Aggressive Deep Driving: Combining Convolutional Neural Networks and Model Predictive Control

Paul Drews
School of ECE
Georgia Inst. of Technology
pdrews3@gatech.edu

Grady Williams
College of Computing
Georgia Inst. of Technology
gradyrw@gatech.edu

Brian Goldfain
College of Computing
Georgia Inst. of Technology
bgoldfain3@gatech.edu

Evangelos A. Theodorou
School of Aerospace Engineering
Georgia Inst. of Technology
evangelos.theodorou@gatech.edu

James M. Rehg
College of Computing
Georgia Inst. of Technology
rehg@gatech.edu

Abstract: We present a framework for vision-based model predictive control (MPC) for the task of aggressive, high-speed autonomous driving. Our approach uses deep convolutional neural networks to predict cost functions from input video which are directly suitable for online trajectory optimization with MPC. We demonstrate the method in a high speed autonomous driving scenario, where we use a single monocular camera and a deep convolutional neural network to predict a cost map of the track in front of the vehicle. Results are demonstrated on a 1:5 scale autonomous vehicle given the task of high speed, aggressive driving.

Keywords: convolutional neural networks, model predictive control, autonomous driving

1 Introduction

A basic challenge in autonomous driving is to couple perception and control in order to achieve a desired vehicle trajectory in the face of uncertainty about the environment. Existing commercial solutions for driver assistance and vehicle autonomy utilize relatively simple models of vehicle dynamics, and emphasize the integration of multiple sensing modalities to characterize the vehicle's environment. Several examples of this approach can be found in the perception and control architectures utilized in the DARPA Grand Challenge Competitions [1, 2, 3].

While many challenging problems remain in order to achieve safe and effective autonomous driving in urban environments, this paper is focused on the task of *aggressive driving*, which requires a tight coupling between control and perception. We define aggressive driving as a vehicle operating close to the limits of handling, often with high sideslip angles, such as may be required for collision avoidance or racing. There has recently been some prior work on aggressive driving using a 1:5 scale vehicle [4, 5]. This work resulted in an open source vehicle platform which we have also adopted in this paper. A disadvantage of this prior work is its reliance on high-quality GPS and IMU for position estimation and localization, which limits the applicability of the method. In this paper, we present an approach to autonomous racing in which vehicle control is based on computer vision sensing, using only monocular camera images acquired from a dirt track in a rally car racing environment. We address the challenge of learning visual models which can be executed in real-time to support high-speed driving. We make the following contributions:

- A novel deep learning approach for analyzing monocular video and generating real-time cost maps for model predictive control which can drive an autonomous vehicle aggressively
- Analysis of the benefits of different representations for the cost maps, with the demonstration that direct prediction of a bird's eye view cost map gives the best performance
- A method for automatic image annotation to support large-scale human-in-the-loop training of deep neural networks for autonomous driving

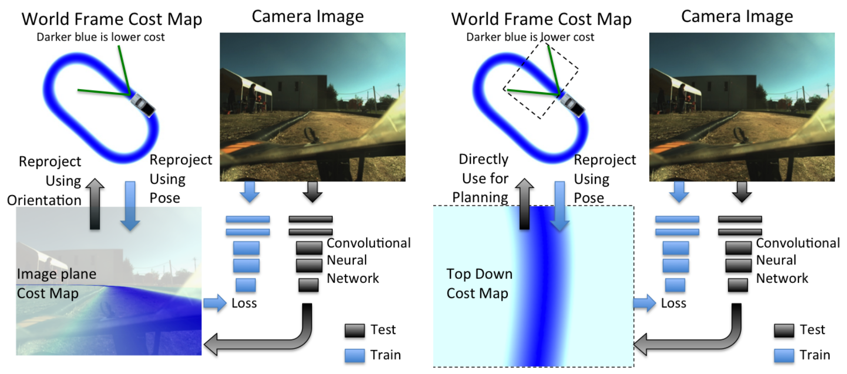


Figure 1: (left) Image plane cost map regression. Camera image and position on a world map are combined to label driveability of image pixels. (right) A top down projection of the cost map can be used as a training target.

