Robotica: (2020) volume 38, pp. 1450–1462. © Cambridge University Press 2019
doi:10.1017/S0263574719001565

Combination of Recurrent Neural Network and Deep Learning for Robot Navigation Task in Off-Road Environment

Farinaz Alamiyan-Harandi†, Vali Derhami†* and Fatemeh Jamshidi‡

†*Computer Engineering Department, Faculty of Engineering, Yazd University, Yazd, Iran*
E-mail: f.alamiyan@yazd.ac.ir
‡*Department of Electrical Engineering, Faculty of Engineering, Fasa University, Fasa, Iran*
E-mail: jamshidi@fasau.ac.ir

(Accepted October 9, 2019. First published online: November 4, 2019)

SUMMARY
This paper tackles the challenge of the necessity of using the sequence of past environment states as the controller's inputs in a vision-based robot navigation task. In this task, a robot has to follow a given trajectory without falling in pits and missing its balance in uneven terrain, when the only sensory input is the raw image captured by a camera. The robot should distinguish big pits from small holes to decide between avoiding and passing over. In non-Markov processes such as the abovementioned task, the decision is done using past sensory data to ensure admissible performance. Applying images as sensory inputs naturally causes the curse of dimensionality difficulty. On the other hand, using sequences of past images intensifies this difficulty. In this paper, a new framework called recurrent deep learning (RDL) with combination of deep learning (DL) and recurrent neural network is proposed to cope with the above challenge. At first, the proper features are extracted from the raw image using DL. Then, these represented features plus some expert-defined features are used as the inputs of a fully connected recurrent network (as target network) to generate command control of the robot. To evaluate the proposed RDL framework, some experiments are established on WEBOTS and MATLAB co-simulation platform. The simulation results demonstrate the proposed framework outperforms the conventional controller based on DL for the navigation task in the uneven terrains.

KEYWORDS: Robot navigation; Supervised deep learning; Recurrent network; Depth data; Uneven terrain.

1. Introduction

The problem of autonomous navigation is a challenging task for wheeled robots especially in an off-road environment with uneven terrain. Colliding to various obstacles and the risk of falling in existed pits and holes make this task more complicated.

Perception, planner, and motion control are conventional processes usually applied in robot navigation.[1,2] The most related researches have represented the robot environment as a grid map to determine the traversability of the outdoor terrains. These maps are created using some devices such as laser scanner, stereo camera, and Kinect camera. They can be a binary representation of the terrain as an obstacle occupancy grid,[3,4] a continuous value representation of the probability distribution for occupancy of each grid cell by an obstacle,[5] or a non-binary mathematical function of the

* Corresponding author. E-mail: vderhami@yazd.ac.ir

slope and roughness of the terrain for each cell of the map.[6] This traversability map has just been used in the path planning process to prepare suitable information which helps the robot besides the output data of positioning systems like GPS to find the robot location and measure the distance the robot moved. This map has not interfered in the robot controller directly.

Creating control architectures based on visual inputs has been recently considered in autonomous navigation tasks. Some vision sensors like Kinect camera simply provide low-cost information such as Red Green Blue (RGB) images and depth measurements about the robot environment. Using depth data as sensory inputs in comparison with RGB images makes the controller more robust against environment variations such as lighting changes and texture variety.[7,8]

Vision sensors prepare the high-dimensional outputs (RGB images and depth data). Using these data directly as controller inputs, the structure of the controller becomes complicated. It makes adjusting the controller parameters more time-consuming and difficult, too. So, it is more rational to reduce the input size by extracting proper features from these data and the controller architecture utilizes them. Some researchers have applied image preprocessing methods, for example, using a CCD color camera, to control the position of a goal-seeking robot,[9] estimating the 3D trajectory in unknown outdoor environments by a trinocular stereo camera,[10] applying both laser range finder and stereo vision to detect trajectory and designing the steering control law based on the kinematic equations of motion,[2,11] utilizing a stereo and mono vision to create a reinforcement learning (RL) approach for a wall follower robot to learn reactive behaviors.[12]

Preprocessing techniques require complex mathematical computation. They also need expert knowledge about models of the operating environment to extract proper control features. Applying these methods, some effective features may not be detected. So, end to end approaches, recently deep learning (DL) structures, have been considered.

Some examples of using end to end techniques are as follows: DL has been applied to design a self-supervised learning (SL) process for a long-range vision to classify complex terrain far from the outdoor-navigator robot and to predict its traversability.[13] A deep convolutional network has been trained using the end to end idea to describe an obstacle avoidance system for off-road mobile robots. This system has mapped raw input images to steering angles in a supervised manner. This structure has too many parameters which are trained simultaneously using massive amounts of training data.[14]

The main advantage of these approaches is training the whole process from data and eliminating hand-crafted heuristics to design and select features. In this way, an objective function is globally optimized automatically using raw data. These approaches make systems robust against the unpredictable variations of the input space. They may potentially detect other useful cues which may not be considered in expert methods.[14] Deep architectures describe various variations behind data and directly represent the data inside the structure of a controller. They introduce some represented features and make feature extraction easier.[15]

Some significant researches have utilized deep neural network (DNN) as a data representation structure and have combined it with other learning techniques in order to solve their problems. For example, merging DNN with RL algorithms in the following works which create a function approximator using DNN and estimate state-action value function to learn their policy: neural fitted Q-iteration, a memory-based method which reuses state transition experiences,[16] deep Q-network (DQN) which combines Q-learning with convolutional neural network (CNN) and updates the parameters of state-action value function using periodically updated targets,[17] deep deterministic policy gradient which is an off-policy actor-critic algorithm and uses soft target updates to make the learning process stable,[18] asynchronous advantage actor-critic algorithm which uses parallel threads of same training process in order to share some needed parameters between actor learners and to stabilize them,[19] and an actor-critic RL algorithm with continuous actions from a stochastic control policy which uses deep auto encoders to prepare the proper features for controlling industrial laser welding processes.[20]

The above mentioned literature worked on learning policies directly on visual inputs (raw images). They had DNNs with plenty of adjustable parameters and needed to store and reuse all state transition experiences. Training large neural networks by RL makes the learning time-consuming. They are still trying to make their methods data-efficient to be applied successfully with large neural networks.

