A LSTM Neural Network applied to Mobile Robots Path Planning

Fiorato Nicola, Yasutaka Fujimoto
Department of Electrical and Computer Engineering
Yokohama National University
Yokohama 240-8501 Japan
fiorato.nicola@gmail.com, fujimoto@ynu.ac.jp

Roberto Oboe
Department of Engineering and Management
University of Padova
Vicenza 36100 Italy
roberto.oboe@unipd.it

***Abstract*—Mobile robots path planning is a central problem in every situation where human intervention is not desired or not possible to accept: full automated industrial warehouses or general stocking areas and every domestic application that involves a mobile robot and special cases where environment is prohibited for human accessing like toxic wastes and bombs defusing [1]. Currently, neural networks are applied to problems related to mobile robot navigation. However, they are not as popular as in applications like image processing, speech recognition or machine translation, where they are commercially relevant. In this paper we propose a *Long Short-Term Memory* (LSTM) neural network as an online search agent to tackle the problem of mobile robots path planning in unknown environments, meaning that the agent relies only on local map awareness realized with a LRF sensor and relative information between robot and goal position. Specifically, a final structure of LSTM network is analyzed and its performance is compared with the A* algorithm, a widely known method that follows the best-first search approach. Subsequently, an analysis of the method developed on a real robot is described.**

***Index Terms*—Artificial Intelligence - LSTM - Path-Planning - Mobile Robots**

I. Introduction

Recently, neural networks have been successfully applied in path planning for mobile robots [2] [3] [4] [5]. The prospect of using recurrent neural networks increased in the last few years, since they showed to achieve the highest performances in various sequence processing tasks like speech recognition and video captioning [6] [7] [8].

The hypothesis of unknown environment is taken into account because in the real world it is hard to guarantee prior information about the map or however a component that updates and store the whole map during travelling could be too much costly in terms of computational resources needed.

The method proposed in this paper aims to be a global path planner, following the definition given in [1], it generates a low-resolution high-level path from start to goal. On the other hand, local path planning is a high-resolution low-level path restricted in a near segment of the global path. In particular, we want to realize an online search agent based on the Long Short-Term Memory neural network with a *supervised learning* approach where the ground truth is provided by the A* algorithm [9]. Datasets are realized with the path solutions found by the supervisor, then the neural network is trained and eventually it is deployed as an online agent.

Supervised learning is faster and easier to perform with respect to reinforcement learning. However, it presents an important limit: the experienced state-action pairs during training are limited to the ones decided by the supervisor, so it leads to a more probable wrong decision during effective deployment when the neural network agent gets itself in a state not experienced in the training process.

Understanding the theoretical concept of planning a path is essential to get the reasons why a recurrent neural network can be applied. Starting from the decision theory basics, the agent, given a bunch of input data of different nature, produces preferences as its outputs. So, it creates a relationship between data describing its current environment awareness, easily called *state*, and selects the next *action* in order to take itself from the current state to next state supposed to be in a better position with respect to the goal point. Considering a single pair state and associated action not independent to their neighbours in terms of time steps, series of states and related series of actions can be examined at the same time. Therefore, planning a path becomes a variable sequence to sequence problem, the ideal target of recurrent neural networks as it can be addressed also to the tasks mentioned at the beginning of this section.

A* is an example of offline search agent, it gives the entire path as output, calculated on the map stored in its memory before starting to move. Meanwhile, an online search computes a single decision and execute the related movement using just the current local information [10].

This paper briefly introduces the LSTM neural network, describes how the proposed model is designed and shows the results achieved in a dedicated simulation environment and the real environment.

