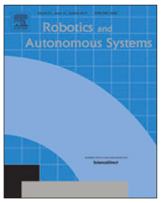
Robotics and Autonomous Systems 138 (2021) 103731

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot



Two-stage visual navigation by deep neural networks and multi-goal reinforcement learning

Amirhossein Shantia [*], Rik Timmers, Yiebo Chong, Cornel Kuiper, Francesco Bidoia, Lambert Schomaker, Marco Wiering

Bernoulli Institute for Mathematics, Computer Science and Artificial Intelligence, University of Groningen, 9747AG, Groningen, The Netherlands

A R T I C L E   I N F O

Article history:
Received 21 March 2020
Received in revised form 30 September 2020
Accepted 4 January 2021
Available online 16 January 2021

Keywords:
Robotic navigation
Reinforcement learning
Deep neural networks
Localization and mapping
Robot simulation

A B S T R A C T

In this paper, we propose a two-stage learning framework for visual navigation in which the experience of the agent during exploration of one goal is shared to learn to navigate to other goals. We train a deep neural network for estimating the robot's position in the environment using ground truth information provided by a classical localization and mapping approach. The second simpler multi-goal Q-function learns to traverse the environment by using the provided discretized map. Transfer learning is applied to the multi-goal Q-function from a maze structure to a 2D simulator and is finally deployed in a 3D simulator where the robot uses the estimated locations from the position estimator deep network. In the experiments, we first compare different architectures to select the best deep network for location estimation, and then compare the effects of the multi-goal reinforcement learning method to traditional reinforcement learning. The results show a significant improvement when multi-goal reinforcement learning is used. Furthermore, the results of the location estimator show that a deep network can learn and generalize in different environments using camera images with high accuracy in both position and orientation.

1. Introduction

Learning by reward and punishment is one of the fundamental learning methods in nature. This learning process is intrinsic in most of the species living on Earth, especially the ones with higher levels of cognitive abilities such as humans [1]. This type of learning, reinforcement learning [2], has been a subject of research for a long time. Its modern form, which is highly based on Markov decision processes, started to emerge in the 1980s [3], and became popular in the second half of the '90s [2,4].

There are two main learning methods in reinforcement learning (RL): model-based and model-free. The model-based approach requires a deep knowledge of the environment which allows us to build decision processes that connect the states with consequences of actions [5,6]. However, the model grows substantially when the number of states and actions increases. In addition, it is often very complex to learn a model of the tasks that the agent needs to solve. Model-free RL plays an important role by allowing the exploration of unknown state spaces and using function approximators (FA) [7–9]. These approximators estimate values of actions in a state through a linear or non-linear mapping. The linear FAs are proven to converge, but they lack the ability to map complex states (e.g. camera images as input) into meaningful value estimates for actions. This is the reason that non-linear FAs are often used to tackle complex problems. One suitable and frequently used non-linear FA is the artificial neural network. The main problem with neural networks and other non-linear FAs is that they can diverge from the optimal solution due to forgetting past experiences or instabilities in the learning process [7]. Therefore, careful thinking should be done during the design of the system. Types of feature inputs, the rewarding mechanism, and the propagation of these rewards are some of the important topics to consider. Due to these problems, the use of model-free RL methods was for a long time restricted to reasonably small sized problems such as robot gait control [10], unit control in games such as StarCraft [11], etc. However, with the recent progress in training deep neural networks, the prospects have changed for RL. Google's DeepMind Research on Atari games [12] was, perhaps, a substantial landmark and more influential than TD-Gammon [13] toward a large scale RL deployment that can solve difficult tasks that matches or surpasses human performance. Later, in 2014, with Google's DeepMind research on the complex board game Go [14], we saw yet another hurdle being removed from the world of non-linear approaches in RL. The trend has not stopped there;

[*] Corresponding author.
E-mail addresses: a.shantia@rug.nl (A. Shantia), r.p.timmers@rug.nl (R. Timmers), y.chong@rug.nl (Y. Chong), c.g.kuiper@student.rug.nl (C. Kuiper), francesco.bidoia@gmail.com (F. Bidoia), l.r.b.schomaker@rug.nl (L. Schomaker), m.a.wiering@rug.nl (M. Wiering).

https://doi.org/10.1016/j.robot.2021.103731
0921-8890/© 2021 Elsevier B.V. All rights reserved.

we have seen many RL publications ranging from completing objectives in games such as Doom [15], besting top tier players at StarCraft II [16] to more real-world applications such as the Google research on robotic manipulation [17].

All the above applications have one thing in common, the need for hundreds of thousands of epochs which translates to significant training time. This can, however, be reduced by using a combination of simulation, transfer learning, or multiple agents. For example, in order to learn to avoid obstacles in the environment with a robot, one can first train it in a simulator, and then use the same network to continue the training in real life [18]. However, avoiding obstacles using a camera requires learning the optical flow. While it is possible to partially learn optical flow in simulation and later complete the learning on a real robot, the same does not apply to place recognition, which is necessary for navigation. Nevertheless, it is possible to use simulations to speed up parts of the learning process. For this reason, we propose a method that can tackle robotic navigation tasks using a deep neural network architecture and model-free RL benefiting from simplified and complex simulations.

In this paper, we propose a novel two-stage framework to alter the common end-to-end RL scheme and reduce the required time to learn and navigate in the environment. In the first stage, we train a deep neural network to localize the robot in the environment using supervised learning, and in the second stage, a multi-goal RL method is used which uses the estimated positions given by the deep network to drive the robot toward the given goals. In stage one, we use a traditional grid-mapping algorithm (GMapping) to extract the geometrical topology of the environment for the supervised training section of our approach [19,20]. A set of images is recorded during the data gathering phase that are tagged with the estimated location from the GMapping algorithm. In our previous research [21], we showed that a stacked denoising autoencoder (SDA) can learn the geometrical relation between the images and the robot location in a small 3D simulated environment. In this paper, we use deep convolutional neural networks (CNNs) [22] to test the scalability and precision of this approach and compare the results to that of SDAs in order to select the best type of network architecture. In parallel, we use a grid-based approach to train our second stage multi-goal RL method. In the first step, we extract a maze from the given map, and train a multi-goal Q-function. When the training is finished, we continue the training of the same Q-function in a 2D simulator using the same map. Finally, we combine the position estimator network, which provides the global location, and the multi-goal Q-function, which was trained in the maze and 2D simulator, to navigate the robot to different destinations in the 3D simulated environment.

We summarize our contributions as follows:

- Proposing a novel framework for robot navigation.
- Testing the scalability and localization performance of convolutional deep neural networks that learn to map camera images to positions.
- Proposing a multi-goal reinforcement learning framework to learn to navigate to several different goals at once.
- Transfer learning from maze to the 2D, and 3D physics simulator.
- Comparison of the results of the proposed multi-goal framework with traditional reinforcement learning.

In Section 2, we further investigate the state of the art in robotic localization and the advances in deep RL. In Section 3, we describe the different methods used in this paper for position estimation and multi-goal RL. In Section 4, we portray in detail the experiments done in which different types of position estimator neural networks are compared, and further elaborate on the environments that are used to carry out the tests. We continue the section by demonstrating the results for the multi-goal RL method. Finally, we discuss the benefits and shortcomings of our proposed method and conclude the paper in Section 5.

2. Previous work

Extensive research has been done on robot navigation, from indoor mobile robots [23], to drones [24], and vehicles [25]. Most of the research is focused on creating a map and localizing the robot in this map using a variety of sensors, such as 2D/3D LIDARs [24,26], cameras [27], or a combination of both [28]. Often these maps are created based on the notion of grid cells. These methods extract geometrical information from the scene and stitch them together by combining the robot motion model and the information that comes from the sensors using probabilistic approaches [29]. In the end, the robot has to be able to plan a path (e.g. $A*$, or Dijkstra's algorithm, [30]) and avoid static and dynamic obstacles to maintain safe and reliable trajectories. This is done mostly through different control algorithms which often, in a predictive manner, forward simulate the movement of the robot and check the sampled path versus a variety of criteria (e.g., proximity to obstacles, distance to global path, oscillation, etc.) [31,32]. The performance and scalability of these methods, however, is directly related to the precision of the sensors, and the available computational power. This is one of the reasons that most mobile robots or vehicles use dedicated 3D sensors such as laser range finders or time-of-flight cameras to map and navigate in the environment. This, however, comes with a price, the range finders are very expensive.

There have been attempts to solve this problem by using cheaper sensors such as cameras to map and navigate the environment(Visual SLAM). In these methods, either feature extraction and matching are used to find correspondences between multiple images using corner detectors [33,34], SIFT descriptor [35], ORB descriptor [36], or featureless approaches are used that generate a global map using direct image alignment, and probabilistic depth maps [37]. By applying geometric and motion constraints, these algorithms separate static and moving features from one another to localize the robot and build a map. A recent survey paper by Saputra et al. [38] gives a comprehensive view of these methods.

With the popularity of deep neural networks [39], different approaches were made to tackle the navigation problem using deep neural networks. Previously, we transferred the knowledge from a traditional map to a stacked denoising autoencoder (SDA) in which the robot used grid mapping for training data, and could localize its position using a camera after training in a small environment [21]. Bidoia et al. [40], used a semi-supervised approach to create a graph map using deep CNNs. QR codes were scattered in the environment while the robot randomly moved throughout the environment. Later, using these codes, several graph nodes were created which allowed the network to predict its location. Moving through the graph was done by remembering the geometrical distance between the nodes.

Wang et al. [41], proposed a deep visual odometry method using a recurrent convolutional neural network architecture that receives a pair of images in each forward pass and outputs the poses. In this architecture, the CNN learns to extract features from pairs of images while a long short-term memory (LSTM) block learns the motion model and movement in the environment. Zhou et al. [42], presented an unsupervised learning framework for monocular depth and camera motion estimation from unstructured video sequences. While the results are promising and have comparable performance to supervised methods, the current framework requires intrinsic camera parameters and has difficulties with dynamic scenes. In our approach, we evaluate how well our architecture can learn positions using only one image before applying additional recursion complexity. In addition, our proposed architecture tackles the kidnapped robot problem.

Researchers have also used end-to-end learning schemes to solve this problem. Kemkpa et al. [15] used RL to solve different sub-tasks in the game of Doom. Although this research lacks the constraints of a mobile robot due to an unconstrained movement in the virtual world, it shows how well the deep networks can condense information in the form of images and make decisions to reach certain goals. Kulhánek et al. [43] used a long short term memory (LSTM) as part of their CNN network in combination with a modified batched advantage actor–critic (A2C) algorithm [44] to solve end-to-end visual navigation. The LSTM part of the network was added to solve the partial observability of the navigation task. In addition, the CNN network predicted the depth map and image segmentation of the current observation and the goal image to enhance the training process and increase the generalizability of the network. Kahn et al. [18] developed a generalized computational graph using deep recurrent neural networks to navigate a remote control (RC) robot in a hallway. The focus of this research was mainly to train the local control mechanism of the robot to perform movements without hitting obstacles using a fixed longitudinal and a controllable angular velocity. The researchers showed that by pretraining their model in simulation, they can achieve faster convergence and better results in the real world.

There is one main issue regarding these end-to-end approaches, and that is the difficulty of extracting meaningful and human understandable data from the system. It would be very difficult to tell the RC robot to go to a certain place (e.g. next to the kitchen counter), or extract information from the system where it thinks it is. Therefore, the training procedure has a hit-miss characteristic. One may never know whether the system will converge, nor which problems are causing the divergence.

