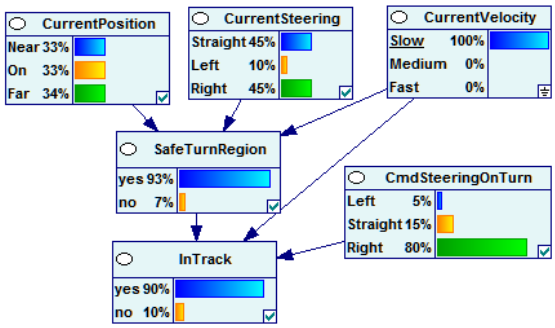


Fig. 6: Bayesian Inference when *current-velocity* is set to *Slow*

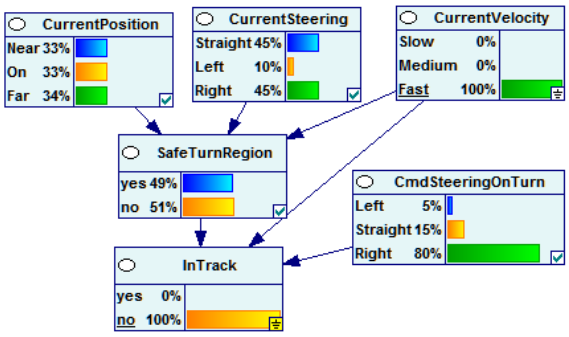


Fig. 7: Bayesian Inference when *current-velocity* is set to *Fast*

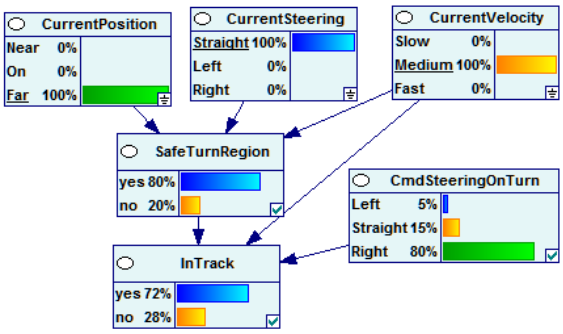


Fig. 8: Bayesian Inference when *current-velocity* is set to *Medium*, *current-steering* is set to *Straight* and *current-position* is set to *Far*

used to compute the confidence metric (the probability of car being on the track and the probability of car being in the turn region to execute control action) under different situations. This was done by setting the evidence on the root (observation) nodes and executing the Bayesian inference engine to compute the posterior probabilities.

The model predicts that when the *current-velocity* is set to *Slow*, there is a high probability of the car being in the safe turn region and remaining on the track (Figure 6). Alternately, figure 7, shows that when the speed is set to *Fast*, the chance of being in the safe turn region to execute a control action is greatly reduced and there is no chance of remaining on the track. Figure 8 shows that when the *current-velocity* is *Medium*, *current-steering* is *Straight* and the *current-position* is *Far* from the safe turn region, there is a good chance of the car executing a control action in the safe turn region (80%) and remaining on track (77%). The results of the Bayesian Network agrees with our experimental observations.

V. Resource Management and System Integration

In this section, we discuss the integration of fog computing for runtime task offloading. We also integrate different system components and discuss their functioning for efficient operations.

A. Managing Resource Constraints

The complex computations of simplex strategies increase the workload on the resource constrained RPi3. This workload increases the power consumption, CPU utilization and the temperature of RPi3 beyond 70°C (configured soft limit). Beyond the soft limit, the clock speed and the operating

voltage of RPi3 are reduced [19]. To address the increasing temperature and CPU utilization problem, we add multiple computing devices on-board the DeepNNCar, and distribute the tasks among them. However, this approach requires additional external power sources, which increases development cost of the platform. Alternatively, as the RPi3 supports WiFi connectivity, we setup wireless communication with other fog or edge nodes and offload some tasks. We use the second approach to keep the development costs low, and utilize the wireless communication capability that enables fog computing. We offload only non-critical tasks (like RL-Actor, which only has to access the Q-Table and select actions). No critical components (Decision Manager) will be offloaded.

