Robotics: Science and Systems 2019
Freiburg im Breisgau, June 22-26, 2019

Value Iteration Networks on Multiple Levels of Abstraction

Daniel Schleich, Tobias Klamt, and Sven Behnke
Rheinische Friedrich-Wilhelms-Universität Bonn, Autonomous Intelligent Systems, Bonn, Germany
Email: {schleich@ais.uni-bonn.de, klamt@ais.uni-bonn.de, behnke@cs.uni-bonn.de}

***Abstract*—Learning-based methods are promising to plan robot motion without performing extensive search, which is needed by many non-learning approaches. Recently, Value Iteration Networks (VINs) received much interest since—in contrast to standard CNN-based architectures—they learn goal-directed behaviors which generalize well to unseen domains. However, VINs are restricted to small and low-dimensional domains, limiting their applicability to real-world planning problems.**

To address this issue, we propose to extend VINs to representations with multiple levels of abstraction. While the vicinity of the robot is represented in sufficient detail, the representation gets spatially coarser with increasing distance from the robot. The information loss caused by the decreasing resolution is compensated by increasing the number of features representing a cell. We show that our approach is capable of solving significantly larger 2D grid world planning tasks than the original VIN implementation. In contrast to a multiresolution coarse-to-fine VIN implementation which does not employ additional descriptive features, our approach is capable of solving challenging environments, which demonstrates that the proposed method learns to encode useful information in the additional features. As an application for solving real-world planning tasks, we successfully employ our method to plan omnidirectional driving for a search-and-rescue robot in cluttered terrain.

I. Introduction

While search-based and sampling-based methods are well investigated for motion planning [8, 17, 10], they tend to perform extensive, iterative searches for complex high-dimensional tasks. We hypothesize that this issue might be addressed by a higher level of scene understanding.

In other domains, especially in perceptual contexts, hierarchical convolutional neural networks (CNNs) are highly successful, because they learn increasingly abstract representations of their input by decreasing resolution and increasing the number of feature maps [15, 22]. A number of works applied CNNs to robot motion planning in recent years. This is promising since CNNs, which can be parallelized efficiently on e.g., GPUs, enable planning without extensive search. Standard CNN architectures have been used to map system state observations directly to actions [19, 4]. However, those approaches have difficulties to understand the goal-directed behavior of planning and to generalize to unseen domains.

This issue is addressed by, e.g., Value Iteration Networks (VINs) [25] or Universal Planning Networks (UPNs) [24]. Instead of following a strict feed-forward approach, values iterate multiple times in an inner loop to be propagated through the representation. Those methods show promising results in terms of goal-directed behavior and generalization to unseen domains. However, they have only been applied to small, low-dimensional problems. Planning in larger state spaces requires more complex network designs and significantly more training data which becomes at some point infeasible on currently available hardware. Thus, it is challenging to apply them to most real-world planning problems.

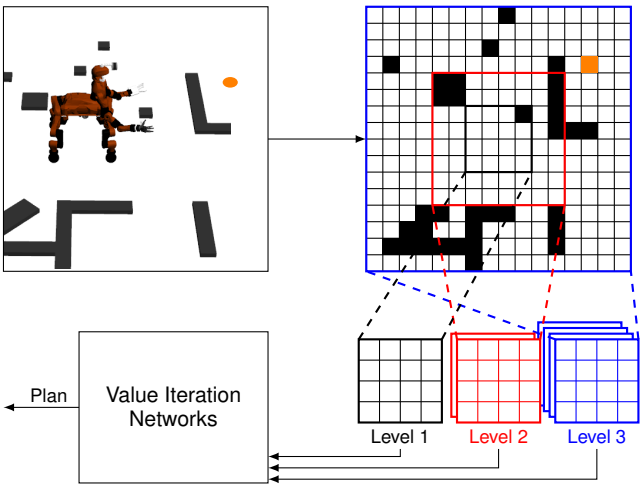

Fig. 1. The general idea of VINs on multiple levels of abstraction (AVINs).

A well-established idea to handle large state spaces is abstraction [16, 12]. An abstract representation describes neighboring states in a spatially/temporally coarser resolution while enriching the representation with additional features.

We propose a method to combine multiple environment representations with increasing level of abstraction with VINs to obtain a learning-based planner which is capable of handling more complex tasks (AVINs) (Fig. 1). With increasing distance from the robot, the level of abstraction increases. While the spatial resolution decreases with an increasing level of abstraction, the number of cells is constant for all levels which results in larger covered areas for more abstract maps. In addition, an increasing level of abstraction comes along with an increasing number of descriptive features for each cell.

Experiments show that our proposed AVINs outperform VINs in 2D grid worlds in their original implementation. While the result quality of AVINs is comparable or even better, they are capable of planning for environments which are up to 16 times larger while the memory requirement significantly decreases. In comparison to Hierarchical VINs (HVINs) [25], which employ multiresolution representations in coarse-to-fine

planning without the introduction of additional features, we obtain a better result quality with lower memory requirements. We especially can show that AVINs learn to encode useful information in their abstract representations since the performance on challenging environments is considerably better in comparison to HVINs. As a demonstration of the applicability to challenging real-world problems, we apply AVINs to plan omnidirectional driving locomotion for a search-and-rescue robot while considering its individual configuration of ground contact areas (which we refer to as the robot footprint).

II. Related Work

Most planning problems can be described as a Markov Decision Process (MDP) which consists of state and action spaces, state transition probabilities for the actions, and reward expectations for each transition [3]. The goal is to find a policy which results in high long-term rewards. One common algorithm to find such an optimal policy is Value Iteration (VI). By applying the Bellman equation [2] multiple times, it calculates the expected long-term reward (value) for each state. The optimal policy is obtained by greedily choosing the action based on the value of possible successor states.

