2020 IEEE International Conference on Robotics and Automation (ICRA)
31 May - 31 August, 2020. Paris, France

One-Shot Multi-Path Planning for Robotic Applications Using Fully Convolutional Networks

Tomas Kulvicius[1], Sebastian Herzog[1], Timo Lüddecke[1], Minija Tamosiunaite [1,2] and Florentin Wörgötter[1]

***Abstract*— Path planning is important for robot action execution, since a path or a motion trajectory for a particular action has to be defined first before the action can be executed. Most of the current approaches are iterative methods where the trajectory is generated by predicting the next state based on the current state. Here we propose a novel method by utilising a fully convolutional neural network, which allows generation of complete paths even for several agents with one network prediction iteration. We demonstrate that our method is able to successfully generate optimal or close to optimal paths (less than 10% longer) in more than 99% of the cases for single path predictions in 2D and 3D environments. Furthermore, we show that the network is — without specific training on such cases — able to create (close to) optimal paths in 96% of the cases for two and in 84% of the cases for three simultaneously generated paths.**

I. INTRODUCTION

Motion (path) planning is one of the fundamental issues in the development of truly autonomous robots, let it be a mobile robot or a robotic-manipulator. In robotics, motion planning is defined as the problem of finding a temporal sequence of valid states (e.g., robot positions or configurations), which brings a robot (-arm) from a start- to a goal-position (configuration) given some constraints (e.g., obstacles) [18]. In this work, we specifically address the issue of multiple path planning for multi-agent systems.

Classical methods for path planning are the Dijkstra algorithm [5] and the A* search [11]. Dijkstra and A* algorithms provide always the optimal solution (i.e., shortest path) on grid-based path finding problems, but they can be slow when having to find long paths especially in higher dimensions.

Sampling based methods such as the rapidly-exploring random tree algorithm (RRT, [19], [15], [14], [8]) can also be used for path finding and operate well in continuous spaces. Their performance, however, is worse on grids when comparing them to Dijkstra or A* [17], [1], i.e., RRT-paths are often not optimal. In addition, one has to tune parameters, which is not the case for the classical algorithms. This makes sampling based methods problematic also because they are on grids computationally more expensive [17], [1].

Bio-inspired neural networks [9], [10], [2], [30], [21], [26] have also been employed for path finding. In these approaches, one uses the activation of a target neuron and propagates its activity to its neighbouring cells where the activity gradient defines the path. One observes that these methods are, thus, related to the Dijkstra algorithm.

Recently, several path planning methods have been proposed using deep learning approaches such as deep multilayer perceptrons (DMLP, [25]), long short-term memory (LSTM) networks [1], and deep reinforcement learning (deep-RL) approaches [28], [22]. All these methods generate paths iteratively by predicting the next state or the next action (in case of deep-RL)based on the environment configuration, the current state, and the target position until the target is reached. Thus, the network has to be exploited many times until a complete path can be constructed.

Path planning for multi-agent systems, usually, is performed by decentralised approaches [29], [4], [3], [7], [20]) where the path is planned for each agent separately such that computation time scales with the number of agents. In case of centralized approaches, paths are computed for each agent simultaneously by solving complex optimization problem which does not scale well as the number of agents increases [20]. Different from the afore mentioned approaches, in this study we present a method based on a fully convolutional network (FCN), which allows multi-path planning in one-shot, i.e., complete multiple paths can be generated by our network with a single prediction iteration.

II. METHODS

A. Overview

The task is to predict an optimal path (or several paths) for an unseen environment with obstacles (not used in the training set), given any start- and end-point (goal). In this study, we have considered both 2D and 3D environments.

We use environments defined as occupancy grids in the shape of a binary image of linear size n[1]. Free spaces are given as zero (white) and obstacles are given as one (black). For start- and end-points we use two more binary images with start/end point set to one (black) and the all other points as zero (see input in Fig. 1).

After training, the model predicts a collision-free path again represented as binary image using the same encoding (black) for the path grid elements(see output in Fig. 1). The trajectory for the agent is then created by combining forward- with backward-search (from start- and end-point, respectively) along the marked path grid elements.

*The research leading to these results has received funding from the European Community's H2020 Programme (Future and Emerging Technologies, FET) under grant agreement no. 732266, Plan4Act.

[1]T. Kulvicius, S. Herzog, T. Lüdecke, M. Tamosiunaite and F. Wörgötter are with Department for Computational Neuroscience, University of Göttingen, 37073 Göttingen, Germany tomas.kulvicius@uni-goettingen.de

[2]M. Tamosiunaite is also with the Faculty of Computer Science, Vytautas Mangnus University, Kaunas, Lithuania

[1]Note that our approach can also be used for environments of size $n \times m$.

