2017 American Control Conference
Sheraton Seattle Hotel
May 24–26, 2017, Seattle, USA

Learning a Deep Neural Net Policy for End-to-End Control of Autonomous Vehicles

Viktor Rausch[1,2], Andreas Hansen[1], Eugen Solowjow[2], Chang Liu[1], Edwin Kreuzer[2], and J. Karl Hedrick[1]

Abstract— **Deep neural networks are frequently used for computer vision, speech recognition and text processing. The reason is their ability to regress highly nonlinear functions. We present an end-to-end controller for steering autonomous vehicles based on a convolutional neural network (CNN). The deployed framework does not require explicit hand-engineered algorithms for lane detection, object detection or path planning. The trained neural net directly maps pixel data from a front-facing camera to steering commands and does not require any other sensors. We compare the controller performance with the steering behavior of a human driver.**

I. INTRODUCTION

Self-driving cars are expected to have a huge impact on the automotive industry. More than 90% of all car accidents are caused by human errors and only 2% by vehicle failures [1]. Self-driving cars can become a safer alternative to human drivers and save thousands of lives yearly. Tremendous efforts are undertaken in industry and academia on hardware and algorithmic research. Self-driving cars have to cope with different challenges such as recognition tasks (traffic, humans, road, lanes etc.), path planning and controls. For each of these tasks a vast amount of different sensor data is gathered and needs to be processed. In the last years, deep neural nets emerged as a promising method for several applications where complex data needs to be processed.

Neural nets are referred to as deep, if they possess a sufficiently large number of layers. Deep learning refers to training a deep neural net. Due to the complex network structure deep neural nets can be used to approximate or to learn highly nonlinear functions [2]. They are firmly established in machine learning and are commonly used in computer vision, speech recognition and text processing. The convolutional neural net (CNN) is a type of a deep neural net which is particularly well suited for image recognition [3]. They perform a dimensional reduction of a high-dimensional input through convolution. Convolutional neural nets lead to superior performance even on large data sets [4]. Applied to autonomous driving CNNs could in principle be used to learn all required road image features without any manual optimization or initialization.

Very recently deep learning was introduced as an end-to-end framework for control policy search [3], [5], [6], [7]. Unlike in classification tasks, the approximated function is continuous and maps complex sensor readings such as camera frames to control actions. Deep learning based end-to-end systems improved the policy performance significantly in several robotic tasks such as grasping [5]. Since explicit mapping of observations to control actions has not to be determined a priori in a design process anymore, deep learning turns out to be very efficient for certain high level controls objectives [6].

Our goal is to demonstrate a deep learning based end-to-end system that uses deep neural nets to encode a control policy for steering a car. Instead of engineering each autopilot component separately, we combine all efforts within one framework. Hence, lane detection, object detection, path planning and controls are not being explicitly considered. Camera systems allow to collect large amounts of information as frames. We use frames from a camera inside a vehicle as our control input which is comparable to the visual information a human driver receives. The controller (a CNN) maps the camera frames to steering angle. Thereby, the high level objective is lane following.

In summary, a deep learning based end-to-end controller could be a promising strategy for many problems related to autonomous driving.This paper intends to provide a basis and an overview for implementing a simple deep-learning system. Researchers associated with classical control techniques, specifically in autonomous vehicle research, are the main audience for this contribution. We describe a neural net architecture and how to train a deep neural net successfully for an end-to-end steering control system of autonomous vehicles. Thereby, we use a car simulator as the testbed. We also compare different kinds of solvers to generate the parameter updates during the net training.

The remainder of this paper is organized as follows. Section II presents the concept of deep control policies. Steering control algorithm design is carried out in Section III. Section IV present simulation results. Finally, conclusions are drawn in Section V.

II. DEEP CONTROL POLICIES

A control policy can be regarded as a generalization of a control law, whereby the mapping from inputs to outputs is not necessarily an analytical functional relation. It can either be deterministic or stochastic and maps observations

This work was supported by the German Academic Exchange Service (DAAD).

