Training a Remote-Control Car to Autonomously Lane-Follow using End-to-End Neural Networks

Bryce Simmons, Yazeed Alhuthifi, Huong Pham, Pasham Adwani, and Artur Wolek
Department of Electrical Engineering and Computer Science
The Catholic University of America, Washington, D.C. 20064
Email: {01simmons, 20alhutaifi, phamh, adwani, wolek} @ cua.edu

***Abstract*—This paper describes the implementation of an end-to-end learning approach that enables a small, low-cost, remote-control car to lane-follow in a simple indoor environment. A deep neural network (DNN) and a convolutional neural network (CNN) were trained to map raw images from a forward-looking camera to steering and speed commands (right, left, forward, reverse). The mechanical, electrical, and software development of the autonomous car is discussed and the performance of both types of neural networks is reported. The car was also trained to detect stop signs using a Haar classifier. Obstacles were detected using an ultrasonic sensor. Inspired by other related projects, the goal of this work was to introduce undergraduate electrical engineering and computer science students to concepts in machine learning and hands-on system integration as part of a senior design capstone course.**

***Index Terms*—Self-driving car, end-to-end learning, deep neural network (DNN), convolutional neural network (CNN), vision-based lane-following, engineering education**

I. Introduction

Self-driving car technology is expected to significantly benefit society by improving road safety, increasing road capacity and reducing travel time, providing new opportunities for mobility-limited individuals, and stimulating economic productivity [1]. Recent research efforts aimed at advancing this technology have focused on integrating complimentary sensors (e.g., Google's "Waymo" [2] or Tesla's "Autopilot" [3] autonomous car use a combination of LiDAR, cameras, radar, sonar, and GPS) and developing related perception algorithms. Simultaneous localization and mapping (SLAM) [4] has allowed vehicles to detect the obstacles and represent the navigable structure of the environment. Machine learning has played an important role in providing semantic understanding through the detection and classification of road signs [5], lane markings [6], terrain [7], and other objects or obstacles. In typical self-driving architectures, the outputs of such perception algorithms are coupled with a decision making framework to generate a specification for a motion planning algorithm which in turn generates a reference path or trajectory for a feedback controller to follow [8].

However, researchers have recently proposed that machine learning techniques could be used, in some circumstances, not only for detection and classification but also for complete "end-to-end" learning (e.g., learning a direct mapping from sensor inputs to steering commands and supplanting the need for motion planning and control). Using less than a hundred hours of driving data, a Convolutional Neural Network (CNN) was trained to map a raw image from a forward-facing camera to steering commands that allow lane and road follow in a variety of weather conditions and environments [9]. End-to-end lane-following has also been demonstrated using a low-cost, remote-control car and hobby-grade components in [9] – a project which has been an inspiration for this work.

The objective of the work described in this paper was to introduce undergraduate students to concepts in machine learning through the hands-on development of a lane-following remote-control car. Four undegraduate students participated in the "Autonomous Car" senior capstone design project in the Electrical Engineering and Computer Science Department at The Catholic University of America during the 2017-2018 academic year. This paper summarizes their effort and reports:

- the mechanical, electrical, and software design of an autonomous car platform and indoor arena that is constructed from commercial off-the-shelf components
- the architecture of deep artificial neural network (DNN) and convolutional neural network (CNN) that was used train the autonomous car to lane-follow with the purpose of comparing their performance
- results describing the capability of the autonomous car to drive around a closed indoor track and a discussion of the challenges encountered and suggestions for future work

II. Design of the Autonomous Car

A. Mechanical and Electrical Design

The autonomous car, shown in Fig. 2, was developed by modifying a hobby Raster BMW M3, 1:14 scale, remote-controlled (RC) car. The RC car consisted of two motors (a steering motor and a drive motor), a chassis, remote-control electronics, and was powered by 5 AA batteries. The RC car was originally designed to receive pulse-width modulated (PWM) signals from a hand-held transmitter to control the steering and drive motor speed. The polarity and function of each wire was determined, and the width of the required pulses for various transmitter inputs was characterized using an oscilloscope. An Arduino UNO [10] was used as an on-board micro-controller that could be easily programmed to control the motors. A KAO3 Arduino Motor Shield [11] was integrated into the system to enable the motors to be

