Driving Like a Human: Imitation Learning for Path Planning using Convolutional Neural Networks

Eike Rehder, Jannik Quehl and Christoph Stiller[1]

Abstract— **Human-like path planning is still a challenging task for automated vehicles. Imitation learning can teach these vehicles to learn planning from human demonstration. In this work, we propose to formulate the planning stage as a convolutional neural network (CNN). Thus, we can employ well established CNN techniques to learn planning from imitation. With the proposed method, we train a network for planning in complex traffic situations from both simulated and real world data. The resulting planning network exhibits human-like path generation.**

I. Introduction

Motion planning is an essential component of autonomous agents navigating through the world. Especially in the context of autonomous robots and vehicles that operate in large open spaces, motion planning is a crucial task.

In research, planning an agent's actions has been understood as an optimization problem in which the optimal actions given a cost function should be selected. Approaches only differ in the shape of both actions and cost functions.

In terms of actions, one may consider discrete or continuous actions and outcomes. For discrete state spaces, planning will take the form of graph optimization, e.g. in grid maps or state lattices [1], [2]. If the state and action space is continuous, nonlinear optimization has been proposed [3]. Also, one can think of a combination of both [4]. For a detailed study on models and their solutions, see [5].

In the end, however, all concepts for planning have in common that the function to be optimized has to be prespecified. In most works, this is done by the careful design by the researcher, e.g. for shortest paths in presence of obstacles or minimum jerk in automated driving. This imposes demands on the perception of autonomous systems as they have to be capable of inferring the boundary conditions of the optimizer at hand.

Thus, some works have aimed to deduct the planning cost function from given sensor data [6], [7]. In this concept, an agent learns to plan its motion from imitation of observed behavior of others, thus called *Imitation Learning* (IL).

In the area of machine learning, recent advances in Deep Learning and Convolutional Neural Networks (CNN) have exceeded all expectations in a broad variety of tasks such as image classification and segmentation, optical flow computation, etc. [8], [9], [10], [11].

With this, it is no wonder that deep learning has also found its way into planning. While some researches aim to construct an end-to-end pipeline that can create control outputs directly from sensor readings, this approach can only be reactive but not strategic [12]. Recently, long term planning has been solved using *Value Iteration Networks* (VIN) for Markov Decision Processes [13], [14]. While these works achieve great results, to our understanding, none was trained directly on observed paths. In the work of Shankar *et al.* the planning cost function was static for one network. Thus, the network could only plan for scenes it had been trained on.

In this work, we propose to model motion planning of an intelligent vehicle as Value Iteration Network. We show how the network can be trained from previously observed paths. As a training input, we rely exclusively on observed paths and not on any kind of manually annotated data. We demonstrate the performance of the network by training a cost function from aerial images to resemble human driving behavior.

II. Imitation Learning for Planning

In this section, we demonstrate how to model the entire planning task as a connection of recursive neural networks. This augmentation makes the planning tasks fully differentiable and thus allows for full back propagation. This way, we can train an underlying cost function network from input data. In this work, we follow the intuition of both Tamar *et al.* as well as Shankar *et al.* [13], [14] for full Imitation Learning of path planning.

In this work, planning is executed in a state grid. For simplicity, the state grid is just a equidistant discretization of the state space. However, one could easily incorporate other state variables, such as orientation, by extending the state grid by additional dimensions. All other steps then scale accordingly.

A. State Transitions within Grids

Planning within a state grid is commonly modeled as a graph where the edges of the graph are represented by neighboring cells in the grid. However, in this work, we aim to model these transitions as components of a neural network for differentiability.

For this, we make use of the property of the Dirac Delta Function. When a Dirac Delta function shifted by a, $\delta(t-a)$,

*The research leading to these results has received funding from the German collaborative research center "SPP 1835 - Cooperative Interacting Automobiles" (CoInCar) granted by the German Research Foundation (DFG).