[1]Authors are with the Department of Mechanical Engineering, University of California, Berkeley, USA
a.hansen@berkeley.edu
changliu@berkeley.edu
khedrick@me.berkeley.edu

[2]Authors are with the Institute of Mechanics and Ocean Engineering, Hamburg University of Technology, Germany
viktor.rausch@tuhh.de
eugen.solowjow@tuhh.de
kreuzer@tuhh.de

to control actions. Recently CNNs have been deployed to encode control policies. However, CNNs have to be trained to generate useful control signals, this is usually referred to as policy search. In this paper, we deploy guided policy search.

Figure 1 demonstrates the two phases considered in this paper. During the training phase a human driver demonstrates the appropriate steering actions given a certain vehicle state. The control signal enters both the plant and the deep neural net. The plant output (e. g. position) is used by the human driver to steer the plant, but at the same is also available to the deep neural net. Once a policy has been trained through guidance, a deep neural net can regress sound control actions from data inputs it has not encountered before. The bottom part of Figure 1 illustrates the system structure after training.

One of the key advantages of such a control structure is the ability to directly exploit complex raw sensor information, such as vision. Hence, deep learned end-to-end systems do not require intermediate hand-engineered steps for feature detection.

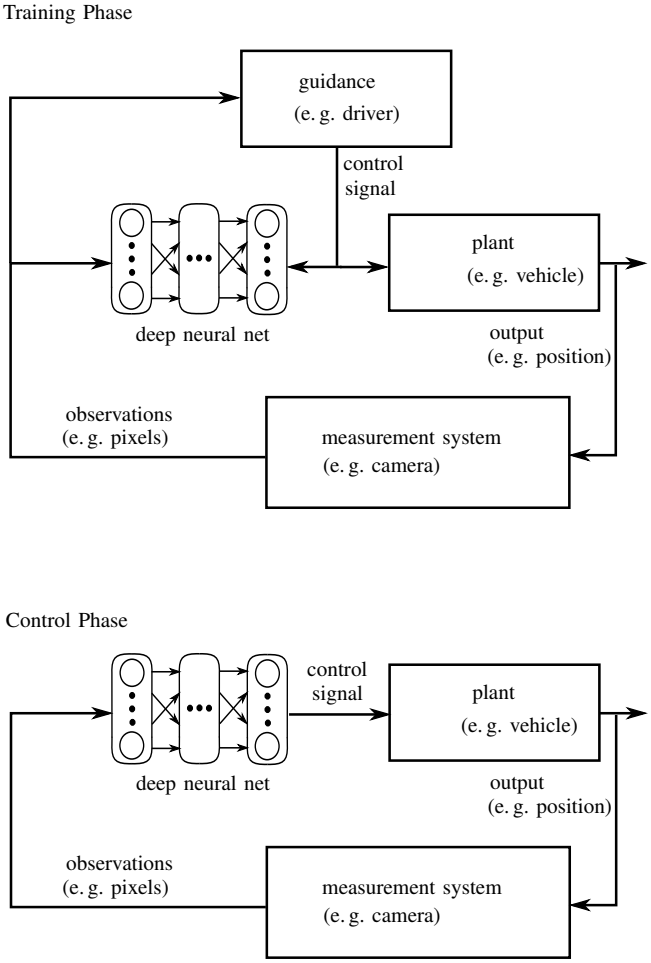


Fig. 1: Illustration of guided policy search within a control loop. During the training phase a guidance system presents the control actions for given observations. Observations and actions enter the deep neural net. During the control phase the deep neural net regresses sound control actions, even for unknown situations.

III. ALGORITHM DESIGN

This section presents the main aspects of the deep neural net policy algorithm.

A. Neural Networks

A neural net consists of different interconnected layers. A network is called a deep neural net, if the net has at least two hidden layers. We use convolutional layers, as the first hidden layers. They abstract features from the training data by themselves and are therefore very well suited for image recognition [3]. Convolutional neural nets use filters to learn features. These filters are called kernels. A kernel takes a segment of the input image and weights it with a parameter that is distinct for each kernel. These parameters are called weights and bias. Those weights and bias are iteratively optimized during the training phase.

