Prospects of Improving the self-Driving Car Development Pipeline: Transfer of Algorithms from Virtual to Physical Environment

Nauris Dorbe, Ingars Ribners, and Krisjanis Nesenbergs

***Abstract*—Problem of transferring and testing self-driving algorithms developed in virtual environment to a physical environment is explored by transferring a Convolutional Neural Network based self-driving car steering algorithm from virtual environment to physical RC card based environment for validation and testing as a step on the way for full scale self-driving car tests. In the process a novel approach for synthetic training data generation from single camera is developed, thus reducing the real world physical requirements for the algorithm and demonstrating the improved self-driving algorithm development pipeline from fully virtual environments to scaled physical models, to full self-driving cars, potentially leveraging the global developer community for development.**

***Index Terms*—Convolutional neural network, training data generation, self-driving cars, virtual and physical models.**

I. Introduction

The development of better and safer self-driving cars has been a focus for many companies, governments and research institutions around the world. A strong focus is on achieving smart, energy efficient, safe, cooperative and low cost mobility, as evidenced, by international strategic documents [1]. The highest level of vehicle automation [2], towards which the research and development are geared is the 5th level, "Full driving automation", which does not require a driver and does not require a person to intervene in the decision making process. Such level of automation requires a high amount of components to be developed and tested, and also the testing with actual cars can be expensive, dangerous and even illegal. Thus any significant success in this field has been mostly shown by large companies with significant resources (e.g. *Google*, *Tesla*, *NVidia*, *Volvo*, etc.). To combat this trend and allow more people to contribute to this complex problem a movement of developing virtual self-driving car algorithms and test them in virtual environments, has emerged. Massive Open Online Courses (MOOC) have been developed [3] where students can develop their own algorithms for self-driving cars and both train and test their neural network based solutions in the virtual environment. This approach provides a large amount of algorithms solving specific problems in the global self-driving picture, but two main problems remain – is it possible to transfer the algorithms from the virtual to the real world and is it feasible, from the perspective of the computing power required, as real world vehicles can have limited computing power because of size, energy requirements and cost.

In this article we explore these questions by examining an image processing and convolutional neural network (CNN) based self-driving car steering algorithm developed in a virtual environment, developing a real-world physical car model (RC), and transferring the algorithm to the physical model for real world testing. In the process we also develop an improved method for generating augmented CNN training data from a single camera, thus reducing the real-world hardware requirements while still preserving the benefits of multiple virtual cameras for a more stable vehicle trajectory. The model cars use a combined data processing approach with a low power local computer and wireless communication to a more powerful server and the process is geared towards preserving the lessons and work done in the virtual environment and minimizing the steps required for the transfer to physical environment.

As a result, this article shows that it is possible to develop physical RC testbed as an intermediary between fully virtual environments for development of self-driving car algorithms and expensive real car testing thus reducing the entry barriers for self-driving car algorithm developers and testing of the resulting algorithms during each step of the development pipeline.

II. Algorithm in Virtual Environment

The first step in the proposed self-driving car development pipeline is developing an algorithm in a virtual environment. In this work the specific algorithm selected for implementation is a neural network based behavioral cloning algorithm implemented in the *Udacity* self driving car virtual environment [3].

Behavioral cloning approach means, that instead of basing complex decisions on a full model of the environment, the system is developed to provide similar behavior than a human would, based solely on some input data. In this specific case, the algorithm only considers image input and provides steering angle and acceleration data output, thus cloning basic driving tasks.

In general behavioral cloning based on visual input is a non-trivial task involving complex processing of high resolution frames. This task is made significantly easier by the introduction of neural networks allowing training the algorithm by example and potentially generalizing it to unseen environments.

The success of training a neural network to drive like a human is highly dependent on the ability to provide precise

Manuscript received January 18, 2018; revised March 7, 2018.

The authors are with the Institute of Electronics and Computer Science, Dzerbenes 14, Riga, Latvia (e-mail: naurisdorbe@gmail.com, e-mail: ram3a12@gmail.com, krisjanis.nesenbergs@gmail.com).

doi: 10.18178/ijmlc.2018.8.2.684

quality training data, robust training process protecting the neural network from over-fitting or under-fitting and the speed and latency of inference for the final result to be applicable to real-time driving systems.