While CNNs are well investigated for tasks such as image classification [15] and robot perception [22], their application to motion planning arose in recent years. Traditional CNN architectures have been used to learn policies and directly derive actions from state observations. Levine et al. [19] and Bojarski et al. [4], for example, trained CNNs to map raw images to robot motor torques for real-world manipulation tasks and autonomous car steering, respectively. Although the results of these applications are impressive, such approaches have poor capabilities to plan long-term goal directed behavior and generalization to unseen domains is also an issue.

In 2016, Tamar et al. [25] proposed VINs. An explicit planning module approximates the VI algorithm by rewriting the application of Bellman equations (which we refer to as Bellman update) as a CNN. Since this planning module is fully differentiable, standard backpropagation can be used to learn the parameters of the model, like a suitable reward function or state transition probabilities. The embedded planning operation enables VINs to generalize well to unseen environments and understand the desired goal-directed behavior. However, VINs do not scale well to larger map sizes and higher-dimensional state spaces since the number of required Bellman updates depends on the path length and larger state spaces require considerably more training data, longer training times, and have large memory requirements. Hence, evaluation was limited to small 2D grid worlds. In the appendix of [25], HVINs were proposed to reduce the number of necessary Bellman updates. Value iteration is first performed on a down-sampled copy of the input map to generate rough state-value estimates, which are up-sampled and used as initialization for another value iteration module working on the full resolution. This model can be extended to multiple hierarchical levels. However, the information loss through down-sampling is not compensated. Furthermore, all levels operate on the whole environment size resulting in only slightly decreasing memory requirements.

VINs have been applied in other domains. Niu et al. [21], proposed Generalized Value Iteration Networks which work on arbitrary irregular graph structures and can be applied to real world data like street maps. Karkus et al. [9] proposed QMDP-nets which handle partially observable environments and express VI through a CNN. Gupta et al. [7] propose a Cognitive Mapper and Planner to plan actions from first person views in unknown environments. They combine a neural network processing first person images to generate a latent representation map of the environment with a hierarchical planning module based on VINs.

UPNs by Srinivas et al. [24] learn useful latent state representations from images of the current scene and the desired goal scene. They infer motion trajectories by performing gradient descent planning and iterating over action sequences in the learned internal representations. Considered environments may have more than two dimensions but are rather small. The gradient descent planner is very time consuming and hinders scaling to larger environments. Impressively, UPNs are able to generalize to modified robot morphologies.

In other domains, abstraction is an established method to handle large state spaces. Abstract states unify multiple detailed states. This can be realized through coarser resolutions or lower-dimensional representations while the loss of information is compensated by additional features which increase the semantic expressiveness of the representation. In [12] the search-based approach for the high-dimensional problem of hybrid driving-stepping locomotion planning [11] is extended to plan on multiple levels of abstraction which results in significantly shorter planning times while the result quality stays comparable. In [16] temporal abstraction is applied to reinforcement learning which generates an efficient space to explore complex environments.

We propose a method to combine VINs with the idea of planning on multiple levels of abstraction to obtain a learning-based planner which is capable of solving planning tasks on challenging, larger state spaces. The information loss in coarser representations is compensated by increasing the number of features. In addition, detailed representations are only generated for parts of the environment which decreases memory requirements. This increases the applicability of learning-based planning approaches to real-world problems.

III. Method

VINs internally represent each state as one cell of a multi-dimensional grid and compute a reward and state-value for each of these grid cells. To enable information flow from the goal to the start state, Bellman updates are performed repeatedly within the VI module. The number of required Bellman updates depends on the maximum possible path length. For large and high-dimensional grids, this leads to large computation graphs for the gradients during backpropagation, resulting in long training times and high memory consumption. Since the number of states within the VI module is limited, we

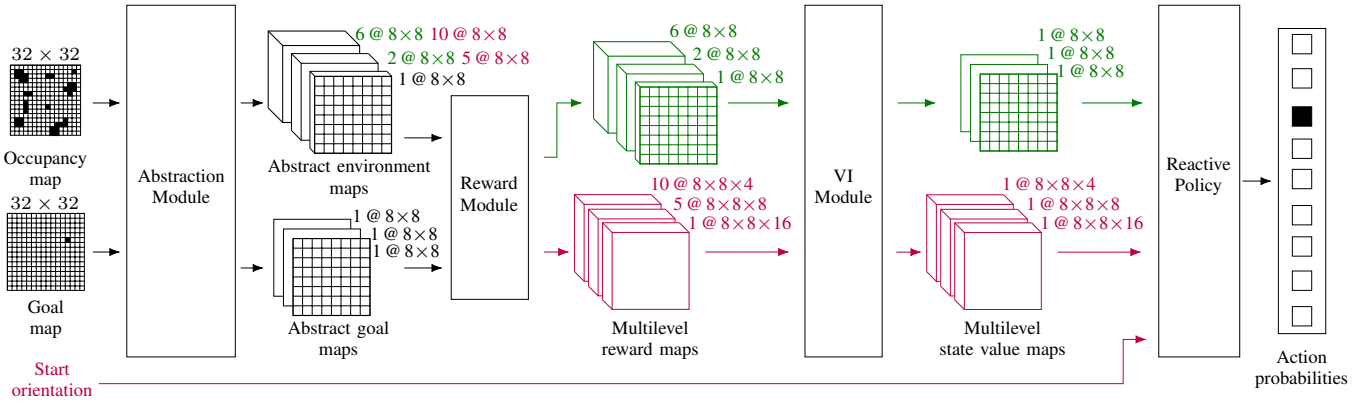

Fig. 2. Network architecture. Elements which only account to 2D grid world planning are shown in green. Elements for 3D locomotion planning are purple. The depicted map sizes correspond to 32 × 32 input maps.

change what each state represents. In the vicinity of the robot, which is defined to be always in the center of each map, spatial precision is needed to plan the next robot action. Regions which are further away from the robot can be described in a coarser, more abstract representation.