Details of all algorithmic components are given next using annotations for the 2D case. The 3D case is similar.

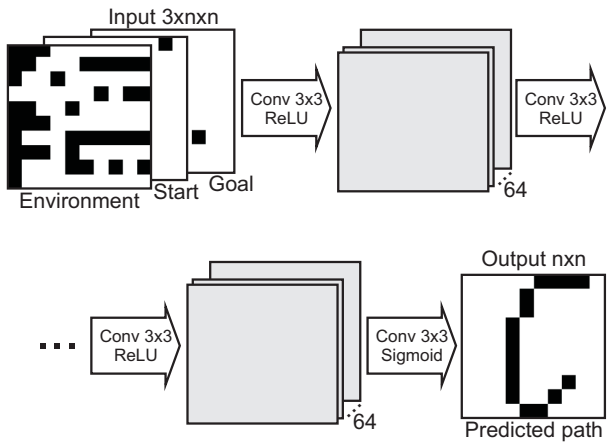

Fig. 1. Proposed network architecture. We used a fully convolutional neural network (FCN) with 21 convolutional layers. We used ReLU activation units in the layers 1 to 20 and a sigmoid activation unit in the last layer. We used a batch normalisation layer after each convolutional layer and in addition a dropout (10%) layer after the last convolutional layer (not shown). For more details please refer to the main text.

B. Data

Inputs and maps are defined as described above. In addition to the case of single start- and end-points we also predict multiple-paths from several start-positions to the same end position. In these cases several grid cells in the start map were set to 1. The output map, which contains the found path, is then represented in the same way as all other maps by setting the grid-cell to 1, if the path traverses this grid cell and 0 everywhere else. Optimal paths were created as ground truth by using the A* algorithm, where eight movement directions (vertical, horizontal and diagonal) were allowed. Movement costs were calculated using Euclidean metric.

Data generation: The input maps for training and testing were generated randomly by setting each grid cell to 1 (i.e, obstacle) with a probability $p_o = 0.6$, or to 0, otherwise. We disallowed the following two diagonal configurations of obstacles: $\{I^e_{i,j} = 0, I^e_{i,j+1} = 1, I^e_{i+1,j} = 1, I^e_{i+1,j+1} = 0\}$ and $\{I^e_{i,j} = 1, I^e_{i,j+1} = 0, I^e_{i+1,j} = 0, I^e_{i+1,j+1} = 1\}$. In these cases A* algorithm would go through two diagonally arranged obstacles that touch each other only by a corner, which in real scenarios is impossible. In case of 3D grids, we assumed that objects (obstacles) are standing on the ground (like buildings in a city) and do not float in the air.

We generated two datasets (2D grids and 3D grids) for learning and testing of single paths and one additional dataset for multiple paths with up to three start-positions. Training did not include multiple paths.

In general, we used environments with linear sizes $n = 10, 15$, and 20 (see Fig. 2 for $n = 15$) and here we used 30,000 environments for 2D and 97,000 for 3D with different obstacles and different start- and end-points. The minimum Euclidean distance between start- and end-points was 5 to avoid short (trivial) paths.

1,000 environments of size 15×15 with different obstacle configurations but fixed start- and end-positions where created for the prediction of multiple paths. We fixed start- and end-points to positions $\{1, 1\}$, $\{1, n\}$ and $\{n, 1\}$ (corners) and the goal was in the middle (position $\{8, 8\}$) (see Fig. 4).

C. Proposed Network

Network architectures: For the 2D case, a fully convolutional network, as shown in Fig. 1, was used. The three above discussed 2D binary images of size $n \times n$ constitute the input layer and the output a 2D image that contains the path candidates. 20 identical 2D convolutional layers with 64 filters of size 3×3 (stride 1) were used and finally also one more convolutional layer at the end with just one filter of size 3×3. A batch normalisation layer (not shown) was used after each convolutional layer, and after the last convolutional layer we included a dropout layer with 10% dropout. In all but the last convolutional layer we used ReLU activation units. In the last convolutional layer sigmoid activation was used. Network output is given by $0 \leq \hat{O} \leq 1$. Zero padding was used in all layers to keep the same dimensions.

For the 3D case, 3D convolutional layers were used with filters $3 \times 3 \times 3$, but we needed more filters in the first four layers (1024, 512, 256 and 128), because of the higher dimension of the input.