To manage the offload challenges, the middleware framework has a Resource Manager (RM) that performs the following tasks: (1) continuous monitoring of resource state (temperature and CPU Utilization), (2) selection of an optimal fog node for task deployment, and (3) adjusting the vehicle speed (V) according to variations in the inference pipeline times. The RM continuously monitors the temperature and CPU utilization of the on-board computer and may offload one or more tasks to the other fog nodes if necessary.

In the background, the RM continuously performs a latency test every 30 seconds using the iPerf [20] networking tool to select a fog node with lowest latency (as it increases the inference time T_R). If the temperature exceeds 70°C, the RM stops the tasks on RPi3 and seamlessly connects to identical tasks running on the selected fog nodes using ZeroMQ (ZMQ) [21]. Once the temperature has fallen below the threshold (70°C), the RM reactivates the tasks on the RPi3.

Task offload will keep the increasing temperature in check; however, the latency overhead of the wireless communication increases the inference pipeline time T_R (discussed in section V-B) of the system. To compensate for the increased time, the RM instructs the DM to saturate (limit) the top speed of the car. The saturated maximum speed V_{MAX} is calculated using the safe distance (d_S) which is the closest distance to the track turns at which the car will have to take a decision to avoid going off the track. The d_S is a track specific quantity which was found to be 0.09m for our track (see Figure 2). Therefore, any decision taken before reaching this distance will give the car a good turning radius. However, any decision taken after this distance will leave the car with insufficient space to turn which will result in a safety violation. V_{MAX} is computed as: $V_{MAX} = \frac{d_S}{T_R}$, where T_R is the inference pipeline time. During task offloads, the DMA has to wait longer for a reply from the offloaded component due to the latency overhead, which increases the T_R.

B. System Integration

The components of DeepNNCar and the data flow among them is shown in Figure 9. DeepNNCar uses ZMQ for communication among its components. The camera provides new images at 30 Hz and the IR opto-coupler speed sensor continuously collects data to compute the speed of the car.

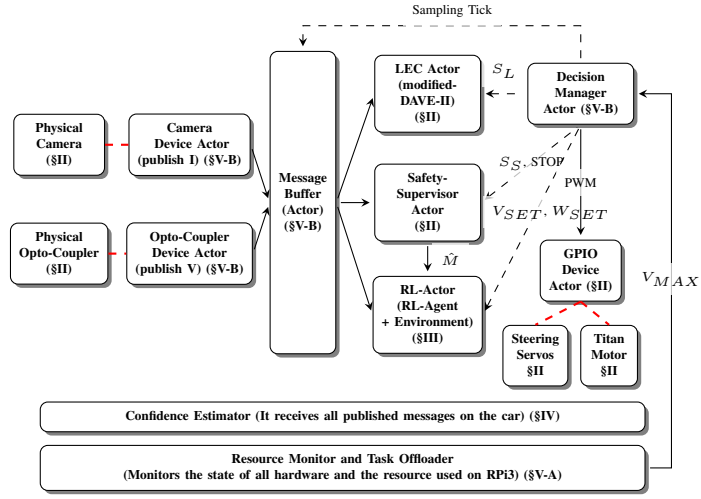


Fig. 9: A block diagram of DeepNNCar along with components. There are asynchronous interactions among various components and thus different messaging patterns were used. The request-reply communications are shown with dotted lines, the publish-subscribe communications are shown in solid lines, and the red dotted lines indicate the hardware connections. The descriptions for all the symbols can be found in Table I.

Then, the camera device actor[3] and the opto-coupler device actor periodically publish the images (I) and speed (V) to all the subscriber components. However, the LEC actor, the SS actor, and the RL-Actor are aperiodic consumers (see [23]) which do not consume the sensor values until prompted by the DMA. The interactions between the periodic publishers and aperiodic consumers are handled with the help of a Message Buffer Actor (MBA), which has a one buffer queue to store the published data (both I and V) along with a sequence label. The data in the MBA gets updated according to the sampling period of the sensors. However, this data cannot be published until the MBA has received a sampling tick and a request from the DMA to publish the data of a certain label. Once the MBA receives this request, it publishes the I and V messages to all subscribed components. Using this data, the LEC actor predicts S_L, the SS computes S_S, track position $\hat{M}$, and $STOP$ (a command issued if the car goes out of the track), and the RL-Actor computes W_L, W_S and V_{SET}.