II. Long-Short Term Memory Neural Network

The Long-Short Term Memory network belongs to the Gated Recurrent Networks (GRN) family, a variant of the basic recurrent neural network. A specific characteristic is that, in their atomic structure, called *memory cell*, there are additional

978-1-5386-4829-2/18/$31.00 ©2018 IEEE

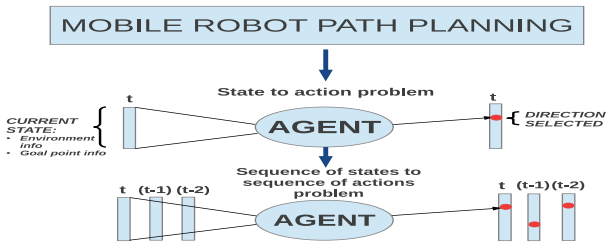

Fig. 1: Conceptual steps on how to consider path planning

elements working as gates, the LSTM case includes gates applied to the input, previous state and output.

From the first idea of LSTM in 1997 [11], there have been proposed many variants of the basic cell structure, but the most common is showed in Figure 2 [12].

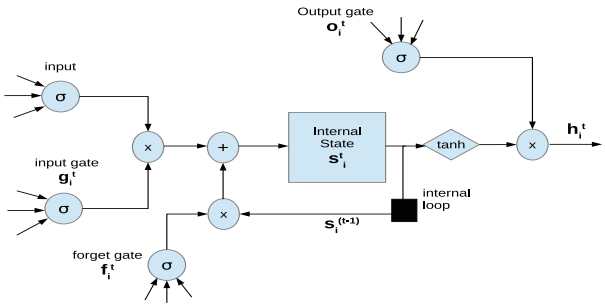

Fig. 2: Representation of the LSTM cell

The input and output signal, and the previous state value are multiplied by their own related gate function: input gate, output gate and forget gate, respectively. Suppose a LSTM network has one layer with I inputs and H cells, the internal state $s_i^{(t)}$ of the i-th cell at time step t is updated by the following equation:

$$s_j^{(t)} = f_j^{(t)}\, s_j^{(t-1)} + g_j^{(t)}\, \sigma\left(b_j + \sum_{i=1}^{I} u_{ij} x_h^{(t)} + \sum_{h=1}^{H} w_{hj} h_h^{(t-1)}\right) \tag{1}$$

where $f_i^{(t)}$ and $g_i^{(t)}$ are the forget gate and the input gate:

$$f_j^{(t)} = \sigma\left(b_j^f + \sum_{i=1}^{I} u_{ij}^f x_i^{(t)} + \sum_{h=1}^{H} w_{hj}^f h_h^{(t-1)}\right) \tag{2}$$

$$g_j^{(t)} = \sigma\left(b_j^g + \sum_{i=1}^{I} u_{ij}^g x_i^{(t)} + \sum_{h=1}^{H} w_{hj}^g h_h^{(t-1)}\right) \tag{3}$$

where σ stands for the *sigmoid* non-linear transformation. The forget gate can be seen as a sort of factor multiplying the previous state just like in a running average, this allows to have a variable weight associated to the past data, if $f_i^{(t)} \to 0$ the recent states information is quickly forgotten, meanwhile if $f_i^{(t)} \to 1$ the past story has an important influence on the next output computation.

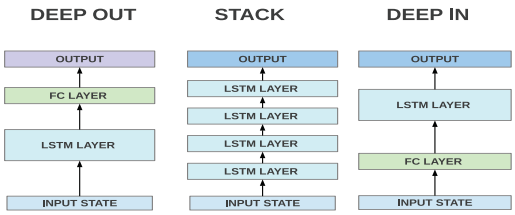

Fig. 3: The deep LSTM neural networks considered

Now, the output of the cell $h_j^{(t)}$ can be calculated as follows:

$$h_j^{(t)} = \tanh\left(s_j^{(t)}\right) q_j \tag{4}$$

$$q_j^{(t)} = \sigma\left(b_j^o + \sum_{i=1}^{I} u_{ij}^o x_i^{(t)} + \sum_{h=1}^{H} w_{hj}^o h_h^{(t-1)}\right) \tag{5}$$

where $q_i^{(t)}$ represents the output gate. The network complexity, directly proportional to the total number of weights, is strongly increased with respect to the basic recurrent neural network, however this leads to the main feature and benefit of the LSTM: ensuring a continuous information flow from the first step to the last step of the time sequence, thanks to the variable gates behaviour, this is the reason why this recurrent network variant is primarily known to learn better the long-term dependencies [13].