Our focus is to make the navigation process goal oriented and more explainable. We would like to know in which location the robot has difficulties reaching or localizing itself, and why. In addition, we would like to make the robot learn multiple objectives at the same time while exploring the space for the current objective. If the robot is traversing a home environment looking for the kitchen, it may gain knowledge about reaching the bedroom as well, or by using reverse logic, it can know which new places it has to explore.

There has been prominent research in the past years regarding the utilization of experiences and exploration, but the initial idea has been around for a long time [45]. The main drive to use the past experiences is to avoid spending a lot of time searching for different goals again and again, while avoiding that the neural networks forget the old experiences. From this perspective, any attempt to reuse experiences is an advantage. The more recent reuse of this idea combined with deep neural networks was done by Mnih et al. [46] which allowed multiple agents to share their experiences while a single server computes the gradients and sends the newest neural network function approximator back to the agents. As we mentioned before, neural networks as function approximators have problems with forgetting past experience. Therefore, in DQN [12] the agent uses a replay buffer as well. Every now and then, this buffer is sampled from, and the state–action pairs are used to train the system.

While using neural networks as function approximators, one can wonder how to create one network that remembers all the Q-values for different goals. The universal value function approximator [47] showed that this can be done by augmenting the feature input with the position of the goal. This way, the network will not have any problem remembering and assigning different Q-values to different goals.

Although this approach allows us to use one function approximator for multiple goals, it solely relies on the generalizability of the FA for providing correct Q-values for unseen goals. However, every action results in reaching a new state which can become a goal in the future. For example, if we are driving for the first time to the supermarket, we may also see the gas station on the way. Next time, if we want to go to the gas station, we do not have to search for it. This is the idea behind hindsight experience replay [48].

Veeriah et al. [49] continued this trend to apply multi-goal RL using an unsupervised mastery in scenarios where there are no apparent goals. For example, in most of the Atari games, the goal is to increase the reward intake, and therefore, no specific single goal can be set.

Our research focuses on the specific navigation problem with obstacles and environmental boundaries in place. The simulated robot should be able to keep track of certain fixed goals and be able to use the experiences to learn to navigate to a new goal. In addition, due to the two-stage training style of the proposed framework, our method does not have to use the universal value function approximator and includes goals in the input. Instead, look-up tables are used in our architecture in which each goal has a separate Q-function.

3. Methodology

In this section, we address in detail the methodology that we used to design and test the robot navigation pipeline. Fig. 1 shows the complete framework proposed in this paper. First, we elaborate on how we solve the exploration and path finding problem through our proposed multi-goal RL method, and then explain our position estimator network.

3.1. Multi-goal reinforcement learning

The first step of our algorithm is to learn the correlation between the estimated position of the robot and the robot's utilities of different actions in different states. In end-to-end deep RL, the system learns to optimize (state, action) pair Q-values using an image as input. This approach has several drawbacks. The most prominent is the required number of trials for a complex task such as navigation. In order to learn to navigate in the environment, one would need a deep network to be able to extract meaningful information from raw images. In the case of model-free methods such as Q-learning [50,51] and Sarsa [52, 53], exploration of possible new states adds to the difficulty of the task at hand. Parallelizing the experiences in deep RL and multi-objective approaches, however, has allowed to reduce this number significantly in applications such as robot manipulation [54]. Our proposed method focuses on the idea of sharing the experiences between multiple goals and using a goal selection technique that gives us the most useful information from exploration throughout the trials.

In robotic navigation, the state space has six dimensions with three position and three orientation axes. Most robots, apart from drones and robots that operate in uneven terrain navigate only in three dimensions (X, Y, and θ). One can define this state space as a discrete set of blocks or a continuous set of numbers. In practice, low level control of the robot base is done in a continuous space while the location estimation and general path planning use a discrete approach.

We use a discrete separation of locations with fixed resolution. While the selected action of the robot is decided through the multi-goal RL method, the movement of the robot between the cells is done using the dynamic window approach (DWA) [31]. We only use the X and Y dimensions in order to reduce the

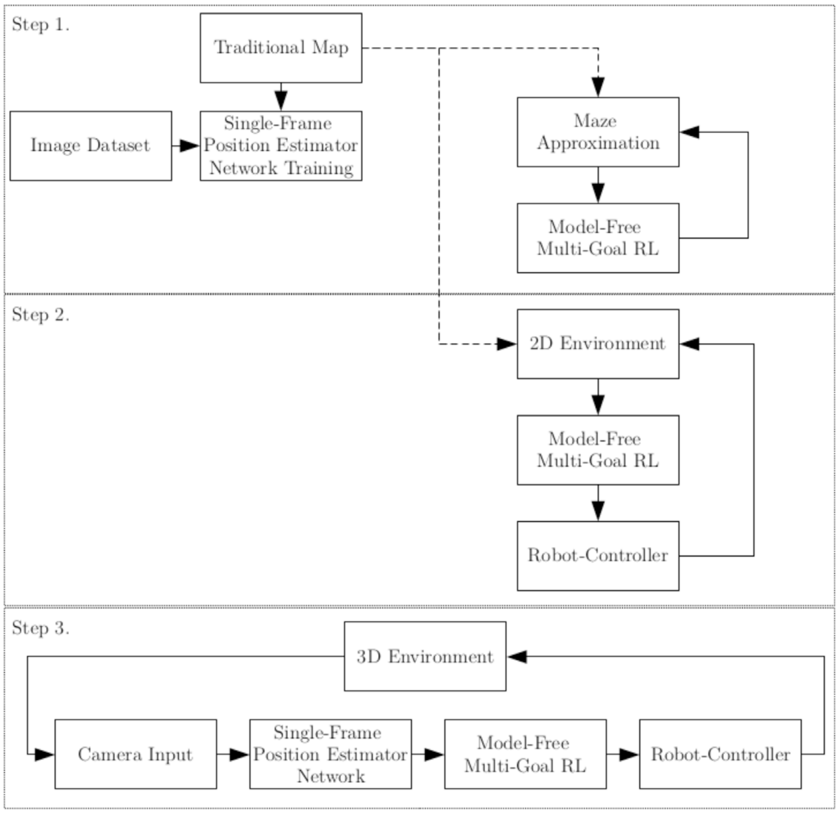


Fig. 1. The complete proposed framework in a nutshell. In the first step, the position estimator is trained with positions extracted from a traditional mapping method while the model-free RL agent learns the environment using an approximated maze extracted from the map. In the second step, the RL agent continues to learn the effects of the robot controller on its actions in a 2D simulated environment. Finally, in the 3D environment, the agent uses position estimation from the proposed CNN and continues learning.

size of the state space. Our localization method, as described in Section 3.2, accurately estimates the orientation of the robot which allows the local controller to reach the desired orientation. Fig. 2 shows a discrete and down-sampled maze extracted from a higher resolution 2D map.

We consider that the agent's knowledge of the environment is incomplete and requires exploration. Therefore, we select a model-free RL approach. In the model-free domain, we can either select on-policy or off-policy RL. On-policy RL algorithms, such as Sarsa are suitable when we want to evaluate the policy that also generated the current outcome of the agent's actions. Off-policy RL methods such as Q-learning, on the other hand, allow to train policies that did not generate the current data. This is the main reason why RL methods that share experiences with one another should always use off-policy RL. Google's DeepMind DQN [12], and a promising robotic manipulation system [54] are examples of such approaches. Our proposed approach uses a model-free multi-goal off-policy RL algorithm. For example, in Fig. 2(b), the agent can use the experiences during exploration of goal 1 and use it later to reach goal 4.

3.1.1. Q-learning

Q-learning requires a way to store Q-values for each state–action pair (s_t, a_t). These values estimate how good the given action is to reach the goal. The Q-values are updated based on rewards that are given to the agent when it reaches a goal or rewards that are given during the navigation phase. Eq. (1) shows the general update rule for the Q-learning method [50].

$$Q_{k+1}(s_t, a_t) = Q_k(s_t, a_t) + \alpha\left(r_t + \gamma \max_a Q_k(s_{t+1}, a) - Q_k(s_t, a_t)\right) \quad (1)$$

After each action, the agent moves and receives a reward r_t. The Q-value of the previously selected action is updated based on the reward of the current state, the performed action, and the prospect of the next state based on its highest Q-value estimation. The γ parameter is the discount factor which determines the importance of future rewards versus immediate rewards and α is the learning rate. The function that approximates the Q values from each state–action pair can either be linear or non-linear. For our problem, since the position estimation network provides the location of the robot, the state space of the RL can be a grid, hence we use a look-up table. For problems in which the state space and the connection to actions are not trivial, a more complex non-linear approximator is required. Considering the grid state space, the available actions to perform in each state are up, down, left, and right movements. We discard the rotation dimension to reduce the size of the state space. This is again possible because the position estimation network gives global coordinates, and before the robot needs to go to the next position, it can rotate to the desired angle.

Due to exploration with Q-learning and the required time to move a robot, the convergence to the optimal solution takes a considerable amount of time. We speed up the convergence by using previous experiences of the agent in the environment. We accumulate all the state, action, reward, and next state experience tuples (s_t, a_t, r_t, s_{t+1}) in a replay buffer. After a certain number of online updates, we re-calculate target values. Then, we shuffle the buffer to remove correlations between data points that are next to each other in time. We take a mini batch from the dataset and train the Q-function. This process is continued until all the past experiences in the replay buffer are used.

Using the above approach, we can initialize a separate lookup table for each goal. After the Q-function for the first goal is

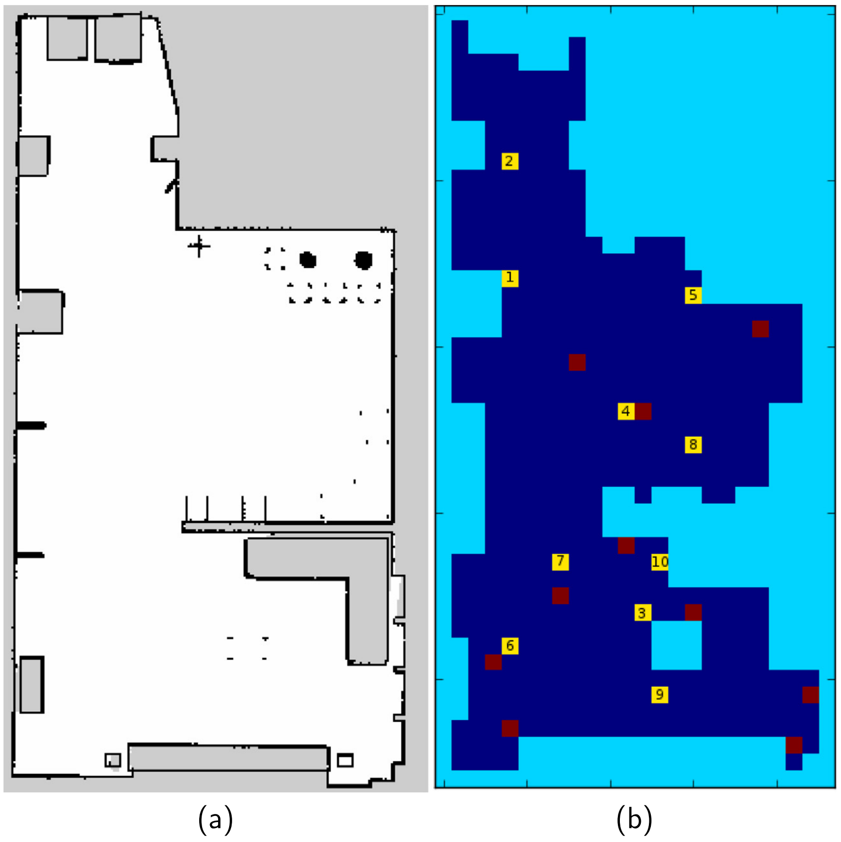


Fig. 2. Fig. 2(a) is the 2D generated map of the big apartment using the Rao-Blackwellized grid mapping method with 5 cm resolution. Fig. 2(b) shows the down-sampled approximated maze of the map with 40 cm resolution, which needs to be learned by the RL algorithm. The red squares show the initial positions, and the yellow squares show the goals for the RL experiments. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

optimized, we go to the next one until all the goals can be found. The downside is that for each Q-function, we must reset the values to the initial ones since these Q-values are optimized for one goal.