Vision sensors have a limited field of vision and cannot cover area around the robot completely. If only the current sensory data are used to define the environment states, this limitation will lose the well-known Markov property of memorylessness. Mounting several devices on the robot can extend the field of vision but increases weight, costs, and raw data preprocessing time.[21] It also intensifies

https://doi.org/10.1017/S0263574719001565 Published online by Cambridge University Press

the curse of dimensionality problem of image inputs and results in a complex controller structure. So, it is better to keep some of the previous environment states and robot actions as short-term memory. It compensates the relatively small field of vision while determining the next command of the robot controller.

The memory-exploitation capability of a recurrent neural network (RNN) can be used to represent the environment states in the most compacted input data supporting the Markov property. RNNs create some internal states with dynamic temporal characteristics using directed cyclic connections between their neurons. They form an internal memory which is suitable for processing arbitrary input data sequences. RNNs have been utilized in speech and handwriting recognition, program code generation, etc.[22]

RNNs can be used as a controller in robotic researches, too. A two-link planar robot manipulator has been controlled using a control system consisted of an RNN controller, fast-load adaptation, and robust PID controller.[23,24] This RNN has modeled the relation between inputs and outputs in given dynamic systems. It has two kinds of feedback connections. Some are the hidden feedbacks of the Elman network and some are from the output layer to the context layer. The simulation results have demonstrated this RNN can control the trajectory of the planar robot arm better than classical feed-forward and diagonal RNNs.

Two series RNNs have been used for the robot navigation task in an unknown flat environment.[25] One of them has solved the localization problem and another has been considered for path planning. The robot was equipped with three infrared sensors to detect obstacles. The kinematics of the platform has been considered to develop the motion of the robot. In localization problem solving, the current robot position in a determined Cartesian system coordinates and the robot steering command have been fed to an RNN and the net output has provided the next robot position. The current position of the robot and the position of target besides a defined function of existed obstacles have been used to find the robot steering command in path planning RNN.

In the abovementioned robotic researches, the input size of the RNN structure was small. Use of the RNN architecture purely is not suitable when network inputs are images considering the huge number of weights which should be trained. Deep recurrent Q-network (DRQN)[26] is a modified version of DQN. This architecture has added recurrency to a DQN framework. It has replaced the first fully connected layer with a recurrent long short-term memory (LSTM) to handle the noisy and partial observability characteristic of the environments in some Atari games. The parameters of the convolutional and recurrent layers of this network are learned simultaneously from scratch. DRQN gives solutions to problems with continues and high-dimensional state spaces, but they are applied to low-dimensional discrete action spaces. The discretization of action space makes DRQN applicable to continuous spaces, but it encounters the curse of dimensionality and may lose essential information of action space structure.[18]

To cover the robot working place comprehensively by exploiting the previous state features and robot actions, and to moderate the process of training plenty of network weights, the SDL framework[27,28] has been proposed which has combined SL, DL, and RL.

An end to end object tracking approach[29] has been represented by combining the DL technique with an RNN. The RNN has exploited the sequence models and a mapping from sensory data to object tracks has been learned. The used network has been composed of input-preprocessing layers (the Encoder), the hidden state propagation section (the Belief tracker), and the final layers (the Decoder). The Encoder analyzes the data captured by a planar laser scanner to detect visible objects. The results of the Belief tracker have been fed to the Decoder to compute a probabilistic occupancy grid related to the corresponding pixels to find non-visible objects in the current view.

In this paper, a new memory-exploitation framework is proposed to create robot controllers. A set of represented features obtained by DL as well as expert-defined features are fed to a recurrent network to combine memory-based information of features with current inputs. Then, a robot controller is created for tasks with complex environments. The results are demonstrated that the previously visited features improve the control commands to be more effective and provide a Markov representation of environment states.

The main contributions of this paper are as follows: (1) introducing represented and task-based expert features to define the environment states (generating features from depth images captured by a camera mounted on the robot in order to represent the environment states, determining two groups of task-based expert features considering the navigation task to improve this representation), (2) combining deep auto-encoders and RNN to propose a new structure called recurrent deep learning

https://doi.org/10.1017/S0263574719001565 Published online by Cambridge University Press

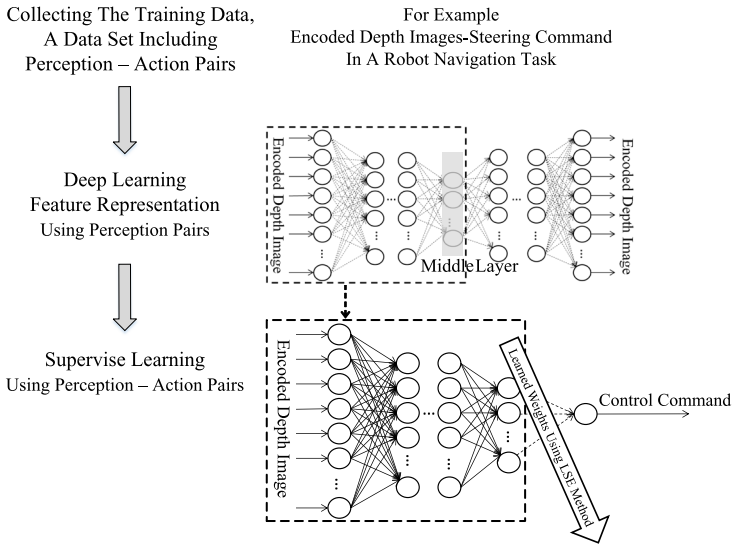

Fig. 1. The schema of the basic SDL approach (solid and dash lines in neural networks show the fixed and under-trained weights, respectively).

(RDL) which introduces a memory-exploitation controller for problems with continuous state and action spaces, the proposed framework includes two separate phases: the features are generated using a separate learning phase and then these features are used as the input of the controller part which is tuned with SL based on the control task. Generating the represented features does not need control data, so it can be obtained easily. This means the robot can move in the environment randomly and robot movements do not have any limitation, (3) applying the proposed controller for a difficult robot navigation task in uneven terrain with plenty of various holes and pits when the only sensory input is the raw depth image captured by a camera which is the least facility, while in the same literatures more than one sensor are applied. This advantage results in decreasing the equipment cost. The robot should distinguish big pits from small holes to decide between avoiding and passing over which is not considered in similar research.