Training procedure: As loss we employed the mean squared error (MSE) between network output $\hat{O}$ and ground truth solution O. Batch sizes were 64 samples (for 2D) and 16 samples (for 3D), where the latter was limited by our hardware. If the accuracy on the validation set did not increase within the last 10 epochs, early validation stopping was used to prevent over-fitting. We used 26,000 samples for training and 2,000 samples for validation in 2D, while in 3D 90,000 samples for training and 5,000 samples for validation were used. Finally, all networks were tested on 2,000 new, unseen samples. The ADAM optimiser with default learning parameters was employed for training both, 2D and 3D, networks.

For network implementation we used Tensorflow and Keras API[2], where the A* algorithm was implemented in Python 3.6.7. We used a PC with Intel Xeon Silver 4114 (2.2GHz) CPU and NVIDIA GTX 1080 (11GB) GPU.

D. Path Reconstruction

As explained above, the output of the network is a value map $\hat{O}$ of linear size n, with $0 \leq \hat{O} \leq 1$. The numbers reflect "how certain the network is" that the respective grid cell is part of the path. However, this map does not yet represent the sequence of how to traverse these grid cells one after the other and an additional step, called path reconstruction, is needed. Here we used bidirectional search such that we represent forward and the backward path-segments as

[2]The data and source code is available online at `https://alexandria.physik3.uni-goettingen.de/cns-group/datasets/path_planning/`

temporal sequence of points on the grid with $\{x_t^f, y_t^f\}$ and $\{x_t^b, y_t^b\}$, respectively. Start-point is given by $\{x_s, y_s\}$ and end-point by $\{x_g, y_g\}$. Initially we set $\{x_1^f, y_1^f\} = \{x_s, y_s\}$ and $\{x_1^b, y_1^b\} = \{x_g, y_g\}$. From any current position $\{x_t, y_t\}$ of the forward/backward path-segment, the next position of the forward/backward path is the maximum-valued grid cell $\{i, j\}$ in the nearest neighbourhood of the current position, hence, that cell where:

$$\{x_{t+1}, y_{t+1}\} = \arg\max_{i,j} \hat{O}_{i,j} \{i \neq x_t, j \neq y_t, \quad i \in [x_t - 1, x_t + 1], j \in [y_t - 1, y_t + 1]\}. \tag{1}$$

Then we set $\hat{O}_{i,j} = 0$ and continue until one of the three following conditions is met:

1) End-point is reached, $\{x_t^f, y_t^f\} = \{x_g, y_g\}$;
2) Start-point is reached, $\{x_t^b, y_t^b\} = \{x_s, y_s\}$;
3) The paths cross each other, $\{x_t^f, y_t^f\} = \{x_k^b, y_k^b\}$ or $\{x_t^b, y_t^b\} = \{x_k^f, y_k^f\}$, where $1 \leq k \leq t$.

Depending on the condition met, the final path $P(x, y)$ is then constructed as follows:

$$P(x, y) = \{(x_1^f, y_1^f) \dots (x_M^f, y_M^f)\}, \tag{2}$$
if condition (1) is met;

$$P(x, y) = \{(x_N^b, y_N^b) \dots (x_1^b, y_1^b)\}, \tag{3}$$
if condition (2) is met;

$$P(x, y) = \{(x_1^f, y_1^f) \dots (x_t^f, y_t^f), (x_{k-1}^b, y_{k-1}^b) \dots (x_1^b, y_1^b)\}, \text{ or} \tag{4}$$
$$\{(x_1^f, y_1^f) \dots (x_{k-1}^f, y_{k-1}^f), (x_t^b, y_t^b) \dots (x_1^b, y_1^b)\},$$
if condition (3) is met.

Here, M and N are the lengths of the forward and the backward path, respectively. In real applications, the points of the path $P(x, y)$ can then be used as via points to generate trajectories using conventional methods such as splines [6], [27] or more advanced state-of-the-art methods such as dynamic movement primitives (DMPs, [13]), Gaussian mixture models (GMMs, [16]), probabilistic movement primitives (PMPs, [23]) or optimal control primitives (OCPs, [12]). Note that in some cases paths could not be reconstructed, because none of the stopping conditions were met. In this case we treated network's prediction as "path not found".

E. Evaluation Measures and Procedure

We compared our approach against the A* algorithm [11]. We have chosen A* against rapidly-exploring random trees (RRT [19] or RRT* [15]), since it has been shown that algorithms such as Dijkstra and A* perform better on grid structures as compared to RRTs [17], [1].

Thus, for comparison we used the following criteria: success rate, path optimality and run-time of the algorithm. Below we describe our evaluation measures and procedure in more detail.

Success rate: Success was scored if the path $P(x, y)$ could be reconstructed, otherwise we counted a failure (path not found). We define:

$$\boldsymbol{SR} = 100\% \cdot (N_S / N_T), \tag{5}$$

where N_S is the number of successfully found paths and N_T is the total number of tested environments, which gives a percent value of success.

