Value Refinement Network (VRN)

Jan Wöhlke[1,2] , **Felix Schmitt**[1] , **Herke van Hoof**[3]
[1]Bosch Center for Artificial Intelligence
[2]UvA-Bosch Delta Lab, University of Amsterdam
[3]AMLab, University of Amsterdam
{JanGuenter.Woehlke, Felix.Schmitt}@de.bosch.com, h.c.vanhoof@uva.nl

Abstract

In robotic tasks, we encounter the unique strengths of (1) reinforcement learning (RL) that can handle high-dimensional observations as well as unknown, complex dynamics and (2) planning that can handle sparse and delayed rewards given a dynamics model. Combining these strengths of RL and planning, we propose the Value Refinement Network (VRN), in this work. Our VRN is an RL-trained neural network architecture that learns to locally refine an initial (value-based) plan in a simplified (2D) problem abstraction based on detailed local sensory observations. We evaluate the VRN on simulated robotic (navigation) tasks and demonstrate that it can successfully refine sub-optimal plans to match the performance of more costly planning in the non-simplified problem. Furthermore, in a dynamic environment, the VRN still enables high task completion without global re-planning.

1 Introduction

Reinforcement learning (RL) trained policies have been demonstrated to play games at super-human levels [Mnih *et al.*, 2015; Silver *et al.*, 2018] and to convincingly act in robotic manipulation [Andrychowicz *et al.*, 2017] and locomotion [Schulman *et al.*, 2016] tasks. This is because RL can deal with unknown, complex (stochastic) dynamics. Nevertheless, sparse reward signals and long decision horizons still pose an obstacle to achieving high performance and data-efficiency. Curriculum learning [Florensa *et al.*, 2017; Wöhlke *et al.*, 2020], intrinsic motivation [Pathak *et al.*, 2017; Haber *et al.*, 2018], or hierarchical RL [Bacon *et al.*, 2017; Levy *et al.*, 2019; Vezhnevets *et al.*, 2017; Nachum *et al.*, 2018] try to address these problems.

Francis *et al.* [2020], Wöhlke *et al.* [2021], and Christen *et al.* [2021] demonstrated that a combination of sub-goal planning in a problem abstraction and sub-goal guided RL in the original state and action space can efficiently solve complex sparse reward navigation tasks. Planning sub-goals has the advantage of better generalization to unseen environment layouts and improved data-efficiency over learning-based approaches like (deep) Q-learning. However, since the transition dynamics in the problem abstraction are unknown and access to the original environmental dynamics is not assumed in an RL setting, "classic" planners (like A* or RRT) are insufficient for sub-goal planning. Hence, Wöhlke *et al.* [2021] learn a transition model for the abstraction.

Yet even this approach struggles with (stochastic) changes in the environment, like moving agents, which may cause costly re-planning or decreased performance. Furthermore, the state space abstraction needs to contain all essential details of the original state space in the necessary granularity. Besides that, we observe that global planning is often only required for a small subset of the state variables: In case of a navigating agent, it is sensible to plan globally a (shortest path) route to the goal, whereas other state variables like the velocity or orientation or other moving agents are not globally relevant. Instead, they impose local motion constraints, to be accounted for locally. Taking inspiration from Chen *et al.* [2019] on combining planning in a low-dimensional state space abstraction with full state information, we therefore propose to refine an initial, global plan in a simple 2D problem abstraction based on the full current state.

To this end, our work presents a novel neural network, the Value Refinement Network (VRN), which implements this promising strategy. Similar to other works [Ichter *et al.*, 2018; Tamar *et al.*, 2016; Lee *et al.*, 2018], we assume a discrete layout map given, which allows for easy computation of value maps reflecting shortest paths. During an episode, our VRN locally refines this value "prior" by passing a local crop of it and the layout alongside full state information through a suitable network architecture whose parameters are updated via RL. This way, we combine the benefits of planning and learning value functions: The generalization and sample-efficiency of planning and the handling of high-dimensional state spaces and stochasticity by learning the local refinement.

Evaluating on simulated robotic navigation tasks, we provide evidence for the following research hypotheses[1]:

H.1: The local refinement of a simple initial 2D value function learned by our VRN can match the performance of more costly planning in a significantly more detailed state space.

H.2: Our VRN maintains high performance in dynamic environments, without the necessity for global re-planning.