3.1.2. *Multi-goal Q-learning with experience replay*

Using the Q-learning method allows us to train multiple policies at once. This means that while the agent is searching for one goal, it can learn about reaching other goals at the same time. To this end, the first step is to update the Q-values for all the goal Q-functions at the same time. In the worst-case scenario, none of the other goals will be traversed during the exploration of the current goal. However, the Q-functions for all the other goals can learn from all the negative results due to obstacles in the state space. In the best case, some of the other goals will be traversed during the exploration phase of the first goal, and therefore, the robot can learn to optimize its path toward them. With multi-goal Q-learning, multiple Q-functions are trained each time step which all have their own reward function that emits a reward r_t^g (Eq. (2)):

$$Q_{k+1}^g(s_t,a_t) = Q_k^g(s_t,a_t) + \alpha\left(r_t^g + \gamma \max_a Q_k^g(s_{t+1},a) - Q_k^g(s_t,a_t)\right) \tag{2}$$

The selection of which policy to use or on which goal to train is an important factor and determines the total time of convergence. We use the notion of temporal difference (TD) errors to measure how many times a secondary goal was traversed. The TD error is the squared difference between the new target of a Q-value of a state–action pair, and its previous value (Eq. (3)).

$$TD_{err}^g = (r_t^g + \gamma \max_a Q^g(s_{t+1},a) - Q^g(s_t,a_t))^2 \tag{3}$$

When the TD error is high for some Q-function, it shows that the specified goal location has not been fully learned by the policy. When the TD value is low, it means that the robot has visited this state several times, and the Q-values are converging to an optimal value. We set up a TD-error matrix for each of the Q-functions with a high initial value. After reaching and meeting the convergence criteria for the current goal, we select the new goal with the highest TD error. This makes sure that the robot traverses the environment in an efficient way and increases the chances of reaching other goals on the way.

3.1.3. *Robot movements and transfer learning*

We discussed our approach toward solving a maze navigation problem using the multi-goal approach. However, learning to solve a maze is straightforward compared to a robot moving in an environment. A robot has a footprint which is a projected polygon of the 3D shape of the robot on the 2D floor plan. For easier calculations, all the concave footprints are often changed to a convex version. The robot also has an axis of rotation which is often on the center of the robot. Movements of the robot are given through a velocity vector. In a 2D navigation scenario this velocity consists of a positional speed in X, Y, and a rotational speed in θ. If the robot has a differential drive base, the positional speed in X direction, and rotational speed in θ can be used, while an omni-directional robot can freely move in all directions. The robot base program will translate these velocities to wheel speeds. In order to have a safe navigation system, we need to apply the speeds given to the robot and change the position and orientation of the footprint. Using obstacle detection sensors, such as infrared, sonar, or laser, the robot can identify its position in relation to these obstacles and processes the movement commands in such a way that the edges of its footprint polygon never touch the obstacles on the way. We use the dynamic window approach (DWA) method to control the robot locally. In this method, the robot has a global path to the sub-goal, the location of the sub-goal, and a cost-map that shows the surrounding obstacles and the current footprint of the robot. The global path in our case is very short because the actions just move the robot from one cell to another neighboring cell (sub-goal). The location of the set sub-goal is therefore one step away from the robot in either X or Y direction. The cost-map is a matrix with a selectable resolution. Based on the sensor, if a certain cell is occupied, it will be marked. Multiple sensors can mark and clear cells in the cost maps. DWA will sample velocities in the available dimensions based on the type of robot. Then, it will forward simulate the robot movements for a short amount of time based on these samples. In the end, a trajectory will be selected which causes no collision, and keeps the robot close to the designated path with a safe distance from obstacles. This accomplishes having the robot move to neighboring cells.

In order to connect the maze navigation results to a more realistic setting, we must adapt the continuous movement of the robot in the 3D simulated environment, so it matches the cell centers in the maze. To this end, our first step is to move from the approximated maze to a 2D simulator. The 2D simulator has a simpler physical model and does not require any rendering, allowing us to speed up the simulation by a factor of ten in comparison to the 3D simulator. In the 2D simulator, we use the map that was made from the 3D simulation.

The actions in the simulator do not necessarily have the same results as in the maze due to the higher resolution of the map, and the nature of the DWA method. Actions can fail, especially in corners, and the RL method needs to be able to cope with it. We use the trained Q-function from the maze, and continue learning in the 2D simulator. It is notable that the robot position in the 2D simulator is always correct. Therefore, the robot is exactly at the location that it thinks it is. However, in the 3D simulator this is not the case, because the position is inferred from a camera image and this brings us to our next challenge, dealing with approximation errors.

Each robot has an odometry error model. There are 5 different noise types that affect the odometry of the robot. Three of which are the direct deviations in X, Y, and θ dimension. The other two are co-dependent errors across these dimensions. When the robot moves forward, it can slightly rotate, or when the robot rotates, the position of the robot can also change due to wheel skid. These 5 error types make a fully functional navigation based on odometry impossible. This problem is solved by estimating the global position of the robot using a variety of sensors and combining it with the odometry model through filtering. Non-linear Kalman filters (e.g. extended [55], and Unscented Kalman Filter [56]), and particle filters [57] are the mostly used methods. The idea is to predict the new position of the robot after a series of motion commands (Eq. (4)) and update this belief with the estimation of the position from any sensor (Eq. (5)).

$$\hat{bel}(x_t) = \int p(x_t|x_{t-1}, u_t)bel(x_{t-1})dx_{t-1} \tag{4}$$

$$bel(x_t) = \eta p(z_t|x_t)\hat{bel}(x_t) \tag{5}$$

Where x_t is the state vector (e.g. X, Y, and θ), u_t is the given motion command, and z_t is the sensor reading of the robot at time t. η is a normalization factor since the multiplications of the beliefs of the robot at time t by the probability $p(z_t|x_t)$ may integrate to a value unequal to one. By knowing the noise model of the robot movement, and the probability of being in the current location by the position estimator, one can keep a good track of the location of the robot for small movement steps, such as used in our methods.

For the final step of transfer learning, the trained Q-functions in the 2D simulator for all the goals will be further trained in a 3D simulator. In the 3D simulator, the position estimator networks give the estimated position of the robot, while the RL method must deal with wrong location estimations. Since the position estimation of the neural networks is accurate and the distance between the cells are small, the odometry error can be ignored. Therefore, we leave out Eq. (4), and use the global estimation from the network to determine the grid-cell of the robot's position. We are curious to see whether the RL method is able to handle the measurement errors of the network without the use of motion prediction and learns to navigate the robot to all goals along the shortest path.

3.2. Position-estimator networks

In our previous research, we have shown that SDA can map images to positions in a small 3D simulated environment with adequate generalizability [21]. However, the question is how well the network scales with the size of the environment. SDAs have shown promising results since their introduction in academia in 2009 [58]. Due to the fully connected architecture of the network, the number of weights to train increases significantly when the input vector is large. Each additional layer will increase the number of weights significantly which requires a large amount of training data in order to learn a good position estimator. In addition, by flattening the two-dimensional image into a one-dimensional vector, we lose topological information of the pixels which is crucial in the process of learning. Perhaps, these are the main reasons that SDAs were outdated quickly with the advent of faster learning and more accurate CNNs. The CNN was first proposed by LeCun et al. [59] but they became widely popular after Alex Krizhevsky et al. [60] bested the ImageNet Large Scale Visual Recognition Competition (ILSVRC) in 2012. Combining the processing power of GPUs with the new ReLU activation function, dropout [61], and contrast normalization of the images in the CNN architecture allowed them to perform much better than all other approaches in the object recognition field. Since then, researchers have been working to find ways to optimize the training and design of the networks – creating deeper and deeper networks – such as the Google Inception architecture [62,63] while others focused on broadening the application of CNNs in different fields such as object detection [64] and semantic segmentation of the scene [65]. By changing parts of the CNN architecture, we use its potential for scalability and generalizability, and compare the results of CNNs and SDAs to select the best network to estimate the position of the robot in the environment.

3.2.1. Convolutional neural networks

When we talk about CNN architectures, it is important to see what type of problem we are trying to solve. CNNs are used extensively in the area of object recognition, where one would like to recognize the main object that can be displayed in different parts of the image. Therefore, one would like the networks to have scale and translation invariance. This is where the pooling layers play an important role. The pooling layers not only reduce the size of the feature map of a hidden unit, they also carry either the maximum or the average response over the input window. When applied in multiple layers, it works as a scale, and shift invariant feature for the network. This property, although useful for object recognition, is detrimental for precise location

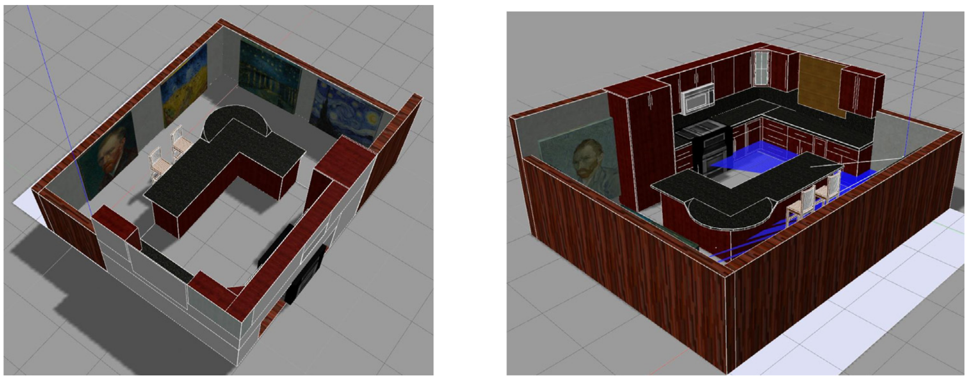


Fig. 3. Small kitchen room of size 7 × 5 meters.

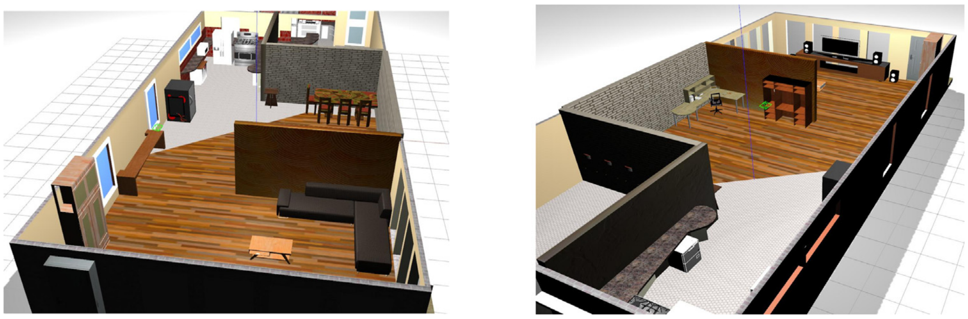


Fig. 4. Big apartment of size 14 × 9 meters.

estimation [40]. We would like the network to observe exactly where an edge was in the image and with what size, since this is important for localizing the agent. Therefore, we do not use pooling layers in our architecture. This has a drawback as well, the input resolution should be limited, because the number of parameters to optimize are greatly higher than that of networks with pooling layers.