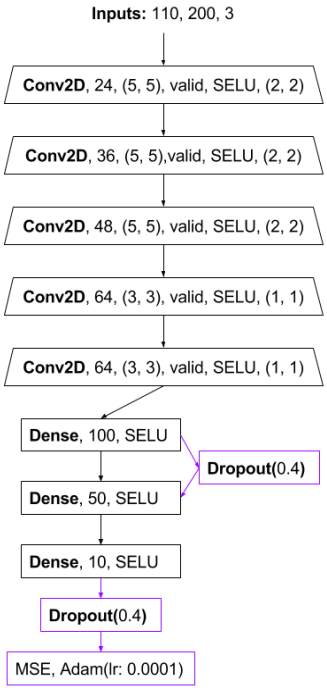

Fig. 1. Architecture of the neural network used.

Thus a CNN based algorithm was developed and implemented in the virtual environment based on the work of M. Bojarski *et al*. [4] at *NVidia Corporation*. The original model was developed for an end-to-end learning solution aimed at learning to select steering angle based on input of three cameras (left, right and center).

Even though our model similar, it differs in size and input data normalization. The full architecture of our neural network is shown in Fig. 1. In order to decrease over-fitting a Dropout [5] was introduced after first and third Dense layers. Additionally we introduced Scaled Exponential Linear Unit (SELU) activations instead of Rectified Linear Unit (ReLU) activations [6]. The network was trained for steering first and for both steering and velocity afterwards, providing sufficient basis for reinforcement learning.

The source code for our implementation of the virtual environment is available at [7].

The training work-flow in the virtual environment is shown in Fig. 2. Training follows the numbering in graph. First we choose training mode and first track to gather training data and train it with additional data augmentation then we test our model in autonomous mode in second track.

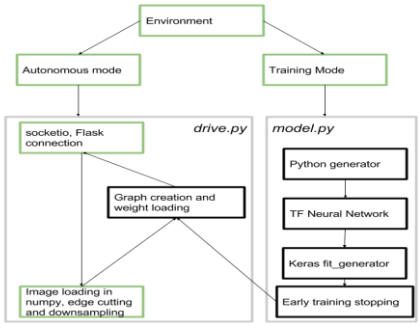

Fig. 2. Training work-flow in the virtual environment.

Data augmentation is one of the key aspects for training both in the virtual environment and later in the physical environment, because it provides robustness and re-usability by adding noise, changing the lighting conditions of the image, randomized movement of the camera and random shadows. Also the data augmentation reduces the amount of training data, that needs to be gathered reducing the data gathering time to less than an hour.

Thanks to multi-threading and python generator we were able to create effective training flow which loaded, preprocessed, augmented input images from local directories in background using CPU without sacrificing GPU utilization time and filling whole RAM with input images. We trained our model only about 54 seconds which are 24 epochs on GTX NVidia 1080 and i7-6800K.

III. Single Camera Data Generation Method

The original algorithm by M. Bojarski *et al*. [4] was developed with powerful hardware in mind, including three separate video cameras for visual input, multiple Graphic Processing Units (GPU) for training the neural network and a lot of gathered real-life training data. Two side cameras were only used for training and were not used for actual driving, thus reducing their utility.

In order to reduce the complexity of the steps in the pipeline and lower the real world hardware and data bandwidth requirements for the hardware steps thus, also reducing real world costs, an improved visual input generation method was developed requiring only a single camera.

The original requirement for three cameras stems from the problem, that training data from a single camera only contains information about how to stay on the preferred trajectory, but not, how to return to it from sub-optimal car placement, such as on the side of the road, which can can happen in real world driving because of drift introduced by non-perfect steering precision and road conditions. The two additional cameras thus introduced additional training data for returning to the trajectory from the left and right side respectively by increasing or decreasing the training steering angle by a constant amount calculated based on the car geometry.

This works, because the original measured steering angle α is increased or decreased by a constant angle α_{Δ} as if the center and heading of the car was placed at one of the side cameras as seen in Fig. 3, thus producing three times as many training data sets with additional information on how to return to the selected trajectory emulating additional "virtual" cars on the left and right of the actual car.