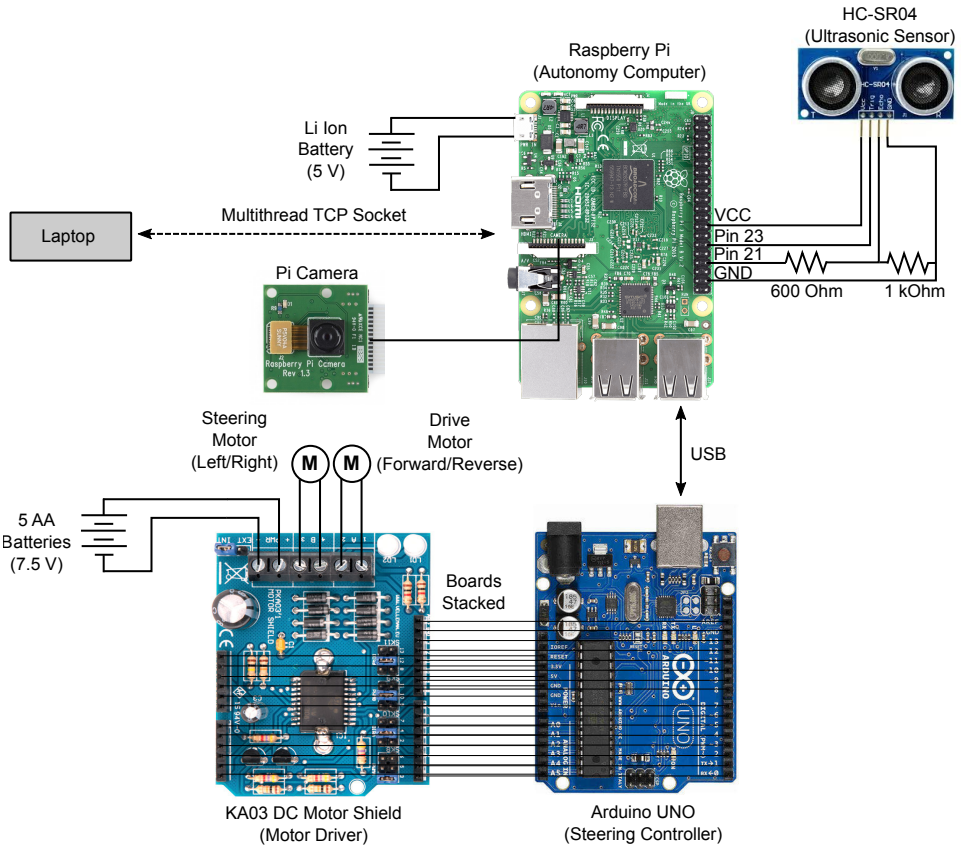


Fig. 1: Electrical and communication integration of components in the CUA Autonomous Car

modulated at a higher voltage and current than the Arduino alone would allow. An ultrasonic sensor (model: HC-SR04 [12]) was mounted on the front of the RC car to serve as a safety feature to prevent collisions by detecting obstacles that are in front of the vehicle. The sensor provides a non-contact measuring range from 2 cm to 4 m with an effectual angle of less than 15 degrees [12]. The Arduino and motor shield interfaced with a more capable Rapsberry Pi single-board computer which, in turn, received wireless commands from a laptop. The power for the Raspberry Pi and Arduino came from a 5 volt (10,400 mAh) Lithium Ion (Li-Ion) battery with USB connectivity. Lithium Ion batteries have a high energy density per unit mass which makes it ideal for keeping the overall weight of the vehicle low while, simultaneously, providing enough power to both boards.To avoid damaging the Raspberry Pi a voltage divider circuit was used to decrease the 5 volt input to 3.3 volts. By installing this separate Li-Ion power source (in addition to the 5 AA batteries), the motors and on-board computers could be separately powered for reliable operation. The electrical integration of all the components on the vehicle, as well as communication with the laptop computer, is illustrated in the schematic in Fig. 1. The list and cost of all hardware components is given in Table I.