In order to use model predictive control, we need a cost function to minimize. One portion of this function predicts cost of being in a position relative to the front of the vehicle, similar in concept to an occupancy map. As shown in Figure 1, our framework is able to take as input a single monocular camera image and output a cost map of the area in front of the vehicle. This cost map image is fed directly into a model predictive control algorithm, with no pre-processing steps necessary. Because the cost map learned by the neural network is independent of the control task being performed, we can use any driving data, including human data, as training data and still generalize to different tasks. Additionally, because we learn an interpretable intermediate representation, it is much easier to diagnose failure cases.

1.1 Related Work

Several approaches have been taken to solve the problem of aggressive autonomous driving, and autonomous driving in general. In [6], an analytic approach is explored. The performance limits of a vehicle are pushed using a simple model-based feedback controller and extensive pre-planning to follow a racing line around a track. More recently, [7] showed the benefits of model predictive control on a 1:10 scale vehicle following waypoints through a challenging obstacle course. [5] also shows some of the benefits of model predictive control in an outdoor, dirt environment. However, these approaches all rely on highly accurate position from an external source such as GPS or motion capture.

There are several ways to approach this problem. Many SLAM approaches that use cameras [8, 9], LIDAR [10], or other sensor combinations [11] can provide accurate position. These systems typically provide position relative to a generated map. However, this approach can be very challenging when localizing in a map created in significantly different conditions [12]. Because a large map needs to be created, and position calculated relative to this map, these methods tend to be computationally expensive. An alternative method to providing absolute position uses deep neural networks to directly regress a position estimate in an area previously visited [13]. However, this method of localization is not yet sufficiently accurate to be directly used for control. Our method does not require any type of absolute position.

Instead of relying on accurate localization, one can instead derive actions from images in an end-to-end trained system, bypassing the need for explicit position information. In [14], a strong case is made for end to end learning, or behavior reflex control in the context of autonomous driving. This work follows from seminal work by Dean Pomerleau in the Alvin project [15]. In [16], a neural network is trained as a policy from images to manipulator arm torques using guided policy search. Because these solutions do not separate image understanding and control, it is very difficult to generalize to new dynamics or control objectives.

An alternative to end-to-end systems learns a drivability function directly from image data that can be used by a lower level controller. By utilizing accurate short range data provided by stereo vision,

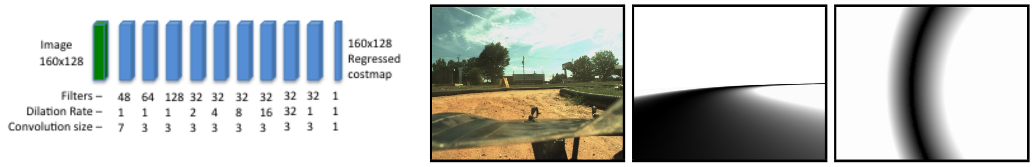


Image
160x128
160x128
Regressed
costmap
Filters – 48 64 128 32 32 32 32 32 32 1
Dilation Rate – 1 1 1 2 4 8 16 32 1 1
Convolution size – 7 3 3 3 3 3 3 3 3 1

Figure 2: Network architecture with input and training targets. Left: Neural network architecture used to produce top down cost maps. Right: example input image, image plane training target and top down training target, respectively

[17] learns a neural network to predict far-field traversibility from images, which is then fed into a separate planning and control framework. However, this approach requires significant geometric image pre-processing, and the resultant map is suited to planning, not high speed control. More recently, [18] directly learns affordances necessary for autonomous driving by a low level controller. However, these learned affordances are not rich enough for a model predictive control framework such as [5]. In [19], a neural network is able to produce an image plane driveability map. However, this method requires the use of additional obstacles sensors such as LIDAR.

Semantic segmentation may also be used to obtain driveability information from an image. Lately, deep neural network architectures have achieved excellent results on semantic segmentation datasets such as [20] and [21]. These models aim to produce a per-pixel labeling of an input image. Many techniques to improve the accuracy of these models, such as conditional random fields (CRFs) [22] and dilated convolutions [23] have advanced the state of the art in this field.