The rest of this paper is organized as follows. In Section 2, the principles of deep-architecture controllers is explained. The proposed approach is given in Section 3. Section 4 includes the experiments and comparison results. Finally, concluding remarks are given in Section 5.

2. Designing Controllers Using Deep Feature Representation Learning

Data representation is a prominent issue of machine learning algorithms because the performance of these methods depends on the represented data called features. Feature engineering is an effort in achieving suitable representations using human ingenuity and prior knowledge. It includes the design of various data transformations and preprocessing techniques. Since this way of engineering is laborious and needs human interference, it is desired to enhance learning algorithms such that they can learn to identify and distinguish the underlying explanatory factors behind the observed low-level sensory data.[15,30]

A good representation can be useful as an input of a supervised robot controller, the problem considered in this paper. DL is one of the solutions to learning representations. DL techniques learn a hierarchical representation of data. They compose of multiple nonlinear transformations created through the multi-layer architecture neural networks and can represent data in a more abstract and general representation using supervised and unsupervised learning algorithms.[15]

There are various types of DNNs such as auto-encoder networks, CNNs, RNNs, and recursive neural networks. Considering the structure of input spaces in different problems, the combination of these networks can be used, too.[22,31] More layers in these networks, more complicated the optimization problem. Hence, layer-wise training algorithms can be used to train these networks.[32–34]

The SDL framework[27,28] illustrated in Fig. 1 has an appropriate structure to be expanded and to be used to adjust the robot controller. So, the proposed idea is developed based on this schema.

https://doi.org/10.1017/S0263574719001565 Published online by Cambridge University Press

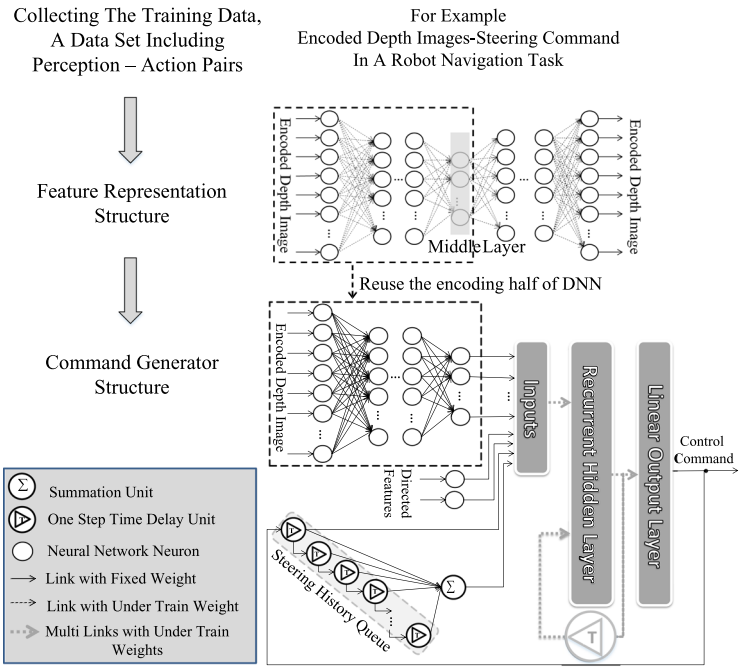

Fig. 2. The structure of RDL controller.

This framework integrated a data representation architecture into a command generator to create an effective robot controller. First, a DNN as a data representation structure is trained to convert raw sensory data to compact informative features in order to represent the environment state.

Various kinds of DL techniques like auto encoders are applicable in this phase. Here, a deep feed-forward neural network named deep auto-encoder is used as a function approximator to generate represented features. This architecture has a symmetric layer organization. The encoding half of the network includes hidden layers with a decreasing number of neurons from the input layer to the middle compressed representation layer. The second set of hidden layers makes up the decoding half and its belonged layers increase the number of neurons consecutively. The output layer tries to recover input data from the compressed meaningful representation using the same number of neurons that exist in the input layer.[20] Since these outputs are used as an activation degree of features, neurons of the middle layer have a transfer function with a positive output range.[28] This network is trained by SL and error back-propagation rule and the adjusted weights of its encoding half are frozen after the train.

The linear weighted combination of represented features is used as a command generator to create a robot controller. The output of this controller is a real continuous value. Linear least square errors method is applied to compute the weights of the linear composition of represented features and some expert-defined features as the basis functions using training data.

3. The Proposed RDL Framework

In order to create the controller architecture using the proposed framework, the phases shown in Fig. 2 are followed. Training a DNN as explained in Section 2 prepares the represented features.

Assume that each captured depth image is summarized to a grid map with p blocks. So, there are p neurons in the input and output layers of state representation structure. Consider that there are q_i neurons in the ith encoding hidden layer and each neuron has an activation function $f_i : \mathbb{R} \to \mathbb{R}$, for example, the sigmoid function and $x \in \mathbb{R}^p$ is a vector from the input layer. The feature value at the jth neuron in the first and ith hidden layers are then computed as:

$$h_{1,j}(x) = f_1\left(w_{1,j}^T x + b_{1,j}\right) \tag{1}$$

and

$$h_{i,j}(h_{i-1}) = f_i\left(w_{i,j}^T h_{i-1} + b_{i,j}\right) \tag{2}$$

https://doi.org/10.1017/S0263574719001565 Published online by Cambridge University Press

where $w_{1,j} \in \mathbb{R}^p$ and $w_{i,j} \in \mathbb{R}^{q_i}$ denote the weights associated with the jth neuron of the first and the ith encoding hidden layer, $b_{1,j} \in \mathbb{R}$ and $b_{i,j} \in \mathbb{R}$ are their corresponding biases, respectively, and h_{i-1} is the representation vector generated by the $(i-1)$th encoding hidden layer.