A custom platform for mounting the wide-angle camera to the vehicle chassis was designed using CAD software and 3D printed. The platform was also used to house the Raspberry Pi and Arduino and was raised above the chassis, using spacers, to provide room to store the external 5 volt battery and to neatly hide the accompanying wires. The complete system is shown in Fig. 2.

Component	Model and Description	Cost
Remote Control Car	Raster BMW M3, scale 1:14	$35
Proximity Sensor	Sparkfun HC-SR04: ultrasonic ranging sensor	$3.95
Control Computer	Arduino UNO	$23.38
Autonomy Computer	RaspberryPi 3	$34.99
Camera	Pi Wide-Angle Camera	$13.56
Motor Shield	KA03	$20.89
Data Storage Device	Micro-SD Card	$13
Indoor Track	Miscellaneous items	$\sim$ $20

TABLE I: Main components and associated costs for the CUA Autonomous Car (Total Cost : $\sim$ $170)

B. Indoor Testing Environment

An indoor track was developed consisting of an outer loop and several intersections (Fig. 3). Wide highlighted tape was used to construct the outline while paper cut to an appropriate size was used to fill in islands and the perimeter of the environment. The paper was used to provide a uniform background and mask any features on the floor that could be misinterpreted as lanes. This significantly improved the accuracy rate during training the neural networks.

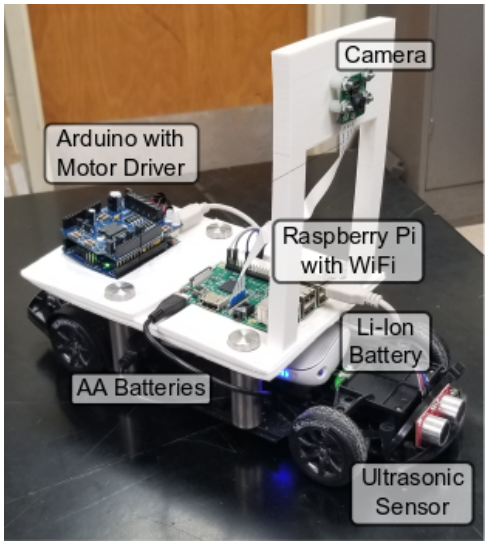


Fig. 2: CUA's Autonomous Car

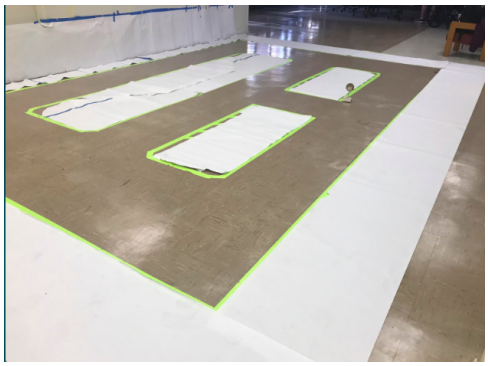


Fig. 3: The laboratory testing environment

C. Software Design

1) Operating System, Firmware, and Libraries: The Raspberry Pi ran a native Linux-based operating system (Raspbian) which facilitated the use of many open-source tools to support this project. The Pi Camera firmware was installed to enable the on-board autonomy computer to capture images and store them as .JPEG files on the Raspberry Pi. Python libraries used in this project include, OpenCV [16] with Numpy [17], and Pandas [18] for preprocessing data and training the cascade classifier and neural networks provided and built by functions from OpenCV library. The *thread* and *socketserver* modules were used for data transfer between the laptop and Raspberry Pi.