The training data set consists of input values or vectors (e. g. an image) and the labeled output value or vector (e. g. steering angle). The last layer of the training net is a Euclidean loss layer. This layer calculates the loss E, which is given by

$$E = \frac{1}{2N} \sum_{n=1}^{N} \|\hat{y}_n - y_n\|_2^2 \tag{1}$$

between the computed value $\hat{y}$ and the labeled value y after an iteration with the batch size N. A batch is a subset of input frames. The loss is required for the optimization algorithm to minimize the loss of the next iteration and to update the weights and bias [8]. The loss function L depends on the weights W and is given by

$$L(W) = \frac{1}{D} \sum_{i=1}^{D} E_i + \lambda r(W), \tag{2}$$

where D is the number of the entire training data set entries, E_i is the Euclidean loss and $\lambda r(W)$ is a regularization term weighted by λ. The training data set can be very large therefore we use an approximation for a batch N of the training data set with $N \ll D$ instead. The new loss function is given by

$$L(W) \approx \frac{1}{N} \sum_{i=1}^{N} E_i + \lambda r(W). \tag{3}$$

In every forward pass the model computes the loss E and in the backward pass the gradient ∇E. The weight updates ΔW dependent on ∇E, the regularization gradient $\nabla r(W)$, and other for each method specific parameters [9]. Typically stochastic gradient descent (SGD) or similar methods are used. The SGD implementation updates the weights via

$$\Delta W_{t+1} = \mu \Delta W_t - \alpha \nabla L(W), \tag{4}$$

$$W_{t+1} = W_t + \Delta W_{t+1}, \tag{5}$$

where the momentum μ and the learning rate α are parameters, that might need to be tuned [10]. A typical value for the momentum is $\mu = 0.9$ and the learning rate should start

at $\alpha = 0.01$ to 0.001 and should drop down after a certain number of iterations [4]. The described optimization can be stopped as soon as the performance of the neural network on the training data set is satisfactory.

B. Implementation

The implementation environment consists of two main parts. The first part simulates the vehicle dynamics and generates the training data and the second part deals with the neural net algorithm.

1) Car simulation: For a first safe policy training step the simulation environment CARSIM is deployed, which is a commercial tool for simulating car behavior. CARSIM computes the vehicle dynamics based on the driver inputs such as steering or braking. The driving environment (road course, traffic, wind) is freely selectable. Screen captures and matching steering angles can be easily recorded via CARSIM and fed to a neural net.

2) Deep neural network framework: We use CAFFE to implement the deep neural net. It is a C++ based library with MATLAB and Python interfaces. Developed at UC Berkeley CAFFE supports training, testing, and fine tuning of neural nets and is usable with CPU and for faster computation with GPU [9].

C. Data collection

In order to train a CNN we need labeled frames, therefore each road screen capture requires a matching steering angle. A human driver steers a car with a joystick wheel in a CARSIM simulation to provide the labeled data for the training step. Here, the considered scenario includes a two lane road without traffic. We sample the simulation with 12 frames per second (FPS). More FPS would only lead to more similar frames without more relevant information [3]. The car drives with a constant speed of 60 km/h to further simplify the scenario. The steering angles vary between $-45°$ and $40°$ and the frame resolution is 1912×1036 pixels. The used steering angle is the wheel steering angle

$$\phi_w = R \cdot \phi_t, \tag{6}$$

where ϕ_t is the tire steering angle and R is the steering ratio [11]. The car model has a steering ratio of $R \approx 15$. After completing the data collection, we reject the non-proper frames by hand. Non-proper frames result from bad human driving behavior or graphic errors in the CARSIM simulation.

Figure 2 shows one sample screen capture used during the training phase. Every pixel has a value between 0 and 255. As common practice we normalize each pixel to values from 0 to 1 and the steering angles to values from -1 to 1. As a result the training loss function will converge faster. If the steering angles are not scaled the loss could become too large, and might influence the convergence of the training loss function. The larger the frames the more computational power is required for training the net. Therefore, we scale the frame size to 190×100 pixels. In order to solve the regression task we generate a hierarchical data format (HDF5) file which includes the scaled and normalized frames and steering angles. This file is the input for the training process in CAFFE. We use a 15 minute driving video to train the net resulting in 10,800 training frames.