Depth in neural networks is, in general, addressed to a sequential application of non linear functions in a single iteration in order to be able to get compositional representations of the input [8].

Given this definition, recurrent neural networks have already this kind of structural property in the temporal dimension. However, it is possible to define specific structures of deep LSTM neural network, the following structures represent the simplest deep LSTM neural network to build:

- deep input to hidden, abbreviated *deep-in*
- deep hidden to output, abbreviated *deep-out*
- stacking LSTM layers, abbreviated *stack*

In Figure 3 are showed the graphical representations of the three structures mentioned above. More detailed and articulated discussion about deep recurrent neural networks can be found here [14] [15].

When applied as an online path planner, a generic neural network configuration would be as showed in Figure 4, where the input state is a flattened vector containing information about the environment, like label values of a grid map or proximity ranges, and heuristic data about the goal point. The output is represented by a full connection layer followed by the softmax transformation in order to create mutually exclusive probabilities, each one associated to a particular allowed action. Eventually, the highest value denotes the choice selected by the agent.

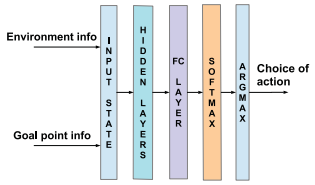

Fig. 4: Generic configuration of a neural network as a path planner

III. Proposed model

The simulation environment consists on a large collection of maps realized with ASCII characters and associated list of scenarios, each one defined by start and goal coordinates and optimal path length [16]. The simulation environment takes into account just static maps.

At time step t, the neural network outputs eight probability values corresponding to a preference given to each of the eight directions allowed, cartesian and diagonal, the choice of action $a^{(t)}$ is the output with highest probability as it corresponds to the most preferred direction. Instead of using a grid map for obstacle awareness, it was found that for this purpose proximity ranges improves the learning performance with respect to a grid map [4]. As relative information between agent position and goal point, relative angle and euclidean distance are included. In addition, we noted that the network predictions benefit from adding the action decided one time step earlier. Summarizing, consider current agent position at coordinates (x_A, y_A) and goal position at coordinates (x_G, y_G), the input state components are defined below and represented in Figure 5:

- $d_i \; \forall \; i = 1, ..., 8$:= proximity ranges
- $\theta_{A \to G} = arctan2 \left(\frac{y_G - y_A}{x_G - x_A} \right)$:= relative angle
- $e_{A \to G} = \sqrt{(y_G - y_A)^2 + (x_G - x_A)^2}$:= euclidean distance
- $a^{(t-1)}$:= the previous step choice of action

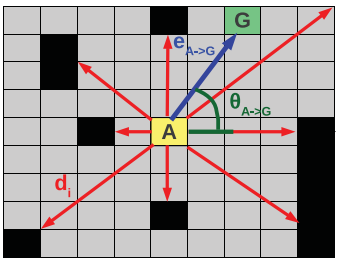

Fig. 5: Depiction of information included in the input state

The ground truth for the training process is provided by the conventional A* algorithm. The training set and validation set specifics are showed in Table I, the two sets are built with paths randomly selected from different maps which belong to the following categories: mazes with corridor having 4 steps width, mazes with corridor having 2 steps width, random filled for 25% map size, random filled for 40 % of the map size. Furthermore, as a sort of complexity limit, the paths which optimal length exceeds the maximum value specified are not included. The fine tuning of the hyperparameters lead to the final configuration described in Table II.