Our method data gathering method relies on the fact that the width of the camera sensor is several orders of magnitude smaller that the distance to the heading point, thus resulting in a small distance between the front and back side of the virtual camera denoted with $\in$ in Fig. 4. Thus the image from the left and right cameras can be safely approximated as an image from a camera faced the same way as the central camera, and only shifted to left or right, respectively. As a result, if the single central camera, has a sufficiently wide field of view, the sides of the resulting

image can be cut and used for generating simulated left and right camera images for use with the appropriate augmented angles ($\alpha+\alpha_\Delta$ and $\alpha-\alpha_\Delta$ respectively) as seen in Fig. 4.

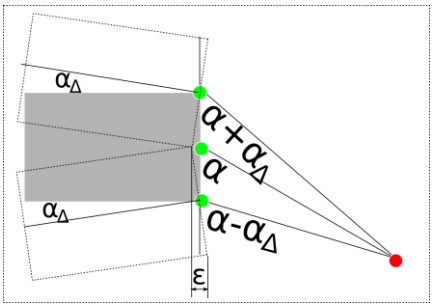

Fig. 3. Schematic of three camera placement (green), destination point (red), and both the real car placement and "virtual" car placements related to side cameras.

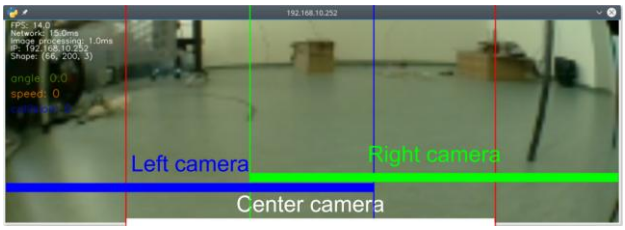

Fig. 4. Virtual left and right camera data generation from a single image from central camera.

This simple method for side camera image generation allowed us to successfully train the neural network with only one real camera input.

IV. Hardware

After the algorithm was implemented in the virtual environment, where it was possible to test and debug it much faster than in actual hardware it was necessary to build a 1/18 scale model hardware platform as a next step in the development pipeline.

As the basis for the platform a Wltoys A969 [8] Radio Controlled (RC) car was selected.

The original *Wltoys RC* and motor control unit was removed and a replacement mini-car board control board was developed allowing fully programmable control. To provide the high level control to the car a *Raspberry Pi* Version 2 mini-computer was fitted as well.

A standard *Raspberry Pi* Camera v1.3 with a "fisheye" lens with a 180 degree view angle was selected to capture road scene. This camera has the advantage of using the *Raspberry Pi* camera port instead of USB connection, providing faster data rates and also has night vision capabilities which might be beneficial for future work. The wide angle lens was selected in order to support the multiple camera view data generation method described above as the camera itself provides a narrow field of view. This introduces lens distortion to the captured image, which was corrected using chessboard pattern image as a baseline [9]. This requires taking multiple pictures from different angles of the chess board and counting corners inside them. We can then observe how distance between corners changes at different places and calculate distortion. This was achieved using *OpenCV* [10] function "*findChessboardCorners()*" for automatic chess corner detection in image, and "*calibrateCamera()*" for camera calibration using these points. The resulting distortion coefficients can be used to un-distort images with function "*undistort()*". The source code for the calibration used is available at *calibrate_git* [11].

For wireless communication a *WI-FI* dongle was used to receive control signals from *Logitech* steering unit and for sending captured pictures from the model car to the server for displaying and for neural network training in learning mode.

The model chassis is equipped with a powerful brushed motor (size 390) powered from a 7.4V LiPo battery. Average measured current for maximum speeds and loads of this model car is around 12A with peaks of up to 16-20A. Estimated theoretical maximum current of such motor is 27A so a specific h-bridge circuit that tolerates such currents was developed. For driving the steering motor a standard 1.5A L298N h-bridge board was used controlling the original model car 5 wire servo motor. In additional a PID controller for steering angle feedback was developed as 5-wire servos don't have it built in and a similar feedback controller was developed for the speed control of the main motor using motor current as simple speed feedback. To measure voltage an ADS1015 12-bit board from *Adafruit* was used.