We experimented with presence and absence of pooling layers, different depths of the network, and single or multiple kernel sizes in the same layer which we will explain in detail in the next section.

4. Experiments and results

In this section, we describe the experiments for the position estimator networks, and the reinforcement learning algorithms in the maze, 2D, and 3D simulation. For the environments, we use one maze simulator, the 2D Stage simulator [66], and the 3D Gazebo physics simulator [67]. We used the robot operating system (ROS) framework [68] to develop our software. The environments for the experiments are shown in Figs. 3 and 4. We used OpenCV, Theano, and Tensorflow to extract images and train the SDA and CNNs [69–71]. We created a simulated robot with which we carried out the experiments, which can be seen in Fig. 5.

4.1. Data gathering

We first extract the map of the 3D simulated environment (see Figs. 3 & 4) using the Rao-Blackwellized grid mapping approach [72]. In this method, the map-building process starts from the first laser reading. When the robot moves, the position of the robot is updated by using scan matching of the new laser reading with the old one. If this scan matching fails, the odometry model of the robot is used to update the location and merge the laser reading from the existing map with the previous map. This process led to the map shown in Fig. 2.

We divide the map in 25-centimeter cells. The center of each cell is a location that the robot should traverse in order to gather the training and test data. In order to do this in an automatic manner, we use Dijkstra's shortest-path algorithm to plan a path to the center of the cell [30]. We consider the footprint of the robot in order to make sure that it fits in the destination cell. The unknown locations, occupied locations, and locations where the robot does not fit are ignored. Finally, we send the robot to each of these positions, and record images and corresponding positions given by the grid mapping method while rotating the camera 1 degree at a time in each position.

After data gathering, we create a test, validation, and training set. We split the images from each location into thirty-six sections (using 10-degree steps). For each section, half of the positions and rotations together with the images are assigned to the training set. For the remaining five images, two go to the validation set, and three to the test set. Therefore, 50% of the data is used for training, 20% for validation, and 30% for testing. The total data set size for the small kitchen and the big apartment are 24,000 and 195,000 images, respectively.

4.2. Deep networks and localization

We performed a set of experiments to find the best architecture for the position estimator network. As mentioned in Section 3, we use SDAs and CNNs to estimate the position and orientation of the robot. We want to find out which of these networks has the best global localization performance in the environment and how well they scale with the size of the room.

4.2.1. Stacked denoising autoencoder architecture

For the SDAs, we selected a number of architectures that can be seen in Table 1. The number of SDA layers, the number of neurons in each layer, and the size of the position estimator multi-layer perceptron (MLP) on top of the SDA layers are the

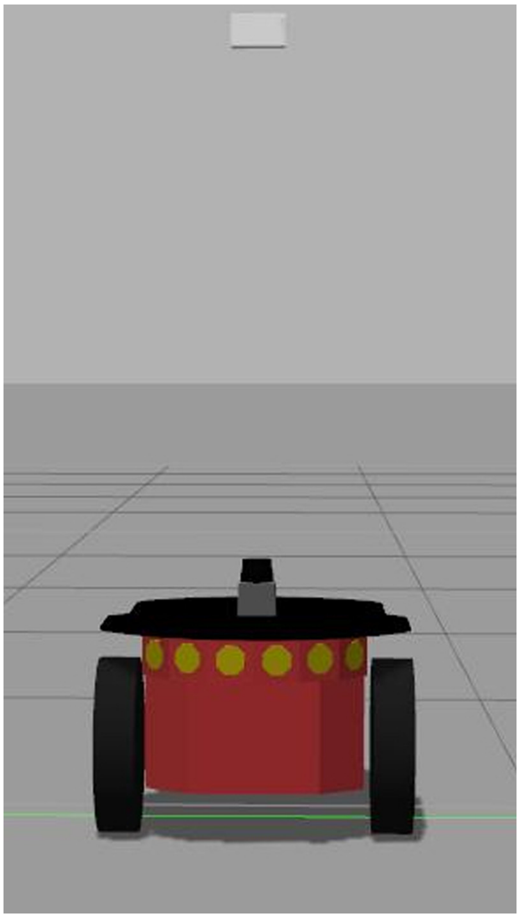


Fig. 5. The differential-drive Pioneer robot that was used for the data gathering and RL experiments. To speed up the RL experiments, we attached an omnidirectional plugin to the simulated robot to minimize the required time to move from a cell to another. During the experiments, the range finder values are cut off to half a meter range and are used as a simple infrared collision detector sensor.

subject of our tests. We train each network for 10,000 epochs, and save the validation results. The network with the lowest validation error for each architecture is used for testing. In our previous research, we did experiments in a room very similar to the small kitchen in Fig. 3. However, there are two main differences in the setup of the localization experiment of this paper in comparison to the previous one. In our previous paper, we had a large pre-training data set of 150,000 unlabeled images. However, since we want to compare the results of SDAs to CNNs, we decided to use the pre-training phase on the smaller labeled data set. We estimate that the performance of the network will drop due to the importance of the unsupervised phase of SDA training. In addition, the environment used in our previous research had a large number of texture-less walls, which forced us to reduce the data gathering locations to positions with some textures in the field of view of the camera. In this paper, however, we added paintings to the walls to allow for complete data gathering of the environment, since the agent has access to meaningful information of the whole environment to localize itself.

The input size of the images for the SDA network are HSV color images of size 36×36 which led to the best results in our previous paper. Due to full connectivity of the layers, increasing the image size results in very high testing errors.

Table 1
SDA architectures for experiments.

Layers	Units per layer	Corruption per layer
1	[4000]	[0.2, 0.2]
1	[1500]	[0.2, 0.2]
2	[4000, 4000]	[0.2, 0.2]
2	[1500, 1500]	[0.2, 0.2]
3	[4000, 4000, 4000]	[0.2, 0.2, 0.2]
3	[1500, 1500, 1500]	[0.2, 0.2, 0.2]

4.2.2. Convolutional neural networks

The same procedure applies to training the CNNs. Table 2 shows the CNN architectures that were trained and evaluated. For the CNNs, presence and absence of pooling layers, the depth of the network, and Convolution type (Inception vs. Normal) were selected as the main criteria for testing different architectures. We use strides in networks without pooling layers to reduce the number of parameters. We compare the use of RGB and HSV images to train and validate the CNNs on the date from the small kitchen, and will train the best performing network on the data from the big apartment. The input size for the CNNs are images of size $84 \times 84 \times 3$. The detailed design of the third architecture from Table 1 is shown in Fig. 6.

4.3. Reinforcement learning

We first compare the normal and multi-goal Q-learning approach on random mazes. The mazes are grids of size 10×10 surrounded by walls. The randomization of the maze is as follows. The $(0, 0)$ cell is empty, and we call it the starting point. For each column, there is a thirty percent chance to have obstacles inside. When the column has obstacles, a random number will generate how long the obstacles will be. Another random number selects the position of the obstacles in the column. The length of the obstacle cannot exceed more than half of the maze. This procedure is repeated for all columns, and we also repeat it for all the rows. After this operation, there may exist obstacles with hollow cells inside. From the starting point, we perform connected component analysis, to find out all the cells that are reachable from this point. We fill the non-blocked remaining cells inside with obstacles. Then, we generate $X = 10$ random goals, the distance of these goals to the starting point $(0, 0)$ should be more than 7 steps. If the random agent cannot find these goals after several iterations, we reject the maze and start over. Finally, we randomize $X - 1$ more initial positions to create in total, 10 mazes with 10 goals and 10 initial positions.

Reaching the goal gives the agent a reward of 100 while hitting blocked cells gives a reward of -2. All other actions receive a fixed punishment of -0.1. We initialize the Q-values to 80 to encourage exploration. The learning rate for the update of Q-values is 1.0 for the maze, 0.1 for the 2D simulator, and 0.8 for the Gazebo 3D simulator. The maze learning rate is set to 1.0 because the environment and actions are deterministic. For the 2D simulator, a smaller value is selected to avoid large changes in Q-values during training due to the stochasticity of the robot control. In the 3D simulator, however, we increased the learning rate because the simulation speed is slower and due to the position estimation errors the Q-values should be updated faster.

The agent uses Boltzmann exploration for the selection of actions for all the maze scenarios which can be seen in Eq. (6).

$$P(a_i|s) = \frac{\exp(\frac{Q(s_t, a_i)}{T})}{\sum_{j=1}^{N} \exp(\frac{Q(s_t, a_j)}{T})} \tag{6}$$

T is the temperature, and for each action in state s_t, we compute action probabilities for the N actions. When the temperature