As an example, we define three levels of abstraction with a constant number of cells but decreasing resolution. *Level-1* has the original input resolution but only covers the vicinity of the robot. For *Level-2*, the resolution is halved resulting in a four times larger covered area. This step is repeated to obtain *Level-3*. Hence, *Level-3* covers an area which is 16 times larger than the *Level-1* area. The spatial arrangement of the three representations is depicted in Fig. 1.

To compensate the information loss in coarser representations, additional features are introduced for each abstract cell and are learned during training. Experiments showed that one, two, and six features for *Level-1*, *Level-2*, and *Level-3*, respectively, achieved best results.

A. Network Architecture

Input to the network (Fig. 2) is an occupancy map of the environment and an equally sized goal map which only contains zeros except for the goal cell (one-hot-map). In contrast to original VINs, we do not provide the system explicit information about the start state, but define that input maps are always robot centered, as also shown in [1].

In a first step, the **Abstraction Module** (Fig. 3) processes the input environment map to three, equally sized abstract environment maps. The *Level-1* map is extracted as a patch around the center of the occupancy map. A convolution and subsequent max pooling operation generate the *Level-2* representation with halved resolution from the input map. While the *Level-2* map is again extracted from the map center, the whole *Level-2* representation is processed similarly to obtain the *Level-3* map. The goal map is processed similarly using max pooling operations without convolutions.

Subsequently, the abstract environment maps and the goal maps are fed into the **Reward Module** (Fig. 4) generating rewards for each state. Since the abstract environment maps have multiple features per cell, this needs to be considered

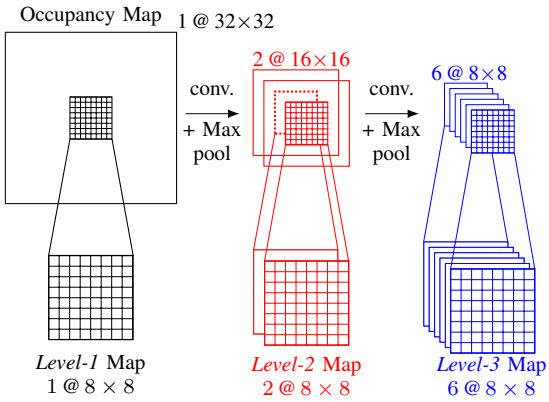

Fig. 3. Abstraction Module. Both convolutions use kernels of size 3 × 3 followed by a 2 × 2 max pooling operation. The goal map is processed using only 2 × 2 max pooling operations without prior convolutions. The depicted map sizes correspond to 32 × 32 input maps.

in the reward computation. It, e.g., might be possible that an abstract map cell can be entered from one direction but not from another, which might be encoded in the features. Hence, multiple reward features are necessary for each cell to represent such information. It is important to understand that information encoded at the same cell position of different abstraction level maps refer to different locations in the environment. We support the network in understanding this relation with the following method: The *Level-1* reward map is obtained by stacking the environment and goal maps and processing them with two convolutions whose parametrization is inspired by original VINs. These convolutions use a padding to keep the map size constant. Thus, the relation between cell position and environment location stays fixed.

To enable information flow between levels, the intermediate *Level-1* features extracted by the first convolution are also used within the *Level-2* reward map generation. Similar to the Abstraction Module, a convolution and subsequent max pooling operation abstract the *Level-1* feature map and match the resolution with *Level-2*. The result is padded with zeros to match the size of the *Level-2* map. This procedure ensures that information at the same cell position in both maps describe the same environment location. The *Level-2* environment and

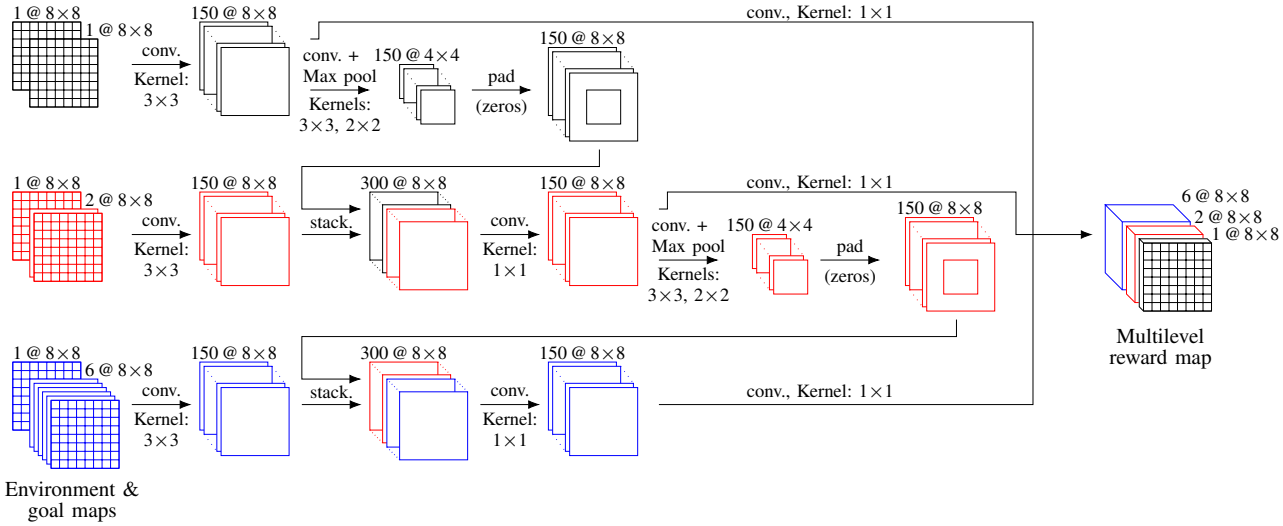

Fig. 4. Reward Module. *Level-1* maps are shown in black, red parts belong to *Level-2* and blue parts to *Level-3*. Depicted map sizes correspond to 32×32 input maps.

goal maps are processed similarly to *Level-1* to obtain the *Level-2* reward map. However, after the first convolution, the intermediate *Level-2* feature map is stacked with the abstracted *Level-1* feature map and processed by a 1×1 convolution. Similarly, the resulting combined feature map is used for the *Level-3* reward map generation.