TABLE I: Training set and validation set specifics

Specific	TRAINING SET	VALIDATION SET
Size (steps)	102400	10240
Number of maps	5	3
Maximum path length (steps)	300	300

TABLE II: Hyperparameters configuration

Hyperparameter	Value
Time sequence length (steps)	512
Learning rate (fixed)	0.1
Weight decay	10^{-5}
Regularization type	L_1
Momentum	0.95
Solver (updating rule)	ADADELTA

The design of the model followed a minimal complexity approach: it consists on achieving the maximum learning performance with the least possible amount of internal weights. The structure of the model proposed is showed in Figure 6, training convergence parameters are revealed in Table III where, for a comparison purpose, also the ones referred to the best multilayer perceptron (MLP) model are included and its structure is showed in Figure 7.

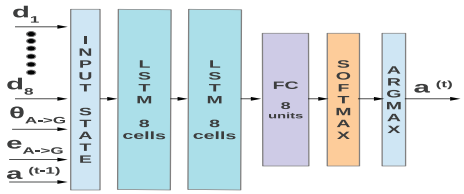

Fig. 6: Final structure of the LSTM neural network

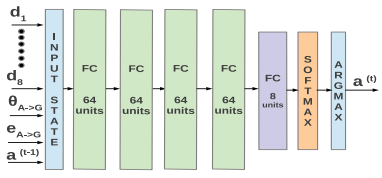

Fig. 7: Final structure of the MLP neural network

TABLE III: Training process results

Configuration	Train loss	Train accuracy	Validation accuracy
proposed LSTM	0.3208	89.97 %	88.60 %
final MLP	0.2276	92.08 %	88.43 %

Here the conclusion is that the best multilayer perceptron relies on a heavier structure in terms of number of learnable parameters:

$$\frac{\text{weights final MLP model}}{\text{weights proposed LSTM model}} = \frac{13768}{1256} = 10.96 \quad (6)$$

IV. Simulation in ASCII maps

In order to better analyze the comparison between the proposed LSTM model and the A* algorithm, it is useful to recall the differences between each other. The former is an online search agent, it can explore only nodes next to the current position and so it is more suitable for real world path planning since distance travelled is equal to the one explored. It aims to solve a path by storing just the current state in its memory, which is given as input to the neural network that calculates a direction immediately executed with a movement by the agent.

Meanwhile the latter is an offline search agent, it requires an a priori map stored in its memory, containing, at least, the supposed path from current to goal position. It is designed to compute the whole path before effectively starting to move. As a consequence, during the searching stage, it is allowed to expand nodes being more than one action far from each other.

Online testing is performed on maps related to videogames, random filled and mazes categories, the trained layers of the configuration represented in Figure 6 are evaluated in these tests, for each one the LSTM online agent is evaluated on 50 paths selected equally from 10 different maps. Results in videogames maps (Dragon Age folder), mazes having corridor 8 steps long and random filled maps for 25 % of the space are showed in Table IV, V and Table VI respectively, where the maximum path length limit is the same used for the training process.

For the sake of precision, here it is described what it is meant for "expanding a node" with respect to the two agents under comparison The A* algorithm expands a node every time the following function is calculated for the adjacent nodes, in ASCII maps environment a total of eight adjacent nodes are taken into account:

$$f(n) = g(n) + h(n) \quad (7)$$

where $g(n)$ is the backward cost from the starting position to node n and $h(n)$ is the heuristic forward cost which is the heuristic distance from node n to the goal node, meanwhile the neural network expands a node one step at a time when a new state is built to be the the new input.