[1]We plan to release code and supplemental material (appendices) here: https://github.com/boschresearch/Value-Refinement-Network

2 Related Work

A group of well-known approaches to incorporate planning operations into policy learning are the differentiable planning modules [Tamar *et al.*, 2016; Nardelli *et al.*, 2019; Lee *et al.*, 2018]: These recurrent neural network architectures are capable of learning to approximately perform value-based planning via backpropagation. Requiring a discrete, grid map input of the environment, differentiable planners are primarily demonstrated on low-dimensional (2D) discrete state and action space maze navigation tasks. The (M)VProp architecture of Nardelli *et al.* [2019] is specifically designed for RL training but struggles to handle non-holonomic agent dynamics [Wöhlke *et al.*, 2021]. While (M)VProp has been shown to be able to handle dynamically changing environments in [Nardelli *et al.*, 2019], this setting requires repeated re-planning, which forced the authors to reduce the training map size. On larger problem sizes, planning is more expensive and repeated re-planning in dynamic environments becomes infeasible. We, instead, aim at developing a method that does not require global re-planning, and therefore can handle larger state spaces, including dynamic elements.

Differentiable planning modules work well in discrete state and action spaces. Continuous state and action spaces have been effectively addressed by hierarchically combining some form of high-level sub-goal planning with a low-level policy operating in the original spaces. Francis *et al.* [2020] use a sampling-based probabilistic roadmap (PRM) planner to set waypoints for an RL trained control policy. Christen *et al.* [2021] (HiDe) and Wöhlke *et al.* [2021] (VI-RL) apply value-based sub-goal planning in a discrete (grid-like) high-level abstraction of selected components of the continuous state space. While Christen *et al.* [2021] use an (M)VProp, Wöhlke *et al.* [2021] employ (exact) value iteration with a learned high-level transition model. With increasing complexity of the original continuous state space, more state components need to be considered in the high-level planning, in order to generate good sub-goals. Unfortunately, value-based planning approaches do not scale well memory- and runtime-wise with the number of state dimensions considered. We aim at reducing the necessary level of detail of the high-level planning by applying local refinement.

In contrast to our work, other authors have considered plan refinement for the case that an exact (environmental) dynamics model *is available*. Chen *et al.* [2019] presented a way of learning to integrate full state information into plans in a lower-dimensional space. In order to choose the next state for tree expansion, their RRT-based NEXT planner uses an attention-based architecture that maps the state into a discrete latent space, which is then processed by a differential planning module. Apart from the additional assumptions, NEXT requires re-planning for dynamically changing environments.

3 Problem Statement

3.1 Original Decision Making Problem

In this work, we consider a distribution of Markov Decision Processes (MDPs) $\mathcal{M}$ that share the same state space $\mathcal{S}$ and action space $\mathcal{A}$. Start states s_0 are sampled from the MDP specific start distribution $\mathcal{S}_{0,m} \subseteq \mathcal{S}$. The goal-dependent (sparse) reward function is of the following form: $r(s,g) = \mathbb{I}_{d(s,g)\leq\epsilon}$ with goal states g sampled from a goal distribution $\mathcal{S}_{g,m} \subseteq \mathcal{S}$ and $d(\cdot,\cdot)$ being some distance measure. $\mathcal{P}_m^{\mathcal{S},\mathcal{A}}(s'|s,a)$ are the MDP specific transition dynamics that model the transition from a state s to the next state s' given as a result of action a. As it is common in the RL setting, the functional form of the dynamics is unknown and can only be experienced by interacting with the environment. Finally, γ is the discount factor and T the time horizon. We can sample specific MDPs $m = \left(\mathcal{S}, \mathcal{A}, \mathcal{P}_m^{\mathcal{S},\mathcal{A}}, r, \mathcal{S}_{0,m}, \mathcal{S}_{g,m}, \gamma, T\right)$ that for example represent different environment layouts.

The overall objective is to find a policy π that maximizes the expected returns under the distribution of MDPs, the goal and initial state distributions, and the dynamics:

$$\max_{\pi} \mathbb{E}_{m\sim\mathcal{M}, s_0\sim\mathcal{S}_{0,m}, g\sim\mathcal{S}_{g,m}, \mathcal{P}_m}\left[\sum_{t=0}^{T-1}\gamma^t r\left(s_{t+1}, g\right)\right] \tag{1}$$

3.2 Problem Abstraction

Similar to Wöhlke *et al.* [2021], we assume that there exists a problem abstraction with finite state space $\mathcal{Z}$, finite action space $\mathcal{O}$, and a time horizon H to execute an abstract action $o \in \mathcal{O}$ in the original state and action space. We also assume that there exists a known mapping $z = f_{\mathcal{Z}}(s)$ defining the state abstraction and that for any z a known $\mathcal{N}(z) \subset \mathcal{Z}$ contains at least all neighbors directly reachable from z.