2 Approach

Our approach combines a high performance control system based on Model Predictive Control (MPC), with deep Convolutional Neural Networks (CNNs) for real-time scene understanding. We show that fully convolutional networks have the ability to go beyond the standard semantic image segmentation paradigm, and can generate a top-down view of the cost surface in front of the vehicle, even generalizing to portions of the track which are outside the camera's field of view, given a single video frame taken from the driver's perspective.

Model predictive control is an effective control approach for aggressive driving [4, 5]. It is based on optimizing a cost function that defines where on a track surface the vehicle should drive. The cost surface must therefore encode the current and future positions of the road, obstacles, pedestrians, and other vehicles. This presents a major barrier for using MPC in novel environments since creating a cost function requires analyzing the local environment of the vehicle on-the-fly. Our solution is to train a deep neural network to transform visual inputs from a single monocular camera to a cost function in a local robot-centric coordinate system. In our implementation, the cost function takes the form of an occupancy-grid style cost map, as shown in Figure 2. The network is trained so that the cost is lowest at the center of the track, and higher further from the center. This cost map can then be directly fed into a model predictive control algorithm.

By factoring the control and perception tasks, we can take advantage of the strengths and mitigate the weaknesses of both deep visual learning and MPC. The perception task of mapping images to cost functions is invariant to the control policy, which means that data can be collected from many different (off-policy) sources. This mitigates the main difficulty in deep learning, which is collecting large amounts of data. However, we are still able to use deep learning for on-the-fly scene understanding. In the case of model predictive control, we are able to operate without an explicitly programmed cost function, enabling its usage in potentially novel environments. However, we are still able to utilize MPC for the difficult problem of online optimization with non-linear dynamics and costs.

2.1 Model Predictive Path Integral Control

Model predictive control works by interleaving optimization and execution: first an open loop control sequence is optimized, then the first control in that sequence is executed by the vehicle, and then state feedback is received and the whole process repeats. This sequence is shown graphically in Figure 3. We use model predictive path integral control (MPPI), which is a sampling based, derivative free, approach to model predictive control which has been successfully applied to aggressive autonomous driving using learned non-linear dynamics [4]. At each time-step, MPPI samples thousands of trajectories from the system dynamics, and each one of these trajectories is evaluated according to its expected cost. A planned control sequence is then updated using a cost-weighted average over the sampled trajectories.

Mathematically, let our current planned control sequence be $(u_0, u_1, \dots u_{T-1}) = U \in \mathbb{R}^{m \times T}$, where m is 2 and T is 60 in our case. Let $(\mathcal{E}_1, \mathcal{E}_2 \dots \mathcal{E}_K)$ be a set of random control sequences, with each $\mathcal{E}_k = (\epsilon_k^0, \dots \epsilon_k^{T-1})$ and each $\epsilon_k^t \sim \mathcal{N}(u_t, \Sigma)$. Then the MPPI algorithm updates the control sequence as:

$$\eta = \sum_{k=1}^{K} \exp\left(-\frac{1}{\lambda}\left(S(\mathcal{E}_k) + \gamma \sum_{t=0}^{T-1} u_t^{\mathrm{T}} \Sigma^{-1} \epsilon_k^t\right)\right) \tag{1}$$

$$U = \frac{1}{\eta} \sum_{k=1}^{K} \left[\exp\left(-\frac{1}{\lambda}\left(S(\mathcal{E}_k) + \gamma \sum_{t=0}^{T-1} u_t^{\mathrm{T}} \Sigma^{-1} \epsilon_k^t\right)\right) \mathcal{E}_k\right] \tag{2}$$

where η is a normalizing constant for updated control sequence U. The parameters λ and γ determine the selectiveness of the weighted average and the importance of the control cost respectively. The function $S(\mathcal{E})$ takes an input sequence and propagates it through the dynamics to find the resulting trajectory, and then computes the (state-dependent) cost of that trajectory sequence, which we denote as $C(x_0, x_1, \dots x_T) = \sum_{t=0}^{T} q(x_t)$. In this paper we only use an instantaneous running cost (there is no terminal cost), and we sample trajectories on a GPU using the dynamics model from [4]. The instantaneous running cost is the following:

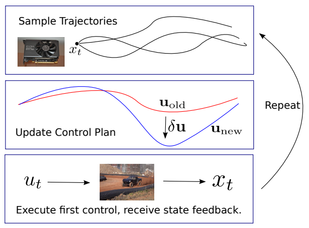

Sample Trajectories
x_t
Update Control Plan
$\mathbf{u}_{\text{old}}$
$\delta \mathbf{u}$
$\mathbf{u}_{\text{new}}$
Repeat
u_t
x_t
Execute first control, receive state feedback.

Figure 3: Model Predictive Path Integral Control algorithm. Trajectories are sampled and updated using a GPU, and the first control executed.

$$q(x) = w \cdot \left(C_M(p_x, p_y), (v_x - v_x^d)^2, 0.9^t I, \left(\frac{v_y}{v_x}\right)^2\right) \tag{3}$$

where term (1), $C_M(p_x, p_y)$, is the output of the neural network which gives the track-cost associated with being at the body frame position (p_x, p_y). The other terms are (2) A cost for achieving a desired speed v_x^d, (3) an indicator variable which is turned on if the track-cost, roll angle, or heading velocity become too high, and (4) is a penalty on the slip angle of the vehicle. The coefficient vector was $w = (100, 4.25, 10000, 1.75)$. Note that the three terms which are not learned (2,3, and 4) are trivial to compute given the vehicle's state-estimate, while the cost map requires analysis of the vehicle's environment. In previous work [4], the cost map was obtained from a pre-defined map of the track combined with GPS localization, which does not generalize to other terrains.

2.2 Convolutional Neural Network Architecture

In this work, we use a CNN to generate costs based on future positions from a single monocular image. Our CNN architecture is constrained to run in real time on the low power Nvidia GTX750Ti available on our platform, and it produces a dense cost map output. We found that a fully convolutional network that outputs a dense cost map with large input receptive fields produces the most accurate result. We trained this architecture to output two different types of predictions (as shown in Figure 1), these are (1) a top-down cost map that can be used directly by MPPI, and (2) an image-plane labeling of pixels that must be projected onto the ground before use.

We experimentally evaluate both the top down and image plane methods with two different neural network structures. The image plane network takes in 640x480 input images and passes them through several convolution layers and 2 pooling layers, followed by a set of 6 dilated convolution layers. The top down network uses a smaller structure, as shown in Figure 2. The dilated convolutions allow each output pixel the full input image as its receptive field while maintaining a reasonable (128x160) output size. This significantly improves the output quality of the network. The cost-map is then taken directly as the output of the final layer without applying normalization.

Using these two network architectures, we are able to maintain low latency and a frame rate of about 10 Hz for the image plane network and 40Hz (full camera frame rate) for the top down network.

Input images come directly from a PointGrey Flea3 color camera at 1280x1024 resolution. These images are downsampled to 640x512, the dataset mean is subtracted, and each pixel is divided by the dataset standard deviation. During training, the 160x128 pixel output is compared with the pre-computed ground truth cost maps obtained from GPS data. It was found that an L1 pixel-wise loss produced a cleaner final cost map than L2 loss. This loss is only computed for points within 10 pixels of the edge of the track in the ground truth image to avoid training the network to output large sections of blank space. The network was trained using the Adam [24] optimization algorithm in Tensorflow [25]. A mini-batch size of 10 images was used during training, and a small random perturbation to the white balance of each image (multiplying each channel by a normally distributed random variable between 0.9 and 1.1) was also applied. For all networks, best driving performance was achieved with training stopped at or near 100,000 iterations. This coincided with the point where testing loss on a held out dataset plateaued.

2.3 Ground truth generation

In order to learn a pixel-wise regression function capable of producing traversal costs at every pixel, training data is needed on the order of 100s of thousands of frames. Labeling all of this data by hand is laborious, slow, and prone to errors. However, the 1:5 scale AutoRally vehicle (Fig. 4) that we use in our experiments, and many autonomous and commercial vehicles are equipped with position sensors and cameras that can associate each image with a full state estimate, including orientation and position. Combined with a surveyed map of a track registered to GPS coordinates, these can be used to create hundreds of thousands of labeled images without any manual labeling of individual images.

