Chapter 6

Path Planning in Rough Terrain Using Neural Network Memory

Nancy Arana-Daniel, Roberto Valencia-Murillo, Alma Y. Alanís, Carlos Villaseñor and Carlos López-Franco

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/intechopen.71486

Abstract

Learning navigation policies in an unstructured terrain is a complex task. The Learning to Search (LEARCH) algorithm constructs cost functions that map environmental features to a certain cost for traversing a patch of terrain. These features are abstractions of the environment, in which trees, vegetation, slopes, water and rocks can be found, and the traversal costs are scalar values that represent the difficulty for a robot to cross given the patches of terrain. However, LEARCH tends to forget knowledge after new policies are learned. The study demonstrates that reinforcement learning and long-short-term memory (LSTM) neural networks can be used to provide a memory for LEARCH. Further, they allow the navigation agent to recognize hidden states of the state space it navigates. This new approach allows the knowledge learned in the previous training to be used to navigate new environments and, also, for retraining. Herein, navigation episodes are designed to confirm the memory, learning policy and hidden-state recognition capabilities, acquired by the navigation agent through the use of LSTM.

Keywords: robot navigation, learning to search, reinforcement learning, LSTM unstructured terrain, rough terrain, cost function

1. Introduction

Autonomous robot navigation in unstructured terrain allows a robot to move through an environment for which the selection of traversable terrain is not a deterministic decision [1]. A mobile robot must make decisions of when to traverse patches of terrain that could be dangerous or that might consume too many resources.

Several approaches have been developed in order to solve this problem; some of them are focused on classifying traversable terrain [2, 3], and others in coupling the perceptual and

planning systems of the robot using functions that map features of the environment to scalar values that represent the traversability of the terrain [1, 4, 5]. These works try to resolve the task of autonomous robot navigation in unstructured terrain; however, these approaches do not address the issue as an integrated system.

In most cases, a human expert provides information for cost map constructions heuristically. In other cases, this information is then used to construct a cost function that automatically maps features to costs; however, note that this cost function is established heuristically. The problem with this methodology is that the traversability of a given feature for a robot is difficult to quantify. In contrast, humans can determine traversal trajectories relatively easily.

An alternative to establishing cost functions is to use an algorithm that automatically constructs and tunes a cost function. Learning to Search (LEARCH) [1] is an algorithm that uses learning from demonstration in order to construct a cost function. In this approach, a human expert exhibits a desirable behavior (a sample path) over certain terrain; then, LEARCH adjusts a cost function in order to match the behavior exhibited by the expert.

LEARCH has the advantage that the cost function to be constructed can be chosen from among linear functions, parametric functions, neural networks and decision trees, to name a few [1]. In particular, neural networks and other learning machines such as support vector machines (SVM) have exhibited considerable generalization capability in different scenarios [6].

However, as will be demonstrated in this paper, the LEARCH generalization capability decreases as the number of sample paths increases; this is because the error decays over time during training.

In this study, this problem with the LEARCH algorithm is addressed using a long-short-term memory (LSTM) neural network and reinforcement learning (RL). Recurrent neural networks are capable of finding hidden states, as shown in [7]. Therefore, we propose a complex learning system that allows a navigation agent to learn navigation policies and determine complex traversability cost functions. Furthermore, this system can retain the knowledge learned in past navigation episodes in memory and generalize this knowledge for use in new episodes.

2. Learning to Search

This section presents an overview of the LEARCH algorithm, along with the results of generalization tests conducted using LEARCH. The objective is to explain the need for memory for this algorithm in order to improve its performance.

The LEARCH algorithm is based on the concept of inverse optimal control [8], which addresses the problem of finding a cost map such that a known trajectory through an environment is optimally navigated using this map. In addition, non-linear maximum margin planning [1] with the support vector regression machine [9] is used to learn behavior from an expert, that is, human expert.

Let S be a state space operated by a path planner. F is a feature space defined over S. Then, for every $x \in S$ a corresponding feature vector $F_x \in S$ exists. The F_x vectors are inputs for the cost

function C, which maps F to scalar values. C is defined as the weighted sums of functions $R_i \in \mathcal{R}$, where $\mathcal{R}$ is a space of limited complexity that maps from the feature space to a scalar [1].

We define a path P as a sequence of states $x \in S$ that lead from the start s point to the goal g. The cost of each state is $C(F_x)$; thus, the cost of the entire path is defined as

$$C(P) = \sum_{x \in P} C(F_x) \tag{1}$$

Consider a path provided by an expert, i.e. sample path P_e, which runs from a start state s_e to a goal state g_e. In order to learn from the expert demonstration, a cost function such that P_e is the optimal path from s_e to g_e is required. This task can be expressed as the following optimization problem [1]:

$$\text{Min}O[C] = \lambda \text{REG}(C) + \sum_{x \in P} C(F_x) - \min \widehat{P} \left[\sum_{x \in \widehat{P}} (C(F_x) - L_e(x)) \right] \tag{2}$$

where λ is a term that scales the regularization term REG(C). $\widehat{P}$ is a path computed by a planner over the cost space, and L_e is a loss function that encodes the similarity between paths. The latter is defined as

$$L_e = \begin{cases} 1 \text{ if } x \in P_e, \\ 0 \text{ otherwise} \end{cases} \tag{3}$$

The sub-gradient is used to minimize $O[C]$. In the cost function space, the sub-gradient is

$$\nabla O_F[C] = \lambda \nabla REG_F[C] + \sum_{x \in P_e} \delta_F(F_x) - \sum_{x \in P_*} \delta_F(F_x), \tag{4}$$

where δ is the Dirac delta and P_* is the optimal path for the actual cost map.

