DeepNNCar: A Testbed for Deploying and Testing Middleware Frameworks for Autonomous Robots

Matthew P Burruss, Shreyas Ramakrishna, Gabor Karsai, Abhishek Dubey
Institute for Software Integrated Systems
Vanderbilt University
Nashville, TN, USA

***Abstract*—This demo showcases the features of an adaptive middleware framework for resource constrained autonomous robots like DeepNNCar (Figure 1). These robots use Learning Enabled Components (LECs), trained with deep learning models to perform control actions. However, these LECs do not provide any safety guarantees and testing them is challenging. To overcome these challenges, we have developed an adaptive middleware framework that (1) augments the LEC with safety controllers that can use different weighted simplex strategies to improve the systems safety guarantees, and (2) includes a resource manager to monitor the resource parameters (temperature, CPU Utilization), and offload tasks at runtime. Using DeepNNCar we will demonstrate the framework and its capability to adaptively switch between the controllers and strategies based on its safety and speed performance.**

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

I. Introduction

Autonomous robots are being ubiquitously used in different cyber physical system (CPS) applications including self-driving cars [1], manufacturing (robotic arms, service robots), agriculture, and search-and-rescue disaster management (autonomous drones [2]). These systems use Learning Enabled Components (LECs), which are trained with deep learning models to perform precise control actions. However, these training scenarios are often limited and lead to unsafe operations in some corner cases. For example, consider trying to operate a robot that was trained individually, to fly in formation with other robots. Such a scenario will most likely lead to collisions. Our solution framework presented in [3] uses reinforcement learning for designing weighted simplex strategies to augment the trained LECs to reduce the safety violations.

The Simplex Architecture [4] has long been used in safety critical CPS applications like aircraft (Boeing 777 [4]), unmanned aerial vehicles (UAV) and mission critical ground rovers to assure safety of the system. These architectures augment unverified, high performance controllers with high assurance safety supervisors and a decision manager (DM) to perform arbitration between the controllers. However, sometimes it is not possible to get a high assurance safety supervisor (SS in our example shown in Figure 3). In such scenarios, using Simplex Architectures will not improve the systems safety guarantees. However, performing an ensemble approach to utilize the weighted sum of outputs [5] could improve

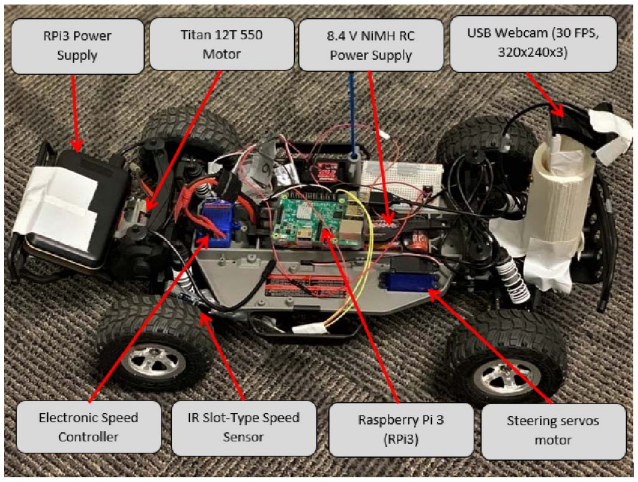
RPI3 Power Supply
Titan 12T 550 Motor
8.4 V NIMH RC Power Supply
USB Webcam (30 FPS, 320x240x3)
Electronic Speed Controller
IR Slot-Type Speed Sensor
Raspberry Pi 3 (RPi3)
Steering servos motor

Fig. 1: DeepNNCar platform with different onboard components and sensors. The car uses the front facing camera images and the IR opto-coupler speed data to compute the continuous steering angle of the car.

the safe operations of the system. We call this approach of computing the sum of weighted controller's output as "Weighted Simplex Strategy".

Demonstration: The testbed for our demonstrations includes the DeepNNCar [3], a closed loop track, and Ultra-Wideband sensors. Using the physical testbed we demonstrate the performance of different weighted simplex strategies and the ability of the framework to adaptively arbitrate between the different controllers based on the safety and speed performance.

II. Overview of DeepNNCar

DeepNNCar[1] is an autonomous testbed built on the chassis of the Traxxas Slash 2WD 1/10 RC car. The Raspberry Pi 3 (RPi3) is the onboard computing unit which generates the PWM signals to control the speed and steering motors. The sensors on the car include a USB webcam that captures images at 30 FPS with a resolution of 320x240x3 and a slot-type IR opto-coupler attached near the rear wheel chassis to measure the RPM and to compute the speed of the car.