Decision Manager Actor: The DMA issues requests for the sequence label and data S_L, S_S from the controllers, and for W_L, W_S, and V_{SET} from the RL-Actor. Once the controller and the RL actor have finished computing, they reply to the DMA their label and values. The DMA then matches the labels and computes S_R using Equation 1 before feeding S_R and V_{SET} to the GPIO device actor, which controls the two motors. After applying the controls to GPIO, the DMA starts a new cycle. This cycle continuous indefinitely until it is terminated by the STOP signal from the SS or manually by the user. The tasks performed between two sampling ticks of the DMA is one control cycle of the system and the time taken to perform one control cycle is referred to as the inference pipeline time T_R. The T_R varies for every control cycle, but

[3]A device actor converts hardware sensor information into topics that can be published and subscribed to, see [22].

the average inference time for CSW-Simplex is experimentally found to be 130ms (see Figure 10).

VI. Evaluation

To evaluate the performance of the proposed weighted simplex strategies, and the resource manager on DeepNNCar, we built three different indoor tracks shown in Figure 2. These tracks were build in our laboratory using 10' x 12' tarps, under controlled lighting condition (higher lighting intensities creates reflections on the tarp, resulting in the LD algorithm to fail), and they had different geometric shapes and turns. The LEC was trained on the images collected from Track1 and Track2, and it was later tested on Track3 to ensure the trained CNN had not overfit (generalized with the training data).

To evaluate the safety performance of the different controllers (LEC, SS) and weighted simplex strategies, we deployed them on DeepNNCar and performed varied trial runs. Figure 3 shows the number of soft safety violations performed by the different controllers and strategies at different speeds. Keeping the speed constant (0.25 - 0.65) m/s we ran the car with different controllers separately for 10 laps around Track1. For the LEC and SS, the data for safety violations were collected by maintaining a constant vehicle speed (ranging from 0.25 to 0.65 m/s) set by a human supervisor, and controlled using a PID controller. The SW-Simplex and CSW-Simplex strategies do not maintain a constant speed. Therefore, the safety violations for these strategies were segregated into different speed groups after completing the experiments. The CSW-Simplex was found to perform most reliably with the lowest number of violations.

As discussed earlier, the primary goal of the weighted simplex strategy is to find the optimal controller weights; we performed experiments to find how the weights of the different simplex strategies varied according to the track segments. For this we clustered the Track1 into three segments: Straight, Near Curved Segment and In Curved Segment, and recorded the weights data. From Table V we see the classical-Simplex uses a binary weights to compute the systems output. In the SW-Simplex strategy the weights remain fixed for different track segments, and in the CSW-Simplex strategy the weights changes dynamically for different track segments. In CSW-Simplex, the weights are same as shown in Table V for most of the trial runs; however, depending on the car's deviation from the center of the track, the weights may vary sightly by (δW_L, δW_S = 0.05)

The second design goal of the weighted simplex strategy is to optimize the speed performance of the system. Figure 11 shows the speed performance of the different controllers, which was plotted using data collected by running the car with different controllers and strategies on Track1 for 10 laps. As seen the SS and LEC have a similar speed pattern due to the PID controller they use. However, for SW-Simplex and CSW-Simplex the speeds are continuously updated depending on the mode of the track. It can be seen that CSW-Simplex optimizes the speed of the car better compared to the other strategies.

Simplex Strategy	Straight Segment	Near Curved Segment	In Curved Segment
Classical-Simplex (W_L, W_S)	(1, 0)	(1, 0)	(0, 1)
SW-Simplex (W_L, W_S)	(0.8, 0.2)	(0.8, 0.2)	(0.8, 0.2)
CSW-Simplex (W_L, W_S)	(0.95, 0.5)	(0.85, 0.15)	(0.8, 0.2)

TABLE V: Comparing the ensemble weights of different simplex strategies. For Classical-Simplex, the weights show that the LEC was chosen in the straight segments, and SS in the curved segments. For the SW-Simplex we have a fixed weight for all the track segments, these weights were manually tuned by a human supervisor. For the CSW-Simplex the weights were dynamically updated by the Q-learning algorithm.