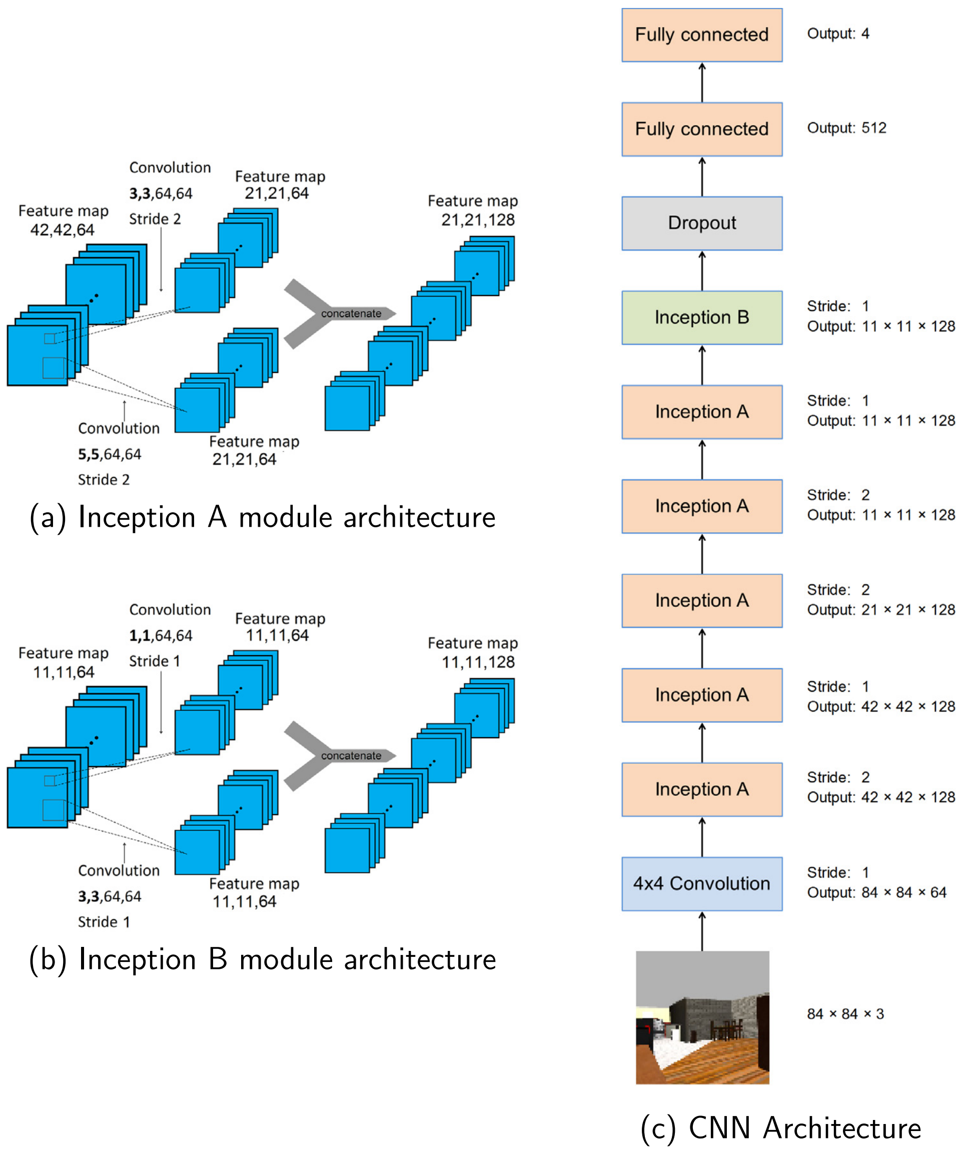

Feature map 42,42,64
Convolution 3,3,64,64 Stride 2
Feature map 21,21,64
concatenate
Feature map 21,21,128
Convolution 5,5,64,64 Stride 2
Feature map 21,21,64
(a) Inception A module architecture
Feature map 11,11,64
Convolution 1,1,64,64 Stride 1
Feature map 11,11,64
concatenate
Feature map 11,11,128
Convolution 3,3,64,64 Stride 1
Feature map 11,11,64
(b) Inception B module architecture
Fully connected Output: 4
Fully connected Output: 512
Dropout
Inception B Stride: 1 Output: 11 × 11 × 128
Inception A Stride: 1 Output: 11 × 11 × 128
Inception A Stride: 2 Output: 11 × 11 × 128
Inception A Stride: 2 Output: 21 × 21 × 128
Inception A Stride: 1 Output: 42 × 42 × 128
Inception A Stride: 2 Output: 42 × 42 × 128
4x4 Convolution Stride: 1 Output: 84 × 84 × 64
84 × 84 × 3
(c) CNN Architecture

Fig. 6. The nine-layer position estimation convolutional neural network architecture with inception. The input to the network is an 84×84 image; The output is estimated as X, Y, $\sin\theta$, $\cos\theta$. Images (A) and (B) show two different Inception modules and (C) shows the complete CNN architecture.

Table 2
CNN Architectures for Experiments.

Layers	Convolution type	Kernel size	Dense layer size	Optimizer
7	Convolution – no pooling	5×5	512	Adam
7	Inception convolution – pooling	5×5, 3×3, and 1×1	512	Adam
9	Convolution – no pooling	5×5	512	Adam
9	Inception convolution – no pooling	5×5, 3×3, and 1×1	512	Adam
9	Inception convolution – pooling	5×5, 3×3, and 1×1	512	Adam

T is high, actions will be assigned similar probabilities. When T drops, higher Q-values will have a higher probability to be selected. We initialize the temperature T to 2 with a decay value of 0.998. After each trial, the temperature is multiplied by the decay value. For the 2D and the 3D environment, however, we use the $\epsilon-$greedy method to reduce exploration because the Q-functions have already been trained and too much additional exploration costs more time in the simulators. The ϵ value is set to 0.1.

For the maze experiments, we consider a goal learned when the success ratio at reaching the goal is 100 percent for fifty consecutive trials from each starting position, and the average difference in number of steps to reach the goal is smaller than 10.

Table 3
The experimental results for position and orientation approximations with the stacked denoising autoencoders.

Small Kitchen - HSV						
Layer size	No. layers	Corruption	Position error(m)		Angular error (degree)	
			Mean	Std	Mean	Std
1500	1	20%	0.218	0.213	12.6	14.0
4000	1		0.217	0.207	12.8	13.6
1500	2		0.204	0.221	11.6	13.7
4000	2		0.193	0.205	11.3	12.8
1500	3		0.198	0.214	11.6	13.9
4000	**3**		**0.182**	**0.205**	**10.7**	**12.5**
Big apartment - HSV						
4000	**3**	20%	**0.560**	**0.869**	**28.8**	**48.9**

The results on the mazes allow us to reliably measure the performance of the multi-goal RL approach versus the sequential goal selection method. To assess the temporal difference goal selection, we perform a separate set of maze experiments with different convergence criteria. In these experiments, the required success ratio of 100 percent is only necessary for fifteen consecutive trials. We also remove the average difference in the number of steps to show the difference between the two approaches better. However, the final goal is to use our method to navigate a robot in a realistic environment. Therefore, we perform experiments in the 2D and 3D simulator as well. For these environments, we use the same big apartment map as during the data gathering phase. We have to, however, downscale the map and convert it to a maze with the correct cell sizes. We start the map approximation by selecting the top left corner of the map images as $(0, 0)$. We divide the width and height of the map with the required cell resolution to extract rows and columns. Any cell with an occupied map pixel will be considered as an obstacle, and the rest will be free cells. We apply the same closing method as we perform for the random mazes.

4.3.1. Big apartment and transfer learning

For the big apartment in Fig. 4, the map size has a width of 17 and length of 9 m. The cell resolution is 40 centimeters, and therefore the approximated maze size is 47×24. We run the RL tests on the approximated maze, 2D simulator, and 3D simulator. The location of the robot in the maze is just a cell index. For the 2D simulator, we use the ground truth pose of the robot to determine the location of the robot in the cells. In the 3D simulator, the position estimator network outputs the predicted position of the robot. These experiments are repeated 10 times.

In order to test the performance of transfer learning, we train the agent in the maze using RL, and then use the trained Q-function in the 2D simulator to cope with the possible problems of local navigation. Finally, the trained Q-function in the 2D simulator is used for the 3D simulation where the position estimator neural network predicts the location of the robot, and the RL method deals with the errors in the predictions. We expect to see a sharp decrease in the required time to converge to reliably finding the goals. These experiments are done on the big apartment (Fig. 4).

4.4. Experiment results

In this section, we discuss the results of the position estimation networks and the RL experiments.

4.4.1. Position estimator results

Table 3 shows the results of the SDA experiments. The best results for the small kitchen room are for the network with 3 layers and 4000 hidden units with 20 percent corruption of the input data. We used the same network for the big apartment as well. The high position and angular errors in the big simulated room show that these networks cannot scale well with the size of the environment. Fig. 7(a) shows a better overview of the error throughout the environment for the big apartment. For each location, the robot estimates the position for each angular rotation of the camera. Then, we average this error for each cell. The green color shows the minimum mean cell error value from the network and the shades toward red mean higher errors. Positions in the center of the map have a better localization in comparison to locations close to walls and corners. The main reason is that the robot has a wider view and can see a larger part of the room when it is in the center. This gives the network more information to distinguish its location robustly. In the corners and next to the walls on the other hand, the estimations are poor. Plain looking walls or objects do not have much information, and therefore the estimations suffer from this lack of information.

Table 4
The experimental results for the convolutional neural networks.

Small kitchen RGB				
Network Type	Position error (m)		Angular error (degree)	
	Mean	Std	Mean	Std
7 Layer	0.066	0.087	4.9	1.96
9 Layer	0.069	0.101	6.1	2.59
Inception 9 layer	0.056	0.079	4.9	2.07
Small kitchen HSV				
Network type	Position error (m)		Angular error (degree)	
	Mean	Std	Mean	Std
7 Layer	0.065	0.086	6.1	2.46
9 Layer	0.069	0.101	6.2	2.60
Inception 9 Layer	0.056	0.080	5.1	2.10
Big apartment RGB				
Network type	Position error (m)		Angular error (degree)	
	Mean	Std	Mean	Std
7 Layer	0.156	0.190	3.9	2.67
9 Layer	0.113	0.125	3.1	2.21
Inception 9 layer	**0.076**	**0.067**	**3.8**	**1.63**
Inception pool 7 layer	0.324	0.087	5.6	1.94
Inception pool 9 layer	0.276	0.174	4.5	2.51

Table 4 shows the results of the position estimation for the CNNs. The best results are achieved by the Inception architecture in both rooms. In addition, the results for the RGB color space is slightly superior to that of the HSV space. The best 9-layer network with the RGB color space has 0.056 cm position error and 4.9 degrees angular error with 0.079 cm and 20.7 degrees standard deviation. The use of the RGB color space works slightly better with the CNNs which is confirmed by other researches as well [73]. Having more layers allows the system to encapsulate the environment better, while using the inception architecture in each level allows the network to use coarse and fine information

Table 5
The experiment with 10 different random mazes. The values in the table are the total number of steps required to solve the maze from each initial location to each goal.

Random mazes											
	Maze number										
RL Method	1	2	3	4	5	6	7	8	9	10	**Average**
Normal RL	184k	209k	204k	185k	255k	212k	186k	172k	203k	201k	**201k**
Multi-Goal RL	76k	82k	90k	73k	10k	91k	83k	75k	78k	93k	**84k**

Table 6
Comparison between temporal difference error based goal selection and random goal selection with 10 different random mazes with different starting temperatures and initial Q-values.

Random mazes - multi-goal RL											
T = 1.0, Q = 0	Maze number										
Goal selection	1	2	3	4	5	6	7	8	9	10	**Average**
Random	23k	31k	28k	27k	46k	29k	32k	52k	38k	29k	**34k**
TD-error	23k	30k	23k	26k	43k	28k	31k	41k	40k	28k	**31k**
T = 2.0, Q = 80	Maze number										
Goal selection	1	2	3	4	5	6	7	8	9	10	**Average**
Random	26k	39k	31k	27k	26k	33k	26k	50k	26k	31k	**32k**
TD-error	33k	41k	29k	27k	28k	31k	21k	47k	28k	34k	**32k**

at the same time. The smaller 3 × 3 kernels only use the surrounding pixel information while the bigger 5 × 5 kernels also include a larger neighborhood which adds more global information. The result of the inception network on the bigger room is also interesting. The positional and angular error has remained similar while the size of the room is doubled. This shows that the CNNs can scale with the size and type of the environment. The results on the bigger room also clearly show that pooling layers should not be used for precise localization purposes.

The results from Tables 3 and 4 clearly show that not only the CNN network performs better in a small environment, but it also scales much better when the environment is larger. For this reason, the 9 layer inception network is used to estimate the positions for the RL method in the 3D simulator.

Fig. 7(b) shows the distribution of the errors in the big environment. The same procedure for the SDA heat map is repeated here. The higher error values are strictly for the positions that are closer to walls. In addition, we also observe that the network gives higher position estimation errors when the scene has a large depth. Since the network uses only a single image, it will be quite hard for it to correctly estimate the depth, and the appearances of the objects that are further away do not change a lot when the camera moves toward them.

4.4.2. Reinforcement learning results

Table 5 shows the results of experiments with the normal and multi-goal approach for random mazes. The data show the average required number of steps to learn to navigate to all the goals from all the initial positions. The multi-goal approach on average has a 239% faster convergence time. Table 6 shows the comparison between random and temporal difference based goal selection with different starting temperatures for the Boltzmann exploration. Note that we relaxed the convergence criteria for this experiment. When the starting temperature and initial Q-values are high, the agents are encouraged to explore the complete environment, and most of the new goals do not need additional training. However, with a reduced starting temperature and initial Q-values, the agents will not encounter all the goals where smart goal selection can positively impact the convergence time. With this setting, The TD-based goal selection performs on average 9 percent better than random goal selection.