2) Data Communication: We used Transmission Control Protocol/Internet Protocol (TCP/IP) sockets as the tools for data transfer between the laptop and Raspberry Pi. The server side of the TCP/IP socket, running on the laptop, was configured with three ports to communicate with the PI camera, the ultrasonic sensor, and the Arduino UNO used to drive the motors. The client side, running on the Raspberry Pi autonomy computer, acted as a inter-communicator facilitating the transfer of commands and data. After the connections were established and while the car was self-driving, images from camera and distance measured by the sensor were transferred to the laptop via the Raspberry Pi so that they could be tested and compared with the training data from the classifier and neural networks. The result from each test returned a unique label (e.g., stop-sign) or command (e.g., right, left, forward, reverse) which would be sent to Arduino from laptop via the autonomy computer.

3) Object Detection: A Haar cascade classifier algorithm was used to detect stop signs. Both positive and negative sample images of the stop signs were collected and stored into an Extensible Markup Language (XML) file. The XML file became the database for the algorithm to be able to detect the stop signs. This algorithm uses Haar-like features which consider adjacent rectangular regions at a specific location in a detection window to gather the sum of the pixel intensities in each region, and calculate the difference between these sums.

4) Object Distance Measurement: A monocular vision [21] technique was used to make distance measurements using the Pi camera module. The distance is calculated using trigonometric relations between a point in space that is detected, the focal length of the camera, the tilt angle, and camera height. The OpenCV function CalibrateCamera was used to calibrate the camera to correct for aspect distortion that may be caused by detecting an obstacle from different vantage points.

5) Neural Network: Keras [19] and Tensorflow [20] were installed on the laptop and used as the frameworks necessary to construct neural networks as described in the following section.

III. Neural Network Architectures

In this work, both a deep neural network (DNN) and a convolutional neural network (CNN) were implemented and compared. The inputs to each network consisted of a camera image while the output was a motor command (one of four actions: left, right, forward, reverse).

A. Pre-processing Data

For any supervised learning model the performance of the model depends greatly on the training data used. To obtain training data we drove the car manually using the directional arrow keys from the laptop (i.e., the right, left, forward, reverse arrow keys). Each time a key was pressed the action was associated with a training label that contained a byte sized message. This training label was appended to the image corresponding to the moment the key is pressed. The image that is captured is at first in color (RGB) and .JPEG format but is immediately convert into gray scale. Then, the gray scaled image is converted into a Numpy array that was further normalized (by dividing each pixel by 255). This decreased the values of each pixel from zero to one making the network converge faster during training. After the action was appended to the image it was saved as a sequence of ordered frames and converted into a .NPZ file. This process was repeated throughout the track environment to learn what actions to take

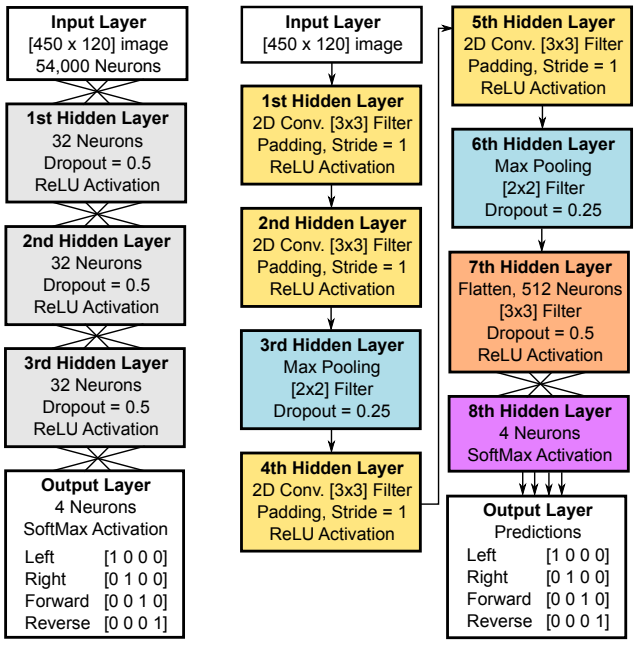
when the vehicle is at different locations. After gathering a sufficient amount of training data the .NPZ file could be used to train the different neural networks to test and compare each models performance.