By calibrating the transformation between the IMU (where position and orientation estimates are calculated) and the camera, a homography matrix can be computed that transforms the surveyed track map from world coordinates to image plane coordinates:

$$H = kT_{im}^{car}T_{car}^{world} \tag{4}$$

Where T_{car}^{world} is the position of the car in world coordinates (estimated at the IMU), T_{im}^{car} is the transformation between the IMU and camera reference frames, and k is the camera intrinsics matrix. Given this, points in the ground coordinate frame can be projected into the image using:

$$p_{im} = \hat{H}p_{world}; \hat{H} = \begin{bmatrix} H_{11} & H_{12} & H_{14} \\ H_{21} & H_{22} & H_{24} \\ H_{31} & H_{32} & H_{34} \end{bmatrix} \tag{5}$$

where p_{im} and p_{world} are homogeneous points. Using this scheme, ground truth images can be produced for each image in our training set. This mapping is not perfect due to small errors in time synchronization and violations of the assumption that the camera is a constant height above the ground. However, despite these small errors, the reprojected cost maps are very good, and networks are able to learn from them. To produce ground truth images for the top down network, a 160x128 section of the cost map directly in front of the vehicle (in vehicle centric coordinates) is used.

Using this method, we created approximately 300,000 images with corresponding ground truth cost maps. These training images were taken from 64 different runs spanning 9 different days over the course of 8 months. It includes substantial variability in lighting conditions, people and equipment present at the collection site, and poses of the camera on the track. This data is split into approximately 250,000 training images and 50,000 test images (selected as full sequences, not randomly sampled from all images).

2.4 Implementation

In order to truly test the performance of a neural network designed for autonomous driving, it must be implemented and tested on a physical platform. In our case, we choose the AutoRally platform (see Figure 4). This is based on a 1:5 scale RC chassis capable of aggressive maneuvers and a top speed of nearly 60 miles per hour. During testing, all sensing and computation is performed on the vehicle in real time, including neural network forward inference and model predictive control

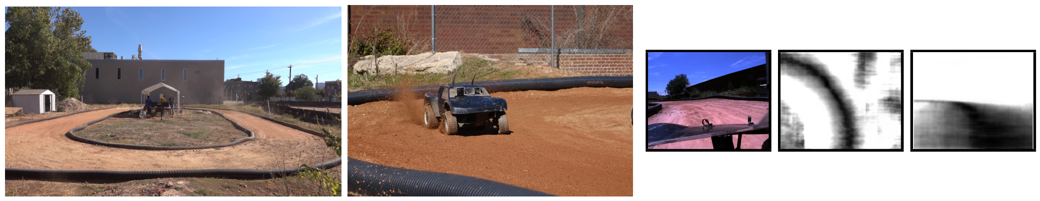


Figure 4: Testing setup and example output images. Left: Oval dirt test track where all test data was taken. Center: Photo of vehicle during testing. Right: Neural network input, top down output, and image plane output.

computation. Point Grey cameras are used to collect images, and perception and control is computed on the onboard Nvidia GTX750Ti GPU.

Forward inference through the network is handled asynchronously, and the cost maps are fed to the MPPI control algorithm at approximately 10Hz for the image plane network and 40Hz for the top down network. The MPPI controller runs at 40Hz. Velocity and acceleration information is obtained from the on-board GPS-IMU system, but absolute position is not used. We use GPS derived velocity for experimental simplicity, however this velocity could be derived from visual odometry in a purely vision based system. For the top down case, the camera orientation (from the IMU) is used to generate a homography transform. The neural network output and associated homography matrix are used by the MPPI algorithm to plan and execute controls until another cost image is available.