In order to avoid overfitting, a cost function space is considered. C is now defined as the space of weighted sums of functions $R_i \in \mathcal{R}$, where $\mathcal{R}$ is a space of functions of limited complexity, which maps from the feature space to a scalar. The possible choices for $\mathcal{R}$ include linear functions, parametric functions, neural networks and decision trees. Thus,

$$\begin{gathered} C = \left\{ C | C = \sum_i \eta_i R_i(F), R_i \in \mathcal{R}, \eta_i \in \mathbb{R} \right\} \\ \mathcal{R} = \{ R | R : \mathcal{F} \to \mathbb{R} \wedge REG(R) < v \} \end{gathered} \tag{5}$$

The functional gradient is projected onto the direction set by finding the element $R_i \in \mathcal{R}$ that maximizes the inner product $\langle -\nabla O_F[C], R_* \rangle$. This maximization can be regarded as a learning problem. Here,

$$\begin{aligned} R_* &= \arg\max_R \langle -\nabla O_F[C], R_* \rangle \\ &= \arg\max_R \sum_{x \in P_e \cap P_*} \alpha_x y_x R(F_x) \end{aligned} \tag{6}$$

where

$$\alpha_x = |\nabla O_{F_x}[C]| y_x = -\operatorname{sgn}(\nabla O_{F_x}[C])$$

As in [1] the projection of the functional gradient can be regarded as a weighted classification problem. It can be seen that the regression targets y_x are positive in regions of the feature space for which the planned path visits more than the sample path and negative in the opposite case. Here, this approach is viewed as minimizing the error induced by visiting states that are not in the sample path. Then, the visitation count U is the cumulative count of the number of states $x \in P$ such that $F_x = F$. The visitation counts can be split into positive and negative components, depending on whether they correspond to the current planned path or the sample path:

$$\begin{aligned} U_+(F) &= \sum_{x \in P_*} \delta_F(F_x) \\ U_-(F) &= \sum_{x \in P_e} \delta_F(F_x) \end{aligned} \tag{7}$$

$$U(F) = U_+(F) - U_-(F) = \sum_{x \in P_*} \delta_F(F_x) - \sum_{x \in P_e} \delta_F(F_x) \tag{8}$$

Ignoring the regularization term of Eq. (4), the regression targets and weights can be computed as functions of the visitation counts. Then, the regressor targets can be obtained using these visitation counts. Further, with this regressor, the cost function can be expressed as

$$C_j = C_{j-1} * e^{n_j R_j} \tag{9}$$

where $j = \{1, 2, 3, \ldots, n\}$, with n being the number of iterations; R is the regressor; and η is the learning rate.

2.1. Learning to Search experiments

This section describes the experiment conducted to test the LEARCH generalization capabilities. Satellite-like images were selected for feature extraction. Further, patches of terrain were divided into grid cells, with a vector being created for each cell. These vectors represented a value for each of the following features of the environment in each dimension: the vegetation density, slope, the presence of gravel or rocks and the presence of water; each scalar value represents an abstraction of the feature; for example, the scalar value for vegetation represents its density, a patch of terrain with grass would be represented with a low value, and a patch of terrain with a tree would be represented with a high value of vegetation. In these experiments vectors of dimension 4 were used, that is, for the patch of terrain with grass, the vector [0, 2, 0, 0] would be its representation. A human expert-traced sample paths over the terrain, as shown in **Figure 1**.

http://dx.doi.org/10.5772/intechopen.71486

Figure 1. Left: Satellite like image. Right: Grid cells and lines with different colours representing sample paths.

With this information a support vector regressor (SVR) was used to learn the cost function R_i of Eq. (5). Note that, after LEARCH is executed, the trained SVR can map the features of the terrain directly into traversal costs. **Figure 2** shows a diagram of the algorithm used for training.

The procedure is explained as follows:

- A cost map M is constructed using a feature map and a cost function ($C(F)$).
- The path planner D^* computes an optimal path P^* from start point s_e to the goal point g_e over M.
- Using the sample path from the expert path P_e and P^*, the vector U indicating the visitation counts is constructed as shown in Eq. (8).
- Using U, the regressor targets are computed, and this regressor is trained.
- The cost function is updated using Eq. (9).
- This process is repeated until P_e and P^* are equal.

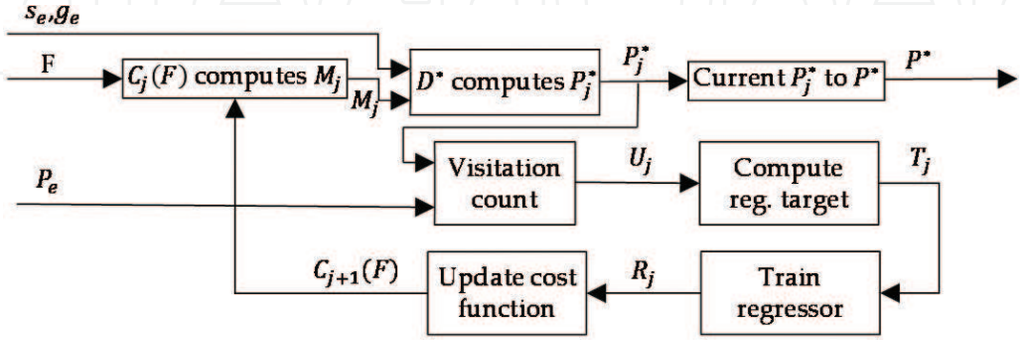

Figure 2. LEARCH diagram. F is the feature map, M is the cost map, s_e and g_e represent the start and the goal points, respectively, P^* is the optimal path, P_e is the sample path, T is the vector of regressor targets values for training a regressor R, C (F is the cost function and $j = \{1,2,3,\ldots, n\}$, where n is the number of iterations needed to train the cost function.

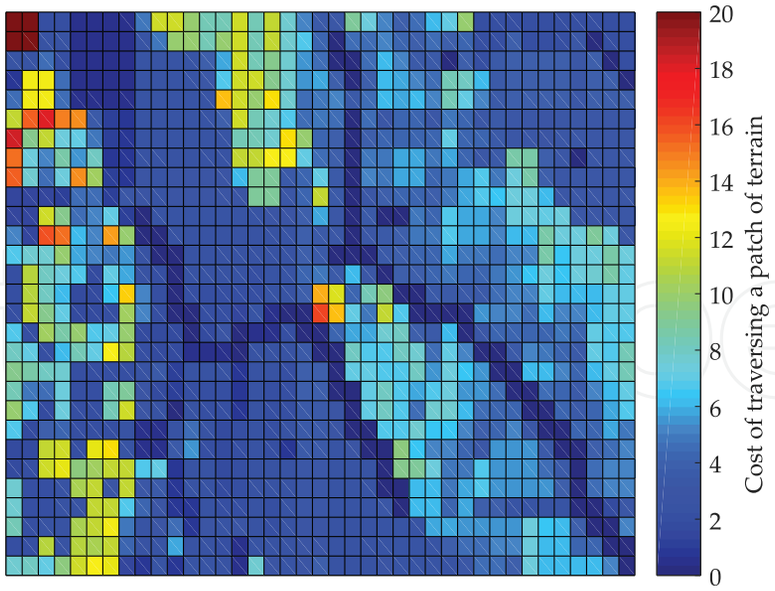

Figure 3. Example of a cost map which belongs to the real map shown in **Figure 1**.