[1]Eike Rehder, Jannik Quehl and Christoph Stiller are with the Institute of Measurement and Control Systems, Karlsruhe Institute of Technology, Karlsruhe, Germany {eike.rehder, jannik.quehl, stiller}@kit.edu

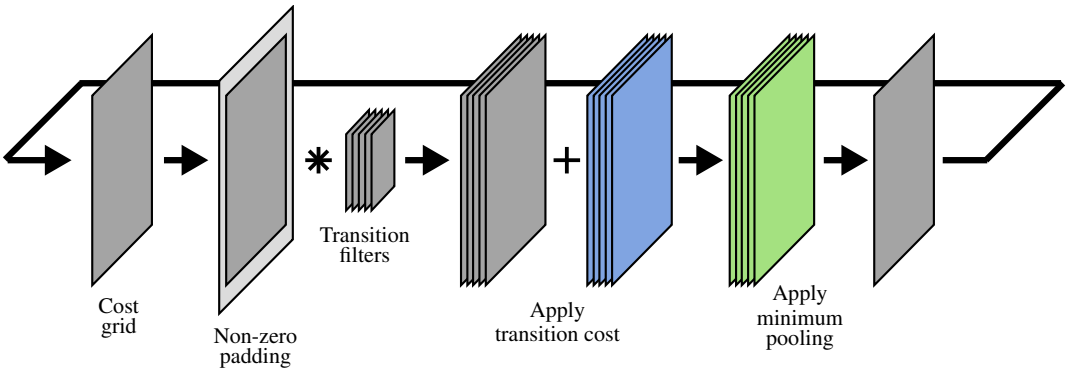


Fig. 1: Value Iteration Module as Recurrent Neural Network. The two layers that define planning behavior are the transition cost map (blue) and the cost per state and transition (green).

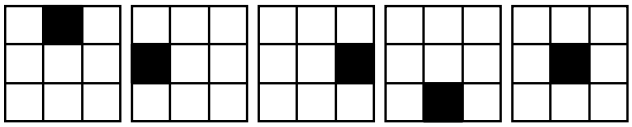


Fig. 2: Transition filters masks for four-connected neighborhood and idle (last). Black cells represent ones, white cells zeros.

is convolved with a function $g(t)$ the result is the function g shifted by a

$$\delta(t-a) * g(t) = g(t-a). \tag{1}$$

The same applies to discrete convolutions. Thus, a shift of a state in a grid can be modeled as a convolution with a two dimensional transition filter mask that resembles a discrete Dirac Delta Function.

In the context of Convolutional Neural Networks, this means that we can model transitions in a state grid as a convolutional layer with known filter masks. Examplary filter masks for the 4-connected neighborhood are depicted in Figure 2. Note especially that the last filter mask represents the *idle element*, a centered dicrete Dirac Function. Convolution with this mask has no effect on the state grid.

B. Value Iteration Module

The Value Iteration Module computes the cost to reach each state within the grid map from a given starting point. For this, it takes the following steps:

1) **Initialization** - Initialize cost grid with arbitrary but very large values in every state. Set the cell of the starting state to zero.
2) **Non-zero padding** - Pad the state grid with arbitrary but very high values. Since zero represents the starting state, zero padding would introduce faulty results in the border regions.
3) **Cost propagation** - Execute all possible transitions by convolving the cost grid with transition filters.
4) **Cost accumulation** - Add cost per state and transition as an additive layer.
5) **Assign minimum state cost** - Compute minimum cost per state by min-pooling over the transition direction of the cost grid.
6) **Recurse** - With the result of 5), restart at 2) until convergence is reached.

Figure 1 demonstrates this process. At convergence, the output of 4) (green layer in Figure) stores the transition policy. This is due to the fact that this stage represents the cost per state and possible action to end up in that specific state. An argmin operation in the direction of possible transition will result in the cheapest possible action to end up in each state.

C. Path Evaluation Module

The Value Iteration Module only computes the minimum cost per state. However, it does not compute the optimal path from starting state to goal state. For this, another recurrent network has to be introduced. Again, this network consists of several steps:

1) **Initialization** - Initialize state grid with all zeros. Set the cell of the destination state to one. Flip the transition filters so that they are mirrored around the centerpoint and input and output directions are exchanged. Create a one-hot representation per cell and action from the argmin of 4) in the Value Iteration Module. This is the transition selection mask. For full differentiability, this may also be a softmin operation.
2) **Transition selection** - Multiply the current state grid with the transition selection mask. The output will be a grid with a single one in the cheapest possible transition into the current state.
3) **Zero padding** - Pad the state grid zeros. Now, zeros represent non-occupied states.
4) **Propagate state** - Convolve with the flipped transition filters.
5) **Recurse** - With the result of 4), restart at 2) until starting state is reached.