Path optimality: We also measured whether successfully predicted paths were optimal (i.e., shortest path) or not. For that, we compared path lengths of the paths obtained by using the FCN and A* algorithms, denoted as L_{FCN} and L_{A*}, respectively. Path length was computed by (we skip FCN and $A*$ notation for clarity):

$$L = \sum_{t=1}^{n-1} \|P(x_{t+1}, y_{t+1}) - P(x_t, y_t)\|, \tag{6}$$

where $P(x, y)$ is the path predicted by the FCN or A* algorithm, and n is the number of points in the path sequence. Here, $\|\cdot\|$ is the Euclidean distance. The path predicted by the FCN is optimal if $L_{FCN} = L_{A*}$. Hence, the percentage of optimal paths OP is:

$$\boldsymbol{OP} = 100\% \cdot (N_O / N_T), \tag{7}$$

where N_O is the number of optimal paths. Because paths predicted by FCN are not always optimal, we also computed the path length ratio LR of non-optimal relative to optimal paths:

$$\boldsymbol{LR} = L_{FCN} / L_{A^*}. \tag{8}$$

This allows us to analyse how much longer non-optimal paths are compared to the A* solutions.

Algorithm run-time: For run-time evaluation, path length was measured in steps, i.e., how many steps it takes to move from the start-point to the end-point. Thus, for run-time comparison we measured how much time it takes to predict (in case of FCN) or to find (in case of A*) a single path for different path lengths. For run-time calculation we included both path search/prediction time and path reconstruction time. Note that A* algorithm (similar to FCN) consist of two steps: path search and path reconstruction from visited states.

Evaluation procedure: Two types of experiments were performed: prediction of single paths in 2D and 3D and prediction of multiple paths in 2D. For the first set of experiments, we trained the network to predict single paths on three different grids with linear sizes $n = 10, 15, 20$) and then tested each model on all three grids. For the second type of experiments we trained on single, but tested on multiple paths (two or three paths). Grids with linear size $n = 15$ were used for this.

III. RESULTS

A. General Performance of the Proposed Network

In Fig. 2 we show examples of single path predictions on 2D grids of linear size $n = 15$, where the path suggestions by the network are marked by blue dots[3]. Small dots represent values close to zero and large dots close to one. Crosses mark the optimal solution, which is the one obtained when

[3]More examples of 2D and 3D path predictions can be found at https://alexandria.physik3.uni-goettingen.de/cns-group/datasets/path_planning/.

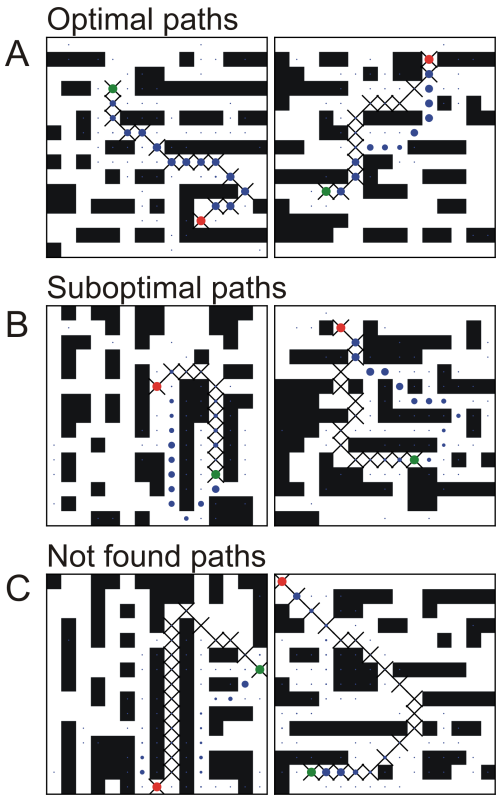

Fig. 2. Single path predictions on unseen environments for grids with linear size $n = 15$ when training and testing on the same grid size. **A)** optimal (shortest) paths, **B)** suboptimal paths, **C)** not found paths. Crosses annotate the A* solution, where blue dots denote the predicted path using our network. Dot size represent small (≈ 0.0) and large (≈ 1.0) values of the network output. Start and endpoint are shown by green and red dots, respectively.

using the A* algorithm. The network is able to predict optimal paths in most of the cases (see statistical analysis in Table I). In panel A we show an example of an optimal path (left), which is identical to the A* solution and an optimal path which differs from the A* path (right). Most often this happens when going around an obstacle from the other side. In panel B one can see paths that are suboptimal but still feasible. Hence these paths take more steps than the A* solution but in most of cases they are only less than 10% longer. Panel C shows cases where paths could not be reconstructed from the network output.