Reward maps are input to the **VI Module** (Fig. 5) where they are processed to state-value maps. Each iteration of the Bellman update is realized by a convolution and subsequent max pooling operation. The kernel is chosen such that it covers the set of possible actions and thus can propagate state values through the map, respectively. Unlike the reward maps, state-value maps consist of only one channel as they describe the expected long-term reward for a state. To enable information flow between levels, we apply a padding to the input maps at the beginning of each iteration, as shown in Fig. 6. The padded area contains values of the neighboring cells of the next higher abstraction level. Since the reward maps vary in their number of features, a mapping from higher-level to lower-level features is required. We found that the best result quality was achieved by using the average over all features of one higher level-cell as the padding value for all corresponding lower level-cells. Learning a mapping from higher-level to lower-level features with fully connected layers performed worse.

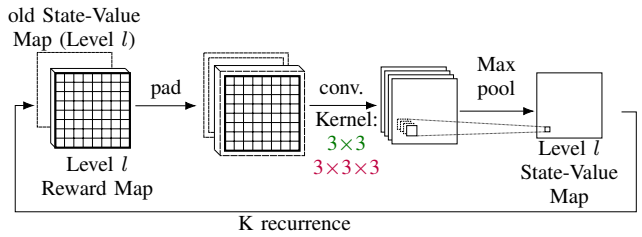

Fig. 5. Value Iteration Module. The depicted operation is performed for each level individually. The padding operation (Fig. 6) enables information flow between levels. Elements which only belong to 2D grid world/3D robot locomotion planning are shown in green/purple.

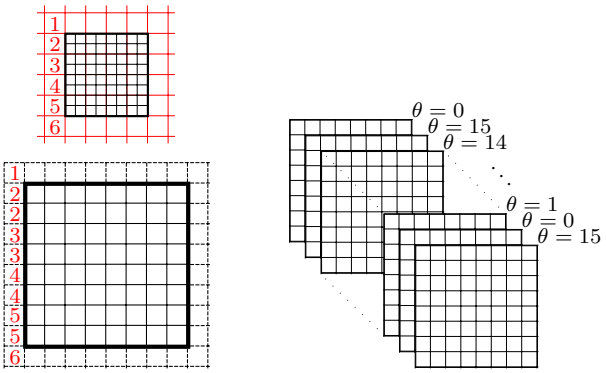

Fig. 6. Left: Padding the map of abstraction level l (bottom) to allow information flow from the map of level $l+1$ (top) to level l. The numbers indicate which values are copied where. Right: Orientation padding during 3D VIs to emphasize that the orientations $\theta = 15$ and $\theta = 0$ are neighbors.

Finally, for all neighbors of the start state, their state-values are mapped to probabilities over actions through a **Reactive Policy**, which simply is a fully connected layer.

We apply the proposed architecture to two planning problems: 2D grid worlds and 3D robot locomotion. The former is used to compare against original VINs and HVINs while the latter demonstrates the capabilities of our approach to handle problems of higher complexity. Necessary specifications and modifications for each planning domain are described in the following. The network is implemented using Python 2.7 and PyTorch 0.4.1. Respective source code is available online[1].

1) 2D Grid Worlds: The planner is given queries for a point-like agent in 2D grid worlds. The goal map is input as a one-hot map. As actions, the agent can move to one of the eight adjacent neighbor cells (Fig. 7 a).

2) 3D Robot Locomotion: Given is a robot that can perform omnidirectional driving and has a fixed footprint. Possible actions for the agent are (Fig. 7 b, c):

[1] https://github.com/AIS-Bonn/abstract_vin

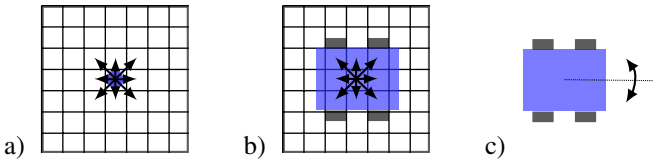

Fig. 7. Possible actions for planning domains. a) Moving to an adjacent neighbor cell in 2D grid worlds, b) drive to an adjacent neighbor state with fixed orientation in 3D robot locomotion planning, and c) turn to the next discrete orientation with fixed position in 3D robot locomotion planning.

- Move to one of the eight adjacent neighbor states with fixed orientation and
- turn to the next discrete orientation with fixed position.

When generating training data and evaluating the network, collision checking is done by checking if any cell which is occupied by the robot footprint is also occupied by an obstacle. Hence, for robots with modular footprints, it is possible to, e.g., take obstacles between their legs.

To enable the network to handle 3D agent states, reward and value maps are extended by one additional dimension for the orientation (Fig. 2). We represent the robot orientation for *Level-1* in 16, for *Level-2* in eight, and for *Level-3* in four discrete orientations of equal angular distance. Due to the increased complexity of the agent states, we increase the number of features for *Level-2* to five and for *Level-3* to ten. Furthermore, we increase the number of convolutions within the Reward Module by two additional convolutions for processing the *Level-1* map and one additional convolution for the *Level-2* map. To consider detailed collision checking for the robot footprint, we transform the reward map at the end of the Reward Module: For each possible robot base pose, we sum over the four cells corresponding to the wheel positions and assign the result to the cell of the robot base pose.

In the VI Module, the convolution kernel needs to cover all possible actions which results in a 3D kernel. Since the neighborhood relation for the orientation is cyclic, we pad the reward maps and state-value maps on the orientation channel on each end with the values of the opposite end (Fig. 6).

Other than 2D planning, the architecture for 3D planning needs information about the start and goal orientation. The start orientation is fed into our system as an additional parameter. It is only used within the Reactive Policy to select those state-values which belong to neighbor states of the start state. The goal orientation is encoded in the goal map in which all cell entries are 0, except for the goal cell which carries the index of the discrete orientation $(1-16)$.