This module is equivalent to the optimal predecessor backtracking in Dijkstra's Algorithm: the network traces back the entire optimal actions that led to the goal state. The

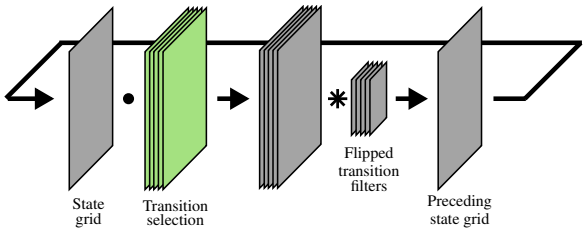


Fig. 3: Recurrent Neural Network to trace back optimal path. The transition selection layer (green) is the argmin of the cost map in Figure 1 at convergence

corresponding network is depicted in Figure 3. The transition selection policy is derived from the cost layer of the network depicted in Figure 1, both colored in green. For backtracing of the optimal path, all transitions have to be reversed. This means that for the filters, we need to flip them in spacial direction and exchange input for output. Convolutions in the forward direction mapped from a map of dimension $W \times H \times 1$ to $W \times H \times K$ for K possible transitions. For the backtracing, since we select 1-of-K possible transitions, the convolution now maps from $W \times H \times K$ to $W \times H \times 1$.

D. Transition Cost Network

To this end, the entire planning process was fully deterministic and predefined by the user. The only parameter to be changed is the cost per state and action. In this part of the network, all learning techniques may be applied. Since the planning network as explained above is fully made up from CNN components, all well-known techniques and toolboxes are at hand [15], [16].

Thus, it is left to the user to specify the input data and network architecture that will generate a cost map. This cost map then specifies the graph topology that is evaluated in the planning stage. Please also note that at time of deployment, this is the only part of the CNN that is still necessary to be a Neural Network. The planning stage may also be exchanged for any other shortest path algorithm as long as the cost map is constrained accordingly.

In this work, to demonstrate the capability of the approach, we employ a Fully Convolutional Network (FCN) operating on aerial views of roads. It is trained to predict cost per state and possible transition to imitate human behavior in traffic. See Section III for details.

E. Loss Function for Imitation Learning

Path planning can be understood as an image segmentation task. Parts of the grid may either belong to the class *path* or *not path*. Accordingly, all loss functions that are used for segmentation can be employed. Specifically, cross entropy remains available for both paths and trajectories. In the context of trajectories, one grid is evaluated against ground truth per planning step. For the path however, one single grid has to be computed for the entire planning task. Note that the output of the Path Evaluation Module is a single grid per planning step. We compute the path from N planning

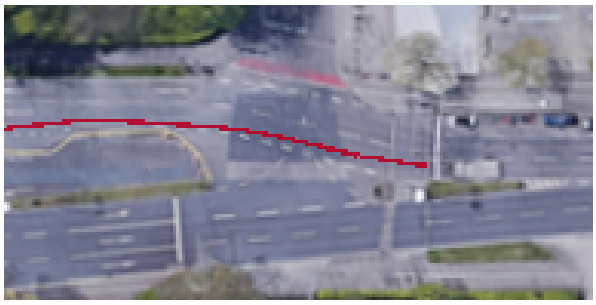


Fig. 4: Example aerial image from the dataset together with one sample path from real world trajectories. Image data from Google Maps [17]

step grids G_i with $i = 1, \ldots, N$. Here, every entry in the grids G_i is in range $[0, 1]$, where the value 0 is an definitely unoccupied cell and 1 represents a definitely occupied cell. Thus, the path grid P can be computed as

$$P = 1 - \prod_{i=1}^{N} (1 - G_i). \tag{2}$$

Note that in the case of an argmin operation for transition selection, every grid per planning step will only feature one unique one in the entire grid. In case of the softmin operation, however, the grids can take arbitrary values in $[0, 1]$ which is necessary for differentiability.

III. Experiments

To show the performance of the proposed approach, we design a network to plan drivable paths from aerial images. The network is trained on both, real an simulated data. We then show its capability on new aerial images that were not included in training.

A. Data

We collected a set of only six aerial images from Google Maps and created ground paths both in simulation and from real world data [17]. For the simulation, we manually annotated roughly 100 possible paths within the images. For real world data, we recorded vehicle trajectories from the road side of an intersection. The maps had a total size of 75m×37.5m and paths had varying length.

