Revisiting End-to-End Deep Learning for Obstacle Avoidance: Replication and Open Issues

Alexander K. Seewald[1]
[1]*Seewald Solutions, Lärchenstraße 1, A-4616 Weißkirchen a.d. Traun, Austria*
alex@seewald.at

Keywords: Deep Learning, Obstacle Avoidance, Autonomous Robotics

Abstract: Obstacle avoidance is an essential feature for autonomous robots. It is usually addressed with specialized sensors and Simultaneous Localization and Mapping algorithms (SLAM, Cadena et al. (2016)). Muller et al. (2006) have demonstrated that it can also be addressed using end-to-end deep learning. They proposed a convolutional neural network that maps raw stereo pair input images to steering outputs and is trained by a human driver in an outdoor setting. Using the ToyCollect open source hardware and software platform, we replicate their main findings, compare several variants of their network that differ in the way steering angles are represented, and extend their system to indoor obstacle avoidance. We discuss several issues for further work concerning the automated generation of training data and the quantitative evaluation of such systems.

1 INTRODUCTION

Obstacle avoidance is an essential feature for autonomous robots, the lack of which can make the estimation of the robot's intelligence drop dramatically. As benchmark task it has been known from the beginning of autonomous robotics and is usually solved with specialized sensors and Simultaneous Localization and Mapping algorithms (SLAM, Cadena et al. (2016)).

An intriguing but less well researched way to implement obstacle avoidance is using end-to-end deep learning to capture the obstacle avoidance skill of a human operator. One advantage here is the ability fo use cheap and power-efficient cameras – or in fact arbitrary sensors – for obstacle avoidance rather than more commonly used laser-range sensors. This is the approach by Muller et al. (2006) and Muller et al. (2004) almost fifteen years ago. They used a convolutional neural network with six layers that directly processes YUV input frames from a stereo camera pair and outputs a steering angle to control the robot. Their network has around 72,000 tunable parameters and about 3.15 million multiply-add operations (MAC).

Here, we revisit deep learning for obstacle avoidance to determine whether it is possible to replicate their main findings. Contrary to other fields such as biological, medical or psychological research, the need for replication studies as a tool for validating existing research approaches has not yet been adequately understood for Deep Learning.[1] However, the complexity of Deep Learning systems are such that to determine more exactly what makes a certain combination of data, network structure and training/testing methodology reliably work for a certain well-defined task is of utmost importance to better understand these systems and their limitations. In fact we found that although the original paper is quite optimistic and states "Very few obstacles would not have been avoided by the system" (Muller et al. (2006): caption of Fig. 3), much later one of the original authors states that "DAVE's mean distance between crashes was about 20 meters in complex environments" (Bojarski et al. (2016): Introduction, last sentence of paragraph 5) both of which cannot be easily true. Our results strongly support the latter.

Improvements in hardware and software have made several optimizations feasible. Firstly, it is no longer necessary to send the stereo camera images to another machine for processing. Even small embedded platforms such as Raspberry Pi (RPi) 4 can now run such small deep-learning networks in real-time.

Secondly, due to miniaturization it is now feasible to build small robots ($<$ 10x10x10cm) with similar capabilities and test the same approach on indoor ob-

[1]This can be inferred from some reviewers' comments.

Table 1: Robot Hardware Overview

Type	Robot Base	Motors	Motor controller	Camera	Controller	Chassis	Power	LocalDL
v1.3 (*K3D*)	Pololu Zumo (#1418)	2x Pololu 100:1 brushed motors (#1101)	Pololu Qik 2s9v1 Dual Serial (#1110)	1x RPi Camera Rv1.3 w/ Kúla3D Bebe Smartphone 3D lens[2]	1x RPi 3B+	Modified transparent chassis[3]	4x 3.7V 14500 LiIon	Yes 8fps
v1.21 (*R2X*)	Pololu Zumo (#1418)	2x Pololu 100:1 brushed motors (#1101)	Pololu Qik 2s9v1 Dual Serial (#1110)	2x RPi Camera Rv2.1	2x RPi ZeroW	Three-part 3D-printed chassis[4]	4x 3.7V 14500 LiIon *or* 4x 1.5V AA Alkaline *or* 4x 1.2V AA NiMH	No <1fps
v2.2 (*OUT*)	Dagu Robotics Wild Thumper 4WD (#RS010)	4x 75:1 brushed motors (included)[5]	Pololu Qik 2s12v10 Dual Serial (#1112)	2x RPi Camera Rv2.1	1x RPi CM3 on official eval board	None (open case)	1x 7.2V 2S 5000mAh LiPo	Yes 8fps

stacle avoidance, which has specific challenges and issues that differ from outdoor obstacle avoidance.