TABLE IV: Online testing in Dragon Age maps

Agent	Success Rate	Average time (s)	Average length	Average nodes expanded
A*	100 %	$3.37 \cdot 10^{-4}$	45.82	47.029
LSTM	68 %	$1.69 \cdot 10^{-3}$	53.39	44.41

TABLE V: Online testing in mazes with corridor 8 steps long

Agent	Success Rate	Average time (s)	Average length	Average nodes expanded
A*	100 %	$2.96 \cdot 10^{-4}$	50.90	53.69
LSTM	26 %	$1.98 \cdot 10^{-3}$	61.71	52.92

TABLE VI: Online testing in random filled maps for 25 % of the space

Agent	Success Rate	Average time (s)	Average length	Average nodes expanded
A*	100 %	$2.84 \cdot 10^{-4}$	61.60	49.85
LSTM	82 %	$2.04 \cdot 10^{-3}$	71.57	56.02

The success rate represents a weakness for the LSTM, while the performance regarding the other three parameters can be considered close to the A* algorithm, this means that the proposed agent has been successfully trained to replicate the supervisor behaviour when the goal is reached. In particular, the average computation time calculation is in favor of the A* because in a real robot it would have to include the time spent to build the map from sensor readings, on the contrary, the LSTM agent uses the proximity ranges which directly generated by a LRF sensor.

A focus on the success rate is necessary, it was found, easily expected, that it increases as the maximum path length limit reduces, the trend is showed in Figure 8 and 9 referring, respectively, to mazes and random filled maps, while in Dragon Age maps it is almost equal to the one related to random filled maps. Furthermore, this analysis includes a comparison with the online agent using the MLP model of Figure 7. This last comparison states that the LSTM sets an absolutely higher success rate with respect to the MLP.

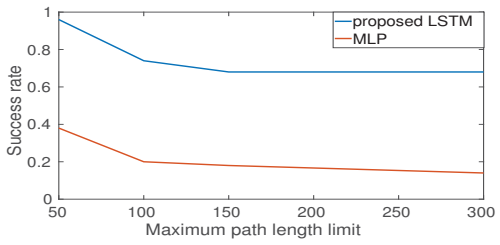

Fig. 8: Success rate trend in mazes maps

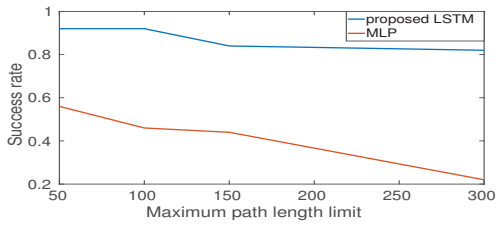

Fig. 9: Success rate trend in random filled maps

V. Experiment on a real robot

This section deals about the evaluation of the proposed method on a real mobile robot platform which is the iRobot Create 2. The proximity ranges are produced with a Hokuyo UTM-30-LX-EW, the ROS framework is utilized to develop the proposed method. Also the real environment is considered static.

Regarding the input state, the differences with respect to the definition in Section III is that the ranges considered are comprised in $[-90°, +90°]$ and the absolute orientation is added. A total amount of 720 proximity ranges are generated by the LRF sensor and divided into eight sections (Figure 10), then the average value of each section is taken as an input of the network. The output directions allowed are delimited in the same angle span related to the proximity ranges, a single direction is $45°$ distant to its adjacent, an example of a sequence of actions is showed in Figure 11.

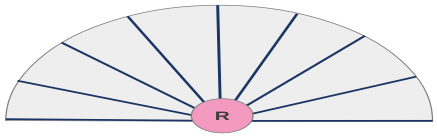

Fig. 10: Angular sections

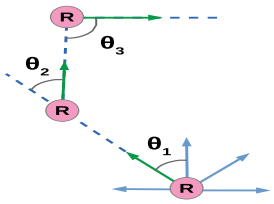

Fig. 11: Example of sequence of actions

Our laboratory is used for the tests as a good example of indoor map. Referring to its representation in Figure 12 where lighter gray means free space, a single path included in a dataset is defined by two red marks randomly selected and representing the start and goal point.

TABLE VII: Training set and validation set specifics

Specific	TRAINING SET	VALIDATION SET
Size (steps)	3000	1000
Time required	$\approx$ 1.5 h	$\approx$ 40 min

The robot travelled following the traces represented at a constant translational speed equal to 0.5 m/s and a rotational speed ranging from 1 rad/s to 1.5 rad/s. Hence, every 0.5 s data are collected to build a sample composed by the input state and the corresponding label represented with θ_i in Figure 11.