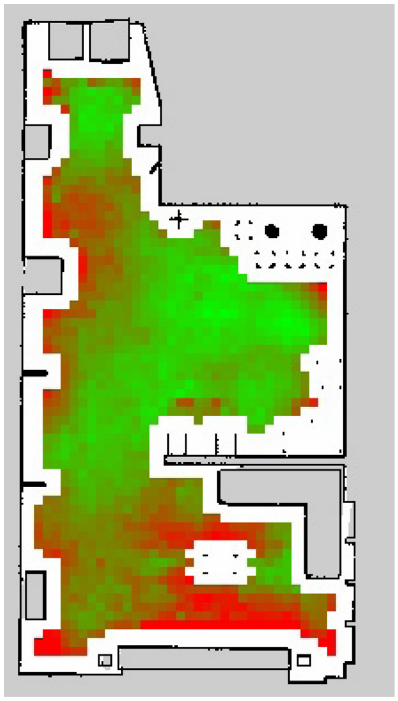
Tables 7 and 8 show the results of the big apartment maze for the single and multi-goal approach. The multi-goal approach

Table 7
The average number of trials to reach the goals from all initial positions in the approximated maze of the big simulated apartment using the single and the multi-goal approach. The multi-goal approach requires 40 percent less trials for convergence.

Goal No.	Single goal		Multi goal	
	Mean	Std	Mean	Std
1	112.1	50.8	112.6	50.6
2	123.2	62.9	107.7	48.6
3	89.6	16.4	50.0	0.0
4	87.2	28.2	50.1	0.1
5	107.2	49.8	50.4	0.3
6	99.7	22.6	51.3	0.9
7	87.3	17.8	50.7	0.5
8	88.4	31.6	51.0	0.7
9	96.0	21.9	52.4	1.6
10	98.9	32.2	50.1	0.1
Average	98.9	32.2	**62.6**	**10.3**

requires 40% less trials and requires half the number of actions as well. The large difference between the random mazes and the approximated maze can be explained by the position of the goals. In random mazes, it is possible that two goals are very close to each other which benefits the multi-goal approach. In our approximated maze, the goals and initial positions are scattered fairly, hence the improvement is less substantial. Figs. 8(a) and 8(b) depict the differences between the single and the multi-goal approach more clearly. The single goal network has to learn to navigate to goals from scratch and therefore the number of trials and actions to learn the optimal path is higher compared to the multi-goal approach. For the multi-goal approach, however, all the goals that were close to the exploration range of the first goal were learned almost immediately, while due to the experience sharing between the Q-functions the agent learned to navigate to all the goals faster. The TD-based goal selection obtained a similar overall performance as the random goal selection in this scenario. While the average number of trials is similar, the TD-based error has a significantly lower standard deviation, making it a suitable candidate for the rest of the multi-goal experiments as well.

Table 9 shows the performance of the multi-goal method in the 2D and 3D simulator. For the 2D simulation, the robot could reach all the goals. The required number of actions to reach the goals were slightly higher than that of the maze simulations. The stochasticity of the ϵ-greedy method, and non-deterministic


(a) Position errors for the best SDA network. Green color represents minimum errors of 0.43 meter per location, and red color has the maximum error of 1.6 meter per location.

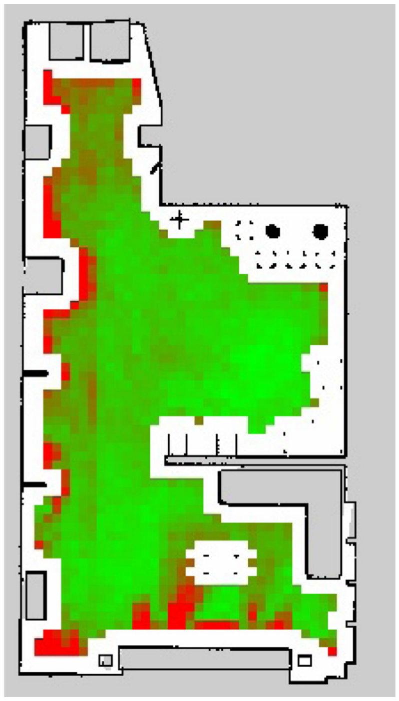

(b) Position errors for the best CNN network. Green color represents minimum errors of 0.12 meter per location, and red color has the maximum error of 0.56 meter per location.

Fig. 7. The error heat map for the position of the robot in the big simulated room. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Table 8
The average number of actions to reach the goals from all initial positions in the approximated maze of the big simulated room using the single and the multi-goal approach. The multi-goal's required number of actions is two times less compared to the single goal method. This shows that goals were quite often reached during exploration to other goals. The temporal difference goal selection has a slightly higher mean but a significantly lower standard deviation in this specific maze.

Goal No.	Single goal		Multi goal - TD		Multi goal - random	
	Mean	Std	Mean	Std	Mean	Std
1	115.0	73.4	117.5	2.8	120.6	20.7
2	103.5	59.7	54.7	5.0	28.3	13.7
3	55.2	15.7	26.6	0.1	19.8	7.6
4	68.0	37.9	17.1	3.2	21.9	7.8
5	79.7	49.0	17.4	2.4	17.7	3.4
6	65.5	23.7	20.8	5.4	24.3	11.4
7	56.0	17.9	16.8	4.1	18.9	7.6
8	75.0	44.2	18.1	2.9	22.7	5.8
9	53.4	18.3	20.3	4.2	24.6	11.2
10	59.0	16.3	17.7	5.9	23.0	9.8
Average	73.0	35.6	32.7	3.6	32.2	9.9

Table 9
The average number of actions (after convergence) to reach the goals from all initial positions in the 2D and 3D simulator using the multi-goal approach. The 2D simulator used ground truth positions for the robot localization while the 3D simulation used the CNN to estimate the positions.

Goal No.	2D Simulation		3D Simulation	
	Mean	Std	Mean	Std
1	35.3	7.3	43.6	10.3
2	42.4	7.8	73.4	16.3
3	19.7	2.6	28.0	4.6
4	25.2	6.4	46.7	15.7
5	33.7	7.9	44.5	11.1
6	23.8	4.0	28.9	5.0
7	22.1	4.2	26.6	5.3
8	27.6	6.3	43.0	12.5
9	19.2	4.4	27.7	9.4
10	21.2	2.1	29.0	3.7
Average	**27.0**	5.30	**39.1**	9.4

behavior of the controller of the robot can explain the higher number of actions. The local navigation system considers the robot footprint and must locally navigate from cell to cell. This can introduce problems when the robot is in tight corners due to insufficient sampling of the velocity space by the DWA approach which may result in failed or incomplete actions.

The 3D simulation results show a higher number of actions to reach the goals (also in Fig. 8(d)). In the 2D simulator, the location of the robot is always correct. However, in the Gazebo 3D simulation, the inception CNN was used to estimate the position of the robot. Therefore, due to erroneous estimations, cell selection will be incorrect at some positions in the room. As can be seen in Fig. 9, it is possible that the actual location of the robot is slightly different than the estimated position. This problem becomes larger if the robot faces plain looking walls, or textureless surfaces, where the output of the CNN is usually the average of all the different positions with the same input. However, the ϵ-greedy method allows the robot to get out of these situations but with a cost of a higher number of actions.

Table 10 shows the results of training the agents directly in the 3D simulator without any transfer learning. From the large standard deviations and higher means compared to Table 9, and

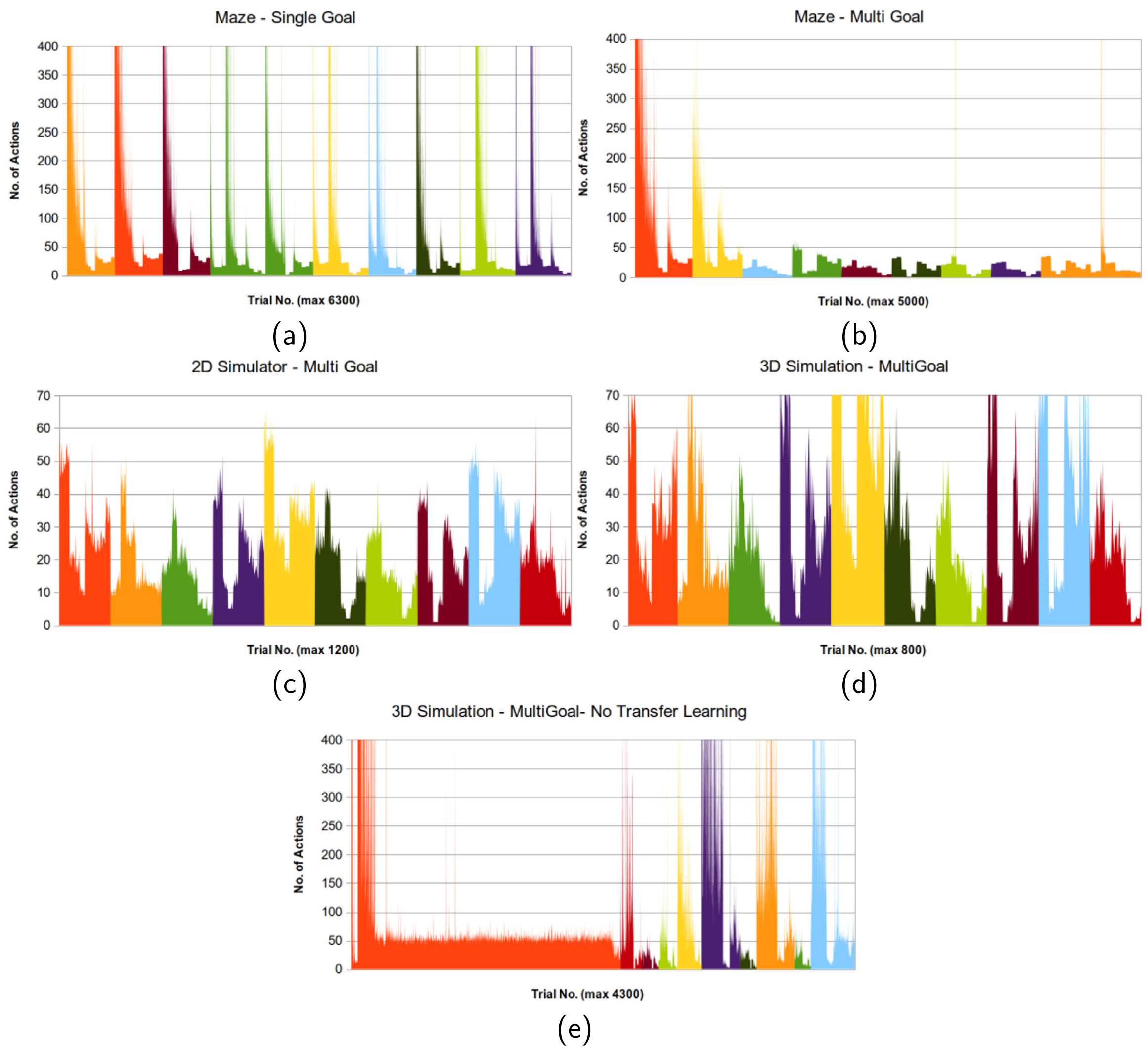


Fig. 8. The number of actions versus the number of trials for the multi-goal and the single goal RL method in the approximated maze, and for the multi-goal approach in the 2D and 3D simulation of the big apartment. Fig. 8(a) shows the result for the single goal approach on the approximated maze. Fig. 8(b) shows the multi-goal results for the approximated maze. Note that the Q-functions have almost converged after the second goal. Fig. 8(c) shows the 2D simulator results with the multi-goal approach. Fig. 8(d) shows the results of the 3D simulation with the multi-goal approach. Note that the number of actions to reach some of the goals are considerably higher than for the 2D simulation due to the errors of the CNN position estimation. Fig. 8(e) shows the results of the multi-goal approach without transfer learning in the 3D simulation.