B. Training

Training data is generated by placing obstacles of random number, size and position into a 2D grid world. In addition, multiple goal states are placed randomly. Since similar data sets are used by Tamar et al. [25], they offer a high comparability to the original VIN implementation. The same method is used to generate training data for the 3D Locomotion application. In addition, more challenging maps are obtained by generating mazes in those 2D grid worlds. For all maps, the start state is defined to be in the map center. Subsequently, we use an A^* planner as an expert to generate optimal paths.

Overall, we generated 5,000 environments of the 2D random obstacle grid worlds and 5,000 environments of the 2D maze grid worlds. Seven planning tasks were defined for each environment, resulting in 35,000 different training scenes for each domain. The validation and test sets both consist of 715 additionally generated environments with seven planning tasks each, resulting in 5,005 different scenes for each set. This applies to the 2D random obstacle grid world, the 2D maze grid world, and the random obstacle grid worlds which are used for 3D planning. To increase data efficiency during training, we do not only use the whole expert paths but also sub-paths, which are generated by randomly placing the start and goal states on the expert path. Hence, the amount of training data increases significantly. We discovered that in the training data set, some actions were chosen more often than other actions. To support the training, we weight the losses for the different actions by the inverse action frequencies.

When evaluating our approach against VINs and HVINs, all networks are trained using the RMSprop optimizer as proposed in [26], which was also used in the original VIN publication. However, when using the RMSprop without any further learning rate scheduler, the network converges to sub-optimal local minima. Employing the cyclic learning rate scheduler proposed in [20] results in a stabilized training performance and better results on the validation set: During training, the learning rate decreases following a cosine annealing scheme. After several training epochs, the learning rate is reset to a higher value. We call the time between learning rate resets a learning rate cycle. Initially, the length of a learning rate cycle is set to 48 epochs and the learning rate is 0.001. After each cycle, the cycle length increases to 150% while the initial learning rate decreases to 95% of the previous one. We compare the results of the cyclic learning rate scheduling to a fixed learning rate of 0.001 in the experiment section. Figure 8 visualizes the learning rate behavior, depicts the training performance on the 2D random obstacle grid domain, and compares it to original VINs and HVINs.

The network is designed to output the next action for a given input. To obtain a path for solving a planning problem, we iteratively let the network predict the next action and update the input maps according to the new robot position. A path is considered successful if it reaches the goal without obstacle collision and within no more than twice the optimal number of actions, as determined by the expert A^* planner. The *success* measures if the network was able to plan a path to the goal. While HVINs and AVINs were normally implemented using three representation levels, we implemented an additional version with four representation levels for the 128×128 maps. For AVINs, *Level-4* describes each map cell with ten features.

The training performance in Fig. 8 indicates that our method obtains better *success* rates than VINs and HVINs on the validation set. VINs become unstable for large maps. For the 128×128 maps, it can be seen that both HVINs and AVINs benefit from a forth representation level.

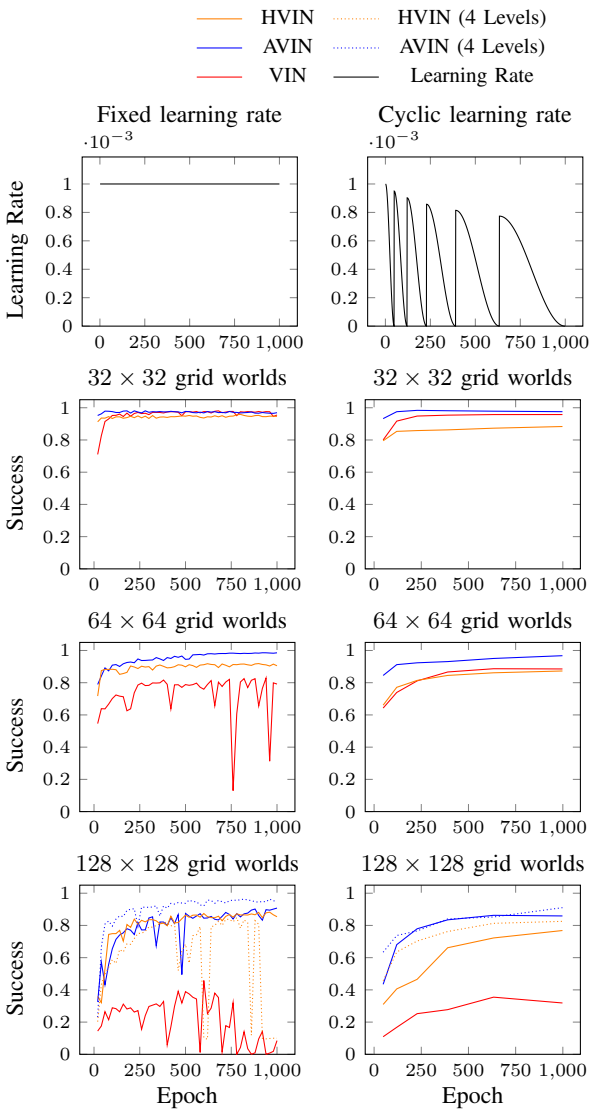

Fig. 8. Training performance of VINs, HVINs, and AVINs on the validation set. Left: Fixed learning rate. Right: Cyclic learning rate scheduling. While the fixed learning rate training was evaluated every 30 epochs, the training with cyclic learning rate scheduling was only evaluated at the end of each learning rate cycle. Especially VINs showed partially unstable training behavior which has also been reported in e.g., [18].

IV. Experiments

All experiments were done on a system equipped with an Intel Core i7-8700K@3.70 GHz, 64 GB RAM, and an NVidia GeForce GTX 1080Ti with 11 GB memory. A video with additional footage of the experiments is available online[2].