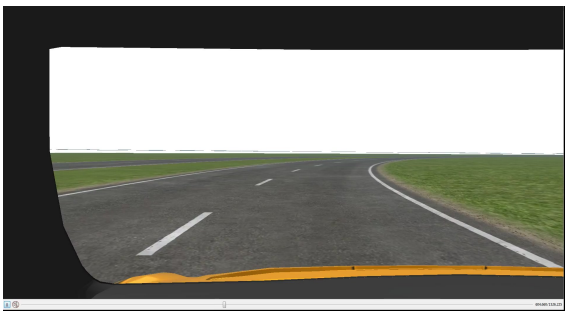


Fig. 2: An exemplary screen capture of a right curve, generated with CARSIM. The steering wheel angle is $-10.17°$.

D. Network structure

The structure of the neural net determines the quality of the results. However, there is more than a single one appropriate net structure. Each task requires a specific net structure. Finding a good net is an iterative process. Some net structures do not lead to convergence of the loss function, some of them lead to convergence but the performance on unknown test data is still bad. We use a CNN with four hidden layers. Three convolutional layers and one fully connected layer. The input data has the size $190 \times 100 \times 3$ and we use a batch size of 128. The first two convolutional layers have a kernel size of 5×5. The first one has 20 feature maps as output and the second one 48. They are used to extract the features of the road frames. After the first two layers we use a pooling layer with 2×2 kernel to scale the frames. The third convolutional layer has a 3×3 kernel size. The fully connected layer has 500 outputs and the last layer has the steering angle as an output. After every layer we use a dropout, which is a regularization technique to prevent the neural net from overfitting [12]. In order to achieve faster learning we use the rectified linear unit (ReLU) instead of the sigmoid function as an activation function [13]. Figure 3 shows the described neural net structure.

IV. RESULTS

This section presents results for the previously described case study. We compare the deep neural net controller performance with a human driver on a novel data set and analyse different solvers.

A. Training of the neural net

We train the net on a CPU, which can be a long process. It can be accelerated by exploiting GPUs. There is no exact rule when to stop the training phase. As soon as the loss converges and does not decrease anymore the net is either trained well enough or the net structure is not suitable for the task at hand. For the case the net is already sufficiently trained and the training process is not stopped, overfitting can

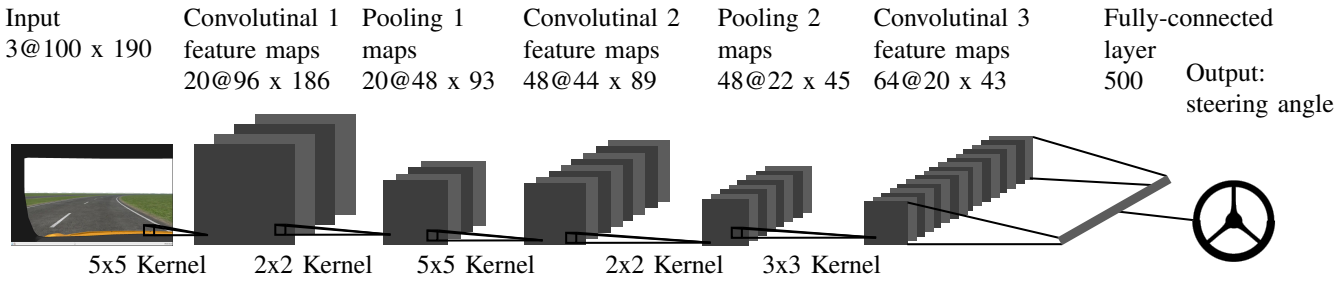


Fig. 3: The net structure of the designed neural net. The net consists of three convolutional layers, two pooling layers and one fully connected layer. We also use dropouts to prevent overfitting and the ReLU activation function.

occur even if dropouts are used. Figure 4 shows a sample loss curve over the training iterations. Note, after the 5,000th iteration the loss does not decrease noticeable, so the training process should be stopped.