Thirdly, the general availability of stable and fast deep-learning frameworks such as Torch, Tensorflow and TensorflowLite (for RPi platforms) has made it much easier to train such networks now. In 2012 it was far more difficult to train convolutional neural networks such as LeNet for handwritten digit recognition (see Seewald (2012)). Now, a simplified version of LeNet is part of the sample code for Tensorflow.

We have built compatible robots within our Toy-Collect open-source hardware and software plattform, and used them to collect training data for this task and deploy the trained models in the field. We ported the original deep learning network from Muller et al. (2006) to Tensorflow as precisely as possible[6] and ran several learning experiments to determine how to best represent steering angles. Lastly, we deployed the trained models on two different robots and analyzed their obstacle avoidance behaviour qualitatively in a variety of settings.

Finally we compared our results and experiences with those mentioned in the original paper and technical report, and discuss relevant issues for future work, then conclude the paper.

2 RELATED RESEARCH

Muller et al. (2006) describe a purely vision-based obstacle avoidance system for off-road mobile robots that is trained via end-to-end deep learning. It uses a six-layer convolutional neural network that directly processes raw YUV images from a stereo camera pair. For simplicity's sake we will henceforth refer to this network (resp. our as-precise-as-possible approximation) as *DAVE*[7]*-like*. They claim their system shows the applicability of end-to-end learning methods to off-road obstacle avoidance as it reliably predicts the bearing of traversible areas in the visual field and is robust to the extreme diversity of situations in off-road environments. Their system does not need any manual calibration, adjustments or parameter tuning, nor selection of feature detectors, nor designing robust and fast stereo algorithms. They note some important points w.r.t. training data collection which we have also noted for our data collection.

Muller et al. (2004) extend the previously mentioned paper with additional experiments, a much more detailed description of the hardware setup, and a slightly more detailed description of the deep learning network. A training and test environment is described. An even more detailed description of how training data was collected is given. They found that using just information from one camera performs almost as good as using information from both cameras, which is surprising. Modified deep learning networks which try to control throttle as well as steering angle and also utilize additional sensors performed disappointingly.

Bojarski et al. (2016) describe a system similar to Muller et al. (2006) that is trained to drive a real car using 72h of training data from human drivers. They note that the distance between crashes of the original *DAVE* system was about 20m which is roughly comparable to what we observed in our tests. They add artificial shifts and rotations to the training data – something we also could have done. They report an autonomy values of 98%, corresponding to one human intervention every 5 minutes. However, their focus is on lane following and not on obstacle avoidance. They used three cameras – left, center, right – and a more complex ten layer convolutional neural network. It could still be interesting to test their network on our task of obstacle avoidance, however their network is about ten times bigger, precluding real-time performance on the smaller RPi platforms.[8]

[2] A double periscope that divides the camera optical path into two halves and moves them apart using mirrors, effectively creating a stereo camera pair from one camera.

[3] Sadly, no longer available

[4] Can be plotted as one part for perfect alignment.

[5] Left two and right two are connected.

[6] Some ambiguity remains, see Section 5.

[7] Named after the **D**ARPA **A**utonomous **VE**hicle project.

[8] Except possibly RPi4 pending further testing.

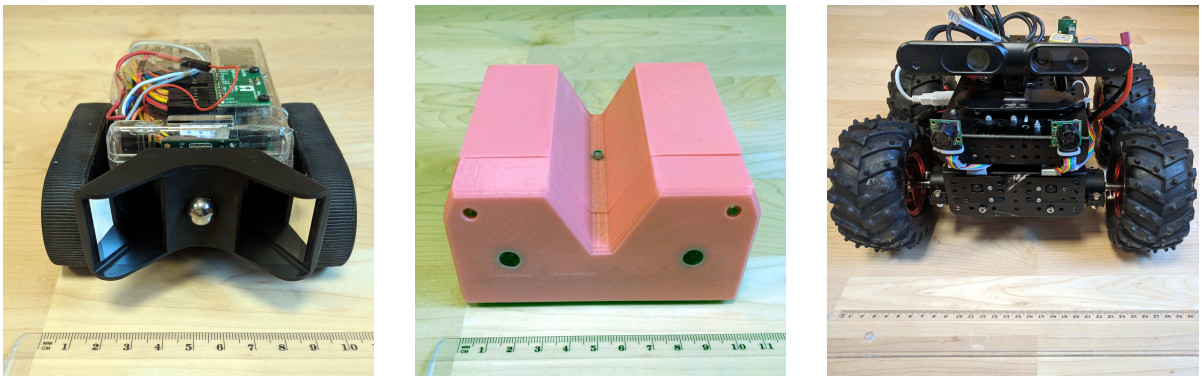

Figure 1: Robots *K3D*, *R2X*, *OUT* (left to right, ruler units: cm)