The traversal costs can be color coded for demonstration purposes, as shown in **Figure 3**, where values near 20 represent the patches with the highest crossing difficulty and those near zero represent terrain that is easy to traverse.

In this experiment, in order to prove the generalization capabilities of LEARCH (i.e. its ability to use policies learned in past navigation episodes during new episodes), we employed the following methodology. First, we trained the LEARCH system described in **Figure 1** using an initial map and one sample path. Then, we incrementally added to this learned knowledge (using the same initial map) by incorporating more paths to be learned by LEARCH (one by one). The results of these experiments are shown in **Figure 4**, where the image (a) of the figure shows the cost map obtained with a cost function trained using one sample path and the image (b) shows the results obtained by adding a path to the training, and so on until five sample paths are used. The image (f) of **Figure 4** shows the cost map obtained with a cost function trained using eight sample paths.

From the cost map (e) of **Figure 4**, in comparison with the cost map (f), it is apparent that information from the environment is missing after the cost function is trained with more sample paths. That is, some states are no longer recognized as states with a high traversal cost. It is important to note that the costs that are most affected are those furthest from the sample paths, in comparison with the costs of the corresponding states on the original path; therefore, the generalization capability of the LEARCH system is very poor. This problem renders the task of finding the optimal path difficult. In addition, the path planner could compute a path that traverses dangerous terrain. Further, note that, the use of only a few sample paths is not a solution to the problem of obtaining a system with knowledge of a greater number of area costs than those attached to the sample paths. This is because such sample paths cannot contain all the information necessary for a good and complete representation of the environment.

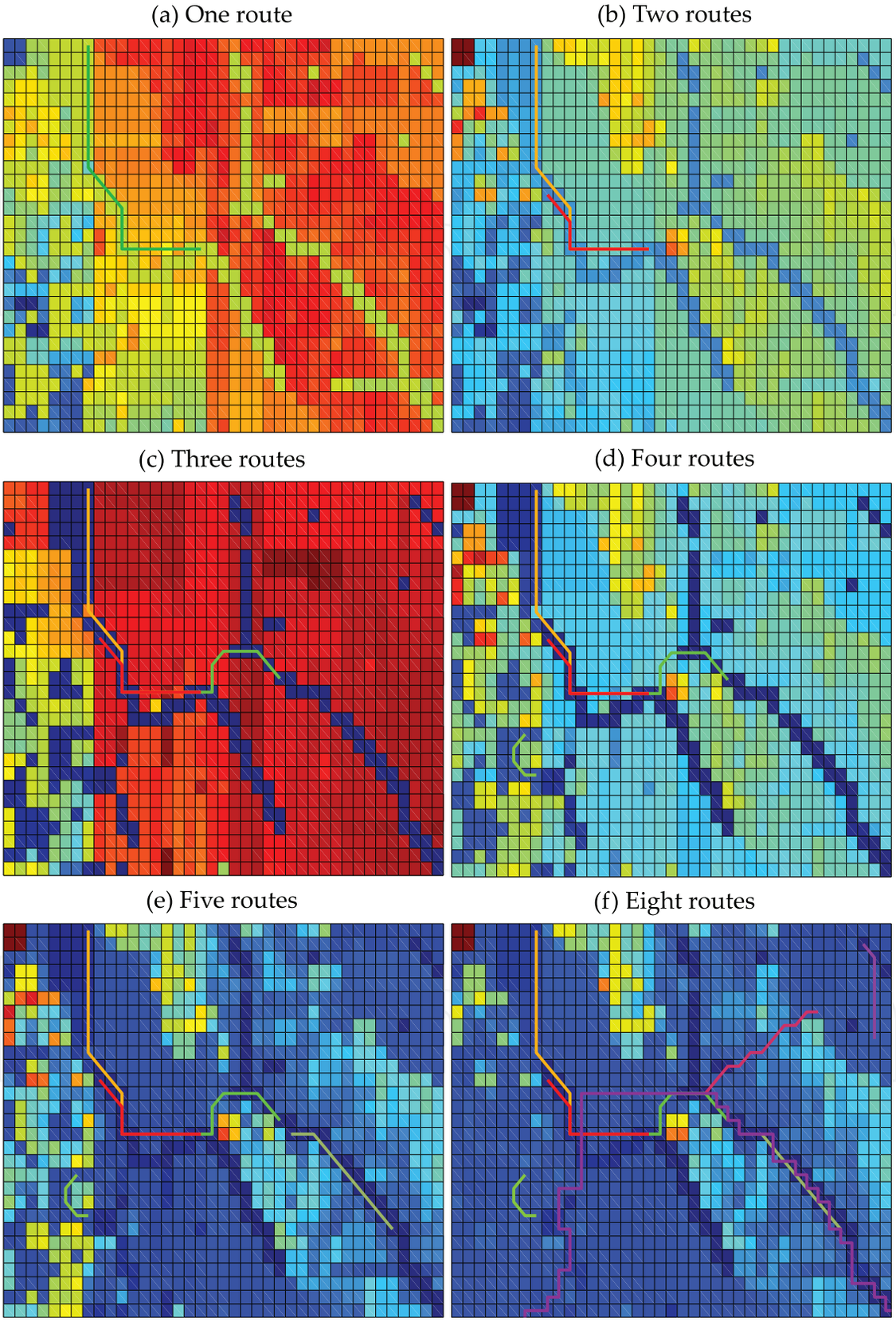

Figure 4. Costs maps obtained using different numbers of paths of terrain.