B. Network Architecture and Training

The Value Iteration Module requires three individual design aspects: the size of the map, the choice of transition filters and the computation of transition costs.

In our experiment, we discretize the input maps into 192×96 cells for planning. Thus, each cell respresents a space of roughly 0.4m×0.4m in the real world. We found this sufficient for initial proof of concept, however, for more accurate planning, one might want to increase resolution.

The transition filters in our experiment represent all possible transitions from one cell to its eight neighbors as well as the idle transition. This means that the filter masks shown

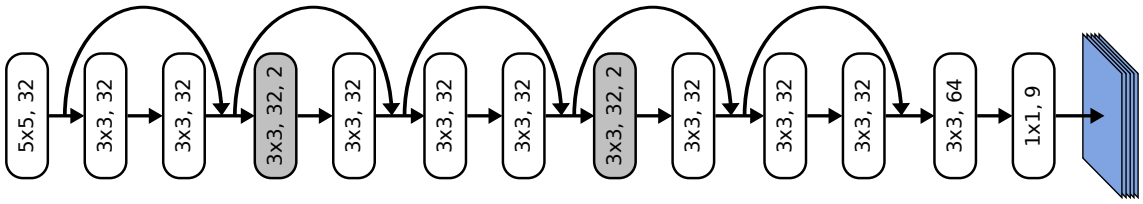
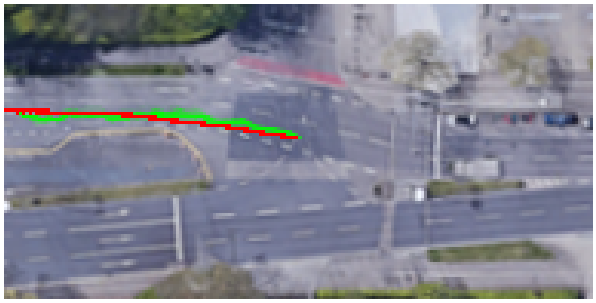
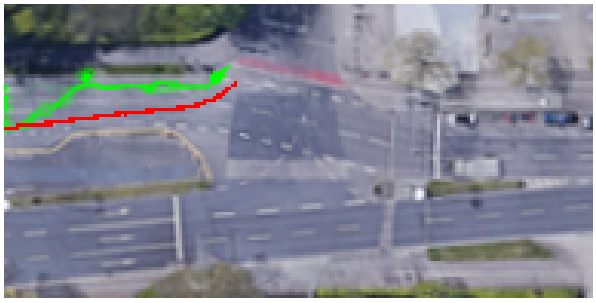


Fig. 5: ResNet for cost map computation. Non-linearities are ReLU. The shaded layers perform dilated convolutions with dilation parameter printed inside. The blue maps that are output are the same as in Figure 1

Fig. 6: Training results. Path planned by network (green) vs. human paths (red). Image data from Google Maps [17]



(a) Plan for lane following on test data



(b) Plan for right turn with multiple lane changes on test data

Fig. 7: Paths planned on test data by the network (green) vs. human paths (red). Image data from Google Maps [17]

in Figure 2 are extended by the four filters representing the diagonal transitions.

Lastly, we computed the transition cost map from the aerial images. For this, we trained a fully convolutional residual network with dilated kernels that predicts cost per cell and transition (the blue layer in Fig. 1) [11], [18], [19] The network architecture is depicted in Figure 5. Since our task is fairly simple and run on low resolution images, we can use a comparably shallow and narrow network. To keep the dimensions of the output same as the input dimension without the need for upsampling we use dilated kernels for every other residual block. In training, to avoid overfit due to our very limited data set, we apply dropout of 20% to the output of the first and second-to-last convolutional layer. All layers use rectified linear units (ReLU) as nonlinearity.

C. Experimental Results

We split the data into two parts, a training set of five road scenes and one as the test scene. We select the single scene for which we have human example paths for the test set. This has a very simple reason: for planning of paths, we have no means to evaluate *right* or *wrong*. We can only compare paths to see if they are *human-like* or not.

Figure 6 shows an example path from the training set together with the network's planning result. As it can be seen, the network in general is perfectly able to replicate human path planning.

We can now look at planning results for previously unobserved scenes. For this, we run the network to re-plan paths that we have previously observed in real life.

Figure 7 shows planning results for two different maneuvers. The first test case is a simple maneuver of lane following as depicted in Fig. 7a. The network performs well and can actually generate paths very similar to the actual human behavior.