To evaluate the network performance, we consider two measures. The above-mentioned *success* rate evaluates the network performance on the desired task: path planning. In addition, to compare against VINs, we evaluated the *accuracy* describing how often the network chooses the same next action as the expert A^* planner. Please note that in many cases there is more than one optimal next action. However, as in original VINs, the network is trained to output one next action which is compared to the planner. Hence, there occur cases in which the output of the network is different from the output of the A^* planner but the network still unrolls an optimal path although the *accuracy* measures a mistake. Stated planning times and memory consumption of our approach include input maps shifting after each network inference to concatenate the next-action network outputs to paths.

[2]https://www.ais.uni-bonn.de/videos/RSS_2019_Schleich/

A. Path Planning in 2D Random Obstacle Grid Worlds

In a first experiment, we compared our AVINs against VINs and HVINs on the test sets of the random obstacle grid world domain. Since a similar test set was used in the original VIN publication, this experiment provides good comparability of the methods' capabilities. We used an implementation with three representation levels for HVINs and AVINs, each level halving the resolution of the previous one. For HVINs, the lowest resolution level used the same number of Bellman updates K as proposed for original VINs. This coarse state-value initialization was then refined by two Bellman updates on the medium resolution map and two consecutive Bellman updates on the fine resolution map.

In [25], grid world sizes from 8×8 to 28×28 are considered. We perform tests on slightly larger maps with 32×32, and significantly larger maps with 64×64 and 128×128 cells. For the largest map size, we tested the additional implementation of HVINs and AVINs using four representation levels. Table I states the results for training with the fixed learning rate while results of a training with cyclic learning rate scheduling are given in Tab. II. Figure 9 depicts paths on a 128×128 map.

The results indicate that our AVINs outperform VINs and HVINs on all map sizes with both learning rate behaviors in terms of *accuracy*, *success*, and memory consumption. While VINs and AVINs show a consistently better performance with the cyclic learning rate scheduling, HVINs perform better with the constant learning rate. It can be furthermore seen that on the 128×128 maps, both HVINs and AVINs benefit from a forth representation level in all measures.

TABLE I
Results for 2D random obstacle grid worlds with fixed learning rate training. All stated numbers are averaged over five network instances with different random seed initializations. For the 128×128 map size, results of HVINs and AVINs with three and four representation levels are given.

32×32	VIN	HVIN	AVIN
Accuracy	80.38%	80.08%	**84.52%**
Success	91.72%	93.84%	**97.18%**
Path difference	2.48%	2.23%	**1.63%**
Graphics memory [MB]	761	739	**685**

64×64	VIN	HVIN	AVIN
Accuracy	70.08%	77.42%	**81.60%**
Success	71.98%	86.82%	**94.02%**
Path difference	2.94%	2.55%	**1.34%**
Graphics memory [MB]	1815	1399	**969**

128×128	VIN	HVIN-3	HVIN-4	AVIN-3	AVIN-4
Accuracy	55.96%	76.50%	78.04%	77.70%	**83.54%**
Success	31.56%	83.24%	84.00%	78.52%	**88.72%**
Path diff.	8.46%	2.09%	2.80%	3.60%	**1.84%**
GRAM [MB]	8247	4085	4049	2189	**1167**

TABLE II
RESULTS FOR 2D GRID WORLDS WITH CYCLIC LEARNING RATE SCHEDULING. GRAPHICS MEMORY USAGE IS SIMILAR TO TAB. I.

32×32	VIN	HVIN	AVIN
Accuracy	84.92%	81.36%	**85.00%**
Success	95.26%	88.27%	**97.56%**
Path difference	**1.01%**	**1.01%**	1.56%

64×64	VIN	HVIN	AVIN
Accuracy	78.88%	80.46%	**83.75%**
Success	89.17%	88.33%	**94.99%**
Path difference	**1.02%**	**1.02%**	1.28%

128×128	VIN	HVIN-3	HVIN-4	AVIN-3	AVIN-4
Accuracy	66.41%	77.46%	79.34%	84.28%	**85.01%**
Success	34.89%	77.40%	83.56%	86.85%	**91.59%**
Path diff.	1.02%	1.02%	1.02%	**0.80%**	1.31%

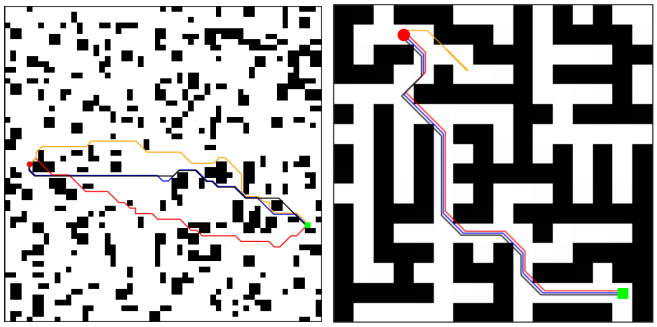

Fig. 9. Result paths. The figure only depicts the corresponding map sections. The start is marked with a red circle and the goal with a green sqaure. Left: 128×128 random obstacle grid world. VINs (red) fail to find a collision free path, HVINs (orange) react to obstacles when approaching them while AVINs (blue) shows better long-term understanding. A optimal path obtained by the A^* planner is depicted in black. Right: 32×32 maze. HVINs (orange) are not able to find a path. VINs (red), our AVINs (blue), and the A^* planner (black) provide the same optimal solution.

B. Path Planning in 2D Maze Grid Worlds

In a second experiment, we aimed at investigating the limitations of our proposed method and the quality of its abstraction. We compared it to VINs and HVINs on 2D maze grid worlds. Mazes possess a larger information density in comparison to the random obstacle grid worlds since the occupancy of nearly every single grid cell is important. Hence, when generating coarser representations, the effect of information loss is large. This puts the focus on the quality of the abstraction which shall learn to encode all required information in the additional features. Table III states the performance of VINs, HVINs, and AVINs on map size of 16×16, 32×32, and 64×64. Training was performed with fixed learning rate. An example maze and generated paths are depicted in Fig. 9.