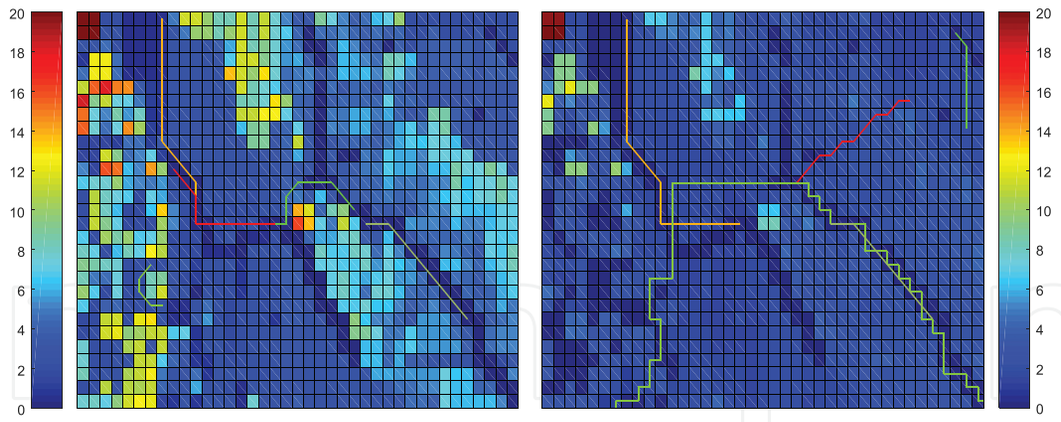

Figure 5. Left: cost map computed with five representative paths. Right: cost map computed with five non representative paths.

In this study, other experiments to prove the limitations of LEARCH were performed, in which we trained the system using nonrepresentative environment paths. That is, the paths taught by the expert traversed many cells of the environment that did not contain sufficient representative features of the environment or cells that did not have significant differences in cost. **Figure 5** shows examples of these paths, which allowed the LEARCH system to acquire nonrepresentative knowledge that was then generalized over the cost map. The cost map at the left of **Figure 5** is less generalized compared with the more descriptive costs shown on the map at the right of **Figure 5**.

Therefore, in order to address the problems with the LEARCH system, we propose the use of an LSTM as part of the system. Inclusion of an LSTM allows the navigation agent to learn navigation policies and complex traversability cost functions and, furthermore, to retain memory of the knowledge learned in the past navigation episodes for reuse during new episodes. The latter capability allows expensive retraining to be avoided when the navigation environment is similar to those already explored by the agent and allows hidden states of the extremely large state space represented by a nonstructured or rough terrain to be recognized. We present the LSTM in the next section.

3. Long-short-term memory neural network

LSTM is a recurrent neural network architecture originally designed for supervised time-series learning. It addresses the problem that errors propagated back in time tend to vanish in multilayer neural networks (MLPs). Enforcing a constant error flow in constant error carousels (CEC) is a solution for vanishing errors [7].

These CECs are processing units having linear activation functions that do not decay over time. CECs can become filled with useless information if access to them is not regulated; therefore, specialized multiplicative units called input gates regulate access to the CECs. Further, their access to activation of other network units is regulated by multiplicative units called output gates. In addition, forget gates are added to CECs in order to reset information

http://dx.doi.org/10.5772/intechopen.71486

that is no longer useful. A combination of a CEC and its input, output and forget gates is called a memory cell (**Figure 6**).

The activation updates at each time step t in this type of neural network are computed as follows. For the hidden unit activation y^h, the output unit activation y^k, the input gate activation y^{in}, the output gate activation y^{out} and the forget gate activation y^{φ}, we have

$$y^i(t) = f_i\left(\sum_m w_{im} y^m(t-1)\right) \tag{10}$$

where w_{im} is the weight of the connection from unit m to unit i. For the activation function f_i, the standard logistic sigmoid function for all units is chosen, except for output units, for which it is the identity function [7]. The CEC activation, also known as memory cell state, is calculated using

$$s_{c_j^v}(t) = y^{\varphi_j}(t) s_{c_j^v}(t-1) + y^{in_j}(t) g\left(\sum_m w_{c_j^v m} y^m(t-1)\right) \tag{11}$$

where g is a logistic sigmoid function scaled to the $[-2, 2]$ range and $s_{c_j^v}(0)$. Finally, the activation update for the memory cell output is calculated from

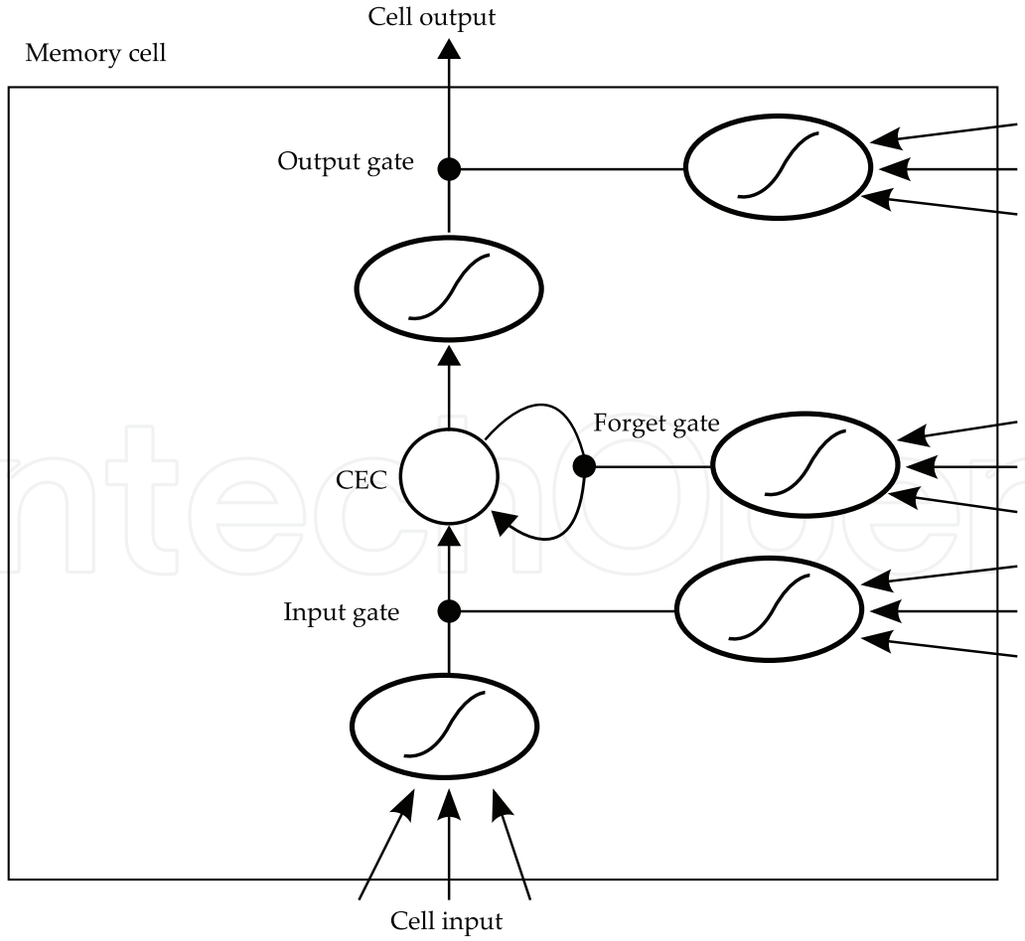

Figure 6. Graphic representation of a memory cell.