3 Experimental Results

In order to evaluate the performance of the proposed system, we tasked the CNN-MPPI algorithm with driving around a roughly elliptical dirt track, using the same 1/5 scale vehicle hardware as [5, 4]. This enables us to compare lap times and speeds achieved with the same controller using a ground truth cost-map. This provides a metric, independent of network validation error, of how well the neural network performs in a real world scenario. Additionally, we compare the performance of the convolutional neural networks (mean L1 pixel distance) on a held out validation data set (from an unseen testing day) to gain some insight into performance discrepancies and failure modes.

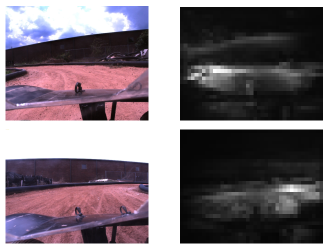
3.1 Network Performance

The accuracy of the neural networks is computed as the L1 distance between the ground truth and the training target on a holdout dataset, taken on a different day than the training data, of approximately 4000 images. In order to achieve a more meaningful metric, we report only the error for pixels where we there is track (i.e. anywhere the ground truth training image is not white). We use this convention in all neural network training we report and report this as: score $= (1 - \text{error})$.

The top down network, which maps input images to a top down (bird's eye) cost map achieved a score of 0.92. The image plane network, which maps input images to an *image plane* cost map achieved a score of 0.82. In addition the having a lower score than the top down network, the image plane cost map must also be projected onto the ground plane causing significant distortion.

3.2 Ablation and Simulation Results

We performed an ablation study in order to identify the features of the input images that play the strongest role in the generation of the cost map. We first obtain as a baseline the cost map which is generated by the network from the full input image. We then ”zero out” a block of pixels at a certain location and size by replacing all pixels within the window with the mean pixel value from the entire dataset. After mean subtraction, this block will have the value zero and will therefore not contribute to the network activations. We systematically examine the influence of different parts of the image on the prediction performance by scanning the window over the entire input image, thereby generating a set of ablation images with zeroed out blocks at different locations. For each ablated image, we compute an accuracy measure (average L1 distance for track pixels). We then construct a sensitivity map by creating an image from these accuracy measures, which each accuracy


(a) Ablation study

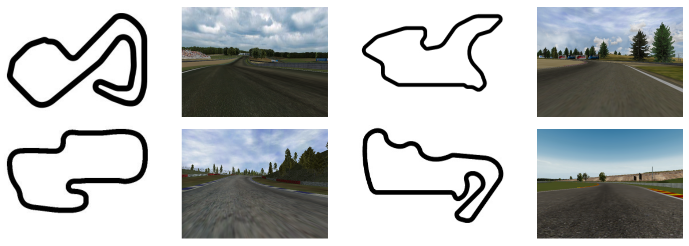

(b) Track images and screenshots

Figure 5: Ablation study and simulation results. (a) In the ablation study, a square window of the input image is replaced with the corresponding square of the dataset mean image. Examples shown of two input image with ablation heatmaps. It is clear in both cases that the inside of the corner is the most important feature for determining track shape, and most image clutter is ignored. (b) TORCS simulation tracks, with example training track in upper left and unseen test tracks. Track overview maps show a great deal of corner variety, and screenshots show texture variety.

value is stored at the corresponding location where a block was ablated. Figure 5 shows sensitivity maps for representative input images. All sensitivity maps are normalized with zero error as black and largest recorded error as white.

In order to test the capability and generalization of this network, we train the network structure from Figure 2 on a dataset of images and corresponding cost maps collected from the TORCS open source driving simulator. We collect approximately 250,000 images taken from several hours of autonomous and manual driving on 4 representative tracks as training. We then validate this network by calculating validation error and tasking the network with completing several laps of three other, unseen tracks. These tracks include similar features, but have new layouts as seen in Figure 5(b). The validation error on the first track is 0.962, significantly better than the trivial output of entirely non-track (white) pixels of 0.896. The system is able to complete the top right and bottom left track of Figure 5(b). The bottom right track produced excellent error (0.961) but required intervention to drive the full track due to errors in one or two corners.

This study demonstrates that the network has learned to use intuitively reasonable input features in real world experiments, and can generalize to unknown scenes in the simulation case. The network can tolerate the removal of small track regions due to ablation and still produce usable cost maps.