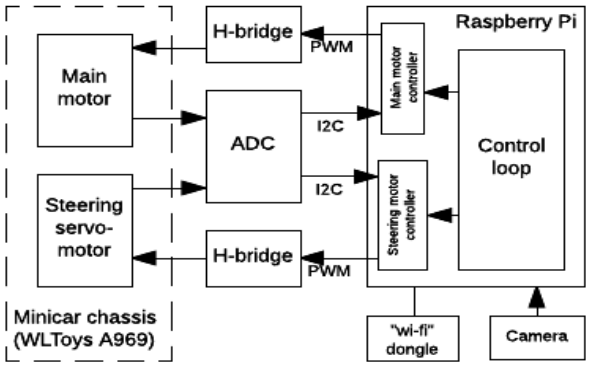

Fig. 5. Hardware solution schematic.

The overall hardware schematic can be seen in Fig. 5.

Additionally an ultrasonic sensor was added as a front collision sensor for future Reinforcement Learning research.

V. Experiment

To test how the algorithm developed in the virtual environment will function on the model hardware a supporting software system (Fig. 6) was set up including a server with connected steering wheel for training the model car capable of supporting multiple clients (car models) thus providing a platform not only for this work, but also future research in multi-agent Reinforcement Learning.

Because after transferring the neural network from virtual environment to the hardware model car we have no applicable training data, it must be gathered again. The gathered training data is basically video frames labeled with steering angle. It is important, that the resulting model is not tied to specific steering angles as those are hardware dependent and can change across different vehicles. To overcome this issue the steering angle for data labels is

converted to curvature which is 1 divided by turning circle radius (based on bicycle model) using formula $C = \frac{\tan(rad(\alpha))}{L}$ where L is wheelbase and α is steering if distance from the center of the mass to back wheels is not given and $C = \frac{1}{\sqrt{\left(A^2 + \frac{L}{\tan(rad(\alpha))}\right)}}$ where A is distance from the center of the mass to back wheel when it is given (Note that the second formula has the danger of division by zero).

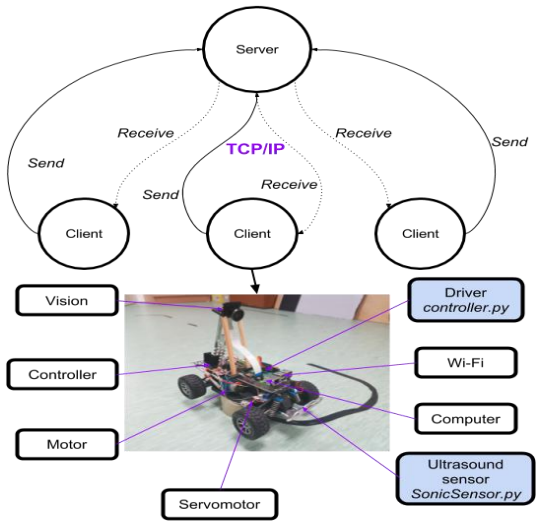

Fig. 6. Overview of experimental setup.

During the hardware experiment for the data improved data gathering method from single camera left and right frames were labeled with +/- 18% of the measured steering angle respectively. This number is derived from the amount of pixels cut from the sides of the full frame - the full frame width is 280 pixels and we divide it in three sections of 200 pixels each. The resulting shift from center frame to left and right is around 14.3%. We add extra 3.7% to simulate more aggressive returning back to center and this results in 18%.

To achieve high quality realistic and smooth training data for behavioral cloning, so that the model can learn best practices of driving and not input mistakes, an interface for *Logitech G27* steering wheel to control the model car instead of simple keyboard input was used.

The model car was trained in a test track by driving the car in one direction and then the model was validated by allowing the car to drive in the opposite direction by itself.

For a training loss function a Mean Square Error $MSE = \frac{1}{n}\sum_{i=1}^{n}(C_i - Y_i)^2$ was used where n is the size of the sample set, C_i is the correct value and Y_i is the predicted value. The MSE was calculated for training and validation data and used for training early stopping, when MSE values for training data were lower than MSE values for validation data.

VI. Results and Conclusions

A self driving car algorithm was successfully developed in a virtual environment and transferred to a physical model car, which based on the pre-trained model was successfully used for behavioral cloning as the first step in transferring algorithms from virtual environment to full size self-driving cars.

A novel approach for synthetic training data generation from single camera was developed, thus reducing the real world physical requirements for the algorithm. Additionally the physical proof-of-concept model car required a small amount of computing power and was able to cope with latency introduced by actual communication channels instead of virtual ones. Even though the initial idea to combat over-fitting was introducing Dropout, in practice it proved inefficient for this scenario and SELU approach provided much better results even though it was a little slower.