$$y^{c_j^v}(t) = y^{out}(t)h\left(s_{c_j^v}(t)\right) \tag{12}$$

The learning process implemented for LSTM in this paper is a variation of real-time recurrent learning (RTRL), as described in Ref. [7] which is a variation of [9]. In this variant, when the error arrives at a cell, it stops propagation further back in time. However, the error is used to update incoming weights when it leaves the memory cell through the input gate.

4. Reinforcement learning y long-short-term memory neural network

In order to teach the LSTM to navigate an unstructured terrain, RL was implemented as described in Ref. [7]. In this approach, an LSTM approximates the value function V of the RL algorithm, which teaches a robotic agent how to navigate a T-shaped maze environment.

This problem is a partially observable Markov decision process, in which the agent is unaware of the full state of the environment and must infer this information using current observations. In this study, these observations are the same feature vectors of the environment that were used for previous LEARCH experiments, and these vectors are the input for the LSTM.

The LSTM outputs represent the advantage values $A(s, a)$ of each action, where a is the action taken in state s. They are used to compute the value of the state $V(s) = \max_a A(s, a)$, which represents the action with the higher advantage value.

To perform weight updates, truncated backpropagation through time was implemented with RL. A function approximator's prediction error at time step t, $E^{TD}(t)$, is computed using Eq. (13) and is propagated one step back in time through all the units of the network, except for the CECs, for which the error is backpropagated for an indefinite amount of time [7]. Thus,

$$E^{TD}(t) = V\left(s(t) + \frac{r(t) + \gamma V(s(t+1)) - V(s(t))}{k} - A(s(t), a(t))\right) \tag{13}$$

where r is the immediate reward, γ is a discount factor in the $[0, 1]$ range and k scales the difference between the values of the optimal and suboptimal actions. It is worth mentioning that only the output associated with the executed action receives the error signal.

During the learning process, the agent can explore the environment using the state values; however, directed exploration (i.e. exploration for which a predictor is used to direct the exploration stage, so as to avoid clueless exploration of the entire state space) is important in order to learn complex terrain navigation. When an undirected exploration is conducted, RL tries every action in the same way over all states; however, in unstructured terrain, some states provide ambiguous information about the environment rendering it difficult for the agent to determine the state of the environment. Other states provide clear information; therefore, the agent must direct its exploration to discover the ambiguous states. In order to explore the environment, an MLP was implemented for directed exploration. This MLP input was the same as the LSTM, and the MLP objective was to predict the absolute value of the current temporal difference error, i.e. $E^{TD}(t)$. This aided prediction of which observations were

associated with a larger error. The desired MLP output was obtained using Eq. (14), and backpropagation was employed to train the MLP:

$$y_d^v(t) = \left|E^{TD}(t)\right| + \beta y^v(t+1) \tag{14}$$

The MLP output $y^v(t)$ is used as the temperature of the Boltzmann action selection rule, which has the form

$$\frac{e^{A(s,a)/y^v(t)}}{\sum_{b=1}^{n} e^{A(s,a)/y^v(t)}} \tag{15}$$

where n is the number of actions available to the agent.

The complete learning process and the manner in which the LEARCH and RL-LSTM systems are connected is shown in **Figure 7**. The entire process occurs offline. First, the LEARCH

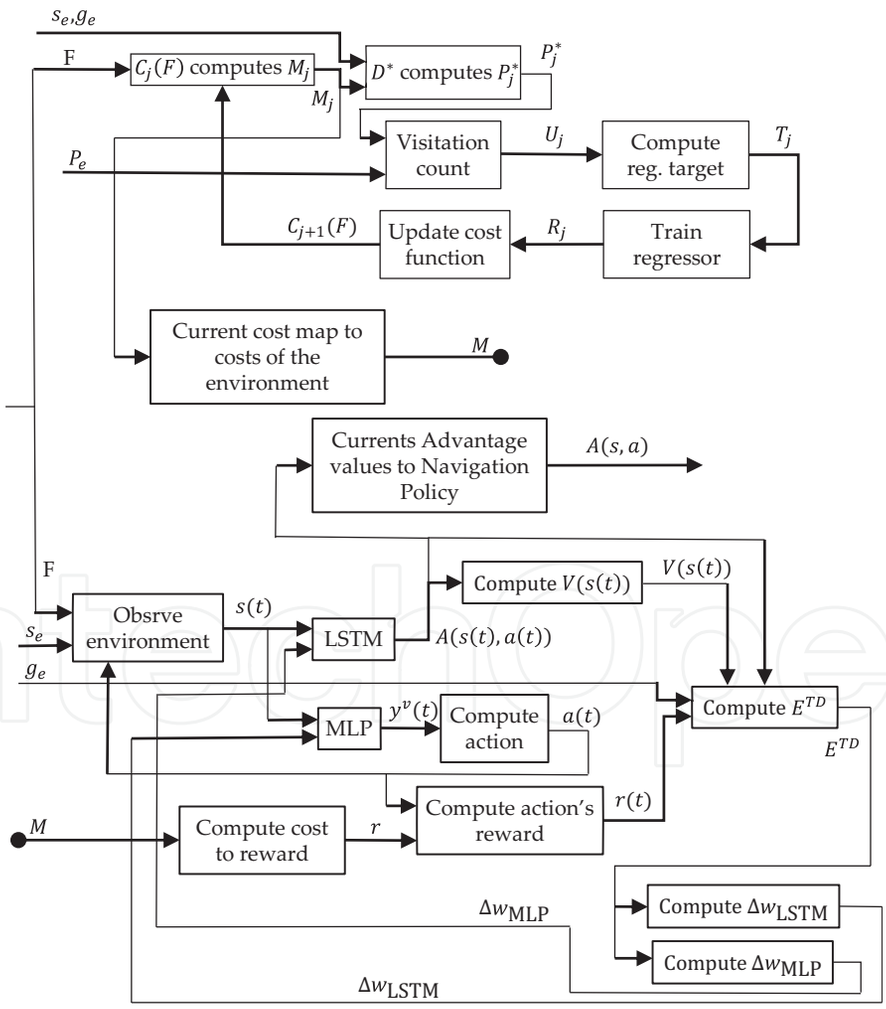

Figure 7. LEARCH-RL-LSTM system showing the manner in which the two systems are connected to train the LSTM. The entire process occurs offline. First, the LEARCH algorithm iterates until the required cost map M is obtained. Then, the RL-LSTM algorithm begins the process of training the LSTM using the costs converted into rewards r. The feature map F is obtained from the robotic agent and used by both systems.