3.3 Driving Performance

Our goal in learning to regress cost maps from images is to plan and execute high speed driving maneuvers. In order to test this end goal, we autonomously drive a 1/5 scale AutoRally vehicle at increasingly aggressive speeds around a flat dirt track. Each method uses the same controller and vehicle physics model, cameras, and track. The form of the controller's cost remains the same, although some parameters such as exploration variance and relative cost weights are tuned slightly to optimize performance. To find the limits of each method, we slowly increased the target speed from 5m/s. If the vehicle was able to performing 10 laps without intervention, the condition was considered a success. If intervention was required, the condition was considered to have failed. Note that the friction limits of the vehicle going around the tracks turns are around 5.5 m/s, so the control algorithm has to intelligently moderate both the steering and throttle in order to navigate successfully. While this single track is a limited environment, there is still significant clutter (such as changing lighting conditions and moving distractors), making this a challenging vision problem. In addition, due to the speeds the vehicle is traveling, small network errors can lead quickly to overall system failure. As with many machine learning systems, our system is sensitive to the training data. Our dataset contains more counterclockwise examples than clockwise examples, possibly explaining the higher failure rate while traveling clockwise.

Using the top down network produced significantly more robust, consistent, and overall faster runs than the image plane network. Using the image plane network, it was only occasionally possible to produce runs of 10 consecutive laps (at the slowest speed). Most of the runs lasted between 1 and 5 laps before intervention was required. Usually, this was due to the network not identifying a turn,

Table 1: Testing statistics for image plane (IP) and top down (TD) networks, 10 lap runs

Method	Counterclockwise travel		Clockwise travel	
	Avg. Lap (s))	Top Speed (m/s)	Avg. Lap (s)	Top Speed (m/s)
(TD) 5 m/s	16.98	4.37	18.09	4.99
(TD) 6 m/s	12.19	6.38	failure	failure
(TD) 7 m/s	10.84	6.91	11.27	6.51
(TD) 8 m/s	10.13	7.47	failure	failure
(IP) 6 m/s	14.48	5.67	failure	failure
[4]	9.74	8.44	N/A	N/A
[5]	N/A	N/A	10.04	7.98

which would result in the vehicle driving to the end of the track and stopping. Figure 4 demonstrates the difference between the two approaches as the vehicle approaches a turn, the top-down network produced much cleaner and crisper cost maps in corners where only a small portion of the track is visible.

Table 1 summarizes lap times and top speeds for our networks and the the method in [5, 4]. In addition, Figure 6 shows some representative trajectories. In [4], using GPS localization in a predefined map, the vehicle and controller were able to achieve an average lap time of 9.74 seconds, only 0.39 seconds faster than the best setting of our method which only uses a single monocular image, body frame velocity, and inertial data as input. The image plane regression network was able to achieve a maximum average lap time of 14.48 seconds over 10 laps, 4.74 seconds slower. This difference is due to the top-down network producing crisper output cost maps, as well as its ability to predict beyond the field of view.

4 Conclusions

In this work, we present field experiments demonstrating novel capabilities of fully convolutional neural networks combined with sampling based model predictive control. We compare two output targets for the neural network, a cost map projected into the image plane and a top down view of the cost map, and find that the top down network only loses 4% lap time over using GPS without needing any absolute position information. We compare them both on a sample of a held-out dataset and in full system experiments driving an autonomous vehicle.

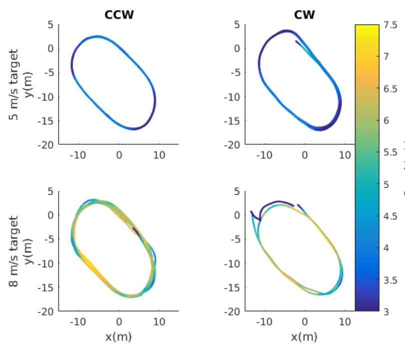


Figure 6: GPS plots of vehicle trajectory with top-down network at 5 m/s and 8 m/s target speeds. Notice how the method is able to reject strong disturbances at the limits of vehicle handling.