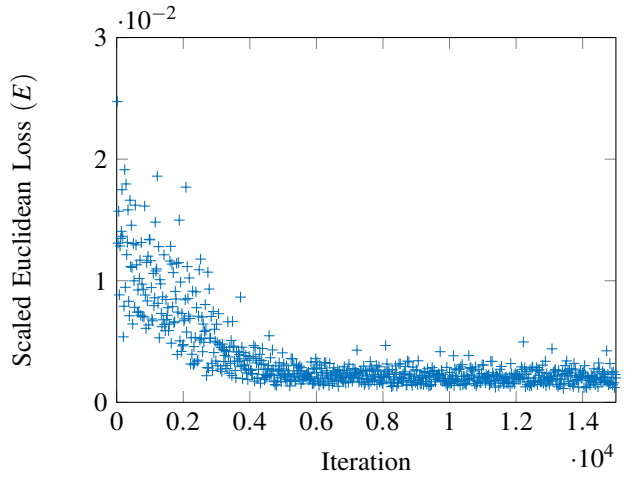


Fig. 4: The Euclidean loss between the predicted and labeled steering angle over the SGD iteration steps.

B. *Validation via autonomous steering for different solvers*

For the validation of the trained network we compare the network with the human driver steering angles in a novel condition and with different solvers. The road frames and steering angles for the validation have not been used during the training phase. Therefore, the trained neural net has never seen those road frames before and we are interested in the question whether the net generalizes well with respect to this data.

Different solvers are available for updating weights and bias after each iteration. We compare different solvers and analyze how they influence the training step. Each model is trained three times, each time with a different solver. We choose the typically used SGD solver, the Adam solver and the Nesterov's accelerated gradient (NAG) solver. The Adam solver computes adaptive learning rates for each parameter and is a gradient based optimization method, so is the SGD and NAG solver [14]. NAG achieves a high convergence rate for convex optimization problems but is also a choice for non-convex problems [15]. Figure 5 shows the Euclidean loss between the predicted and labeled steering angles scaled to the range of -1 to 1, for the different solvers. The three loss curves are Gaussian distributed and quite similar but each solver achieves another loss at the end. Table I shows the converged mean loss of each solver. The Adam solver achieves the lowest loss after 8,000 iterations with 0.0012.

TABLE I: Mean of the converged Euclidean loss between the scaled predicted and labeled steering angles on the training data.

	SGD	Adam	NAG
Converged loss mean	0.002	0.0012	0.0023
Converged standard deviation [$\cdot 10^{-3}$]	0.5541	0.4414	0.3528

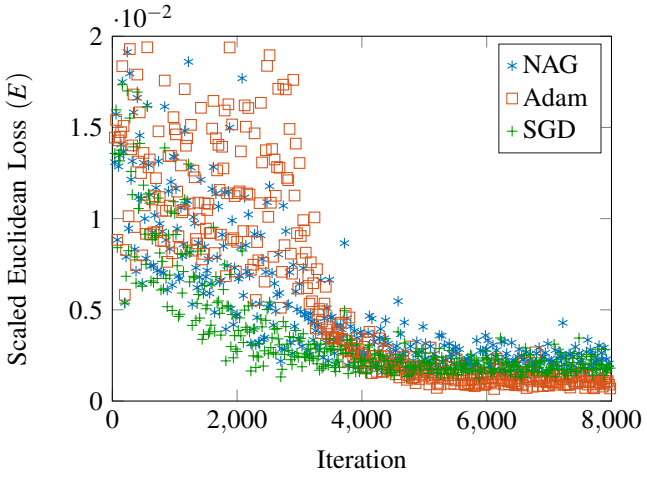


Fig. 5: The loss function over the training iterations for the NAG, Adam, and SGD solvers.