The last hidden representation generated by the middle hidden layer, h_{mid}, will serve as an input for the first decoding layer. The output value at the jth neuron in the kth decoding hidden layer and the output layer (as the final layer) are computed as:

$$y_{k,j}(y_{k-1}) = f_k^{'}\left(w_{k,j}^{'T} y_{k-1} + b_{k,j}^{'}\right) \tag{3}$$

and

$$y_{final,j}\left(y_{final-1}\right) = f_{final}^{'}\left(w_{final,j}^{'T} y_{final-1} + b_{final,j}^{'}\right) \tag{4}$$

where $w_{k,j}^{'} \in \mathbb{R}^{q_k}$ and $w_{final,j}^{'} \in \mathbb{R}^p$ denote the weights associated with the jth neuron of the kth decoding hidden layer and the output layer, $b_{k,j}^{'} \in \mathbb{R}$ and $b_{final,j}^{'} \in \mathbb{R}$ are their corresponding biases, respectively. y_{k-1} is the output vector generated by the $(k-1)$th decoding hidden layer.

The error between the original input vector x and the final reconstruction y can be considered as a measure for the quality of the representation structure and serves as a loss function for the back-propagation algorithm.[35] Here, for real-valued inputs, images, the mean squared error (MSE) is the most common choice resulting in the final cost function:

$$C(W, B) = 1/2n \sum_x \|x - y\|^2 \tag{5}$$

where W and B denote the collection of all weights and all the biases in the network, respectively. n is the total number of training samples. y is the vector of outputs from the network when x is the input. Here the desired output is x, too. The sum is over all training inputs, x. Note that the output y depends on all entries of x, W, and B, but to keep the notation simple, it does not explicitly indicate this dependence. The notation $\|a\|$ just denotes the usual length function for the vector a.[22]

Some expert features are defined and added to the represented features, h_{mid} being generated for each input data after training the representation network and freezing the encoding weights, in order to improve the performance of the controller. Here, two groups of these specific features are introduced considering the navigation task: (1) directed features associated with wall following mission including the positions of guideline in RGB images as a grid map, and (2) features related to the obstacle avoidance task such as the previous applied steering angle and sum of previous steering angles which can manage in a first in first out queue. All the abovementioned features create the vector v with q_v elements which is used as the input of the command generator structure.

The previous condition of the environment and behavioral history of the controller can modify the next command chosen by the robot controller, too. This memory-based information can distinguish situations, especially in an obstacle avoidance task. So, to exploit them and complete the representation of the environment states, all the abovementioned features are fed to a recurrent hidden layer with q_v neurons. This layer is followed by a linear neuron in the output layer and creates the command generator of the robot controller. Defining a recurrence relation over time steps in RNN can deal with sequences of variable length. This relation is typically shown in the following formula:

$$s_j^k = f_{recurent}\left(s^{k-1} \times w_{recj} + v \times w_{vj} + b_j\right) \tag{6}$$

where s_j^k is the state of jth neuron in the recurrent hidden layer at time k. v is an exogenous input at time k prepared by the features representation structure and expert knowledge. s^{k-1} is the vector of all neuron states in the recurrent hidden layer at time $(k-1)$, $w_{recj} \in \mathbb{R}^{q_v}$ and $w_{vj} \in \mathbb{R}^{q_v}$ denote the weights associated with the jth neuron of the recurrent hidden layer, and $b_j \in \mathbb{R}$ is their corresponding bias.

The parameters of these added layers are adjusted in the same way as the basic controller using SL, back-propagation rule, and training data. It is evaluated by the MSE cost function as follows:

$$C(W, B) = 1/2n \sum_v \|O_{desired} - O\|^2 \tag{7}$$

https://doi.org/10.1017/S0263574719001565 Published online by Cambridge University Press

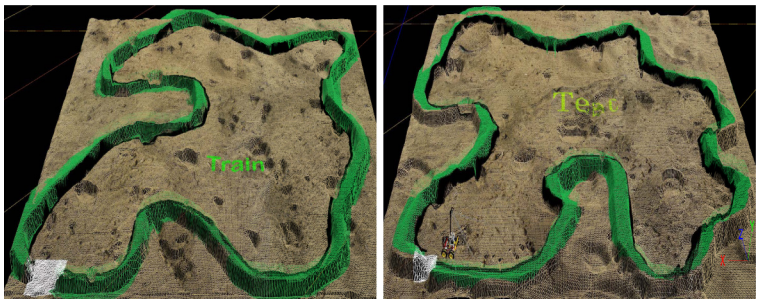

Fig. 3. The Train and Test environments.

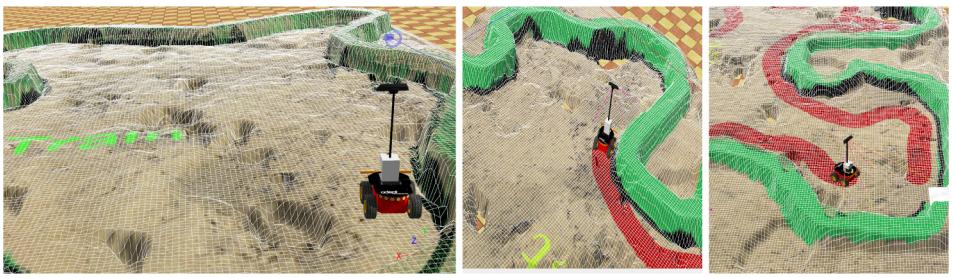

Fig. 4. Some sample closer views of uneven terrains in experiments.

where O is the vector of controller outputs and $O_{desired}$ is the vector of all supervised commands in collected training data. The sum is over all represented training inputs, v.

RDL framework can be summarized as follows:

1. Collect training data, a data set including perception-action pairs.
2. Train the deep feature representation structure using the cost function in Eq. (5) for perception elements of pairs belongs to the training data set.
3. Reuse encoding half of the data representation structure with the frozen weights to generate the represented feature vector h_{mid} for each input of the controller using Eqs. (1) and (2).
4. Add expert features and create v vector for each represented feature vector h_{mid}.
5. Create the command generator structure by adding a recurrent hidden layer and a linear output layer.
6. Train the recurrent structure using the cost function in Eq. (7) for perception-action pairs belongs to the training data set.