The ability of this network to predict around corners beyond the camera field of view was critical in performance for the controller. The model predictive controller only uses information that it can see in the output of the neural network, and plans ahead 1.5 seconds to produce a control signal. This 1.5 second time horizon leads to extremely timid behavior in the case of the image plane regression network because the available look ahead distance is very short. This was not the case with the network that directly regressed the top-down view, and was a large contribution to its success in vehicle performance.

Additionally, the top-down network tends to produce a map with a defined centerline that is still good for planning, even if the exact location of the track is incorrect. This allows the MPPI algorithm to continue planning feasible paths until another image is processed, hopefully rectifying the errors.

Acknowledgments

This work was made possible by the ARO through DURIP award W911NF-12-1-0377, NSF award NRI-1426945, and support from BMW award UR:KAN KA-SVT (Agreement No. 821). Thanks also to Dominic Pattison and Justin Zheng for their help making real-world testing possible.

References

[1] M. Montemerlo et al. Junior: The stanford entry in the urban challenge. *Journal of Field Robotics*, 25(9):569–597, 2008. ISSN 1556-4967. doi:10.1002/rob.20258. URL `http://dx.doi.org/10.1002/rob.20258`.

[2] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer, et al. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics*, 25(8):425–466, 2008.

[3] Urmson et al. Tartan racing: A multi-modal approach to the darpa urban challenge. 2007.

[4] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou. Information theoretic mpc for model-based reinforcement learning. In *International Conference on Robotics and Automation (ICRA)*, 2017.

[5] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou. Aggressive driving with model predictive path integral control. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1433–1440, May 2016. doi:10.1109/ICRA.2016.7487277.

[6] J. Funke, P. Theodosis, R. Hindiyeh, G. Stanek, K. Kritatakirana, C. Gerdes, D. Langer, M. Hernandez, B. Mller-Bessler, and B. Huhnke. Up to the limits: Autonomous audi tts. In *2012 IEEE Intelligent Vehicles Symposium*, pages 541–547, June 2012. doi:10.1109/IVS.2012.6232212.

[7] N. Keivan and G. Sibley. *Realtime Simulation-in-the-Loop Control for Agile Ground Vehicles*, pages 276–287. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014. ISBN 978-3-662-43645-5. doi:10.1007/978-3-662-43645-5_29. URL `http://dx.doi.org/10.1007/978-3-662-43645-5_29`.

[8] J. Engel, T. Schöps, and D. Cremers. Lsd-slam: Large-scale direct monocular slam. In *European Conference on Computer Vision*, pages 834–849. Springer, 2014.

[9] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015.

[10] J. Zhang and S. Singh. Loam: Lidar odometry and mapping in real-time. In *Robotics: Science and Systems*, volume 2, 2014.

[11] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*, pages 127–136. IEEE, 2011.

[12] C. Beall and F. Dellaert. Appearance-based localization across seasons in a metric map. *6th PPNIV, Chicago, USA*, 2014.

[13] A. Kendall and R. Cipolla. Modelling uncertainty in deep learning for camera relocalization. *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2016.

[14] 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.

[15] D. A. Pomerleau. Alvinn, an autonomous land vehicle in a neural network. Technical report, Carnegie Mellon University, Computer Science Department, 1989.

[16] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016.

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

[18] C. Chen, A. Seff, A. Kornhauser, and J. Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2722–2730, 2015.

[19] D. Barnes, W. Maddern, and I. Posner. Find your own way: Weakly-supervised segmentation of path proposals for urban autonomy. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 203–210, May 2017. doi:10.1109/ICRA.2017.7989025.

[20] P. Arbeláez, B. Hariharan, C. Gu, S. Gupta, L. Bourdev, and J. Malik. Semantic segmentation using regions and parts. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3378–3385. IEEE, 2012.

[21] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.

[22] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. *arXiv preprint arXiv:1412.7062*, 2014.

[23] F. Yu and V. Koltun. Multi-scale context aggregation by dilated convolutions. In *ICLR*, 2016.

[24] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL `http://arxiv.org/abs/1412.6980`.

[25] M. Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `http://tensorflow.org/`. Software available from tensorflow.org.