A low loss is a good indicator for a well trained neural net but the most important factor is how well the trained net performs on unknown data. Figure 6 shows the human driver steering angles on the validation data set and the steering angles computed by the neural net. The trained nets do not only perform well on the training data set but on the unknown novel data set as well. They show good performance after

only 4,000 iterations and a training data set consisting of 10,800 frames. Similar to the loss curves the steering angles are in agreement.

Table II shows the mean loss and standard deviations of the results. Even if the Adam solver has the best loss curve, the performance on the validation data set of the NAG trained net shows the best results for unknown road images. Despite the fact that the trained nets perform well on unknown data the mean loss on the validation data is higher than on the training data by one magnitude. However, this is expected and a typical machine learning phenomenon because the average fit of the model is always worse for novel unseen data than for data which was used for training.

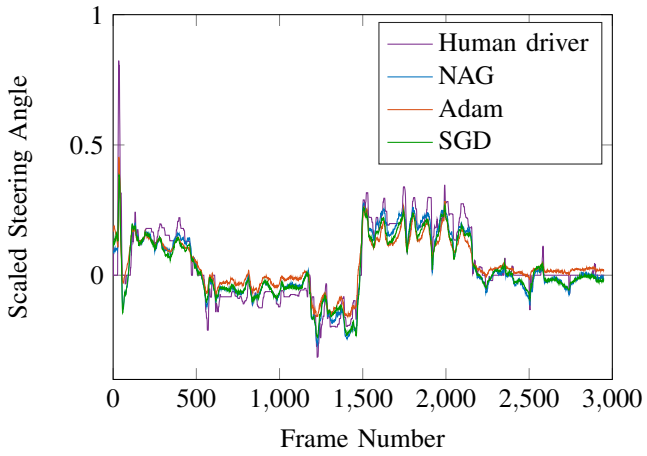


Fig. 6: Comparison of the human driver and the neural net predicted steering angle on a for the neural net unknown data set.

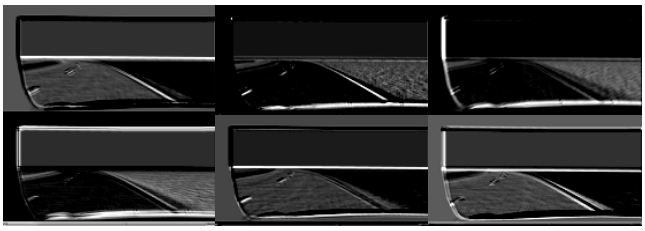
TABLE II: Mean and standard deviation of the Euclidean loss between the scaled predicted and labeled steering angles on the validation data.

	SGD	Adam	NAG
Mean error	0.0395	0.0465	0.0338
Standard deviation	0.0374	0.0391	0.0346

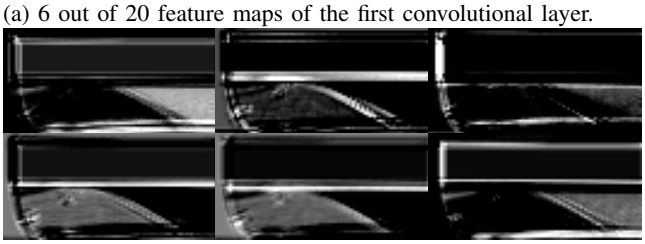
Figure 7 shows six feature maps associated with each of the convolutional layers. The network starts with random weights and bias and adapts from the examples without any manual optimization or initialization. It can be clearly seen that the convolutional layers learn the important features of the road frames during the training.

V. CONCLUSIONS AND FUTURE WORK

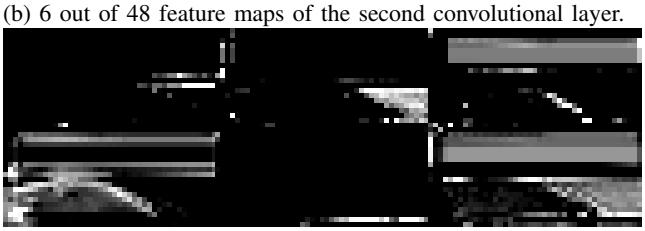
We presented a deep convolutional neural net for autonomous vehicle steering. The deep neural net served as a controller and was trained with human guidance. During the training phase road screen captures and human drivers steering angles, which have been generated with the car simulator CARSIM, were deployed. The neural net framework CAFFE was used to build the net structure and train the



(a) 6 out of 20 feature maps of the first convolutional layer.