4. Experimental Results

To evaluate the proposed approach, a robot navigation task in an off-road environment with uneven terrain was considered. The mobile robot and the experimental environments were established on the WEBOTS simulator and the robot controller was implemented in MATLAB software. The control task was performed in the simulation environment. The performance of Pioneer3_AT[1] which is a four-wheeled mobile robot was evaluated in the experiments. For achieving depth images, a Kinect camera was used. This is the only vision system which the robot was equipped with. Kinect has a depth image resolution of 320×240 pixels with a field of view of $58.5° \times 46.6°$ resulting in an average of about 5×5 pixels per degree.

To create an uneven terrain in simulated environments, various real depth images with admissible roughness were considered. They were combined with some manually built hills and holes to form an elevation map. The generated terrains for Train and Test environments are displayed in Fig. 3. Train environment was used to gather training data and Test environment was used to compare the performance of the proposed controller. Some closer views of uneven terrain used in experiments are shown in Fig. 4, too.

[1]For more information, please visit http://robosklep.com/en/wheeled-robots/154-pioneer-3-at.html.

https://doi.org/10.1017/S0263574719001565 Published online by Cambridge University Press

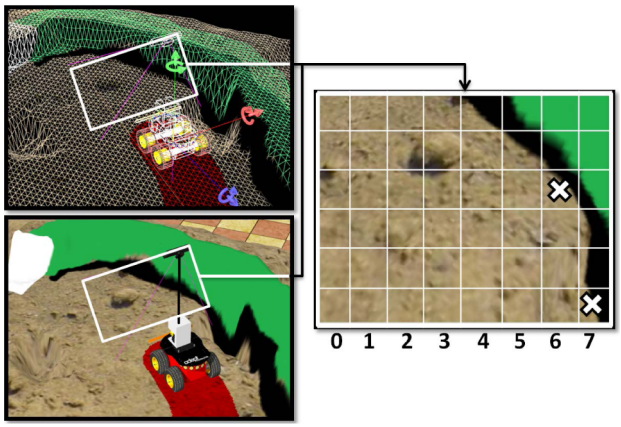

Fig. 5. A sample of RGB grid map where the positions of guideline are determined in the lowest and middle rows.

Here, the navigation task is defined as the following mission; a robot tries to reach and follows a given trajectory (wall) while it prevents falling in the existed holes and pits and keeps its balance in uneven terrain. The robot has to always be on the left side of the trajectory (wall). It has to preserve the desired distance from the trajectory while avoiding crossing the trajectory line (to collide with the wall). The admissible distance from the trajectory line is limited to a boundary. The depth image captured by Kinect was converted to a rectangular grid of 40×40 pixels blocks. They were inputs of the controller. The mean and variance of depth values of all pixels in each block were computed and used to describe that block. These two-dimensional data were ordered in a linear arrangement of rows (a row vector with 96 elements) and were used as the input of DNN to create represented features.

The captured RGB images were used to define expert features. Each image was converted to a grid map with 6×8 blocks. Each block contained 40 pixels of the image in each dimension. The positions of guideline (which exists near the wall) in the lowest and middle rows of this map were determined by utilizing its color. A sample of this operation is shown in Fig. 5. The previous applied steering angle and the sum of the last five steering angles were joined to the above features, too.

Collecting the training data is a common unavoidable task in most of machine learning methods specially supervised vision-based techniques and it is time-consuming.[36] The proposed framework includes two separate learning phases. The first learning phase generates the features. These features are used as the input of the controller part tuned with the second SL phase based on the control task. Generating the features does not need the control data. To gather data for this phase, the robot can move in the environment randomly without considering the determined task missions, so the data can be obtained easily. This is an improvement in manually data collection challenge. In the proposed framework, the command generator part has fewer adjustable parameters, and therefore its tuning needs less training samples. In the experiments, the robot was moved manually in Train environment multiple times using a joystick and the training data were automatically generated during the interaction between the robot and its simulated environment. The controllers were tuned using this set of input–output pairs. The data set includes about 5640 samples where the sample capturing rate is 2.5 images per second. The first elements of sample pairs, encoded depth maps, were duplicated and used as the input–output pairs to train the representation network for generating the represented features. Then the pairs of the encoded depth map and its associated control commands were utilized as the input and output of the controller to adjust the command generator of controllers. The control command is a value belongs to $\{-20, 0, +30\}$ and indicates changes in the direction of robot orientation (turn right, go straight ahead, and turn left). The negative value makes the robot to turn right and the positive one causes a left turn in the robot movement. The greater positive value keeps the robot away from the holes and pits more quickly, while the lower negative value returns the robot to the suitable path slowly and avoids it falling in the holes and pits.

A DNN with nine hidden layers was used in the image feature representation task. The number of neurons in each hidden layer is chosen using expert knowledge as 80, 60, 50, 30, 20, 30, 50, 60,

https://doi.org/10.1017/S0263574719001565 Published online by Cambridge University Press

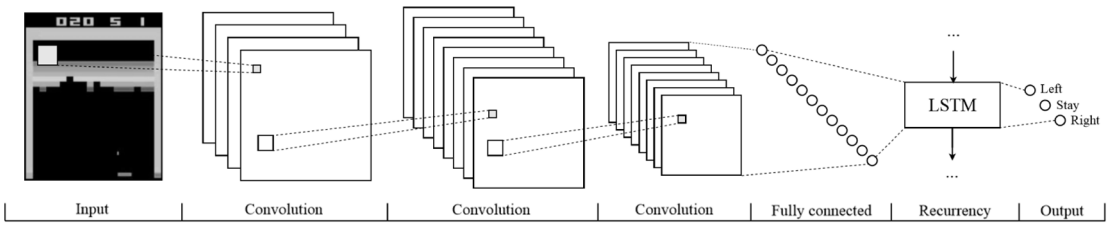

Fig. 6. CNN-LSTM structure with pixel input.[37]

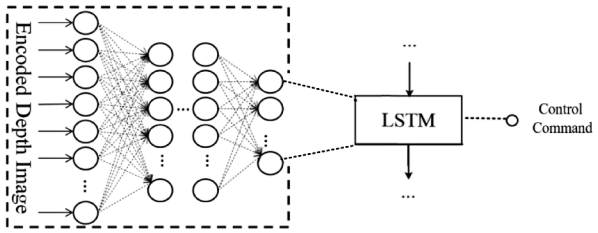

Fig. 7. MLP-LSTM structure with encoded depth image.

and 80, respectively. This network has 96 inputs and 96 outputs. The train operation starts using randomly initialized weights.