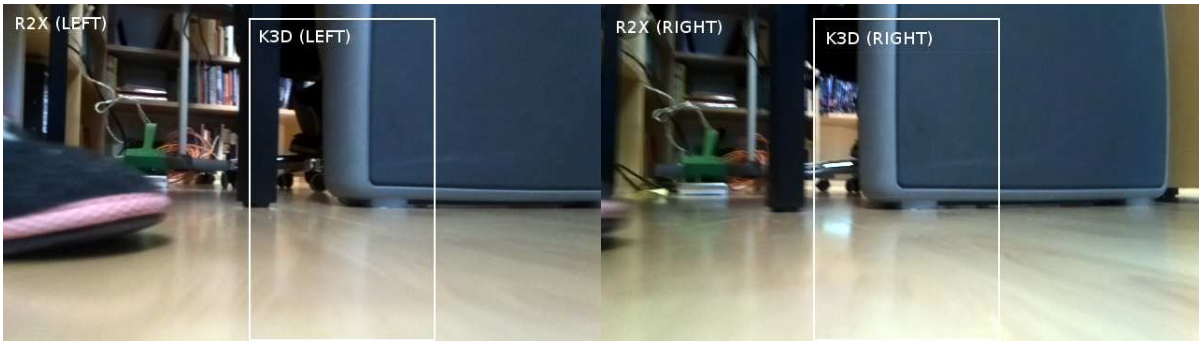

Figure 2: FOV comparison between robot *R2X* and *K3D*

Pfeiffer et al. (2016) describe an end-to-end motion planning system for autonomous indoor robots. It goes beyond our approach in also requiring a target position to move to, but uses only local information (similar to our approach). However, their approach uses a 270° laser range finder and cannot be directly applied to stereo cameras. Their model is trained using simulated training data and as such has some problems navigating realistic office environments.

Hartbauer (2017) describes a system for collision detection inspired by the known function of the collision-detecting neuron (DCMD) of locusts. No machine learning takes place. The necessary computational power for applying this algorithm is very low compared to our trained network and of course no training data must be collected. His approach can be used with a single camera and even computes avoidance vectors. One disadvantage of this approach is that it can only be applied once the robot is moving.

Wang et al. (2019) describe a convolutional neural network that learns to predict steering output from raw pixel values. Contrary to our approach, they use a car driving simulator instead of real camera recordings, and they use three simulated cameras instead of our two real cameras. They propose a slightly larger network than the one we are using and explicitly address overfitting and vanishing gradient which may reduce the achievable performance also in our case.[9] They note several papers on end-to-end-learning for autonomous driving including Muller et al. (2006) – however it should be noted that most of the mentioned papers are concerned with car driving and lane following, and not with obstacle avoidance, which are overlapping but different problems.

Khan and Parker (2019) describe a deep learning neural network that learns obstacle avoidance in a class room setting from human drivers, somewhat similar to our system. As starting point they use a deep learning network that has been trained on an image classification task and reuse some of the hidden layers for incremental training. However, their approach used only one camera and cannot be directly applied to stereo cameras. Still, their results seem promising and will be considered for future experiments.

3 TOYCOLLECT PLATFORM

All experiments were conducted with our ToyCollect robotics open source hardware/software platform

[9] Especially for the smaller *outdoor* dataset.