B. Deep Artificial Neural Network Algorithm

The Deep Artificial Neural Network (DNN) was constructed using Keras [19] with Tensorflow [20] as the back-end. These frameworks allow for hyper-parameter to be tuned easily. The DNN architecture uses as a training input a recorded key arrow input and a flattened vector length of 54,000 which is the number of pixels gathered from the 450 x 120 image. Each pixel has a dedicated neuron at the input layer of the network which is then connected to three hidden layers comprised of 32 neuron each. Dropout was used at each hidden layer. Dropout is a regularization technique used to reducing overfitting by canceling out neurons, randomly, at each layer. For our case we chose to set this parameter to 50 percent. Also, at each hidden layer besides the last we used a rectified linear unit (ReLU) activation function. Much like a rectifier, ReLU function discards the irrelevant and garbage negative values and outputs only the others. That is, if the input is greater than 0, the output is equal to the input. When comparing ReLU with other activations such as Tanh or sigmoid, the ReLU had a much higher accuracy rate when visualizing the accuracy and loss from the training data. The output layer is made up of four neurons that link directly to the predicted actions. To accommodate for a multi-neuron output layer, a Softmax activation function was used. The entire network architecture is shown in Fig. 4a.

C. Convolutional Neural Network Algorithm

The Convolutional Neural Network (CNN) was also constructed using Keras [19] with Tensorflow [20] as the back-end. The CNN was designed with a total of 8 hidden layers (see Fig. 4b). The input layer consisted of a 2D image matrix of size 450 x 120 that has been normalized as mentioned in Section III-A. Keras has a tunable function called ”padding”, this technique maintains the input dimension of the convolutional layers by adding zeros to the boundaries. This prevents losing valuable data from the edges of the image during convolution. The size of the filter that was used has a dimension size of 3 x 3, without the padding option, the matrix would essentially be cropped by a total of 1136 pixels symmetrically around the perimeter. Once convolved, a ReLU activation function was used. The output is composed of 32 feature maps that detect different edges or patterns across the data. The next layer involved another 2D convolution, ReLU activation, and a max pooling layer with a window size of 2 x 2. The objective of max pooling is to down-sample an input representation, reducing its dimensionality and allowing for assumptions to be made about features contained in the sub-regions grouped. After the max pooling layer a dropout function is applied followed by two more 2D convolutional layers and another max pooling layer with dropout function set to 25 percent. The output of the max pooling layer was flattened to a 1D vector which is called a dense layer or fully connected layer that is made up of 512 neurons. Next, another ReLU activation is applied as well as dropout set to 50 percent. The final layer or the output layer has four neurons which is directly linked to the predicted actions, forward, reverse, right, left. To accommodate for a multi-neuron output layer, a softmax activation function was, again, used.



(a) DNN Architecture (b) CNN Architecture

Fig. 4: Comparison of architectures of the Deep Neural Network (DNN) and the Convolutional Neural Network (CNN)

IV. Implementation and Results

A. Training Process

The collection of training data was divided into two phases. The first phase consisted of accumulating roughly 5,000 frames and the second phase consist of roughly 10,000 frames. At each phase the frames were used to train both the DNN and CNN models while testing the accuracy and loss between training data and testing data. For each phase, 80% of collected data was used as training data and 20% as testing data. A batch size of 10 was chosen with total epochs ranging from 10 for the CNN, up to 100 for DNN. Using Keras callbacks library the weights derived from training were saved at each epoch which allowed us to choose the weights depending on accuracy and loss.