The first adjusted half of this feed-forward network was used as a function approximator to construct the basic controller. This network has five layers with 80, 60, 50, 30, and 20 hidden neurons with fixed weights and tan-sigmoid and log-sigmoid transfer functions in the first 4 hidden layers and the last hidden layer, respectively. A linear output unit forms the output layer of the basic controller and produces a continuous real value output. Adjusting the weights of the linear combination of features in the output layer is a single-pass operation. The outputs of data representation structure plus expert-defined features were fed to a recurrent layer with 24 hidden neurons with log-sigmoid transfer functions to append feature histories. This layer is followed by a linear output unit in the output layer and the command generator is formed to work as the proposed controller. Adjusting the weights of this network is done using SL and back-propagation rule.

According to review paper[37] and lots of its referred references, a typical network architecture (named CNN-LSTM) used in DL for control tasks with pixel input is as follows: The input (a preprocessed screen image or several stacked or concatenated images) is followed by some convolutional layers without pooling, and a few fully connected layers. After the fully connected layers, there is a recurrent layer, such as LSTM or gated recurrent unit. In the output layer, each combination of actions in the task has one unit. When the input consists of some features instead of pixels, the CNN layers are replaced by MLP structure. In this paper, the input of the controllers included some statistical depth information as a grid map and it was needed to apply some changes in implementation of the network structure of these works. So, to compare the RDL with CNN-LSTM structure shown in Fig. 6, the final implementation was modified to one illustrated in Fig. 7 named MLP-LSTM and it was thoroughly learned using SL.

The result of network training is affected by the initial weights and biases of hidden units, the uncertainty of the environment, and the inconsistency of training data.[36] So, the testing phase was run several times and the evaluation of the experiments was evaluated by the average of evaluation criteria over five independent runs. The average of testing time steps over these runs is around 1800.

The performance of controllers is compared to each other considering some defined failures as follows:

1. "No Visible Trajectory" failure occurs when the robot cannot capture the guideline in camera color images. This failure can happen as a normal event in strongly convex routes and spiral paths. Since losing the trajectory is also possible, this situation should be distinguished in experiments. It is supposed that the robot can capture the trajectory line in convex paths after at most 110 successive "No Visible Trajectory" failures. Exceeding this threshold means the robot loses the trajectory. The appropriate amount of this threshold helps the robot to return to the suitable situation before

https://doi.org/10.1017/S0263574719001565 Published online by Cambridge University Press

Table I. Comparison results in the experiments.

Test environment			
	Controllers		
	Basic SDL	RDL	MLP-LSTM[37]
Number of failures as evaluation criteria	**(24 features)**	**(24 features)**	**—**
Falling and losing balance	13	**1.6**	15.4
Too near to wall	13	**10.6**	73.2

The bold values demonstrate the performance of the proposed RDL controller in comparison with the basic SDL controller and MLP-LSTM controller considering the given criteria. They show lower failures.

losing the path completely. In this way, the robot controller needs less supervisor interference. This threshold is obtained by trial and error according to path curves in environments and the maximum value of angles the supervisor applied to the robot for passing these curves when the training data is gathered.

2. "Falling and losing balance" failure is detected via an inertial unit mounted on the robot.
3. "Too Near to Wall" failure shows the situations when the robot moves besides the wall and crosses over the guideline and a collision is possible.
4. "Too Far from Wall" failure includes the situations when the distance from the wall is greater than the desired threshold and a path loss is possible. It should be noted that this failure can happen when the direction of the robot is not parallel to the curvature of the wall, too.

Here, the performance of the proposed RDL controller is compared with the basic SDL controller and MLP-LSTM controller considering the abovementioned criteria. The statistical results of this comparison are illustrated in Table I. This table shows the performance improvement by applying the proposed RDL. The number of Falling and losing balanceİ and Too Near to Wallİ failures decreases about 87.7% and 18.5%, respectively, in comparison with the basic SDL controller for Test environment. The number of these failures decreases about 89.6% and 85.5% in comparison with MLP-LSTM controller, respectively.

Since "Too Far from Wall" failure sometimes is due to the existence of holes and pits, the number of this failure for RDL controllers is greater than the basic SDL and the qualitative comparison of them is needed to evaluate them. Figure 8 displays the samples of trajectories which are passed by the robot using evaluated controllers in Test environment.[2] As this figure shows, the performance of RDL in following the given trajectory curves is the best. RDL never misses the wall and it avoids holes and pits more precisely because it utilizes the histories of features and previous steering commands which represent previous environment states. As it is visible in the trajectory figures of the basic SDL and MLP-LSTM controller (position determined by a blue rectangle), they missed the wall when they confronted a strongly convex path. RDL is also able to distinguish the smaller holes from bigger pits to cross over them without going far from the wall to avoid failures.

The reasons of the pre-mentioned improvement are as follows. The proposed framework includes two separate learning phases. It uses DL in the first phase to cover the input space thoroughly. The recurrent layer is applied to append memory-dependent features in the next phase in order to compensate non-Markov property of the environment, supervisor errors, and data inconsistency. The command generator part has fewer adjustable parameters. So it is tuned in shorter learning time and needs less training samples.

Making the block size of maps smaller (e.g., 20×20 pixels), the approach is able to encode the environment states in depth images more accurately and the robot can preserve its distance to guide line more precisely.