In Table I we show a statistical analysis on how our networks perform on different 2D- and 3D-grids. When trained and tested on the same grid size we found a very high success rate, which is above 99.5% for 2D as well as 3D cases. Remarkably, we also find that these networks can predict paths with high reliability even on a grid sizes that had not been used for training. With one exception (3D case), success rate is above 98% and 88% for the 2D and 3D, respectively. The exception is the "extreme" case of a network trained on a linear grid size of $n = 10$ and tested on grids of size $n = 20$. Naturally, performance goes down when training on small and testing on large grids, since the large grids contain many "never seen" long paths, which just do not exist in the smaller grids.

The above results had shown that the network is most often able to predict feasible paths, but those are not always optimal (see Table I). Sub-optimal paths are more often obtained in larger grids of linear size $n = 20$ (from 58.10% to 86.55% and from 53.34% to 83.02% for 2D and 3D case, respectively) as compared to smaller grids of size $n = 10$ (from 98.05% to 99.85% and from 92.88% to 93.74% for 2D and 3D, respectively). This is due to the fact that a larger number of longer paths exist in larger grids and those can more easily lead to prediction errors.

Finally we measured paths length asking how much longer non-optimal paths are relative to the optimal (shortest) path. As above, better performance is obtained if the network is trained on the same or on larger grids than those used for testing. Still, non-optimal paths are, on average, less that 10% longer compared to the optimal path (in a range between 5% and 7%, and between 7% and 9% for 2D and 3D, respectively).

In the last experiment we compared run-time, i.e., how much time it takes to process one environment, between A* and FCN. Results obtained on the the 2D and 3D grid of linear size 20 are presented in Fig. 3. Here we show run-time of each realization (in total 2,000 environments) for both A* and FCN. As expected, results show that on average run-time of A* increases quadratically if paths are getting longer, whereas run-time of FCN on average increases linearly. Note that prediction time of the FCN is constant and the linear increase is due to the path reconstruction time (longer paths take more time to reconstruct as compared to shorter paths). Fitted functions predict that in 2D case for paths longer than 30 steps (e.g., in larger environments), FCN would outperform A*, whereas in the 3D case FCN outperforms A* if the paths are longer than 13 steps. Note that in 3D, path reconstruction time for FCN becomes negligible. Moreover, our proposed network is able to predict multiple paths within the same time, whereas in case of A* one would need to run path search several times, which — as a consequence — would scale run-time by k, where k is the number of searched paths.

B. Prediction of Multiple Paths

In Fig. 4 we show examples of two- (panels B1-B4) and three-path predictions (panels C1-C4), where panels A1-A4 are control cases (single-path predictions). As before, also here we show cases of optimal, suboptimal and not found paths. Our results can be summarized into four different cases:

1) Optimal multi-paths are found and path finding is not affected by adding a second or third source (see optimal paths in panels A1-C1).
2) One or more paths become suboptimal when adding more sources. One example of this can be found in panels A2 and B2. Here an optimal path (A2) turns into a suboptimal one when adding a second source (see panel B2).

TABLE I

RESULTS FOR THE PREDICTION OF SINGLE PATHS IN 2D AND 3D ENVIRONMENTS OBTAINED FROM 2,000 TESTED UNSEEN ENVIRONMENTS.

2D		Tested on								
		10×10	15×15	20×20	10×10	15×15	20×20	10×10	15×15	20×20
		Success rate (%)			**Optimal paths** (%)			**Path length ratio** (Mean ± CI [95%])		
Trained on	10×10	**100**	99.75	98.45	**99.85**	83.51	58.10	**1.07**±0.039	1.10±0.011	1.21±0.020
	15×15	99.90	**99.95**	98.90	97.25	**91.00**	75.53	1.07±0.012	**1.07**±0.009	1.09±0.008
	20×20	99.90	99.70	**99.60**	98.05	94.38	**86.55**	1.07±0.015	1.05±0.006	**1.06**±0.015
3D		Tested on								
		10×10×10	15×15×15	20×20×20	10×10×10	15×15×15	20×20×20	10×10×10	15×15×15	20×20×20
		Success rate (%)			**Optimal paths** (%)			**Path length ratio** (Mean ± CI [95%])		
Trained on	10×10×10	**99.85**	90.60	62.05	**93.74**	58.33	53.34	**1.07**±0.007	1.20±0.014	1.36±0.021
	15×15×15	99.70	**99.95**	88.50	93.18	**89.24**	61.69	1.09±0.011	**1.07**±0.008	1.20±0.014
	20×20×20	99.75	99.95	**99.80**	92.88	88.49	**83.02**	1.09±0.010	1.07±0.009	**1.07**±0.009

3) Paths can "disappear" when adding a second or third source (see panels A3, B3 and C3).
4) Or finally, as shown in panels A4 and C4, a path can "appear" when adding more sources.