Figure 8. (a) Example of a real environment modelled as a grid map. (b) Patches of terrain used for training are marked with an orange box.

algorithm iterates until the required cost map M is obtained. Then, the RL-LSTM algorithm begins the process of training the LSTM using the costs converted into rewards r. The feature map F is obtained from the robotic agent and used by both systems.

In order to prove the generalization and long-term memory capabilities of LSTM, training was performed using patches of terrain containing representative features of rough terrain. That is, an entire map is not used to train the LSTM (**Figure 8**). In this way, an efficient training phase is achieved by taking advantage of the above-mentioned capabilities. In the next section, we show the results of the experiments conducted to confirm these capabilities. In addition, we prove the efficacy of the LSTM for mapping tasks that require inference of hidden states, i.e. smoothing or noise recognition.

The LEARCH algorithm builds a cost function; however, as noted in Section 2, the cost function capability for generalization is limited and decays as the number of training paths grows. As the motivation for employing a cost function is to obtain the cost of traversing a patch of terrain so that the path planning system can compute the optimal path with the minimal traversal cost, we propose the extraction of terrain patches having descriptive characteristics of rough terrain for navigation. Hence, the traversal costs for these environment features can be determined using LEARCH, and the costs can be transformed to rewards for a RL algorithm [10].

5. Results

In this section, the results of the experimental tests are presented. Five environments were designed for navigation policy learning using the LEARCH-RL-LSTM system shown in **Figure 7**. Here, each environment was modeled as a grid, and each model was referred to as a map. Each map was a grid having dimensions of 20×20 cells. Further, each cell represented a patch of terrain, and this patch was represented by a vector of dimension 4, where each dimension was a scalar value representing the vegetation density, terrain slope, rock size or the presence of water.

http://dx.doi.org/10.5772/intechopen.71486

Figure 9 shows in the lower right corner the color code used to illustrate the manner in which this environment was designed. Each environment differed by 5% from the previous one, i.e. 20% of the states in map 5 differed from those of map 1. These maps are shown in **Figure 9**.

Table 1 lists the results of experiments conducted using the LEARCH system alone to learn the navigation policies and cost functions of the five maps. In order to test the capability of LEARCH to reuse knowledge learned in previous navigation episodes, the following process was employed. Once LEARCH learned the navigation policies and cost function of map 1, this knowledge was used as initial knowledge to start navigation episodes involving the remaining maps. As is apparent from the first row of **Table 2**, it was not necessary to retrain the LEARCH row shows, and it was not necessary to retrain the LEARCH system to learn the demonstrated behavior and cost function of map 2. In other words, the LEARCH system could apply the knowledge learned from map 1 to map 2. However, this behavior did not occur for the other maps. For maps 3, 4 and 5, and when attempting to reuse the knowledge learned from map 1, it was necessary to retrain the LEARCH system. In these learning episodes, an increased number of iterations were necessary in order to acquire the new knowledge (as is apparent when **Table 1** is compared with row one on **Table 2**, it can be concluded that the previous knowledge learned using map 1 is even detrimental to the system performance when new