5. Conclusion and Future Works

In this paper, DL and RNN were combined to form a new framework called RDL. This framework was applied to control a mobile robot in an off-road navigation task. The robots controller received

[2]You can view a movie of the robot performance when it used RDL architecture to complete its missions during one of these trajectories in Test environment at https://pws.yazd.ac.ir/lcir/?attachment_id=3390.

https://doi.org/10.1017/S0263574719001565 Published online by Cambridge University Press

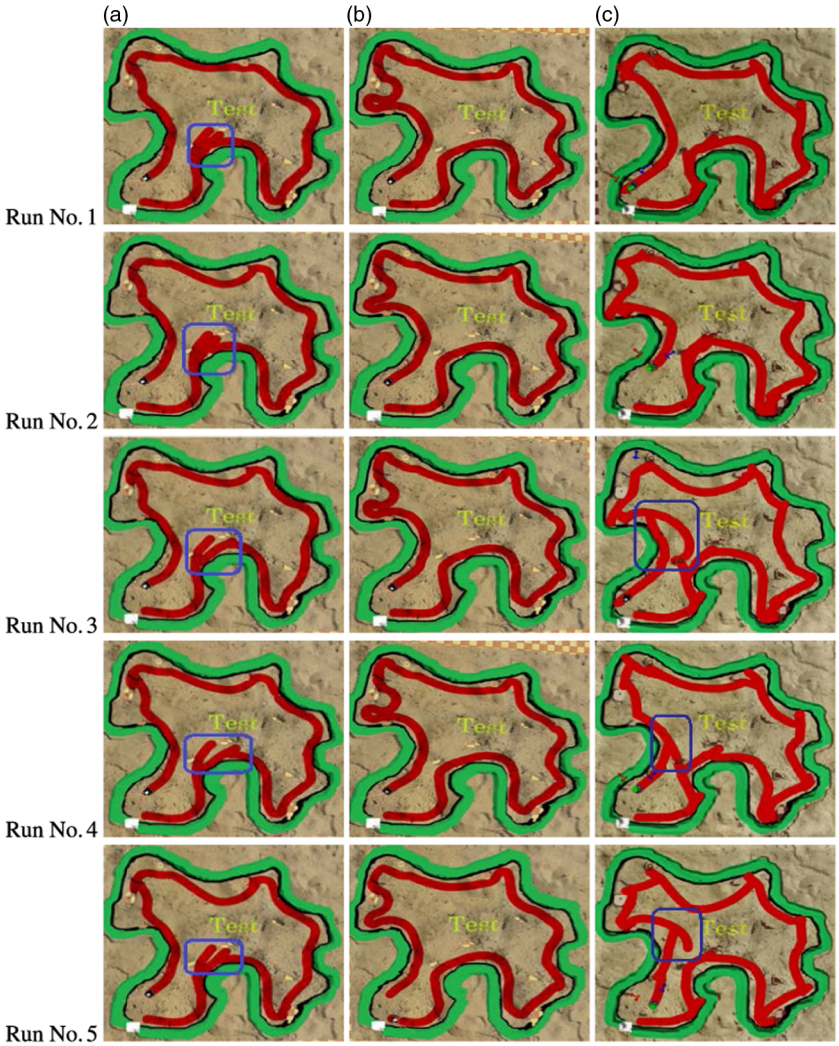

Fig. 8. The samples of trajectories which are passed by the robot using evaluated controllers in Test environment. (a) Basic SDL controller, (b) RDL controller, and (c) MLP-LSTM controller.[37]

depth image data as the input and computed the steering angle as the output. Using DL, the compact descriptive features were extracted from high-dimensional raw images. These features and some expert-defined features were considered as the input of an RNN structure. The adjustable weights of RNN were tuned by supervisory data.

To assess RDL, it was applied to a vision-based robot navigation task. In this task, wall following, obstacle avoiding, and keeping balance should be simultaneously considered in uneven terrain with various holes and pits. Simulation results demonstrated that the proposed framework significantly reduces the number of failures especially Falling and losing balance.

It is concluded that the presented idea based on DL decreases the dimension of states and overcomes the curse of dimensionality. Applying the suggested memory-based RNN architecture in RDL, it is not essential to include the sequence of past states in the controllers input. Hence, RDL is suitable for non-Markov environments. As another advantage, RDL has a few adjustable parameters and low training computational cost.

As future works, RDL can be applied to various control tasks and complicated environments with continuous state and action spaces. Moreover, other learning algorithms can be employed to tune RNNs parameters more precisely.

Supplementary Material

To view supplementary material for this article, please visit https://doi.org/10.1017/S0263574719001565.

https://doi.org/10.1017/S0263574719001565 Published online by Cambridge University Press