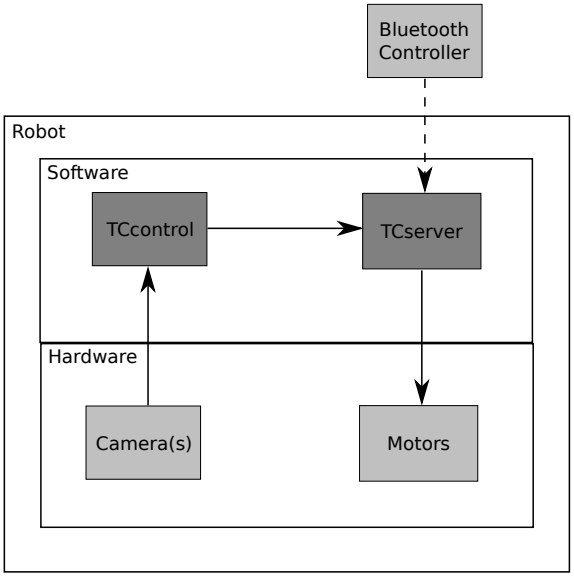

Figure 3: Local Deep Model processing (robots *K3D*, *OUT*)

(`https://tc.seewald.at`). A hardware overview of the three utilized robots can be found in Table 1 while Fig.1 shows robot images.

While *OUT* and *K3D* need only a single main controller and have sufficient computational power to run a DAVE-like model at interactive frame rates (around 8 frames-per-second) thus enabling onboard processing, the need for two cameras on *R2X* necessitates the usage of two controllers, each connected to one camera, and each streaming the frames independently to a processing server.

OUT and *R2X* each use two cameras with a field-of-view of 62.2° horizontal and 48.8° vertical, however for *K3D* we used the older camera which has only 54° horizontal and 41° vertical field-of-view and the horizontal viewing angle is approximately halved again by the 3D smartphone lens. Fig.2 shows the difference in field-of-view between *K3D* and *R2X*. Because of the high spatial distortion of *K3D* when scaling to 149x58 we only ran preliminary experiments but excluded this robot from the final evaluation. In Muller et al. (2006), cameras with FOV of 110° were used. However the nearest equivalent would have been 160° cameras which we could only have used on *OUT* as their flat ribboncable connectors are rotated by 180° which would have necessitated a complete redesign of *R2X*.

OUT also includes a depth camera, GPS module, a 10-DOF inertial measurement unit including an accelerometer, gyroscope, magnetometer and a barometer for attitude measurement as well as a thermometer, and four ultrasound sensors. However these were not used for our experiments.

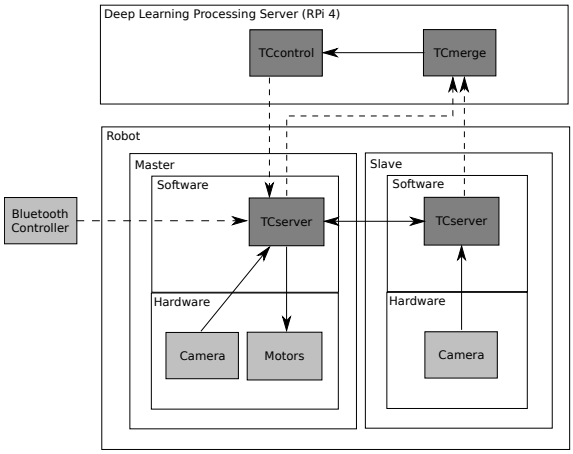

Figure 4: Remote Deep Model processing (robot *R2X*)

R2X includes two high-power white LEDs to allow operation in total darkness and whose brightness can be controlled in 127 steps. However these also were not used for our experiments.

Fig.3 shows the overall architecture for local processing. *TCserver* denotes the robot control program which is responsible for driving the motors, accepting commands[10] and optionally streaming uncompressed video to the respective controller. It also allows to collect uncompressed video and steering direction for deep learning training data. *TCcontrol* starts the camera(s) (configured to output uncompressed YUV video), processes camera input via TensorflowLite and a locally stored model, and sends appropriate commands to *TCserver* using a simple socket-based 3-byte command interface.[11] The *Bluetooth Controller* implements an override that allows to control the robot manually, overriding *TCcontrol* commands, in order to move the robot to an uniform starting position or stop it before it crashes into an undetected obstacle.[12]

Unfortunately it is not possible to connect two cameras to a RPi controller since the necessary connections are only available on the chip and not on the PCB. Only the ComputeModule allows to directly connect two cameras; however the official evaluation boards for the RPi compute module are too large to

[10] We have implemented touch-based controls on Android mobile phones (using one or two fingers); using foot gas pedal, brake and steering wheel; using head-movements from Google Cardboard; and using Bluetooth gaming controllers. Here we used only the last option.

[11] Basically, speed, direction and a sychronization byte.