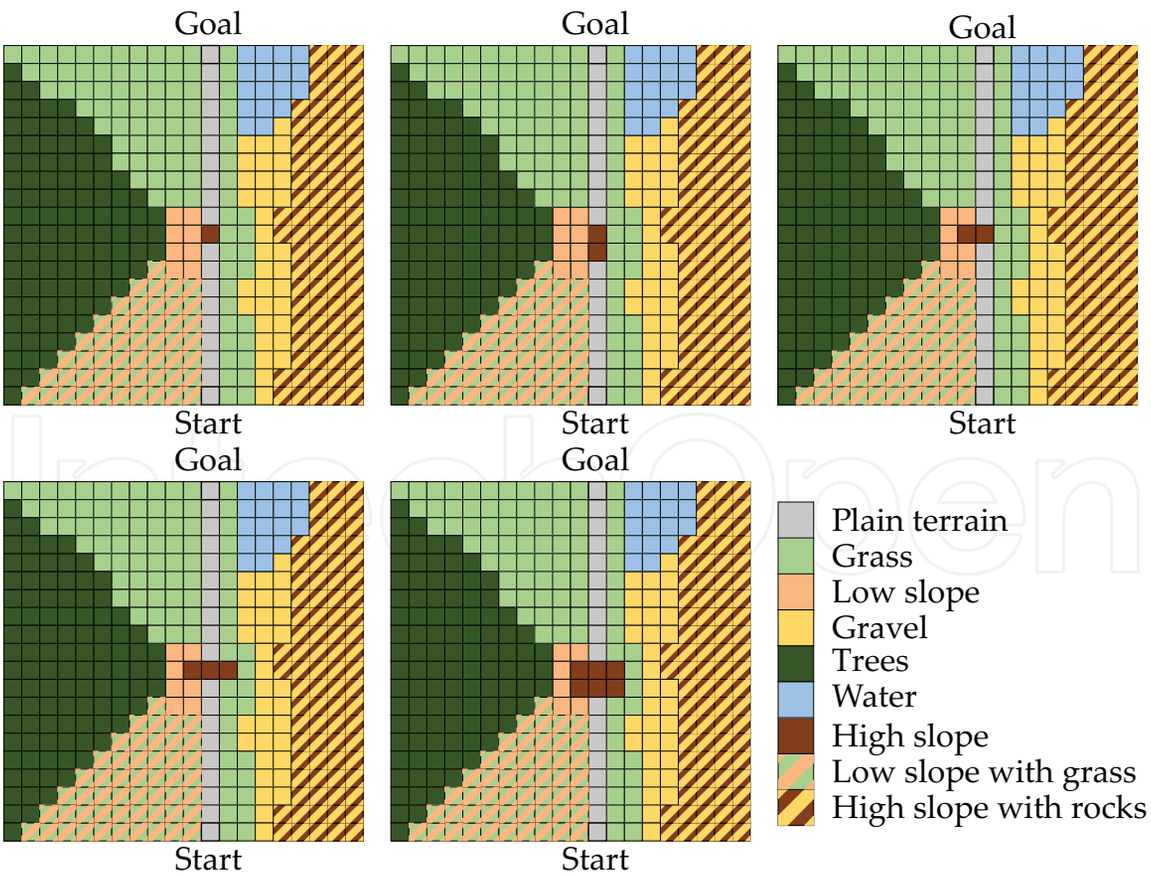

Figure 9. Maps used in experiments. Lower right corner of the second row: colour code used to represent environment features.

Environment	Map 1	Map 2	Map 3	Map 4	Map 5
Iterations	7	3	3	3	4

Table 1. Iterations needed to learn demonstrated behavior using LEARCH system.

Environment	Map 2	Map 3	Map 4	Map 5
Iterations LEARCH	0	5	4	7
Iterations LEARCH-RL-LSTM	0	0	0	0

Table 2. Iterations needed to learn demonstrated behavior using knowledge of map 1, for LEARCH and LEARCH-RL-LSTM system.

maps are processed, even if the new maps are very similar to map 1. Therefore, the LEARCH system was shown to have a very poor generalization capability.

When RL-LSTM was integrated with LEARCH to improve the capability for reusing knowledge learned from previous navigation episodes, there was no need to retrain the system. This is apparent from the second row of **Table 2**, where all the demonstrated behavior for maps 2 to 5 could be learned using the knowledge learned from map 1 only. It is important to note that, although the results were obtained from relatively small maps, it was necessary to retrain the cost function using LEARCH in each of these cases. Further, when LEARCH-RL-LSTM was employed, retraining was unnecessary when the patches of terrain were similar, because this system can generalize knowledge from previous navigation episodes. Note that, when the agent navigates in real time, even small retraining episodes are computationally expensive. Further, the agent is required to stop navigating until the retraining episode ends. However, for LEARCH-RL-LSTM, retraining is unnecessary when the environment is similar to those already known from previous navigation episodes.

Another set of environment maps was also used to test both algorithms. For these new environments, features that were not observed in previous scenarios were included. In this experiment, map 6 was the base of knowledge, and two new features were included in maps 7, 8 and 9. The states differed in the same way as in the previous experiment, with 5% of the states in each map being different from those of the previous maps. However, these differences included new features in order to simulate a dynamic environment, i.e. we simulate that the terrain of the map 6 suddenly changed when the agent navigates again on this map introducing new features on some cells of the grid of map 6. The maps used for this experiment are shown in **Figure 10**.

The LSTM used in these experiments was trained offline. During agent navigation, an efficient training episode was only executed if necessary, i.e. only if the action that LSTM learned to take is dangerous for the agent. These training episodes were efficient, because only a fraction of the environment was used (such as the patch of terrain shown in the lower right corner of **Figure 10** each time the robot encountered a new state or required navigation assistance.

http://dx.doi.org/10.5772/intechopen.71486

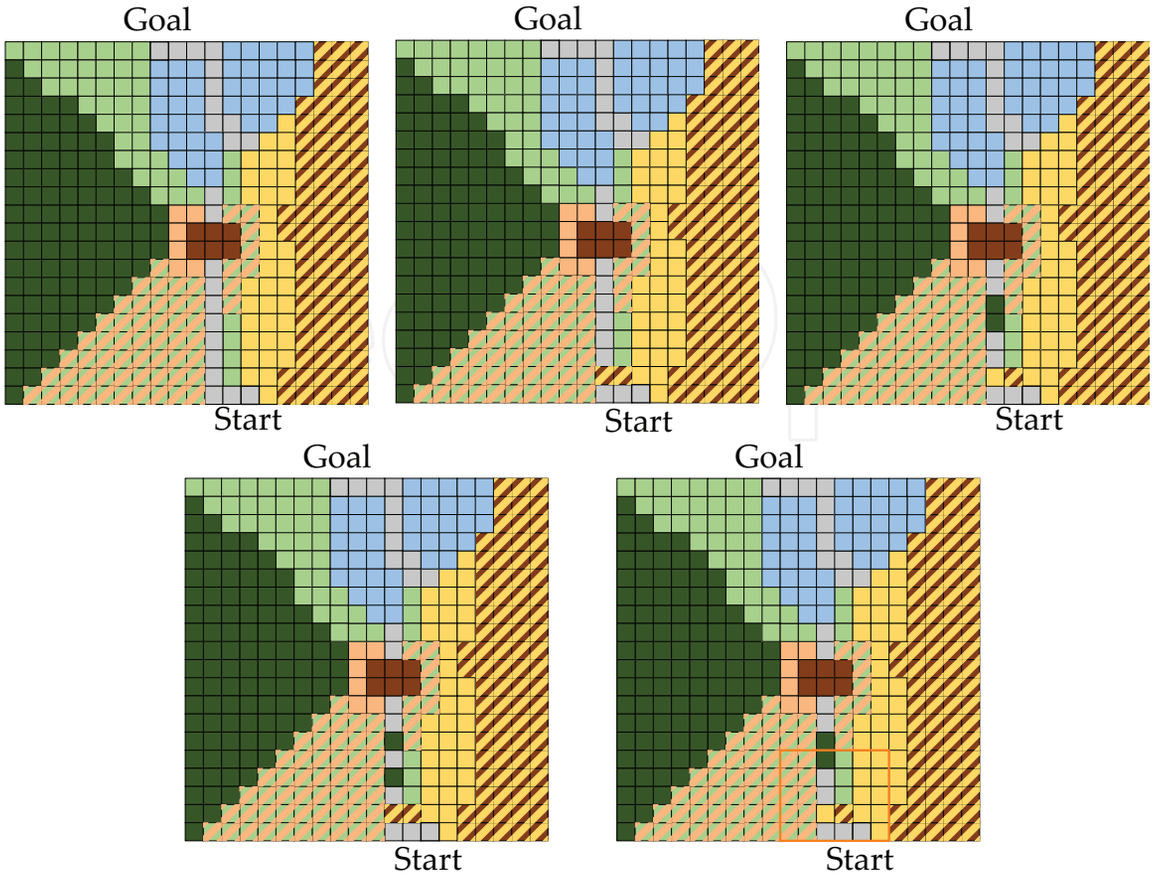

Figure 10. Maps used in second set of experiments to simulate dynamic environments. Lower right corner of the second row: sample of a map with the patch of terrain used for retraining marked by an orange box.