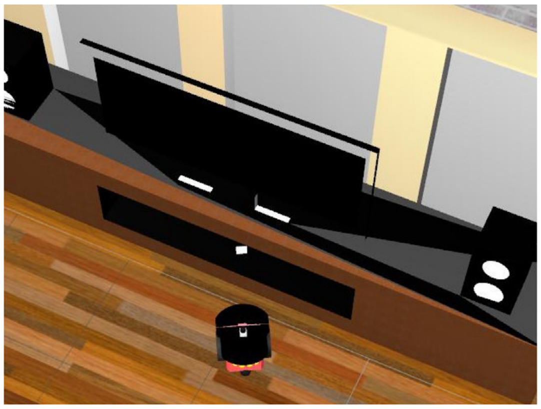
the number of agents that failed to learn all the goals, it is evident that the two-stage transfer learning significantly speeds up the learning process.

5. Discussion

In this paper, we introduced a two-stage visual navigation framework using deep convolutional neural networks and multi-goal reinforcement learning. Our goal was to design a system that is robust, and resilient to localization errors. Therefore, we first investigated whether a deep convolutional neural network is capable of learning position information based on an image and whether it can generalize well for locations that are outside of the training set. We performed several comparisons between different CNN architectures and our previously proposed SDA architectures on a small and a large 3D simulated environment. The proposed inception CNN architecture performed best with 0.076 m position error and 3.8 degrees angular error in the large room and 0.056 m position error and 4.9 degrees angular error in the small room. CNNs proved superior to SDA in both accuracy

Table 10
The average number of trials and actions (after convergence) to reach the goals from all initial positions in the 3D simulator using the multi-goal approach without transfer learning. The right most column is the number of agents that could not learn the respective goals after 14 thousand trials. These experiments took approximately a month to complete.

Goal No.	Trials		Actions		Not reached
	Mean	Std	Mean	Std	No. of agents
1	6849.6	5154.0	74.8	8.4	0
2	376.6	253.2	87.9	40.3	3
3	181.1	104.5	46.9	63.5	1
4	141.8	1.8	24.0	3.8	0
5	432.0	60.0	144.4	45.2	4
6	142.4	5.6	21.3	6.4	1
7	153.1	15.8	33.8	5.8	1
8	473.1	181.5	189.7	31.6	1
9	3141.5	5343.4	131.1	60.9	3
10	160.4	23.9	372.7	26.1	0
Average	1205.2	1114.4	80.1	29.2	


(a)

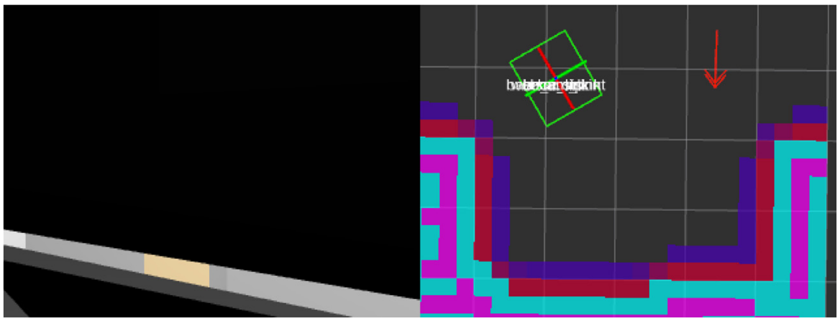

(b) (c)

Fig. 9. An example view of erroneous position estimation from the CNN. Fig. 9(a) shows one of the positions in the big apartment where the CNN estimation is incorrect due to the limited view of the robot. Fig. 9(b) shows the robot's camera view. Fig. 9(c) shows the actual position of the robot versus the estimated pose. The estimated pose is shown by the red arrow, while the robot footprint is the green rectangle. The other colored pixels are the inflated cost maps with an increased range to better visualize the situation. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

and the ability to scale with a more considerable amount of data. However, CNNs, similar to other methods that rely on visual input, suffer greatly from lack of texture, and therefore their output values should be used in combination with the motion model of the robot. Based on the results, we can argue that the proposed CNN has good potential in robot localization. We then tested our proposed multi-goal RL method to see whether a combination of dynamic goal selection, experience sharing, and transfer learning can reduce learning time. We first tested the multi-goal and single-goal approach in 10 different random mazes with 10 random goals and 10 random initial positions. Our multi-goal approach learned to solve all navigation tasks around, on average, 240 percent faster than the traditional method. We then focused on robot RL experiments and tested how transfer learning can reduce learning time. We compared RL agents on the approximated maze of the large simulated room using the multi-goal and the single-goal approach. The multi-goal agent was able to learn to navigate to all the goals using half of the total number of actions required by the single-goal method. We transferred the trained multi-goal agent to the 2D simulation of the large room in which we evaluated the effects of the stochasticity of robot base movements. After learning to navigate to all goals in the 2D simulator, we transferred the agent to the 3D simulated environment. We showed that while the agent could guide the robot to the goals, the number of actions for convergence was slightly higher in the 3D simulator in comparison to the 2D simulator. The higher number of actions was caused by the position estimation errors of the CNN and the stochasticity of the robot moving platform. We can conclude that CNNs can learn and transform images into reliable coordinates for localization and that a multi-goal RL agent can achieve faster convergence by sharing the experiences of one goal with all the others using transfer learning, and smart exploration to reduce the required number of actions to navigate to different goals.

For future work, the first improvement step could be to learn continuous control of the robot. It is possible to use a neural network with the position, angle, velocity, and acceleration information of the robot as inputs, and longitudinal and angular velocity as outputs. This would simplify the method and since no discretization would be needed, the generalization performance would increase, which should lead to less training time.

We also want to research how we can train the position estimator network without needing so much accurate position data. It might be possible to use the locally accurate robot odometry model for this purpose. This would allow to scale up our system to learn to navigate in even larger environments.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] H. Shteingart, Y. Loewenstein, Reinforcement learning and human behavior, Curr. Opin. Neurobiol. 25 (2014) 93–98, http://dx.doi.org/10.1016/j.conb.2013.12.004, Theoretical and computational neuroscience, URL http://www.sciencedirect.com/science/article/pii/S0959438813002286.
[2] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, Vol. 1, MIT Press, Cambridge, 1998.
[3] I.H. Witten, An adaptive optimal controller for discrete-time Markov environments, Inf. Control 34 (4) (1977) 286–295.

[4] D.P. Bertsekas, J.N. Tsitsiklis, Neuro-dynamic programming: An overview, in: Decision and Control, 1995, Proceedings of the 34th IEEE Conference on, Vol. 1, IEEE, 1995, pp. 560–564.

[5] R.A. Howard, Dynamic Programming and Markov Processes, Wiley for the Massachusetts Institute of Technology, 1964.

[6] R. Bellman, A Markovian decision process, J. Math. Mech. (1957) 679–684.

[7] M.A. Wiering, Explorations in Efficient Reinforcement Learning (Ph.D. thesis), University of Amsterdam, 1999.

[8] S.I. Reynolds, Reinforcement Learning with Exploration (Ph.D. thesis), University of Birmingham, 2002.

[9] L. Busoniu, R. Babuska, B.D. Schutter, D. Ernst, Reinforcement Learning and Dynamic Programming Using Function Approximators, first ed., CRC Press Inc., USA, 2010.

[10] N. Kohl, P. Stone, Policy gradient reinforcement learning for fast quadrupedal locomotion, in: Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on, Vol. 3, 2004, pp. 2619–2624, http://dx.doi.org/10.1109/ROBOT.2004.1307456.

[11] A. Shantia, E. Begue, M. Wiering, Connectionist reinforcement learning for intelligent unit micro management in starcraft, in: Neural Networks (IJCNN), the 2011 International Joint Conference on, IEEE, 2011, pp. 1794–1801.

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, et al., Human-level control through deep reinforcement learning, Nature 518 (7540) (2015) 529.

[13] G. Tesauro, Temporal difference learning and TD-Gammon, Commun. ACM 38 (3) (1995) 58–68.

[14] D. Silver, A. Huang, C.J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al., Mastering the game of Go with deep neural networks and tree search, Nature 529 (7587) (2016) 484–489.

[15] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, W. Jakowski, ViZDoom: A Doom-based AI research platform for visual reinforcement learning, in: 2016 IEEE Conference on Computational Intelligence and Games, CIG, 2016, pp. 1–8, http://dx.doi.org/10.1109/CIG.2016.7860433.

[16] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A.S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser, J. Quan, S. Gaffney, S. Petersen, K. Simonyan, T. Schaul, H. van Hasselt, D. Silver, T. Lillicrap, K. Calderone, P. Keet, A. Brunasso, D. Lawrence, A. Ekermo, J. Repp, R. Tsing, Starcraft II: A new challenge for reinforcement learning, 2017, arXiv:1708.04782.

[17] S. Gu, E. Holly, T. Lillicrap, S. Levine, Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates, in: Robotics and Automation (ICRA), 2017 IEEE International Conference on, IEEE, 2017, pp. 3389–3396.

[18] G. Kahn, A. Villaflor, B. Ding, P. Abbeel, S. Levine, Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation, 2017, arXiv preprint arXiv:1709.10489.

[19] G. Grisetti, C. Stachniss, W. Burgard, Improved techniques for grid mapping with Rao–Blackwellized particle filters, IEEE Trans. Robot. 23 (1) (2007) 34–46.

[20] G. Grisettiyz, C. Stachniss, W. Burgard, Improving grid-based SLAM with Rao–Blackwellized particle filters by adaptive proposals and selective resampling, in: Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on, IEEE, 2005, pp. 2432–2437.

[21] A. Shantia, R. Timmers, L. Schomaker, M. Wiering, Indoor localization by denoising autoencoders and semi-supervised learning in 3D simulated environment, in: International Joint Conference on Neural Networks, IJCNN, IEEE, 2015, pp. 1–7.

[22] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86 (11) (1998) 2278–2324.

[23] S. Thrun, Robotic mapping: A survey, Exploring Artif. Intell. New Millennium 1 (2002) 1–35.

[24] P.-J. Bristeau, F. Callou, D. Vissiere, N. Petit, The navigation and control technology inside the AR.Drone micro UAV, IFAC Proc. Vol. 44 (1) (2011) 1477–1484.

[25] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J.Z. Kolter, D. Langer, O. Pink, V. Pratt, M. Sokolsky, G. Stanek, D. Stavens, A. Teichman, M. Werling, S. Thrun, Towards fully autonomous driving: Systems and algorithms, in: Intelligent Vehicles Symposium (IV), 2011 IEEE, 2011, pp. 163–168, http://dx.doi.org/10.1109/IVS.2011.5940562.

[26] A. Hornung, K.M. Wurm, M. Bennewitz, C. Stachniss, W. Burgard, Octomap: An efficient probabilistic 3D mapping framework based on octrees, Auton. Robots 34 (3) (2013) 189–206.

[27] F. Bonin-Font, A. Ortiz, G. Oliver, Visual navigation for mobile robots: A survey, J. Intell. Robot. Syst. 53 (3) (2008) 263–296.