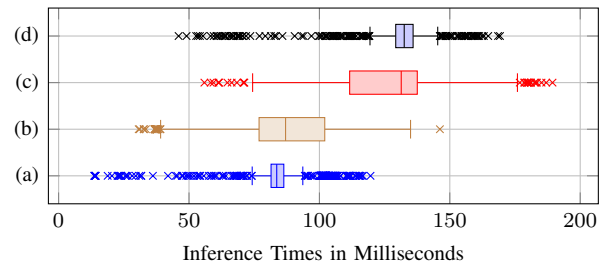


Fig. 10: Inference times in milliseconds of (a) SS: driving only with the safety supervisor, (b) LEC: driving only with the modified Dave-II model, (c) SW-Simplex, and (d) CSW-Simplex.

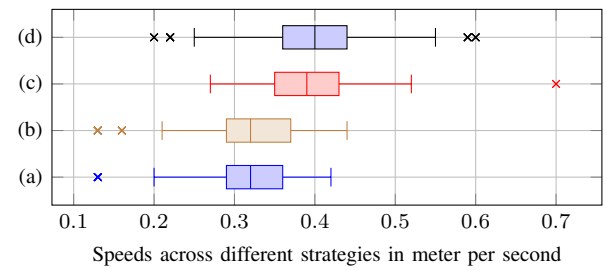


Fig. 11: Speeds in meter per second of (a) SS: driving only with the safety supervisor, (b) LEC: driving only with the modified Dave-II model, (c) SW-Simplex, and (d) CSW-Simplex.

The safety and speed results of weighted simplex strategies indicate an improvement over the LEC and SS controllers, however the increased computations of the weighted simplex strategies increases the inference pipeline times. Figure 10 illustrates the inference times of the different controllers and strategies. We observe that the LEC and SS have lower pipeline times (80 ms) while the SW-Simplex and CSW-Simplex controllers have a higher pipeline time (130 ms).

As discussed in Section V, to compensate for the computations of the controllers and weighted simplex strategies, we setup a testbed for performing the task offloading experiments. We had a wireless reply-request communication between the onboard RPi3, a laptop (with an Intel 4-core processor) and a desktop (with 32 AMD Ryzen Threadripper 16-core processor). These experimental results were collected in real-time when the car was running on Track1.

To evaluate the performance of the task offloader, we performed task offload to one of the above mentioned fog nodes, and evaluated the temperature variations of the RPi3. The Figure 12 shows the temperature of the RPi3 to drop below the threshold (70°C) when the tasks are offloaded to the fog device. It also shows that the tasks were got back to

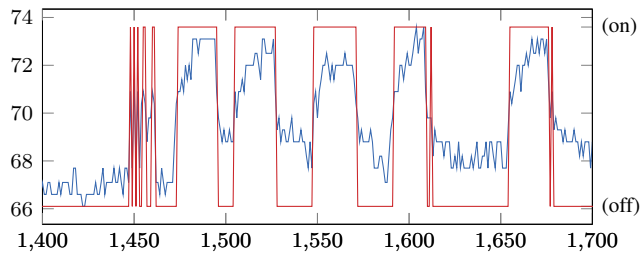


Fig. 12: The effect of offloading the tasks in response to high temperature per iteration of the inference pipeline. The trigger to offload the task is 70°C. The blue line shows the temperature in Celsius. The red line shows when the tasks were offloaded to the fog (on=on fog, off=off fog). The graph shows a subset of iterations (total 10000 iterations).