Tests showed, that ultrasonic sensors used were not appropriate for scale models because of noise from other components and close proximity.

The trained models were evaluated on different tracks and/or in different conditions (lighting, direction etc.). The success of the training was validated as the capability of the model car to independently stay in the track lane and finish the track [12].

VII. Discussion

This work could serve as a base for future work in multi-agent reinforcement learning system where each agent will be connected to already existing server - client architecture. Server will keep central model which will be update continuously based on client feedback. At the same time clients will also keep local models for effective inference. Model cars will be able to learn to drive based on indoor navigation system which is already implemented. Feedback from navigation system will be used as a reward function. Goal of this future project is to create safe learning algorithm where cars would be able to learn to drive by themselves without threatening other drivers and learn not only from their local data but data from other cars as well. To make this process fast and reliable both supervised and human feedback (while learning) methods will be used in hybrid system similar to [13]. A future goal is to transfer the results down the pipeline to an actual self driving car which is already prepared for cooperative driving.

Acknowledgment

This work is supported in part by the Latvian National research program SOPHIS under grant agreement No. 10-4/VPP-4/11.

References

[1] European Commission, "Directorate-General for Mobility and Transport, A European strategy on cooperative intelligent transport systems, a milestone towards cooperative, connected and automated mobility no com/2016/0766," EUR-Lex, 2016.
[2] Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles, *SAE International*, 2016.
[3] Udacity. (2017). Self-driving car engineer nanodegree. [Online]. Available: https://www.udacity.com/course/self-driving-car-engineer-nanodegree--nd013
[4] M. Bojarski, D. D. Testa, D. Dworakowski *et al.*, (2016). End to end learning for self-driving cars. [Online]. Available: https://arxiv.org/pdf/1604.07316.pdf

[5] N. Srivastava, G. Hinton, A. Krizhevsky *et al*., "Dropout: A simple way to prevent neural networks from overfitting," *Journal of machine learning research*, vol. 15, no. 1, pp. 1929-1958, 2014.
[6] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, "Self-normalizing neural networks," A*dvances in Neural Information Processing Systems*, vol. 30, 2017.
[7] N. Dorbe, "Virtual environment implementation source code," *Naurislv/P3-Behavioral-Cloning*, 2017.
[8] Wltoys. (2017). A969 RC Car. [Online]. Available: https://www.banggood.com/Wltoys-A969-Rc-Car-118-2_4Gh-4WD-Short-Course-Truck-p-916962.html
[9] Y. B. Wu, S. X. Jiang, Z. K. Xu, S. Zhu and D. H. Cao, "Lens distortion correction based on one chessboard pattern image," *Frontiers of Optoelectronics*, vol. 8, no. 3, pp. 319-328, 2015.
[10] OpenCV. (2017). Camera calibration. [Online]. Available: http://docs.opencv.org/2.4/modules/calib3d/doc /camera_calibration_and_3d_reconstruction.html
[11] Nauris Dorbe, "Camera calibration source code," Available: *Naurislv/P4-Advanced-Lane-Finding*, 2017.
[12] Nauris Dorbe. (2017). Experimental result video. [Online]. Available: https://www.youtube.com/watch?v=dtxsSOoHhQQ and https://www.youtube.com/watch?v=T5G2UocGi-s
[13] Kory W Mathewson, Patrick M Pilarski. (2017). Actor-critic reinforcement learning with simultaneous human control and feedback. [Online]. Available: https://arxiv.org/pdf/1703.01274.pdf

Nauris Dorbe was born in Talsi, Riga, 1992. Currently he is studying doctor studies in computer science. Presently, his professional occupation is machine learning engineer, assistant researcher. He is interested in space tech, self-driving cars, machine learning, enterprise big data solutions.

Ingars Ribners was born in Riga, Latvia. Currently he is working on PhD studies in computer science. Present his professional occupation is assistant researcher. He is interested in self-driving vehicles, cooperative driving and multiagent systems modeling.

Krisjanis Nesenbergs was born in Latvia, Riga, 1985. Currently he is working on PhD studies in computer science. Presently, his professional occupation is researcher. He is interested in human-computer interaction, CPS, wearable systems, artificial intelligence, sensor networks.