For the first phase of training, the DNN had a total of 83% accuracy where as the CNN had a total of 86% as shown in figure **??**. For both models the car was able to predict straight paths and some of the turns on the testing environment but failed at intersections and some corners. Overall, both the DNN and the CNN performed poorly due to insufficient amount of training data for the wide scope of possible actions on the testing environment. For the second phase, up to 91%

accuracy was noted in CNN and 86% for DNN during training, see fig **??**. We found that tuning the hyper-parameters of the neural network was a difficult task which was complicated by the long time required to train the models.

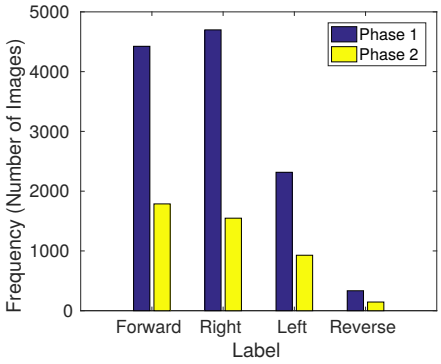


Fig. 5: Composition of labels used in training data for Phase 1 (total of 4408 image-label pairs) and Phase 2 (total of 11774 image-label pairs).

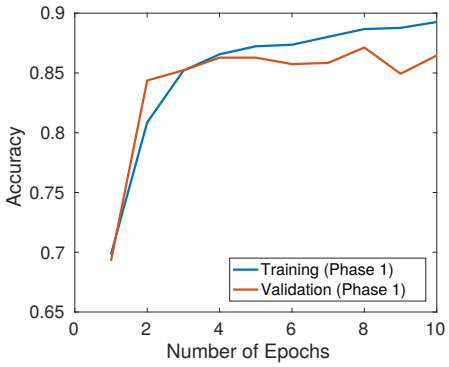


Fig. 6: CNN accuracy during Phase 1 and Phase 2

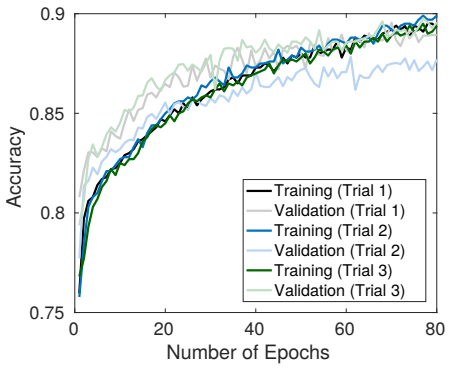


Fig. 7: DNN accuracy during Phase 1 and Phase 2

B. Real-time Lane Following

After training the the DNN and the CNN, the resulting weights were saved in XML and HD5 file extensions on the laptop for use during real-time lane following. At runtime, each captured image was sent from the Raspberry Pi to the laptop where it was converted to gray scale, normalized, and then passed through the network using the trained weights. The resulting command (i.e., predicted best action) was then transmitted back to the car. The network processed about ten frames per second.

Initially, real-time lane following was tested on a "difficult" track that involved several abrupt turns. Table II shows the number of predictions made before failing to lane-follow during five trials in this environment using both the DNN and CNN model. Lane-following was considered a failure if the car went entirely outside of the track. The DNN made the most average predictions before failure, however the results were comparable to that of the CNN. The wide turn radius of the car may have contributed to the difficulties encountered. Interestingly, the CNN was able to learn how to reverse and get back on track after failing to lane-follow while the DNN was unable to learn this tactic while using the same training data.