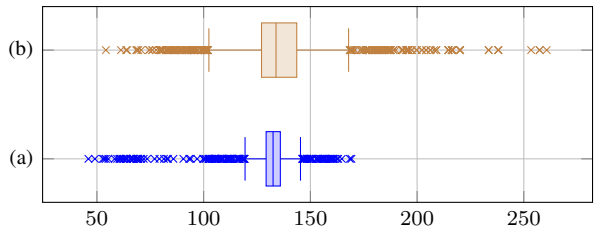


Fig. 13: Inference times in milliseconds (a) CSW-Simplex with all tasks executed onboard (b) CSW-Simplex with RL-Actor offloaded (Q-table was offloaded to the fog node, and RL-Actor had to fetch actions from the offloaded Q-table).

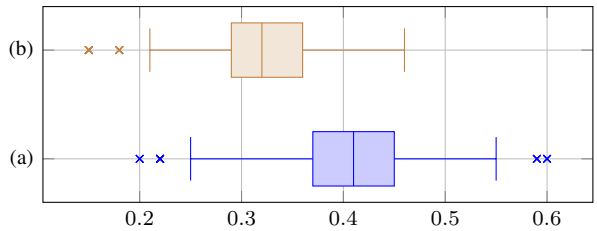


Fig. 14: Speed readjustment during offload (m/s) (a) CSW-Simplex with all tasks executed onboard (b) CSW-Simplex with RL-Actor offloaded (Q-table was offloaded to the fog node, and RL-Actor had to fetch actions from the offloaded Q-table).

RPi3 once the temperature dropped below the threshold.

Results in Figure 12 shows that task offload keeps the RPi3's temperature in check; however it increases the inference pipeline time due to added latency overhead. The pipeline time comparison of the car when tasks get offloaded vs. not-offloaded is shown in Figure 13. From the figure it can be seen that CSW-Simplex with all tasks performed onboard RPi3 to have lower inference time T_R compared to the CSW-Simplex with RL-Actor offloaded (Q-table was offloaded, so RL-Actor had to fetch the actions from the offloaded Q-table).

In Section V we discussed the importance of saturating the top speed of the car in order to compensate for the latency overhead. Figure 14 shows that the DeepNNCar with offloaded RL-Actor runs at lower (safe) speeds in order to compensate for the increased inference time T_R.

VII. Related Work

Our work encompasses several topics including Autonomous system testbeds and Simplex Architectures. These topics are briefly discussed below.

Autonomous testbeds: There have been several ongoing projects related to physical testbeds for autonomous systems. F1/10 [24] is an autonomous racing competition with cars built on Traxxas 1/10 scale RC car (like DeepNNCar) with an expensive NVIDIA's Jetson TX1 onboard computing unit. These autonomous builds use cameras, IMUs, and expensive LIDAR ($1,775) systems for performing simultaneous localization and mapping (SLAM). An F1/10 car is far more expensive compared to the cost of DeepNNCar (approx. $3000 vs. $418). DeepPicar [25] is another platform which is built using a smaller 1/24 scale RC chassis. This platform also uses an RPi3 as the computing unit and performs autonomous driving using NVIDIA's DAVE-II CNN. This build is relatively inexpensive ($70), but has a considerably smaller chassis and uses discrete steering actuation unlike DeepNNCar which performs continuous steering.

Simplex architectures and arbitration logic: Simplex Architectures [10] have been extensively used in CPS to provide safety guarantees to control systems. They have been used in online control systems [26], real time embedded systems [27], and unmanned aerial vehicles (UAV) [11]. These implementations use arbitration logic to switch between the controllers. The two well known switching criteria are linear matrix inequality [26] and hybrid system reachability [28]. Simplex Architectures always choose the output of one controller, depending on the safety criteria, whereas the weighted simplex strategy computes a weighted sum of the different controller's output. Also, with Simplex Architectures it is not possible to combine two unverified controllers; however, weighted simplex strategies allow their combination. In addition, the switching criteria based on reachability analysis could be slow if the sate space is large. However, with the use of RL in CSW-Simplex finding the optimal weights during testing is about 20 milliseconds.

Safety via contracts: Andalam, Sidharta, et al. [29] have discussed CLAIR, a contract-based framework for developing resilient CPS architectures. This work is predominantly a contract-based framework for components in different levels of abstraction. Formal contracts are used to capture the assumptions about the environment and guarantees provided by the systems components. It also employs resilience managers at the component and system levels to monitor the safety contracts. If the A-G contract fails then the component/system violates the safety. In our work, we use the weighted simplex strategy, and we show that context-sensitive weighted simplex strategies can improve the safety of the robotic system.