In Fig. 5 we provide a statistical summary for multi-path prediction. We compare here the prediction performance for the one-path control case with two and three paths. All tests were performed on linear grids of size $n = 15$.

We obtain success rates of 96.4% and 83.9% for finding two paths out of two searched paths and three paths out of three searched paths, respectively. Hence, with increased number of searched paths performance is slightly decreasing. Remarkably, for the difficult case of trying to find three-paths, we still achieve always at least one path, and success rate for predicting two paths was still quite high (99.2%). The here-observed improvement for the two-path-search can be understood by the fact that there is a higher probability to find two paths out of three than two out of two.

For multi-path search we find, however, that there are fewer optimal paths (85.88% and 83.33% for two-path and three-path search, respectively). Also now paths are getting longer (15% and 22% for two-path and three-path search, respectively) when the number of searched paths increases. This is a consequence of the fact that paths are more likely to intersect in the case of multiple sources and this way they are less optimal.

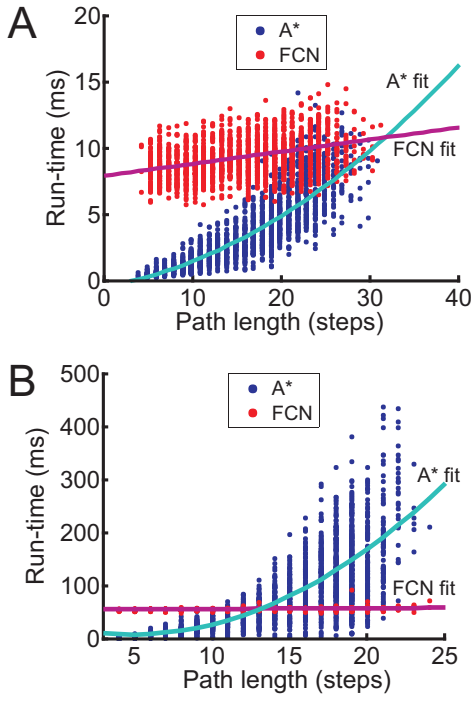

Fig. 3. Results for the run-time comparison between A* and FCN obtained from 2000 tested samples on 2D (panel A) and 3D grids (panel B) of size 20. We used a first order and a second order polynomial functions to fit FCN and A* data, respectively.

IV. DISCUSSION

In the current study, we have investigated how to generate single as well as multiple paths using fully convolutional networks. To the best of our knowledge, this is novel, since it allows predicting multiple paths in a "single shot" (one-time network prediction). We would like to emphasize that there are other deep-learning approaches existing (discussed above) but those generate paths iteratively and they can only plan a single path [28], [22], [25], [1]. If multiple-paths are considered then this happens for each agent separately [20], [3], [7] or they only deal with collision avoidance path planning. Hence, path planning of paths for navigation in maze-like environments is not addressed in these studies.

Recently, an interesting paper by [24] suggested also one-shot path planning with a fully convolutional network. However, their approach addresses only single path planning and they deal with human-aware collision-free navigation in rather simple (object free) environments (not mazes). Note that their method is based on two steps: 1) A network predicts the path and 2) then the RRT* algorithm [15] is employed for path refinement. All this is computationally more expensive as compared to our approach of path reconstruction by bidirectional search.

We have demonstrated that in case of single path predictions our proposed network is able to predict optimal or close to optimal paths successfully in more than 99.5% of the cases (when trained and tested on the same grid size) in both 2D and 3D environments. Furthermore it generalizes quite well on grid sizes different from the trained ones as long as they do not differ too much.

In case of multiple path prediction, we have shown that, although the network has never been trained on multiple