[12] Bluetooth was already available on *K3D* and *R2X* and has lower latency than Wi-fi. Also this way the control would not interfere with video streaming where necessary. We paired each robot to a specific Bluetooth controller for easy testing.

fit on the small robot. So for robot *R2X* we integrated two RPi Zero Ws into a single robot chassis and connected each to a separate camera.[13] However since the RPi Zero W is based on the original RPi 1, it is much too slow for online deep learning model processing and achieves less than 1 frames per second. So for this robot we must stream the video frames to a second more powerful plattform.[14]

Fig.4 shows the overall architecture for remote processing. On each of the two RPi Zeros runs *TCserver*. One of them, *Master*, is connected to the motors and controls them as well as to one camera. The other, *Slave*, is just connected to the camera and via two logic-level GPIO lines to the *Master*. *Master TCserver* detects the motor controller[15] and is the only one receiving commands via Bluetooth or Wi-fi from external sources. When receiving the start command from *TCcontrol*, both *TCserver* processes start their cameras and send the uncompressed YUV frames via Wi-fi to *TCmerge* on the processing server. *TCmerge* is responsible for combining the frames and analyzing timestamps to ensure synchronization, throwing away combinations of frames that are temporally too far apart. The combined frames are then sent locally via Linux pipe to *TCcontrol*, where they are processed exactly in the same way as in the local processing scenario and the computed commands are sent back via Wi-fi. A slightly higher latency is observed, however as long as both robot and deep learning processing server are in the same Wi-fi network, frame rates of up to 8 fps at 416x240 resolution[16] can be achieved using this approach. The main limitation is the Wi-fi transfer rate for uncompressed YUV frames. According to our measurements, the RPi4 can achieve up to 30 fps on DAVE-like deep learning models and up to 8 fps on MobileNet v2.[17]

[13]In the meantime other options have become available, e.g. StereoPi, which we are currently evaluating.

[14]It would however be feasible to add a Raspberry Pi 4 on top of the robot by adapting the chassis. However, using StereoPi and a CM3+ would give almost the same performance at more managable thermal load. The RPi 4 is quite hard to keep cooler than 60° C – the temperature at which the PLA chasiss would melt.

[15]I.e. the same progam runs on both Master and Slave and either role is dynamically detected during startup.

[16]This resolution gives the most similar aspect ratio to the originally collected 640x368.

[17]NVidia's Jetson Nano is far more powerful and could be switched for the RPi 4 in this setting quite easily. It can however not easily be used on most of our robots because 1) It only has one camera connector, 2) Even with active cooling it easily reaches temperatures that melt PLA, so an ABS or metal chassis would have to be built. However, we could still put it on robot *OUT*.

4 DATA COLLECTION

The original datasets used to train DAVE are – as far as we know – not available. We also attempted to find other datasets compatible with our robot platforms and focussed on obstacle avoidance – however because of the non-mainstream task and a focus in the research community on autonomous car driving which is concerned mainly with lane following, we were unable to find any other suitable datasets. So we finally collected two different types of data (consisting of frames and speed/direction control input[18]) ourselves for indoor and outdoor obstacle avoidance. In each case we aimed for a consistent avoidance behaviour roughly at the same distance from each obstacle similar to that described in Muller et al. (2006). However, instead of collecting many short sequences, we collected large continuous sequences and afterwards filtered the frames with a semi-automated approach. All data was collected by students during summer internships in 2017, 2018 and 2019. The students were made aware of the conditions for data collection[19], and were supervised for about one fifth of recording time.

For *indoor* obstacle avoidance (robots *R2X*, *K3D*) we collected 267,617 frames in a variety of indoor settings. These were collected directly onboard *R2X* robots in uncompressed YUV 4:2:2 format on SD cards in 640x368 resolution at 10fps. Control of the robot was via paired Bluetooth controller. We first inspected the recorded sequences manually and removed those with technical issues (e.g. no movement, cameras not synchronized, test runs). Because of synchronization issues the first minute of each sequence (up to the point when each MASTER and SLAVE synchronize with an external time server[20]) had to be removed. Additionally, sequences with very slow speed and with backward movement (negative speed) were removed along with 50ms of context. Since both cameras were independently recorded, we also removed all frames without a partner frame that is at most 50ms[21] apart. We also removed frames where movement information is not available within $\pm$25ms of the average timestamp for the image frames. Lastly, we had to remove 80% of the frames with straight forward movement as other-

[18]Only direction (steering) is used for training.