A more challenging task is depicted in Fig. 7b. Here, we asked the network to plan a path for a right turn with integrated lane change. While the human performs both tasks in one, the network first takes the right turn before taking an abrupt lane change right at the end of the path (left in Fig 7b). Also note the planning artifacts at the small drive right at the bend of the planned path.

From these results, we conclude that the network trained on such a small dataset may perform reasonably for very simple driving situations. However, it does not yet generalize well for more complex tasks.

As a final sanity check, we may look at the feature filter masks trained in the lowest layer of the network. The filter bank is depicted in Fig. 8. Note that the network was not initialized from any existing net but instead was trained from scratch on the five training images.

Judging from the filter masks, the network bases its

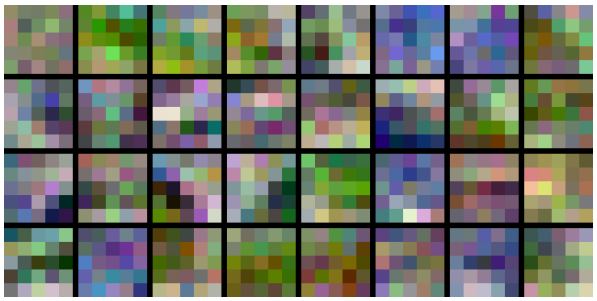


Fig. 8: Trained filter masks in lowest layer

decision in planning on two kinds of features. For once there are filter masks that represent gray-scale and blueish edges. These kinds of features are found on roads. The blueish hue of filters may stem from the blue haze of shadow areas. The other strong feature focus lies on green colors. This is due to roads being bounded by terrain and vegetation.

IV. Conclusion

In this work, we proposed a neural network architecture to execute planning. We trained a Value Iteration Network to imitate human motion planning behavior. A Network trained on simulated trajectories showed the capability to reproduce human actions in simple driving situations. It is especially noteworthy that this result could be achieved from simulated paths in only five different road layouts. We expect the network to generalize well if more training trajectories are provided. In general, we have shown that it is possible to replicated human planning using a unified Neural Network Architecture. This implies that intelligent vehicles might learn strategic planning while in operation in real traffic.

References

[1] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," *Computer*, vol. 22, no. 6, pp. 46–57, 1989.

[2] J. Ziegler and C. Stiller, "Spatiotemporal state lattices for fast trajectory planning in dynamic on-road driving scenarios," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct 2009, pp. 1879–1884.

[3] J. Ziegler, P. Bender, T. Dang, and C. Stiller, "Trajectory planning for berthaa local, continuous method," in *Intelligent Vehicles Symposium Proceedings, 2014 IEEE*. IEEE, 2014, pp. 450–457.

[4] P. Bender, Ö. Ş. Taş, J. Ziegler, and C. Stiller, "The combinatorial aspect of motion planning: Maneuver variants in structured environments," in *Intelligent Vehicles Symposium (IV), 2015 IEEE*. IEEE, 2015, pp. 1386–1392.

[5] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[6] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning." in *AAAI*, vol. 8. Chicago, IL, USA, 2008, pp. 1433–1438.

[7] B. D. Ziebart, N. Ratliff, G. Gallagher, C. Mertz, K. Peterson, J. A. Bagnell, M. Hebert, A. K. Dey, and S. Srinivasa, "Planning-based prediction for pedestrians," in *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2009, pp. 3931–3936.

[8] 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.

[9] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1–9.

[10] P. Fischer, A. Dosovitskiy, E. Ilg, P. Häusser, C. Hazırbaş, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox, "Flownet: Learning optical flow with convolutional networks," *arXiv preprint arXiv:1504.06852*, 2015.

[11] 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*, 2015, pp. 3431–3440.

[12] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "Deepdriving: Learning affordance for direct perception in autonomous driving," in *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[13] A. Tamar, Y. Wu, G. Thomas, S. Levine, and P. Abbeel, "Value iteration networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 2154–2162.

[14] T. Shankar, S. K. Dwivedy, and P. Guha, "Reinforcement learning via recurrent convolutional neural networks," *arXiv preprint arXiv:1701.02392*, 2017.

[15] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*. ACM, 2014, pp. 675–678.

[16] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[17] Google, "Google maps," http://maps.google.com.

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[19] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," *arXiv preprint arXiv:1511.07122*, 2015.