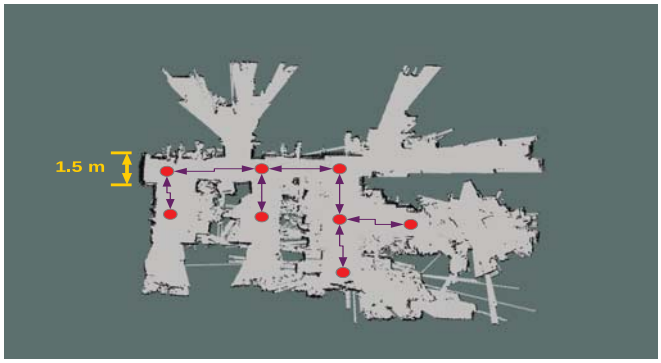

Fig. 12: Traces considered for the datasets

In Table VII, the few important details about the datasets collected for the training process are showed, hyperparameters configuration is almost equal to Table II, here the time sequence length is reduced to 30 time steps. The training process produced a validation accuracy equal to $\approx 94\%$ using the same LSTM network structure of Figure 6.
However, online testing gave acceptable results just for simple targets, meaning goal points no farther than 3 m from the starting position and including one turn at most. Figures 13 and 14 represent good examples of solved cases, the green arrow is the goal point while the red ones stand for the trace marked by the robot and the purple points denote the averaged ranges of each sections in Figure 10 related to the robot position showed.

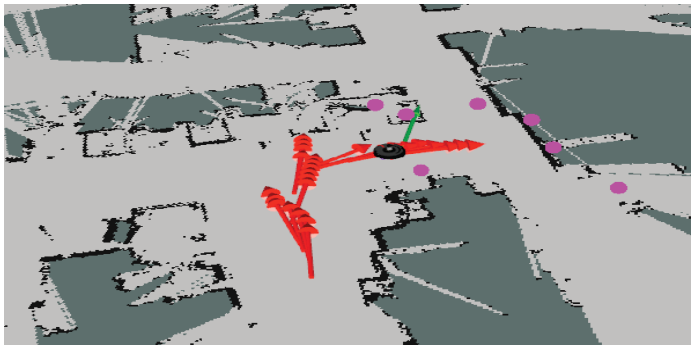

Fig. 13: Example 1 of solved case

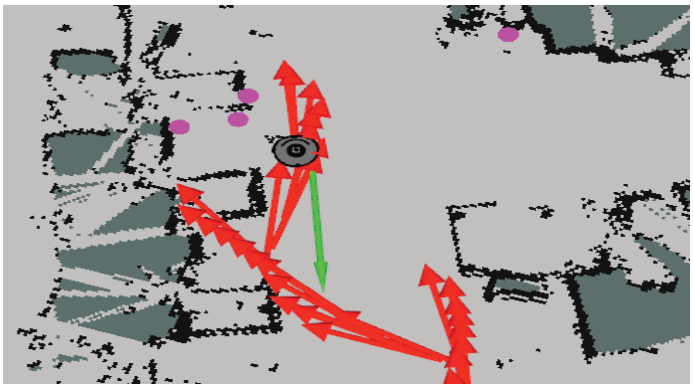

Fig. 14: Example 2 of solved case

VI. Conclusion and Future Works

In the simulation environment used, the proposed LSTM neural network model has been successfully trained to solve online path planning problems, but with limited distance from start to goal point position, otherwise a low success rate is attained. In particular, the failure attempts consisted on inability of getting away from blind spots, this could be due to the fact that these situations were not experienced during the training process. A reinforcement learning (RL) stage could be the best solution. We would think about an initial stage of supervised training then a second stage based on RL online training in order to have the chance to adjust the learnable parameters every time the agent gets in ambiguous positions like blind spots.