Phan, Dung et al. [12] have integrated Simplex Architecture with A-G contracts (to determine the switching logic), and have named it as Component Based Simplex Architecture (CBSA). In addition to the switching logic, A-G contracts are also used to provide a run-time assurance for the systems safety using the components assured contracts. While CBSA uses A-G contracts for the arbitration logic and run-time assurances, the CSW-Simplex uses RL for finding the optimal weights to compute the systems (safe) output.

VIII. Conclusion and Future Work

In this work, we have discussed the problems associated with the LECs and have further implemented it on our physical testbed, DeepNNCar. To improve the safety guarantees of the LEC driven robot, we introduced a framework that allows for the integration of safety supervisors and weighted simplex strategies. We described two weighted simplex strategies (SW-Simplex, CSW-Simplex), and evaluated their effects on speed, steering, and safety of DeepNNCar. Our analysis showed that the CSW-Simplex outperformed other implementations with the fewest safety violations and maximal safe operating speed of the car. Furthermore, we developed a system monitor which estimates a probability of the system staying in the safe working region. In addition, we described a mechanism to compensate for the computation overhead of the simplex strategies by offloading tasks to available fog nodes.

This framework can be integrated onto robots used in factories, warehouse and research laboratories, where they are required to perform safe navigation and coordinated tasks.

As future work, we plan to integrate fog-edge performance monitoring benchmark tool like FECBench [30], and an online confidence estimation scheme, which uses real time data to predict the confidence metric. We also plan to extend the existing framework to perform multi-agent experiments.

Acknowledgements: This work was supported by the DARPA and Air Force Research Laboratory. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA or AFRL.

References

[1] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang et al., "End to end learning for self-driving cars," arXiv preprint arXiv:1604.07316, 2016.

[2] D. A. Pomerleau, "ALVINN: An autonomous land vehicle in a neural network," in Advances in neural information processing systems, 1989, pp. 305–313.

[3] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," Journal of artificial intelligence research, vol. 4, pp. 237–285, 1996.

[4] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton et al., "Mastering the game of go without human knowledge," Nature, vol. 550, no. 7676, p. 354, 2017.

[5] T. Glasmachers, "Limits of end-to-end learning," arXiv preprint arXiv:1704.08305, 2017.

[6] Y. Tian, K. Pei, S. Jana, and B. Ray, "DeepTest: Automated testing of deep-neural-network-driven autonomous cars," in Proceedings of the 40th International Conference on Software Engineering. ACM, 2018, pp. 303–314.

[7] K. Pei, Y. Cao, J. Yang, and S. Jana, "DeepXplore: Automated whitebox testing of deep learning systems," in Proceedings of the 26th Symposium on Operating Systems Principles. ACM, 2017, pp. 1–18.

[8] W. Xiang, H.-D. Tran, and T. T. Johnson, "Reachable set computation and safety verification for neural networks with ReLU activations," arXiv preprint arXiv:1712.08163, 2017.

[9] C. Richter, W. Vega-Brown, and N. Roy, "Bayesian learning for safe high-speed navigation in unknown environments," in Robotics Research. Springer, 2018, pp. 325–341.

[10] L. Sha, "Using simplicity to control complexity," IEEE Software, no. 4, pp. 20–28, 2001.

[11] P. Vivekanandan, G. Garcia, H. Yun, and S. Keshmiri, "A simplex architecture for intelligent and safe unmanned aerial vehicles," in 2016 IEEE 22nd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA). IEEE, 2016, pp. 69–75.

[12] D. Phan, J. Yang, M. Clark, R. Grosu, J. D. Schierman, S. A. Smolka, and S. D. Stoller, "A component-based simplex architecture for high-assurance cyber-physical systems," in 17th International Conference on Application of Concurrency to System Design, ACSD 2017, Zaragoza, Spain, June 25-30, 2017, 2017, pp. 49–58. [Online]. Available: https://doi.org/10.1109/ACSD.2017.23

[13] G. Biswas, H. Khorasgani, G. Stanje, A. Dubey, S. Deb, and S. Ghoshal, "An approach to mode and anomaly detection with spacecraft telemetry data," International Journal of Prognostics and Health Management, 2016.