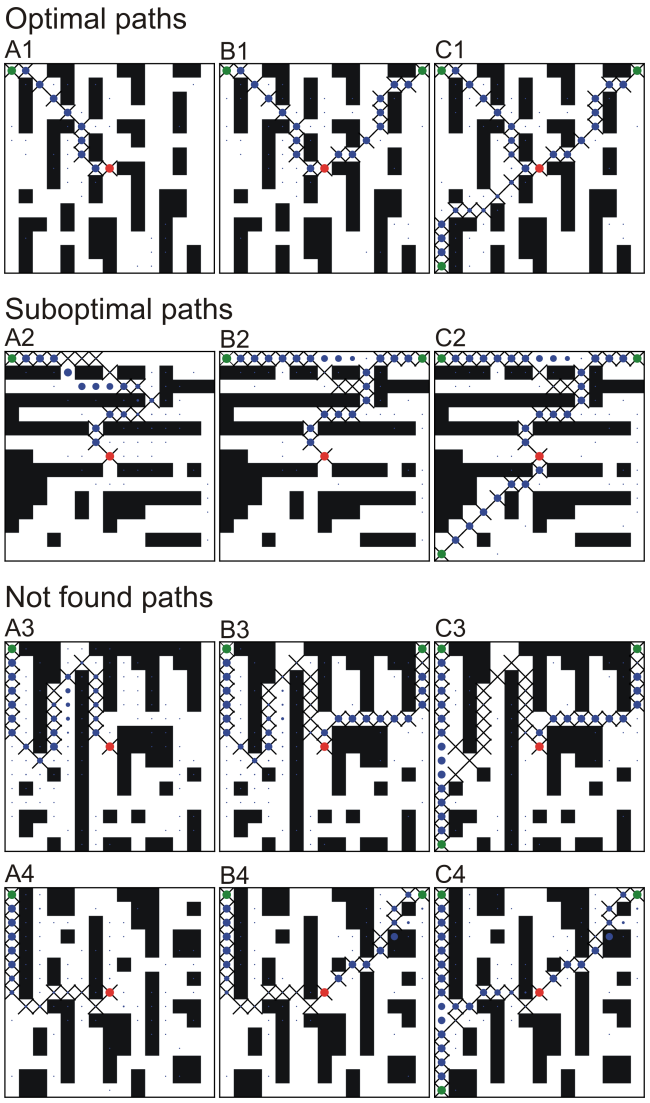

Fig. 4. Examples of multi-path predictions on unseen environments of linear size 15. **A)** Single path predictions (control case), **B)** predictions of two paths, and **C)** prediction of three paths. Crosses denote A* solutions where blue dots denote predicted paths using FCN. Size of the dots correspond to small (close to zero) and large (close to one) values of the network output. Green and red dots correspond to start-points and end-points, respectively. Note that the predicted path shown in panel A2 is an optimal path, and that the predicted path shown in panel A3 is a feasible path.

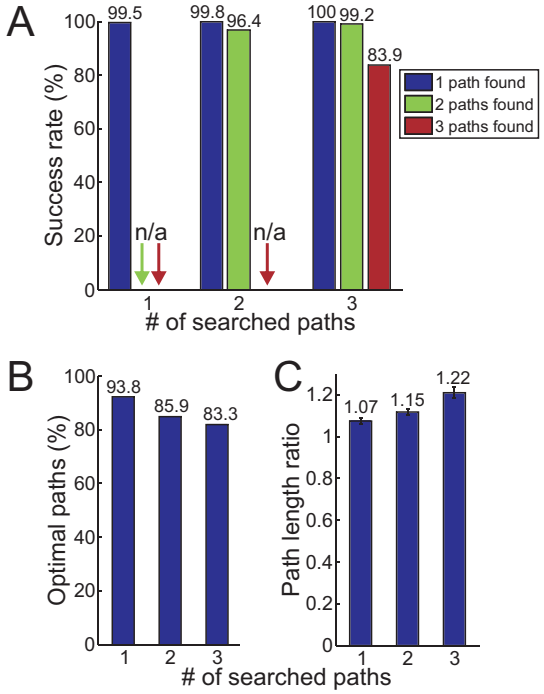

Fig. 5. Results for the prediction of multiple paths obtained from 1,000 tested unseen environments. A) Success rate, B) percentage of optimal paths, C) path length ration of non-optimal paths. Error bars in panel C denote confidence intervals of mean (95%).

paths, it is also able to generate paths from multiple sources to one target. This could be also used to solve single-source-multi-target (or vice versa) problems. We obtained a success rate of 96.4% and 83.9% for the prediction of two and three paths, respectively. Naturally, results should improve by including samples of multiple paths in the training set. The other option would be to repeat the network's prediction one or several more times by giving locations at which the path reconstruction was lost as new start- and end-points. Note, however that the goal of this study was to show how far one can get by not doing this.

An interesting extension of our approach concerns the possibility to use our method also in dynamic (changing) environments. For example, if the maze changes during one run of a robot towards its goal, the agent could just use its current position as the new start-point and perform another one-shot path finding procedure. Thus, this method can be used for fast and efficient on-line (re-)planning in dynamic environments, too.

In summary, there are several quite useful features of our method: 1) single-shot operation, 2) possibility for multi-path planning, 3) constant prediction run-time, 4) relatively high success and optimality rates, and 5) the ability to generalise to environments of different sizes. This should make it a valuable and attractive approach for many robotic applications.

REFERENCES

[1] M. J. Bency, A. H. Qureshi, and M. C. Yip. Neural path planning: Fixed time, near-optimal path generation via oracle imitation. *CoRR*, abs/1904.11102, 2019.