	Trial 1	**Trial 2**	**Trial 3**	**Trial 4**	**Trial 5**	**Mean**
DNN	67	47	42	67	59	56
CNN	44	47	40	70	61	52

TABLE II: Comparison of DNN and CNN number of predictions before failure in a difficult track environment over five trials

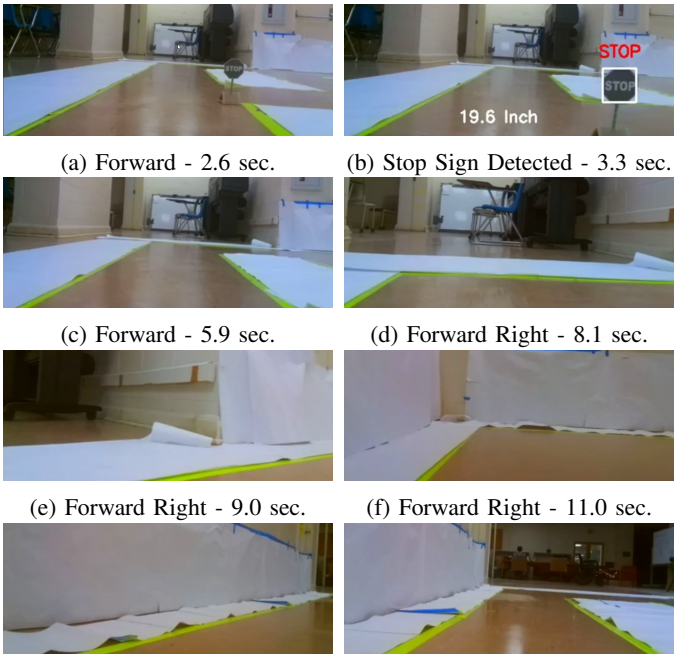
To simplify the problem, the track was modified and the abrupt turns were removed to give an easier testing environment shown in Fig. 3. In this case both the ANN and CNN were able to lane-follow indefinitely. A simple delay was implemented to stop the car when a stop-sign was detected within a certain distance threshold. An example of the car's view and predicted actions during lane-following is shown in Fig. 8.

V. Conclusion

This senior-design project to develop a lane-following autonomous car served as an educational experience that introduced a team of undergraduate students to topics related neural networks, machine vision, and hands-on system integration.

Final results indicated that the CNN achieved a higher degree of training accuracy than the ANN and both architectures enabled the car to lane-follow in a simplified environment. However, one disadvantage of using the CNN algorithm was that the convolutional layers are computationally expensive (with our model having 4 convolutional layers there are over 110 million trainable parameters) and the time required to train the CNN was significantly longer. The Haar classifier algorithm performed well in our test environment but is not suitable to deploy in the real world due to its fragile nature. The algorithm will produce false positives when objects have similar rectangular features as a stop sign.

Potential areas for improvement include the use of additional hardware components such as a greater number of



(a) Forward - 2.6 sec. (b) Stop Sign Detected - 3.3 sec.

(c) Forward - 5.9 sec. (d) Forward Right - 8.1 sec.

(e) Forward Right - 9.0 sec. (f) Forward Right - 11.0 sec.

(g) Forward Right - 13.5 sec. (h) Forward - 16.0 sec.

Fig. 8: Trial run illustrating autonomous car stopping at stop sign and executing a U-turn around the track. Each camera image is labeled with the predicted action and time.

wide-angle cameras and proximity sensors. A LiDAR system could be used to obtain a better representation of the space for localization and training. We would also like to include Natural Language Processing in the framework so that voice commands could be used to control the vehicle. Mechanical upgrades to the RC car components could include the use of DC brushless motors, upgrade AA batteries to a lithium ion battery for more consistent and reliable output power, and to improve maneuverability to allow lane-following in more difficult environments.

ACKNOWLEDGMENT

We would like to thank Zheng Wang who inspired our work through his "OpenCV Python Neural Network Autonomous RC Car" project (developed as part of the COMP 502 course at Bridgewater State University in Bridgewater, MA). His software, made available online, was used for pre-processing the data, ... fill this in. Also, we would like to thank Afshin Nabili for providing the space for creating the testing environment necessary for data collection. Finally, we thank the School of Architecture at Catholic University for assistance with printing the 3D platform used.