5.1. Noise tests

In the previous experiments, we assumed that the agent could infer the current state of the environment model based on the features observed by the agent. However, in a real scenario, the agent must infer the actual state via a perceptual system based on data obtained through noisy sensors such as cameras, a Global Positioning System (GPS) or LiDAR. In outdoor environments, two states (patches of terrain) can be very similar; however, the same action in these similar states could lead to different resultant actions. In case of noisy signals, one state could be interpreted as another similar state or, alternatively, as a new state that is not explicitly represented in the environment model, i.e. a hidden state.

To test these two systems in more realistic environment, a noise signal was induced to the inputs of both systems. A real uniform distribution bounded to a maximum of $[-1, 1]$ (20% of noise) was used. Then, several runs of each system were conducted with an initial limit of $[-0.1, 0.1]$ (2% of noise) and increments of $[-0.1, 0.1]$ in the noise signal, until the maximum limits where both systems failed to infer the real state for the agent were determined. **Tables 3** and **4** show the test results for both systems with noise; the noise range values are

Environment	Map 1	Map 2	Map 3	Map 4	Map 5
Noise-supported LEARCH	2%	2%	2%	2%	6%
Noise-supported LEARCH-RL-LSTM	10%	8%	10%	8%	12%

Table 3. Maximum noise supported by both systems in tests where the desired behavior could be reproduced with maps 1–5.

Environment	Map 6	Map 7	Map 8	Map 9
Noise-supported LEARCH	0%	0%	0%	0%
Noise-supported LEARCH-RL-LSTM	10%	8%	8%	8%

Table 4. Maximum noise supported by both systems in tests where the desired behavior could be reproduced with maps 6–9.

the maximum limits of the noise supported by the system using that map. Note that the results of the maps 6–9 yielded by the LEARCH system are omitted, because this system could not reproduce the desired behavior on these maps under the supplied noise levels.

6. Conclusion

LEARCH is an efficient method for learning a cost function that maps environment features to traversal costs and can then be used to navigate an unstructured terrain. However, as demonstrated by the experiments conducted in this work, this algorithm is incapable of reusing knowledge in an efficient manner. Indeed, zero knowledge is sometimes preferable to reusing previously learned knowledge.

We concluded that LEARCH cannot reuse knowledge because of a lack of memory; because of this lack of memory, the cost function cannot correlate knowledge learned in earlier training episodes with the new information provided by new environments; therefore, an LSTM was proposed. The LSTM can relate knowledge using memory cells, and this knowledge can be used to manage dynamic environments. This performance was demonstrated in experiment, where a dynamic environment was simulated through addition of new features that were not included in previous training episodes.

In addition, we implemented these two approaches to manage real scenarios in which noisy signals were present. The experiments showed that LEARCH-RL-LSTM can reproduce the desired behavior and navigate through the environment.

Author details

Nancy Arana-Daniel*, Roberto Valencia-Murillo, Alma Y. Alanís, Carlos Villaseñor and Carlos López-Franco

*Address all correspondence to: nancyaranad@gmail.com

Department of Computer Science, Universidad de Guadalajara, Guadalajara, Jalisco, México

References

[1] Silver D, Bagnell JA, Stentz A. Learning from demonstration for autonomous navigation in complex unstructured terrain. The International Journal of Robotics Research. 2010;**29** (12):1565-1592. DOI: 10.1177/0278364910369715

[2] Surger B, Steder B, Burgard W. Traversability analysis for mobile robots in outdoor environments: A semi-supervised learning approach based on 3D-lidar data. In: 2015 IEEE International Conference on Robotics and Automation (ICRA); 26–30 May 2015; Seattle, WA, USA. IEEE; 2015. p. 3941-3946. DOI: 10.1109/ICRA.2015.7139749

[3] Häselich M, Jöbgen B, Neuhaus F, Lang D, Paulus D. Markov random field terrain classification of large-scale 3D maps. In: 2014 IEEE International Conference on Robotics and Biomimetics (ROBIO 2014); 5–10 Dec 2014; Bali, Indonesia. IEEE; 2014. p. 1970-1975. DOI: 10.1109/ROBIO.2014.7090625

[4] Kondo M, Sunaga K, Kobayashi Y, Kaneko T, Hiramatsu Y, Fuji H, Kamiya T. Path selection based on local terrain feature for unmanned ground vehicle in unknown rough terrain environment. In: 2013 IEEE International Conference on Robotics and Biomimetics (ROBIO); 12–14 Dec 2013; Shenzhen, China. IEEE; 2013. p. 1977-1982. DOI: 10.1109/ROBIO.2013.6739759

[5] Murphy L, Newman P. Risky planning on probabilistic Costmaps for path planning in outdoor environments. IEEE Transactions on Robotics. 2013;**29**(2):445-457. DOI: 10.1109/TRO.2012.2227216

[6] Valencia-Murillo R, Arana-Daniel N, López-Franco C, Alanís A. Rough terrain perception through geometric entities for robot navigation. In: 2nd International Conference on Advances in Computer Science and Engineering (CSE 2013); 1–2 Jul 2013; Los Angeles, CA, USA. Atlantis Press; 2013. DOI: 10.2991/cse.2013.69

[7] Bakker B. Reinforcement learning with long short-term memory. In: Advances in Neural Information Processing Systems 14. Cambridge: MIT Press; 2002. p. 1475-1482

[8] Kalman R. When is a linear control system optimal. Journal of Basic Engineering. 1964;**86**(1): 51-60

[9] Cortes C, Vapnik V. Support-vector networks. Machine Learning. 1995;**20**(3):273-297. DOI: 10.1007/BF00994018

[10] Sutton R, Barto A. Reinforcement Learning: An Introduction. Cambridge: MIT Press; 1998

IntechOpen