References

1. G. Antonelli, S. Chiaverini and G. Fusco, "A fuzzy-logic-based approach for mobile robot path tracking," *IEEE Trans. Fuzzy Syst.* **15**(2), 211–221 (2007).
2. Y. Yang, M. Fu, H. Zhu, G. Xiong and S. Changsheng, "Control Methods of Mobile Robot Rough-Terrain Trajectory Tracking," *8th IEEE International Conference on Control and Automation (ICRA)*, Anchorage, Alaska (2010) pp. 731–738.
3. R. Hadsell, P. Sermanet, J. Ben, A. Erkan, J. Han, B. Flepp, U. Muller and Y. LeCun, "Online Learning for Offroad Robots: Using Spatial Label Propagation to Learn Long-Range Traversability," *Proceedings of Robotics: Science and Systems (RSS)* (2007) p. 32.
4. S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann and K. Lau, "Stanley: The robot that won the DARPA Grand Challenge," *J. Field Rob.* **23**(9), 661–692 (2006).
5. K. Konolige, M. Agrawal, R. Bolles, C. Cowan, M. Fischler and B. Gerkey, "Outdoor Mapping and Navigation Using Stereo Vision," **In**: *Experimental Robotics* (Springer, Berlin, Heidelberg, 2008) pp. 179–190.
6. C. Castejón, B. Boada, D. Blanco and L. Moreno, "Traversable region modeling for outdoor navigation," *J. Intell. Rob. Syst.* **43**(2–4), 175–216 (2005).
7. D. Hanafi, Y. M. Abueejela and M. F. Zakaria, "Wall follower autonomous robot development applying fuzzy incremental controller," *Intell. Control Autom.* **4**(1), 18 (2013).
8. C. Ye, N. H. Yung and D. Wang, "A fuzzy controller with supervised learning assisted reinforcement learning algorithm for obstacle avoidance," *IEEE Trans. Syst. Man Cybern. Part B (Cybern.)* **33**(1), 17–27 (2003).
9. F. A. Jafar, N. A. Zakaria and K. Yokota, "Visual features based motion controller for mobile robot navigation," *Int. J. Simul. Syst. Sci. Technol.* **15**(1), 7–14 (2014).
10. P. Saeedi, P. D. Lawrence and D. G. Lowe, "Vision-based 3-D trajectory tracking for unknown environments," *IEEE Trans. Rob.* **22**(1), 119–136 (2006).
11. G. M. Hoffmann, C. J. Tomlin, M. Montemerlo and S. Thrun, "Autonomous Automobile Trajectory Tracking for Off-Road Driving: Controller Design, Experimental Validation and Racing," *American Control Conference*, New York City, USA (2007) pp. 2296–2301.
12. P. Quintìa, J. E. Domenech, C. V. Regueiro, C. Gamallo and R. Iglesias, "Learning a Wall Following Behaviour in Mobile Robotics Using Stereo and Mono Vision," *IX Workshop en Agentes Fìsicos,* Vigo, Espana (2008).
13. R. Hadsell, P. Sermanet, J. Ben, A. Erkan, M. Scoffier, K. Kavukcuoglu, U. Muller and Y. LeCun, "Learning long-range vision for autonomous off-road driving," *J. Field Rob.* **26**(2), 120–144 (2009).
14. U. Muller, J. Ben, E. Cosatto, B. Flepp and Y. L. Cun, "Off-Road Obstacle Avoidance Through End-to-End Learning" **In**: *Advances in Neural Information Processing Systems* (Y. Weiss, B. Scholkopf, and J. Platt, eds.) (MIT Press, Cambridge, MA, 2006) pp. 739–746.
15. Y. Bengio, A. Courville and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Trans. Pattern Anal. Mach. Intell.* **35**(8), 1798–1828 (2013).
16. M. Riedmiller, "Neural Fitted Q Iteration - First Experiences with a Data Efficient Neural Reinforcement Learning Method," *European Conference on Machine Learning*, Porto, Portugal (2005) pp. 317–328.
17. V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland and G. Ostrovski, "Human-level control through deep reinforcement learning," *Nature* **518**(7540), 529–533 (2015).
18. T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. M. O. Heess, T. Erez, Y. Tassa, D. Silver and D. P. Wierstra, "Continuous Control with Deep Reinforcement Learning," *U.S. Patent Application 15/217,758* (2017).
19. V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver and K. Kavukcuoglu, "Asynchronous Methods for Deep Reinforcement Learning," *International Conference on Machine Learning,* New York City, USA (2016) pp. 1928–1937.
20. J. Günther, P. Pilarski, G. Helfrich, H. Shen and K. Diepold, "Intelligent laser welding through representation, prediction, and control learning: An architecture with deep neural networks and reinforcement learning," *Mechatronics* **34**, 1–11 (2016).
21. H. Schäfer, M. Proetzsch and K. Berns, "Obstacle Detection in Mobile Outdoor Robots," *Proceedings of International Conference on Informatics in Control, Automation and Robotics,* Angers, France (2007) pp. 141–148.
22. G. Ian, B. Yoshua and C. Aaron, *Deep Learning* (MIT Press, Cambridge, MA, 2016).
23. S. Yildirim, "Design of adaptive robot control system using recurrent neural network," *J. Intell. Rob. Syst.* **44**(3), 247–261 (2005).
24. D. Pham and S. Yildirim, "Design of a neural internal model control system for a robot," *Robotica* **18**(5), 505–512 (2000).
25. H. Brahmi, B. Ammar and A. M. Alimi, "Intelligent Path Planning Algorithm for Autonomous Robot Based on Recurrent Neural Networks," *International Conference on Advanced Logistics and Transport (ICALT)*, Tunisia (2013) pp. 199–204.
26. M. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable MDPs," *CoRR, abs/1507.06527* (2015).
27. F. Alamiyan Harandi and V. Derhami, "Feature extraction from depth data using deep learning for supervised control of a wheeled robot," *J. Control* **11**(4), 13–24 (2018).

https://doi.org/10.1017/S0263574719001565 Published online by Cambridge University Press

28. F. Alamiyan Harandi, V. Derhami and F. Jamshidib, "A new framework for mobile robot trajectory tracking using depth data and learning algorithms," *J. Intell. Fuzzy Syst.* **34**(6), 3969–3982 (2018).
29. P. Ondrúška and I. Posner, "Deep Tracking: Seeing Beyond Seeing Using Recurrent Neural Networks," *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, Phoenix, Arizona, USA (2016) pp. 3361–3367.
30. S. Lange and M. Riedmiller, "Deep Auto-encoder Neural Networks in Reinforcement Learning," *International Joint Conference on Neural Networks (IJCNN)*, Barcelona, Spain (2010) pp. 1–8.
31. L. Shao, Z. Cai, L. Liu and K. Lu, "Performance evaluation of deep feature learning for RGB-D image/video classification," *Inf. Sci.* **385**, 266–283 (2017).
32. Y. Bengio, "Learning deep architectures for AI," *Found. Trends Mach. Learn.* **2**(1), 1–127 (2009).
33. G. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science* **313**(5786), 504–507 (2006).
34. J. N. Liu, Y. Hu, J. J. You and P. W. Chan, "Deep Neural Network Based Feature Representation for Weather Forecasting" *Proceedings on the International Conference on Artificial Intelligence (ICAI)*, Las Vegas, USA (2014).
35. D. E. Rumelhart, G. E. Hinton and R. J. Williams, "Learning representations by back-propagating errors," *Nature* **323**(6088), 533 (1986).
36. F. Fathinezhad, V. Derhami and M. Rezaeian, "Supervised fuzzy reinforcement learning for robot navigation," *Appl. Soft Comput.* **40**, 33–41 (2016).
37. N. Justesen, P. Bontrager, J. Togelius and S. Risi, "Deep learning for video game playing," *IEEE Transactions on Games*, 1–1 (2019). doi: 10.1109/TG.2019.2896986