[14] D. Jiménez, "Dynamically weighted ensemble neural networks for classification," in 1998 IEEE International Joint Conference on Neural Networks Proceedings. IEEE World Congress on Computational Intelligence (Cat. No. 98CH36227), vol. 1. IEEE, 1998, pp. 753–756.

[15] M. Davis, M. Smith, J. Canny, N. Good, S. King, and R. Janakiraman, "Towards context-aware face recognition," in Proceedings of the 13th annual ACM international conference on Multimedia. ACM, 2005, pp. 483–486.

[16] H. Cao, D. H. Hu, D. Shen, D. Jiang, J.-T. Sun, E. Chen, and Q. Yang, "Context-aware query classification," in Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval. ACM, 2009, pp. 3–10.

[17] L. Fridman, B. Jenik, and B. Reimer, "Arguing machines: Perceptioncontrol system redundancy and edge case discovery in real-world autonomous driving," arXiv preprint arXiv:1710.04459, 2017.

[18] C. J. Watkins and P. Dayan, "Q-learning," Machine learning, vol. 8, no. 3-4, pp. 279–292, 1992.

[19] "Raspberry Pi frequency management." [Online]. Available: https://www.raspberrypi.org/documentation/hardware/raspberrypi/frequency-management.md

[20] "iPerf: A tool for measuring network performance." [Online]. Available: https://iperf.fr/

[21] P. Hintjens, ZeroMQ: messaging for many applications. " O'Reilly Media, Inc.", 2013.

[22] A. Dubey, G. Karsai, P. Volgyesi, M. Metelko, I. Madari, H. Tu, Y. Du, and S. Lukic, "Device access abstractions for resilient information architecture platform for smart grid," IEEE Embedded Systems Letters, pp. 1–1, 2018.

[23] N. Mahadevan, A. Dubey, and G. Karsai, "Model-based software health management for real-time systems," in IEEE Aerospace Conference(AERO), vol. 00, 03 2011, pp. 1–18. [Online]. Available: doi.ieeecomputersociety.org/10.1109/AERO.2011.5747559

[24] "F1Tenth autonomous racing competition." [Online]. Available: \url{http://f1tenth.org/}

[25] M. G. Bechtel, E. McEllhiney, and H. Yun, "DeepPicar: A low-cost deep neural network-based autonomous car," arXiv preprint arXiv:1712.08644, 2017.

[26] D. Seto and L. Sha, "A case study on analytical analysis of the inverted pendulum real-time control system," CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, Tech. Rep., 1999.

[27] S. Bak, D. K. Chivukula, O. Adekunle, M. Sun, M. Caccamo, and L. Sha, "The system-level simplex architecture for improved real-time embedded system safety," in Real-Time and Embedded Technology and Applications Symposium, 2009. RTAS 2009. 15th IEEE. IEEE, 2009, pp. 99–107.

[28] S. Bak, K. Manamcheri, S. Mitra, and M. Caccamo, "Sandboxing controllers for cyber-physical systems," in Cyber-Physical Systems (ICCPS), 2011 IEEE/ACM International Conference on. IEEE, 2011, pp. 3–12.

[29] S. Andalam, D. J. X. Ng, A. Easwaran, and K. Thangamariappan, "Clair: A contract-based framework for developing resilient cps architectures," in 2018 IEEE 21st International Symposium on Real-Time Distributed Computing (ISORC). IEEE, 2018, pp. 33–41.

[30] Y. Barve, S. Shekhar, A. Chhokra, S. Khare, A. Bhattacharjee, and A. Gokhale, "Fecbench: An extensible framework for pinpointing sources of performance interference in the cloud-edge resource spectrum," in 2018 IEEE/ACM Symposium on Edge Computing (SEC). IEEE, 2018, pp. 331–333.