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

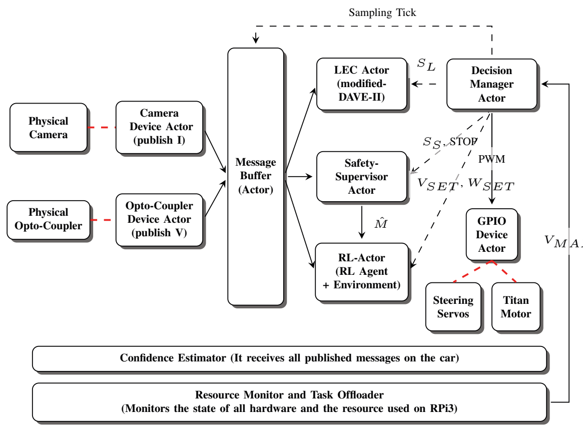
Sampling Tick
Physical Camera
Camera Device Actor (publish I)
Physical Opto-Coupler
Opto-Coupler Device Actor (publish V)
Message Buffer (Actor)
LEC Actor (modified-DAVE-II)
S_L
Decision Manager Actor
S_S, STOP
PWM
Safety-Supervisor Actor
V_{SET}, W_{SET}
$\hat{M}$
GPIO Device Actor
V_{MAX}
RL-Actor (RL Agent + Environment)
Steering Servos
Titan Motor
Confidence Estimator (It receives all published messages on the car)
Resource Monitor and Task Offloader (Monitors the state of all hardware and the resource used on RPi3)

Fig. 2: A block diagram of DeepNNCar along with the different components which communicate using different messaging patterns. 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.

DeepNNCar wirelessly communicates to a fog device, to publish its run-time position, speed (m/s), steering angle, CPU utilization, and temperature. This client-server setup of DeepNNCar allows it to work in different modes including data collection, livestream, and autonomous driving mode.

The car uses an end-to-end (e2e) learning [6] pipeline with a modified NVIDIA's DAVE-II Convolutional Neural Network (CNN) model to infer a steering action based upon the camera image and current speed. The CNN based controller (LEC) is augmented with safety supervisor (SS) which uses a computer vision based lane detection (LD) algorithm to determine a discrete steering values for a given lane segment.

The framework integrates the LEC and SS to design different mode aware simplex strategies including SW-Simplex (Simple Weighted Simplex) and CSW-Simplex (Context Sensitive Weighted Simplex) for improving the systems safety guarantees (see Figure 3). The DM (see Figure 2) allows for adaptive switching between these strategies based on performance. In addition, the framework has a resource manager to monitor the computational resources and offload tasks at runtime. Figure 2 shows the block diagram of DeepNNCar's different components and their interactions. A detailed functioning of the block diagram along with the components is provided in our full paper to be presented at ISORC 2019 [3].

III. Demonstration Example

To showcase the system, we have designed a closed loop track on which we run DeepNNCar using the different controllers (LEC and SS), weighted simplex strategies (SW-Simplex and CSW-Simplex) to monitor for its speed, steering and safety performance. In this scenario, the car is tasked to run around the track while optimizing its speed and reducing the number of soft safety violations (car crossing the track boundaries). Figure 3 shows the number of safety violations

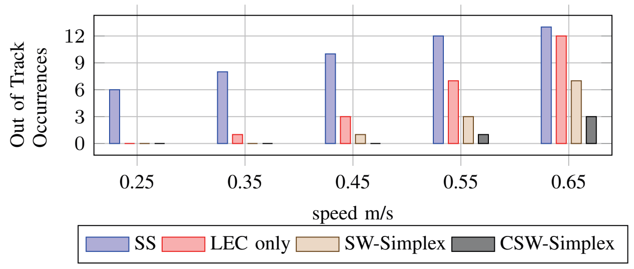

Fig. 3: Safety performance of different controllers deployed on DeepNNCar. (a) SS: driving with OpenCV lane detection code, (b) LEC: driving with the modified Dave-II CNN model, (c) SW-Simplex, and (d) CSW-Simplex. The horizontal axis shows the different speeds of the car during an experiment.

performed by the different controller and weighted simplex strategies while operating at various speeds.

The accurate real time position of the car on the track is obtained using the Archimedes Ultra-Wideband System by Ciholas (CUWB) [7]. Anchor sensors are positioned at the boundaries of the track and a tag is positioned on the mobile agent. The tags communicate with the anchors to provide the real time position of the car. The 2d position, runtime speed, steering and the safety performance of the car is displayed on a dashboard of wirelessly connected fog device. Using this setup we will demonstrate performance based arbitration between different weighted simplex controllers. We will run DeepNNCar around the track multiple times to illustrate the DM's (see Figure 2) capability to adaptively switch between the controllers and strategies based on the current speed and safety performance. This adaptability of the framework allows the car to work with different controllers in changing environments (different tracks, varying lighting conditions).

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] L. Apvrille, T. Tanzi, and J.-L. Dugelay, "Autonomous drones for assisting rescue services within the context of natural disasters," in General Assembly and Scientific Symposium (URSI GASS), 2014 XXXIth URSI. IEEE, 2014, pp. 1–4.

[3] S. Ramakrishna, A. Dubey, M. P. Burruss, C. Hartsell, N. Mahadevan, S. Nannapaneni, A. Laszka, and G. Karsai, "Augmenting learning components for safety in resource constrained autonomous robots," CoRR, vol. abs/1902.02432, 2019. [Online]. Available: http://arxiv.org/abs/1902.02432

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

[5] 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.

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

[7] Ciholas. Archimedes ultra-wideband system. [Online]. Available: https://cuwb.io/docs/v2.0/overview/