[2] N. Bin, C. Xiong, Z. Liming, and X. Wendong. Recurrent neural network for robot path planning. In *Int. Conf. on Parallel and Distributed Computing: Applications and Technologies*, pages 188–191, 2004.

[3] Y. F. Chen, M. Liu, M. Everett, and J. P. How. Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning. In *2017 IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 285–292, 2017.

[4] V. R. Desaraju and J. P. How. Decentralized path planning for multi-agent teams with complex constraints. *Autonomous Robots*, 32(4):385–403, 2012.

[5] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[6] M. Egerstedt and C. F. Martin. Optimal trajectory planning and smoothing splines. *Automatica*, 37(7):1057–1064, 2001.

[7] M. Everett, Y. F. Chen, and J. P. How. Motion planning among dynamic, decision-making agents with deep reinforcement learning. *CoRR*, abs/1805.01956, 2018.

[8] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot. Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *2014 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 2997–3004, 2014.

[9] R. Glasius, A. Komoda, and S. Gielen. Neural network dynamics for path planning and obstacle avoidance. *Neural Networks*, 8(1):125–133, 1995.

[10] R. Glasius, A. Komoda, and S. Gielen. A biologically inspired neural net for trajectory formation and obstacle avoidance. *Biological Cybernetics*, 74(6):511–520, 1996.

[11] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Systems Science and Cybernetics*, 4(2):100–107, 1968.

[12] S. Herzog, F. Wörgötter, and T. Kulvicius. Generation of movements with boundary conditions based on optimal control theory. *Robotics and Autonomous Systems*, 94:1–11, 2017.

[13] J. A. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal. Dynamical movement primitives: Learning attractor models for motor behaviors. *Neural Comput.*, 25(2):328–373, February 2013.

[14] F. Islam, J. Nasir, U. Malik, Y. Ayaz, and O. Hasan. RRT*-Smart: Rapid convergence implementation of RRT* towards optimal solution. In *2012 IEEE Int. Conf. on Mechatronics and Automation*, pages 1651–1656, 2012.

[15] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011.

[16] S. M. Khansari-Zadeh and A. Billard. Learning Stable Non-Linear Dynamical Systems with Gaussian Mixture Models. *IEEE Trans. Robot.*, 27:943–957, Oct. 2011.

[17] L. Knispel and R. Matousek. A performance comparison of rapidly-exploring random tree and Dijkstra's algorithm for holonomic robot path planning. *Institute of Automation and Computer Science, Faculty of Mechanical Engineerig, Brno University of Technology*, 2013.

[18] J.-C. Latombe. *Robot motion planning*, volume 124. Springer Science & Business Media, 2012.

[19] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. *Technical Report*, 1998.

[20] P. Long, W. Liu, and J. Pan. Deep-learned collision avoidance policy for distributed multiagent navigation. *IEEE Robotics and Automation Letters*, 2(2):656–663, 2017.

[21] J. Ni, L. Wu, P. Shi, and S. X. Yang. A dynamic bioinspired neural network based real-time path planning method for autonomous underwater vehicles. *Computational intelligence and neuroscience*, 2017, 2017.

[22] A. I. Panov, K. S. Yakovlev, and R. Suvorov. Grid path planning with deep reinforcement learning: Preliminary results. *Procedia computer science*, 123:347–353, 2018.

[23] A. Paraschos, C. Daniel, J. Peters, and G. Neumann. Probabilistic movement primitives. In C.J.C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 2616–2624, 2013.

[24] N. Pérez-Higueras, F. Caballero, and L. Merino. Learning human-aware path planning with fully convolutional networks. In *2018 IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 1–6, 2018.

[25] A. H. Qureshi, M. J. Bency, and M. C. Yip. Motion planning networks. *CoRR*, abs/1806.05767, 2018.

[26] E. Rueckert, D. Kappel, D. Tanneberg, D. Pecevski, and J. Peters. Recurrent spiking networks solve planning tasks. *Scientific Reports*, 6:21142, 2016.

[27] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo. *Robotics: Modelling, planning and control*. Springer Publishing Company, Incorporated, 2009.

[28] L. Tai, G. Paolo, and M. Liu. Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. In *2017 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 31–36, 2017.

[29] K.-H. C. Wang and A. Botea. MAPP: A scalable multi-agent path planning algorithm with tractability and completeness guarantees. *Journal of Artificial Intelligence Research*, 42:55–90, 2011.

[30] S. X. Yang and M. Meng. Neural network approaches to dynamic collision-free trajectory generation. *IEEE Trans. on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 31(3):302–318, 2001.