REFERENCES

[1] D. J. Fagnant, K. Kockelman, "Preparing a nation for autonomous vehicles: opportunities, barriers and policy recommendations", *Transportation Research Part A*, 77:167-181, 2015.

[2] Waymo, "FAQ - Waymo", *waymo.com*. (Online) Available: *https://waymo.com/faq/* (Accessed: 05/03/2018)

[3] Tesla, "Autopilot — Tesla", *tesla.com*. (Online) Available: *https://tesla.com/autopilot/* (Accessed: 05/03/2018)

[4] G. Bresson, Z. Alsayed, L. Yu, S. Glaser, "Simultaneous localization and mapping: a survey of current trends in autonomous driving", *IEEE Transactions on Intelligent Vehicles*, 2(3):194-220, 2017.

[5] A. Mogelmose, M. M. Trivedi, T. B. Moeslund, "Vision-based traffic sign detection and analysis for intelligent driver assistance systems: Perspectives and survey", *IEEE Transactions on Intelligent Transportation Systems*, 13(4):1484-1497, 2012.

[6] M. Braga de Paula, C. R. Jung, "Automatic detection and classification of road lane markings using onboard vehicular cameras". *IEEE Transactions on Intelligent Transportation Systems*, 16(6):3160-3169, 2015.

[7] R. Hadsell, P. Sermanet, J. Ben, A. Erkan, M. Scoffier, K. Kavukcuoglu, "Learning long-range vision for autonomous off-road driving", *Journal of Field Robotics*, 26(2):120-144, 2009.

[8] B. Paden, M. Cap, S. Z. Yong, D. Yershov, E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles", *IEEE Transactions on Intelligent Vehicles*, 1(1):33-55, 2016.

[9] M. Bojarski et al. "End to end learning for self-driving cars" *arXiv preprint*:1604.07316, 2016.

[10] Arduino, "Arduino Uno Rev3",*Arduino*. [Online]. Available: *https://store.arduino.cc/usa/arduino-uno-rev3*. [Accessed: 09/03/2017]

[11] Velleman, "Velleman Kits KA03: MOTOR & POWER SHIELD FOR ARDUINO" ,*Velleman*. [Online]. Available: emphhttps://www.velleman.eu/products/view/?id=412174 [Accessed: 09/03/2017]

[12] SparkFun Electronics, "HC-SR04", *SparkFun*. [Online]. Available: *https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf* [Accessed: 09/03/2017]

[13] Z. Wang, "Self Driving RC Car", *WordPress*, June 2015. [Online]. Available: *https://zhengludwig.wordpress.com/projects/self-driving-rc-car/*. [Accessed: 09/01/2017].

[14] Z. Wang, "AutoRCCar", *GitHub*, June 2015. [Online]. Available: *https://github.com/hamuchiwa/AutoRCCar*. [Accessed: 09/25/2018].

[15] T. Ball, "Train your own OpenCV Haar Classifier", *Coding Robin*, July 22, 2013. [Online]. Available: *http://coding-robin.de/2013/07/22/train-your-own-opencv-haar-classifier.html*. [Accessed: 10/23/2017].

[16] G. Bradski, "The OpenCV Library", *Dr. Dobb's Journal of Software Tools*, 2000.

[17] T. Oliphant, "A guide to NumPy'', *USA: Trelgol Publishing*, 2006.

[18] W. Mckinney, "Data Structures for statistical computing in Python'', *Proceedings of the 9th Python in Science Conference*, p.51-56, 2010.

[19] F. Chollet et al.,"Keras" [Online]. Available: https://keras.io. [Accessed: 05/07/2018].

[20] M. Abad et al. ,"TensorFlow: A system for large-scale machine learning'', *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation*, p.265-283, 2016.

[21] C. Jiangwel, J. Lisheng, G. Lie, Libibing, W. Rongben,"Study on method of detecting preceding vehicle based on monocular camera", *IEEE Intelligent Vehicles Symposium*, 2014.