[19]See Section 2

[20]The RPi platform does not offer a realtime clock and thus suffers from quite significant clock drift. It would however have been quite simple to synchronize local clock when MASTER and SLAVE synchronize during startup but we forgot.

[21]I.e. half of 100ms which corresponds to the 10fps recording frequency.

wise these would have dominated the training set. After all these filters, 70,745 frames remained which we distributed into 13,791 (20%) frames for testing and 56,954 (80%) for training.

For *outdoor* obstacle avoidance (robots *OUT*) we collected 66,057 frames in a variety of outdoor settings. These were collected on a mobile phone connected via Wi-fi to an *OUT* robot. The phone also translated the phone-paired Bluetooth controller commands to Wi-fi and sent them to the robot. These steps were necessary since at that time no Compute Module with significant onboard memory was available.[22] The frames were collected in 1280x720 resolution in raw H264 format at 15fps. We used the same filtering as above and obtained 27,368 frames which we distributed into 5,351 (20%) for testing and 22,017 (80%) for training.

In both cases the frames were downscaled to 149x58 resolution via linear interpolation (ignoring aspect ratio) and split into equal-sized Y, U, and V components for left and right camera.

5 RESULTS

For training, we used TensorFlow with AdamOptimizer and a learning rate of 10^{-4}. We aimed to reproduce the original model described in Muller et al. (2006) and Muller et al. (2004) (p.28, Fig. 34) as precisely as possible. However there were two points which we could not find out by reference to the original papers.[23] We did not manage to get the number of trainable parameters exactly right, however these differ even between Muller et al. (2006) and Muller et al. (2004).

Firstly, according to Muller et al. (2006) the third layer is connected to various subsets of maps in the previous layer. The paper does not state *which* subsets, only that there are 24 maps and 96 kernels. Since the previous layer contains one map depending only on image data from the left camera (L), one map depending only on image data from the right camera (R), and four maps depending on image data from both cameras (4x A), we chose our 24 maps like this:

- L, R and all 2-element subsets from A (6 maps)
- L and all 3-element subsets from A (4 maps)
- R and all 3-element subsets from A (4 maps)
- All four A maps (1 map)
- L and all A maps (1 map)
- R and all A maps (1 map)
- L, R and all 3-element subsets of A maps (4 maps)
- All 3-element subsets from A which contain the first A map (3 maps)

This configuration yields the same number of kernels and maps as in the original paper. Only the last 3 maps are somewhat arbitrarily chosen – for symmetry we would have added three maps to get all 3-element subsets from A in the last set.

Secondly, it was unclear how the two outputs for steering were represented. We assumed a regression task where each unit encodes the steering angle in one direction (always positive between 0 and 1) while the other unit would be set to zero. Straight forward movement would be represented by both units being set to zero. This corresponds to *Reg2* in the results table. We also formulated steering as a classification problem, representing left, right and straight forward as distinct classes. Left and right were determined at a steering output of ± 65 which corresponds to half the maximum steering output of ± 127. This variant corresponds to *Cl3*. For completeness sake we also added a variant with a regression task and single output unit that directly predicts the steering output (± 127). This was called *Reg1*.

Table 2 shows the results of the different variants *Reg1*, *Reg2* and *Cl3* which only differ in the number and interpretation of output units, on the two datasets *indoor* and *outdoor*. Steering output is measured from hard left (-127) to hard right ($+127$) with 0 being straight forward movement. Columns Acc. $\pm X$ show the corresponding accuracy when using a bin of $0 \pm X$ for the center class, and defining left and right class accordingly. For estimating steering output from the *Cl3*, we used a weighted sum of the estimated probabilities for (left, center, right) where each class is weighted by double the threshold initially used to defined the classes (here $65 * 2 = 130$). Even this very crude method to estimate steering output outperforms both *Reg1* and *Reg2* by a large margin on accuracy, and for *indoor* even on correlation.[24]

Figure 5 shows the predicted steering output for *R2X* and *OUT* versus the human steering output on the test set. The first 2,000 samples are shown.

[22] The RPi ComputeModule evaluation board offers neither an SD card slot nor Bluetooth and the one available USB slot was needed for a Wi-fi USB stick.

[23] We also did not receive an answer to our personal request for clarification.