On the real robot, the method solved just very simple tasks and so the implementation has to be enhanced: datasets need to be enlarged as much as possible, at least including hundreds of thousands samples like in the simulation environment, in this way it may be also possible to achieve acceptable learning performance using more angular sections of proximity ranges and a more resolute output direction set in order to be able to generate a smoother path. However, using the same strategy of building a dataset here presented could easily lead to a required time too high and maybe not affordable.

References

[1] W. Khaksar et al. “A review on mobile robots motion path planning in unknown environments”. In: *2015 IEEE International Symposium on Robotics and Intelligent Sensors (IRIS)*. Oct. 2015, pp. 295–300.

[2] Bram Bakker. “Reinforcement Learning with Long Short-Term Memory”. In: (2002). Ed. by T. G. Dietterich, S. Becker, and Z. Ghahramani, pp. 1475–1482. URL: http://papers.nips.cc/paper/1953-reinforcement-learning-with-long-short-term-memory.pdf.

[3] Aviv Tamar et al. “Value Iteration Networks”. In: *Advances in Neural Information Processing Systems 29*. Ed. by D. D. Lee et al. Curran Associates, Inc., 2016, pp. 2154–2162. URL: http://papers.nips.cc/paper/6046-value-iteration-networks.pdf.

[4] H. Xiao, Li Liao, and F. Zhou. “Mobile Robot Path Planning Based on Q-ANN”. In: *2007 IEEE International Conference on Automation and Logistics*. Aug. 2007, pp. 2650–2654.

[5] Mihai Duguleana and Gheorghe Mogan. “Neural networks based reinforcement learning for mobile robots obstacle avoidance”. In: *Expert Systems with Applications* 62 (2016), pp. 104–115.

[6] Alex Graves and Navdeep Jaitly. “Towards End-To-End Speech Recognition with Recurrent Neural Networks”. In: *Proceedings of the 31st International Conference on Machine Learning*. Ed. by Eric P. Xing and Tony Jebara. Vol. 32. Proceedings of Machine Learning Research 2. Bejing, China: PMLR, 22–24 Jun 2014, pp. 1764–1772. URL: http://proceedings.mlr.press/v32/graves14.html.

[7] H Sak et al. “Sequence discriminative distributed training of long short-term memory recurrent neural networks”. In: *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*. Jan. 2014, pp. 1209–1213.

[8] J. Donahue et al. “Long-Term Recurrent Convolutional Networks for Visual Recognition and Description”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.4 (Apr. 2017), pp. 677–691.

[9] P. E. Hart, N. J. Nilsson, and B. Raphael. “A Formal Basis for the Heuristic Determination of Minimum Cost Paths”. In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (July 1968), pp. 100–107.

[10] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. http://aima.cs.berkeley.edu. Prentice Hall, 2010. Chap. Beyond Classical Search, pp. 429–436.

[11] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-Term Memory”. In: *Neural Computations* 9.8 (Nov. 1997), pp. 1735–1780.

[12] Wojciech Zaremba and Ilya Sutskever. “Learning to Execute”. In: *CoRR* abs/1410.4615 (2014).

[13] Alex Graves. *Supervised Sequence Labelling with Recurrent Neural Networks*. Vol. 385. Jan. 2012, pp. 31–35.

[14] Razvan Pascanu et al. “How to Construct Deep Recurrent Neural Networks”. In: *International Conference on Learning Representations 2014 (Conference Track)*. Apr. 2014. URL: http://arxiv.org/abs/1312.6026.

[15] X. Li and X. Wu. “Constructing long short-term memory based deep recurrent neural networks for large vocabulary speech recognition”. In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Apr. 2015, pp. 4520–4524.

[16] N. Sturtevant. “Benchmarks for Grid-Based Pathfinding”. In: *Transactions on Computational Intelligence and AI in Games* 4.2 (2012), pp. 144–148. URL: http://web.cs.du.edu/~sturtevant/papers/benchmarks.pdf.