Since original VINs perform no abstraction procedure, it was to expect that they obtain the best *accuracy* and *success* rates in this challenging domain. However, while the performance difference between HVINs and VINs is rather small in the random obstacle domain, this difference increases considerably for large maze worlds. For the 16×16 maps, our approach performs worse than HVINs. An explanation for this might be that, for this input map size, *Level-1* only has a size of 4×4 which might be insufficient to detailly plan next actions in the vicinity of the robot. Nevertheless, our method significantly outperforms HVINs on larger maps indicating the advantage of our abstraction method—which introduces additional features to compensate information loss—compared to HVINs, which employ no additional features.

TABLE III
RESULTS FOR 2D MAZE GRID WORLDS.

16×16	VIN	HVIN	AVIN
Accuracy	**94.42%**	87.20%	85.59%
Success	**94.48%**	87.42%	86.88%
Path difference	**0.51%**	2.02%	1.96%
Graphics memory	**569 MB**	575 MB	635 MB

32×32	VIN	HVIN	AVIN
Accuracy	**85.60%**	69.94%	82.17%
Success	**82.10%**	48.54%	71.50%
Path difference	**0.88%**	2.02%	1.09%
Graphics memory	761 MB	739 MB	**685 MB**

64×64	VIN	HVIN	AVIN
Accuracy	**84.58%**	58.82%	81.57%
Success	**78.02%**	14.22%	59.39%
Path difference	1.49%	1.99%	**0.68%**
Graphics memory	1815 MB	1399 MB	**969 MB**

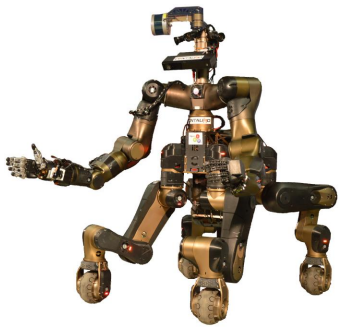

Fig. 10. The Centauro robot.

C. Planning 3D Locomotion with Footprint Consideration

In a third experiment, we investigated the capabilities of the proposed method to solve significantly more complex planning tasks. We employed the 3D version of AVINs to plan omnidirectional driving locomotion while considering the robot footprint. An example platform is the quadrupedal disaster response robot Centauro [14] whose legs end in 360° steerable, active wheels (Fig. 10). We chose a fixed leg configuration with 0.8 m longitudinal and lateral distance between wheels. Environment maps had a resolution of 0.2 m.

At first, we employed our method to plan paths for the described footprint on the 32×32 maps of the random obstacle grid domain. Averaged over five training runs, we achieved an average success rate of 74.20% for the 5,005 tasks in the test set while our paths were on average 1.86% longer than the optimal solution. The required graphics memory was 865 MB. We further compared the planning times for both the A^* planner and AVINs for this 3D locomotion task and the 2D planning task (see Sec. IV-A) for the 32×32 map size.

As can be seen in Tab. IV, the A^* planner is in average about 23 times faster than AVINs on the 2D planning task. However, for the more complex 3D planning tasks with footprint consideration, both planners have similar planning

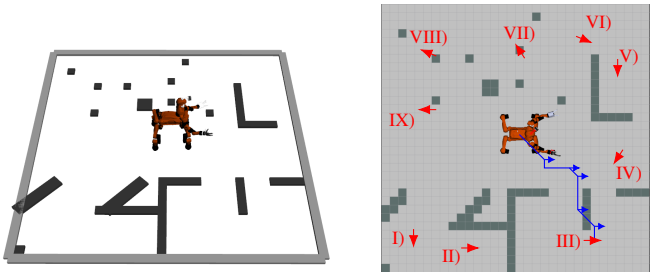

Fig. 11. 3D Locomotion planning experiment. Left: Gazebo arena with Centauro. Obstacle heights were chosen to be rather small to prevent the laser scanner from coping with occlusions. Right: The corresponding occupancy map with the nine chosen goals and one example result path.

times. This observation supports our assumption that learning-based planners are promising for complex planning tasks since they do not perform extensive, iterative searches, as traditional planners do.

Finally, we integrated AVINs into the locomotion planning pipeline of Centauro. A 3D rotating laser scanner with spherical field-of-view perceived the environment. Measurements were processed using the SLAM method of Droeschel et al. [5]. Occupancy maps with a resolution of 0.2 m were generated from these point clouds and were input to AVINs. Robot perception and control was implemented in C++. Communication with AVINs was realized using ROS. We designed a test arena in the Gazebo simulation environment which contained challenging obstacles of different shape and size, as shown in Fig. 11. We placed nine different goal states in the map and compared our approach to the expert A^* planner.

The results in Tab. V indicate that our AVIN planner obtained optimal or close to optimal paths in most cases. Even challenging tasks which require the robot to take obstacles between its leg (e.g., III and VIII) could be solved. However, the AVIN planner did not plan successful paths for problems II and V, which required turning actions in narrow sections. Here, the obtained paths did not reach the goal but ended oscillating between two adjacent poses. Moreover, AVIN planning times had a considerably smaller distribution than the A^* planner. Although AVIN planning times do not show a considerable advantage over A^*, this experiment demonstrates the application of AVINs to significantly more challenging tasks compared to original VIN applications.

However, the above-described comparison between the 2D and 3D application indicates the large potential of learning-based planners to outperform traditional planners in complex planning tasks. Nevertheless, increasing hardware requirements and decreasing success rates currently limit this development. Future work might combine the strengths of AVIN and A^* to create a planner with perfect success rate and low planning times for high-dimensional domains, e.g. by using AVIN to generate an informed heuristic for A^*. Different methods to combine search- and learning-based planners have been proposed in [6], [13] and [23].