[28] T. Whelan, H. Johannsson, M. Kaess, J.J. Leonard, J. McDonald, Robust real-time visual odometry for dense RGB-D mapping, in: 2013 IEEE International Conference on Robotics and Automation, 2013, pp. 5724–5731, http://dx.doi.org/10.1109/ICRA.2013.6631400.

[29] S. Thrun, Probabilistic robotics, Commun. ACM 45 (3) (2002) 52–57.

[30] T.J. Misa, P.L. Frana, An interview with Edsger W. Dijkstra, Commun. ACM 53 (8) (2010) 41–47, http://dx.doi.org/10.1145/1787234.1787249.

[31] D. Fox, W. Burgard, S. Thrun, The dynamic window approach to collision avoidance, Robot. Autom. Mag. IEEE 4 (1) (1997) 23–33, http://dx.doi.org/10.1109/100.580977.

[32] B.P. Gerkey, K. Konolige, Planning and control in unstructured terrain, in: Workshop on Path Planning on Costmaps, Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 2008.

[33] P.H. Torr, A. Zisserman, Feature based methods for structure and motion estimation, in: International Workshop on Vision Algorithms, Springer, 1999, pp. 278–294.

[34] D. Nister, 0. Naroditsky and J. Bergen. Visual odometry, in: Proc. CVPR, 2004, pp. 652–659.

[35] J.L. Schonberger, J.-M. Frahm, Structure-from-motion revisited, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 4104–4113.

[36] R. Mur-Artal, J.D. Tardós, Orb-slam2: An open-source SLAM system for monocular, stereo, and RGB-D cameras, IEEE Trans. Robot. 33 (5) (2017) 1255–1262.

[37] J. Engel, J. Stückler, D. Cremers, Large-scale direct SLAM with stereo cameras, in: Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on, IEEE, 2015, pp. 1935–1942.

[38] M.R.U. Saputra, A. Markham, N. Trigoni, Visual SLAM and structure from motion in dynamic environments: A survey, ACM Comput. Surv. 51 (2) (2018) 1–36.

[39] J. Schmidhuber, Deep learning in neural networks, Neural Netw. 61 (C) (2015) 85–117, http://dx.doi.org/10.1016/j.neunet.2014.09.003.

[40] F. Bidoia, M. Sabatelli, A. Shantia, M.A. Wiering, L. Schomaker, A deep convolutional neural network for location recognition and geometry based information, in: Proceedings of the 7th International Conference on Pattern Recognition Applications and Methods, ICPRAM 2018, Funchal, Madeira - Portugal, 2018, pp. 27–36, http://dx.doi.org/10.5220/0006542200270036.

[41] S. Wang, R. Clark, H. Wen, N. Trigoni, Deepvo: Towards end-to-end visual odometry with deep recurrent convolutional neural networks, in: 2017 IEEE International Conference on Robotics and Automation (ICRA), 2017, pp. 2043–2050.

[42] T. Zhou, M. Brown, N. Snavely, D.G. Lowe, Unsupervised learning of depth and ego-motion from video, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 1851–1858.

[43] J. Kulhanek, E. Derner, T. de Bruin, R. Babuska, Vision-based navigation using deep reinforcement learning, in: 2019 European Conference on Mobile Robots, ECMR, IEEE, 2019, pp. 1–8.

[44] Y. Wu, E. Mansimov, R.B. Grosse, S. Liao, J. Ba, Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation, in: Advances in Neural Information Processing Systems, 2017, pp. 5279–5288.

[45] L.-J. Lin, Reinforcement Learning for Robots Using Neural Networks (Ph.D. thesis), Carnegie Mellon University, Pittsburgh, 1993.

[46] V. Mnih, A.P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, K. Kavukcuoglu, Asynchronous methods for deep reinforcement learning, in: International Conference on Machine Learning, 2016, pp. 1928–1937.

[47] T. Schaul, D. Horgan, K. Gregor, D. Silver, Universal value function approximators, in: International Conference on Machine Learning, 2015, pp. 1312–1320.

[48] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O.P. Abbeel, W. Zaremba, Hindsight experience replay, in: Advances in Neural Information Processing Systems, 2017, pp. 5048–5058.

[49] V. Veeriah, J. Oh, S. Singh, Many-goals reinforcement learning, 2018, arXiv preprint arXiv:1806.09605.

[50] C.J.C.H. Watkins, Learning from Delayed Rewards (Ph.D. thesis), King's College, Cambridge, 1989.

[51] P. Dayan, C. Watkins, Q-learning, Mach. Learn. 8 (3) (1992) 279–292.

[52] G.A. Rummery, M. Niranjan, On-Line Q-Learning Using Connectionist Systems, 37, University of Cambridge, Department of Engineering Cambridge, England, 1994.

[53] R.S. Sutton, Generalization in reinforcement learning: Successful examples using sparse coarse coding, in: Advances in Neural Information Processing Systems, 1996, pp. 1038–1044.

[54] S. Levine, P. Pastor, A. Krizhevsky, D. Quillen, Learning hand-eye coordination for robotic grasping with large-scale data collection, in: International Symposium on Experimental Robotics, Springer, 2016, pp. 173–184.

[55] A. Jazwinski, Stochastic Process and Filtering Theory, Academic Press, 1970, A subsidiary of Harcourt Brace Jovanovich Publishers.
[56] E.A. Wan, R. Van Der Merwe, The unscented Kalman filter for nonlinear estimation, in: Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No. 00EX373), IEEE, 2000, pp. 153–158.
[57] N.J. Gordon, D.J. Salmond, A.F. Smith, Novel approach to nonlinear/non-Gaussian Bayesian state estimation, in: IEE Proceedings F (Radar and Signal Processing), (2) IET, 1993, pp. 107–113.
[58] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.A. Manzagol, Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion, J. Mach. Learn. Res. 11 (2) (2011) 3371–3408.
[59] Y. Lecun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, L. Jackel, H. Baird, Constrained neural network for unconstrained handwritten digit recognition, in: C. Suen (Ed.), Frontiers in Handwriting Recognition, Montreal, 1990, CENPARMI, Concordia University, 1990.
[60] A. Krizhevsky, I. Sutskever, G.E. Hinton, ImageNet classification with deep convolutional neural networks, in: F. Pereira, C.J.C. Burges, L. Bottou, K.Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 25, 2012, pp. 1097–1105.
[61] G.E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Improving neural networks by preventing co-adaptation of feature detectors, CoRR abs/1207.0580 (2012) URL arXiv:1207.0580.
[62] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: Computer Vision and Pattern Recognition, CVPR, 2015, http://arxiv.org/abs/1409.4842.
[63] C. Szegedy, S. Ioffe, V. Vanhoucke, A. Alemi, Inception-v4, inception-ResNet and the impact of residual connections on learning, in: Computer Vision and Pattern Recognition, CVPR, 2016, http://arxiv.org/abs/1602.07261.
[64] S. Ren, K. He, R. Girshick, J. Sun, Faster R-CNN: Towards real-time object detection with region proposal networks, in: Advances in Neural Information Processing Systems, 2015, pp. 91–99.
[65] J. Long, E. Shelhamer, T. Darrell, Fully convolutional networks for semantic segmentation, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 3431–3440.
[66] R. Vaughan, Massively multi-robot simulation in stage, Swarm Intell. 2 (2–4) (2008) 189–208.
[67] N.P. Koenig, A. Howard, Design and use paradigms for Gazebo, an open-source multi-robot simulator, in: IROS, IEEE, 2004, pp. 2149–2154.
[68] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A.Y. Ng, ROS: An open-source robot operating system, in: ICRA Workshop on Open Source Software, Vol. 3, Kobe, Japan, 2009, p. 5.
[69] Itseez, Open source computer vision library, 2015, https://github.com/itseez/opencv.
[70] Theano Development Team, Theano: A Python framework for fast computation of mathematical expressions, arXiv e-prints abs/1605.02688 (2016) URL http://arxiv.org/abs/1605.02688.
[71] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-scale machine learning on heterogeneous systems, 2015, Software available from tensorflow.org, http://tensorflow.org/.
[72] G. Grisetti, C. Stachniss, W. Burgard, Improved techniques for grid mapping with Rao–Blackwellized particle filters, IEEE Trans. Robot. 23 (1) (2007) 34–46.
[73] R. Sachin, V. Sowmya, D. Govind, K.P. Soman, Dependency of various color and intensity planes on CNN based image classification, in: S.M. Thampi, S. Krishnan, J.M. Corchado Rodriguez, S. Das, M. Wozniak, D. Al-Jumeily (Eds.), Advances in Signal Processing and Intelligent Recognition Systems, Springer International Publishing, Cham, 2018, pp. 167–177.

Amirhossein Shantia is a Ph.D. candidate in artificial intelligence at university of Groningen. He completed his bachelor in computer software engineering at Azad university of Tehran in 2009. He then continued his masters in computer science at university of Groningen in 2011. His research is focused on deep learning, neural networks, robotics, reinforcement learning and computer vision. He is currently working as a product owner in VW Car.Software Organization in the field of artificial intelligence.

Rik Timmers is an education and research Technician at the faculty of science and engineering of the University of Groningen. He helps preparing lab practicals and students who are working on their Bachelor and Master thesis. His main interests are Robotic Manipulation, Vision, and Reinforcement Learning.

Yiebo Chong has finished his Bachelor in Artificial Intelligence at the University of Groningen and is currently an Artificial Intelligence Master student there.

Cornelis Geert Kuiper is currently a master student in Artificial Intelligence following the Computational Intelligence and Robotics track at the University of Groningen. Cornelis followed and finished his Bachelor degree in Artificial Intelligence at the University of Groningen between 2014–2018.

Francesco Bidoia received an Engineering Degree in Industrial and Automation Engineering from Bologna University, Italy in 2014. He then consecuted the research master in Artificial Intelligence at the University of Groningen, Netherlands in 2017. He is currently working at Ascent Robotics, a Tokyo based start-up, working on self driving vehicles. His work focuses on full-stack navigation systems, for autonomous driving up to Level 4.

Lambert Schomaker (19-2-1957) is professor in artificial intelligence at the University of Groningen since 2001. He is known for research in simulation and recognition of handwriting, writer identification, style-based document dating and other studies in pattern recognition, machine learning and robotics. He has (co)authored over 200 publications and was involved in the organization of many conferences in handwriting recognition and document analysis. In recent years he and his team have worked on the Monk system: an interactively trainable search engine and e-Science service for historical manuscripts. The availability of up to thousands of training images for single classes of complex patterns has brought pattern recognition and machine learning into the ballpark of big data. Other recent work is in the area of robotics and industrial maintenance, in the EU ECSEL project Mantis. In 2015, he became co-chair of the Data Science and Systems Complexity center at the Faculty of Science and Engineering at the University of Groningen. In 2017, he joined the CogniGron center for cognitive systems and materials in a large-scale seven-year project in neuromorphic computing. He is a member of the IAPR and senior member of IEEE.

Dr. Marco Wiering is an assistant professor in the department of artificial intelligence from the University of Groningen, the Netherlands. He performed his Ph.D. research in the research institute IDSIA in Switzerland and graduated in 1999 on the topic of reinforcement learning. Before going to the University of Groningen, he worked as an assistant professor at Utrecht University. Dr. Wiering has co-authored more than 170 conference or journal papers and has supervised or is supervising 12 Ph.D. students and more than 110 Master student graduation projects. His main research topics are reinforcement learning, deep learning, neural networks, robotics, computer vision, game playing, time-series prediction and optimization.