[24] Center bin size during training of *Cl3* was ± 65. For *Reg1* and *Reg2* – because of the formulation as regression problem – no bin size was used and raw steering output was trained (normalized to [0,1] for *Reg2* and [-1,1] for *Reg1* via division by 127)

Table 2: Results of learning experiments

Dataset	#Samples	Model	#Steps until convergence	Correlation coefficent	Acc. w/ center ±65	Acc. w/ center ±48	Acc. w/ center ±32
indoor	70,745	*Cl3*	305.5k	0.3945	59.93%	59.57%	59.05%
indoor	70,745	*Reg2*	102.0k	0.2353	49.31%	49.51%	47.29%
indoor	70,745	*Reg1*	118.0k	0.2144	48.29%	47.82%	45.98%
outdoor	27,368	*Cl3*	69.5k	0.1807	58.10%	50.27%	40.87%
outdoor	27,368	*Reg2*	193.7k	0.2783	55.97%	49.34%	43.13%
outdoor	27,368	*Reg1*	219.9k	0.2885	54.68%	47.74%	41.43%

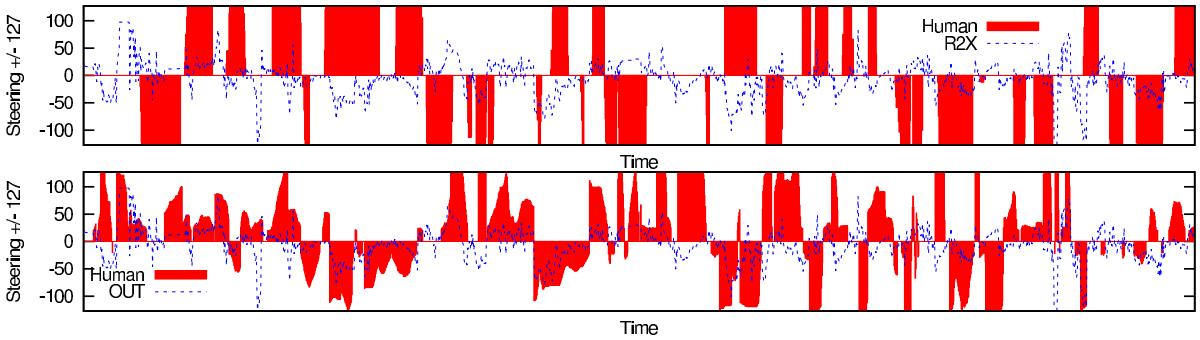

Figure 5: Steering output on test set for robots *R2X*, *OUT* (top-to-bottom)

6 QUALITATIVE EVALUATION

For simplicity and because the results of the learning experiments indicated that *Cl3* was the best model, we only deployed *Cl3*. Because of the very small FOV of *K3D*, we only deployed the indoor model to robot *R2X* and the outdoor model to robot *OUT*.

During testing *R2X* we found that when directly driving towards a wall, left and right directions are alternately activated strongly (obviously both directions are valid in this case) but cancel each other out since the robot cannot react so speedily to steering commands. Therefore we reduced the frame rate from 8 to 2 fps which improved the issue greatly at cost of less responsive reactions. Generally the obstacle avoidance performance is fair, however in some cases the robot steers in the right direction but steers back at the last moment. This may indicate that the training data does not include sufficient examples with very near obstacles. In some cases the steering action comes far too late and can only be observed by analyzing the video logs after each run. We also observed that sometimes only one frame from a sequence of driving towards a wall shows left or right steering output. Wall following behaviour was sometimes observed – in some cases over prolonged periods. In several cases complex obstacle avoidance behaviour sequences were observed (such as driving below an office chair without touching it) which indicates that even this simple CNN can learn surprisingly complex tasks. Still, the system cannot be considered fully autonomous. About once every minute a manual intervention was necessary (autonomy = 90%).

During testing *OUT* we found the model to perform quite well and exhibit fair to good obstacle avoidance performance in extended test runs. Although trained with a floor of green grass, it worked as well with half of the floor covered by colorful autumn leaves. The robot showed avoidance at large and medium distances, however at short distances collisions were quite frequent. Again we speculate that the way data was collected prevented a sufficient sample of very near obstacles. Sometimes collisions also happened when the obstacle was far away initially. Again the system cannot be considered fully autonomous. About once every two minutes a manual intervention was necessary (autonomy = 95%).

We found that obstacle avoidance works better in the outdoor setting, however this may be because there are fewer and wider spaced obstacles. Also the clutter problem (each image shows several levels of obstacles at different distances) and the less diverse object textures observed in indoor settings could explain why the outdoor task is simpler. Finally, the local processing prevented any significant latency between frames and between frame recording and sending the new steering output in the outdoor setting. Surprisingly, we had far less training data for the outdoor task and all evaluation measures were worse there, so from this we would have expected the outdoor model to perform worse, which proved not to be the case.