(b) 6 out of 48 feature maps of the second convolutional layer.



(c) 6 out of 64 feature maps of the third convolutional layer.

Fig. 7: Six feature maps of each convolutional layers. The neural network learns all the important road features just from the examples and without any manual optimization or initialization. We increased the contrast of the gray scale frames for a better visualization.

net. The solver selection is an important factor in training neural nets. We showed that a good loss function curve is not necessarily an indicator for a well trained neural net. The Nesterov's accelerated gradient solver performed best on our net architecture. The convolutional layers learned after a few thousand iterations to detect important road features without any hand-engineered features. On the one hand the trained neural net shows good performance on an unknown validation data set although we used only a small training data set. On the other hand the mean loss on the validation data set could be further improved with more training data or a different net structure. We demonstrated how to learn a deep neural net policy for end-to-end steering control of autonomous vehicles. Remarkably the performance is achieved by just exploiting raw sensor data of a single camera as the the input of the control logic. In future work we will scale our approach by using more data, more efficient and deeper nets. Furthermore, the approach will be implemented on a real car. In order to collect training data which is more proper than the human driver steering angles, we will use a sliding mode controller or a model predictive controller instead the human driver. With the resulting better training data set we should increase the stability of our end-to-end system and reach a higher convergence rate. We expect a growing interest in deep learning within the traditional controls community. While deep learning seems to handle

raw sensor data and high level objectives well, there are no formal guarantees regarding stability or convergence. The authors are convinced that a combination of classical control theory and deep learning will result in a new generation of controllers.

ACKNOWLEDGMENT

The financial support received from the German Academic Exchange Service (DAAD) is gratefully acknowledged. Thanks to e.g. Ziya Ercan from the Department of Mechanical Engineering, University of California, Berkeley, USA, Gregory Kahn from the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, USA, and Wenshuo Wang from the Mechanical Engineering Department, Beijing Institute of Technology, Beijing, China for their technical support and helpful comments.

REFERENCES

[1] S. Singh, "Critical Reasons for Crashes Investigated in the National Motor Vehicle Crash Causation Survey," NHTSAs National Center for Statistics and Analysis, Tech. Rep., 2015.

[2] T. Liu, S. Fang, Y. Zhao, P. Wang, and J. Zhang, "Implementation of Training Convolutional Neural Networks," *arXiv preprint arXiv:1506.01195*, 2015.

[3] 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.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[5] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *Journal of Machine Learning Research*, vol. 17, no. 39, pp. 1–40, 2016.

[6] T. Zhang, G. Kahn, S. Levine, and P. Abbeel, "Learning Deep Control Policies for Autonomous Aerial Vehicles with MPC-Guided Policy Search," *arXiv preprint arXiv:1509.06791*, 2015.

[7] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous Methods for Deep Reinforcement Learning," in *International Conference on Machine Learning*, 2016.

[8] Net-Scale Technologies, Inc. (2016, August) Autonomous Off-Road Vehicle Control Using End-to-End Learning. Final technical report. [Online]. Available: http://net-scale.com/doc/net-scale-dave-report.pdf

[9] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional Architecture for Fast Feature Embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[10] L. Bottou, "Stochastic gradient descent tricks," in *Neural Networks: Tricks of the Trade*. Springer, 2012, pp. 421–436.

[11] R. Rajamani, *Vehicle Dynamics and Control*. Springer Science & Business Media, 2011.

[12] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[13] A. Radford, L. Metz, and S. Chintala, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks," *arXiv preprint arXiv:1511.06434*, 2015.

[14] D. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[15] I. Sutskever, J. Martens, G. E. Dahl, and G. E. Hinton, "On the importance of initialization and momentum in deep learning," *ICML (3)*, vol. 28, pp. 1139–1147, 2013.