TABLE IV
PLANNING TIMES COMPARISON BETWEEN THE A^* PLANNER AND AVINS FOR 2D AND 3D PLANNING TASKS.

32×32	A^* planner	AVIN
2D planning task	0.004 sec	0.093 sec
3D planning task with footprint	0.263 sec	0.283 sec

TABLE V
RESULTS OF OUR APPROACH AND THE A^*-PLANNER FOR THE TASKS DEPICTED IN FIG. 11.

	AVIN		A^*-planner	
	Path length	Planning Time	Path length	Planning Time
I)	24.59	0.431 sec	23.41	0.169 sec
II)	Not found		24.14	0.980 sec
III)	18.49	0.342 sec	17.90	0.102 sec
IV)	18.80	0.363 sec	18.80	0.341 sec
V)	Not found		27.76	2.117 sec
VI)	18.65	0.321 sec	17.01	0.172 sec
VII)	15.55	0.321 sec	15.55	0.051 sec
VIII)	24.67	0.449 sec	22.92	0.223 sec
IX)	21.13	0.405 sec	21.13	0.705 sec

V. CONCLUSION

In this paper, we propose an extension to Value Iteration Networks (VINs) to employ multiple levels of abstractions. While the state resolution gets coarser, additional features compensate the information loss. Our approach outperforms VINs in terms of result quality on random obstacle grid worlds and is capable of solving considerably larger planning tasks while requiring only a fraction of the graphics memory. We can further demonstrate that our approach learns to encode important information in its representation which lets it obtain significantly better results in challenging environments compared to Hierarchical VINs. In addition, we successfully extended our method to plan omnidirectional driving locomotion for the disaster-response robot Centauro while considering its footprint. Comparing planning times to a search-based planner supports the assumption that learning-based planners are promising to outperform traditional planners in complex tasks. In summary, we demonstrated how abstraction enables learning-based planners to handle more complex state spaces—increasing their applicability towards real-world motion planning problems.

ACKNOWLEDGMENTS

This work was supported by the European Union's Horizon 2020 Programme under Grant Agreement 644839 (CENTAURO).

REFERENCES

[1] Sven Behnke. Local Multiresolution Path Planning. In *RoboCup 2003: Robot Soccer World Cup VII*, volume 3020 of *Lecture Notes in Computer Science*, pages 332–343. Springer, 2004.

[2] Richard Bellman. A Markovian Decision Process. *Journal of Mathematics and Mechanics*, 6(5):679–684, 1957.

[3] Richard Bellman. *Dynamic programming*. Courier Corporation, 2013[1957].

[4] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint:1604.07316*, 2016.

[5] David Droeschel, Max Schwarz, and Sven Behnke. Continuous mapping and localization for autonomous navigation in rough terrain using a 3D laser scanner. *Robotics and Autonomous Systems*, 88:104 – 115, 2017. doi: 10.1016/j.robot.2016.10.017.

[6] Edward Groshev, Aviv Tamar, Maxwell Goldstein, Siddharth Srivastava, and Pieter Abbeel. Learning generalized reactive policies using deep neural networks. In *2018 AAAI Spring Symposium Series*, 2018.

[7] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik. Cognitive mapping and planning for visual navigation. *arXiv preprint:1702.03920*, 2017.

[8] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968. doi: 10.1109/TSSC.1968.300136.

[9] Peter Karkus, David Hsu, and Wee Sun Lee. QMDP-Net: Deep learning for planning under partial observability. In *Advances in Neural Information Processing Systems (NIPS)*, pages 4694–4704, 2017.

[10] Lydia Kavraki, Petr Svestka, Jean-Clause Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566580, 1996. doi: 10.1109/70.508439.

[11] Tobias Klamt and Sven Behnke. Anytime Hybrid Driving-Stepping Locomotion Planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4444–4451, 2017. doi: 10.1109/IROS.2017.8206310.

[12] Tobias Klamt and Sven Behnke. Planning Hybrid Driving-Stepping Locomotion on Multiple Levels of Abstraction. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1695–1702, 2018. doi: 10.1109/ICRA.2018.8461054.

[13] Tobias Klamt and Sven Behnke. Towards Learning Abstract Representations for Locomotion Planning in High-dimensional State Spaces. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[14] Tobias Klamt, Diego Rodriguez, Max Schwarz, Christian Lenz, Dmytro Pavlichenko, David Droeschel, and Sven Behnke. Supervised autonomous locomotion and manipulation for disaster response with a centaur-like robot. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–8, 2018. doi: 10.1109/IROS.2018.8594509.

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1097–1105, 2012.

[16] Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3675–3683, 2016.

[17] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. *Computer Science Dept., Iowa State University*, (TR 98-11), 1998.

[18] Lisa Lee, Emilio Parisotto, Devendra S Chaplot, Eric Xing, and Rusian Salakhutdinov. Gated path planning networks. In *35th International Conference on Machine Learning (ICML)*, 2018.

[19] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research (JMLR)*, 17 (1):1334–1373, 2016.

[20] Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. *arXiv preprint:1608.03983*, 2016.

[21] Sufeng Niu, Siheng Chen, Hanyu Guo, Colin Targonski, Melissa C Smith, and Jelena Kovacevic. Generalized Value Iteration Networks: Life Beyond Lattices. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[22] Max Schwarz, Anton Milan, Arul Selvam Periyasamy, and Sven Behnke. RGB-D object detection and semantic segmentation for autonomous manipulation in clutter. *The International Journal of Robotics Research (IJRR)*, 37(4-5):437–451, 2018. doi: 10.1177/0278364917713117.

[23] Markus Spies, Marco Todescato, Hannes Becker, Patrick Kesper, Nicolai Waniek, and Meng Guo. Bounded Suboptimal Search with Learned Heuristics for Multi-Agent Systems. 2019.

[24] Aravind Srinivas, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Universal Planning Networks. *arXiv preprint:1804.00645*, 2018.

[25] Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2154–2162, 2016.

[26] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude, 2012.