7 DISCUSSION

One issue that may be limiting was the lack of consistent training data. While human operators will try their best to consistently drive the robot, boredom and interindividual differences may yield changes in data collection. One option would be to use other sensors[25] to implement obstacle avoidance behaviour and then using this system to collect large datasets for obstacle avoidance training using direct or indirect information from these sensors. It might not be possible to make the system completely autonomous, however even an increased autonomy would help to make data collection more efficient and more consistent.

Another issue is how to evaluate and compare these systems. We can easily compare performance of standardized datasets, however this is not always meaningful (think of a robot driving straight towards a wall – obviously both left and right steering are correct). For publicly available datasets it may be possible to overfit the test set, so the ability to generate arbitrary amounts of data should always be preferred. One option for almost autonomous systems is the number of human interventions over time (Bojarski et al. (2016)'s *autonomy* measure) – however this is again dependent on human input or on the availability of a perfect autonomous systems, which is only feasible within a simulation setting. Another way would be to combine a robust simultaneous localization and mapping system (SLAM, see e.g. Cadena et al. (2016)) that creates a map and localizes the robot, and using this data to evaluate more complex measures such as average distance driven before a collision, minimum distance to an obstacle per run and number of collisions and near-misses.

Finally, larger deep learning networks pretrained in related contexts (such as described by Khan and Parker (2019)) may adapted to this task.

8 CONCLUSIONS

We replicated the findings of Muller et al. (2006) and Muller et al. (2004), and found that they also apply to some extent to indoor settings. We found that training the network in a classification setting yields better results w.r.t. correlation and accuracy using different bin sizes versus training in a regression setting. However, performance is not yet competitive although the ability to use arbitrary sensors remains intriguing. Lastly, we have introduced the ToyCollect open source hardware and software platform.

[25]E.g. ultrasound sensors which are already present on robot *OUT*, perhaps augmented with bumper sensors

ACKNOWLEDGEMENTS

This project was partially funded by the Austrian Research Promotion Agency (FFG) and by the Austrian Federal Ministry for Transport, Innovation and Technology (BMVIT) within the Talente internship research program 2014-2019. We would like to thank all interns which have worked on this project, notably Georg W., Julian F. and Miriam T. We would also like to thank Lukas D.-B. for all 3D chassis designs of robot *R2X* including the final one we used here.

REFERENCES

Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L.D., Monfort, M., Muller, U., Zhang, J., et al. (2016). End to end learning for self-driving cars. Technical Report 1604.07316, Cornell University.

Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., Reid, I., and Leonard, J. (2016). Past, present, and future of simultaneous localization and mapping: Towards the robust-perception age. *IEEE Transactions on Robotics*, 32(6):13091332.

Hartbauer, M. (2017). Simplified bionic solutions: a simple bio-inspired vehicle collision detection system. *Bioinspir. Biomim.*, 12(026007).

Khan, M. and Parker, G. (2019). Vision based indoor obstacle avoidance using a deep convolutional neural network. In *Proceedings of the 11th International Joint Conference on Computational Intelligence - NCTA, (IJCCI 2019)*, pages 403–411. INSTICC, SciTePress.

Muller, U., Ben, J., Cosatto, E., Fleep, B., and LeCun, Y. (2004). Autonomous off-road vehicle control using end-to-end learning. Technical report, DARPA-IPTO, Arlington, Virginia, USA. ARPA Order Q458, Program 3D10, DARPA/CMO Contract #MDA972-03-C-0111, V1.2, 2004/07/30.

Muller, U., Ben, J., Cosatto, E., Fleep, B., and LeCun, Y. (2006). Off-road obstacle avoidance through end-to-end learning. In *Advances in neural information processing systems*, pages 739–746.

Pfeiffer, M., Schaeuble, M., Nieto, J. I., Siegwart, R., and Cadena, C. (2016). From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots. *CoRR*, abs/1609.07910.

Seewald, AK. (2012). On the brittleness of handwritten digit recognition models. *ISRN Machine Vision*, 2012.

Wang, Y., Dongfang, L., Jeon, H., Chu, Z., and Matson, ET. (2019). End-to-end learning approach for autonomous driving: A convolutional neural network model. In Rocha, A., Steels, L., and van den Herik, J., editors, *Proc. of ICAART 2019*, volume 2, pages 833–839.