In this work, we specifically assume the finite state space abstraction $\mathcal{Z}$ to be a regular 2D grid such that it is generally much smaller than the not necessarily finite state space $\mathcal{S}$, $|\mathcal{Z}| \lll |\mathcal{S}|$. In case of continuous state space agent navigation, $\mathcal{Z}$ may result from discretization of the x- and y-coordinates according to a chosen resolution.

Abstract actions $o \in \mathcal{O}$ correspond to the "sub-task" of reaching a neighboring abstract state $z' \in \mathcal{N}(z)$. Since these "sub-tasks" o may be executed by a potentially learned (stochastic) policy or controller within horizon H, the unknown transition dynamics $\mathcal{P}_m^{\mathcal{Z},\mathcal{O}}(z'|z,o)$ of the problem abstraction may be non-deterministic.

4 Value Refinement

4.1 Definition

In order to solve the problem in the original state space $\mathcal{S}$, we need a value function $V(s)$ or $Q(s,o)$ with respect to the original state space. We define value refinement as representing $Q(s,o)$ as a function of s and $\widetilde{V}^{\mathcal{Z}}$

$$Q(s,o) = f\left(\widetilde{V}^{\mathcal{Z}}, s, o\right) \tag{2}$$

with $\widetilde{V}^{\mathcal{Z}}$ being a globally known, not necessarily optimal value function for the state space abstraction $\mathcal{Z}$.

4.2 Motivating Example

As a motivating example for demonstrating the effectiveness of locally refining values, consider the following 25×25 grid-world maze navigation task (Fig. 1) with discrete states and actions. The 3D state space $\mathcal{S}$ consists of the x- and y-coordinate as well as an orientation component that can

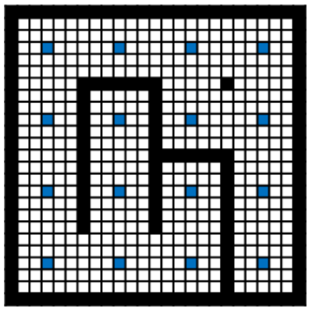


Figure 1: Exemplary maze layout. Layouts were randomly generated with five tiles wide corridors (allowing for turning). Potential start and goal positions are shown in blue. Reaching the goal tile results in a sparse reward of one. The time horizon is $T = 100$.

attain eight different values: 'North', 'North-East', 'East', 'South-East', 'South', 'South-West', 'West', and 'North-West'. The agent's actions correspond to the (eight) neighboring tiles. For this motivating example, the transition dynamics $\mathcal{P}_m^{\mathcal{S},\mathcal{A}}\left(s'|s,a\right)$ are, for once, assumed to be known: An action is only executed if the direction of the target tile deviates from the agent's orientation by maximum one unit. For example, in orientation 'North', only the actions 'North-West', 'North', and 'North-East' will be executed. As a result, the agent changes its orientation to the navigation direction and subsequently moves to the target tile, unless it is occupied.

Consider the problem abstraction case with $\mathcal{O} = \mathcal{A}$ and $H = 1$. Furthermore, the abstract state space $\mathcal{Z}$ ignores the orientation component of $\mathcal{S}$. This results in the 25×25 tiles as abstract states z.

We now demonstrate that the local refinement strategies we want to explore can effectively solve this navigation task. Utilizing our knowledge about the exact agent dynamics $\mathcal{P}_m^{\mathcal{S},\mathcal{A}}\left(s'|s,a\right)$ we can apply value iteration (VI) to the original navigation problem in the (3D) state space $\mathcal{S}$, which we denote "VI 3D". Table 1 shows the success rate of navigating the maze with a greedy policy. Without any surprise, "VI 3D" results in an optimal policy for the task. However, 3D VI is costly memory- and compute-wise.

Performing value iteration in the problem abstraction ("VI 2D") is much cheaper. We therefore need a transition model $\widetilde{\mathcal{P}}_m^{\mathcal{Z},\mathcal{A}}\left(z'|z,a\right)$ in the state space abstraction $\mathcal{Z}$: Here we assume that the agent can transition to free neighboring tiles, regardless the orientation. As a result, "VI 2D" computes shortest path plans. The success rate of "VI 2D" is as low as $25.7\,\%$ since this transition model cannot represent the motion constraints imposed by the orientation.

In the following, we will show that these sub-optimal but simple and cheap to obtain 2D "prior" plans for the problem abstraction only need slight local adjustments, accounting for the motion constraints, to become effective for the original problem. First, observe that we can use the value function $\widetilde{V}^{\mathcal{Z}}$ obtained from "VI 2D" as initialization for 3D value iteration by repeating it along the orientation dimension. As a basic result from Bellman [1957] we know that repeatedly applying VI with the exact (3D) transition model $\mathcal{P}_m^{\mathcal{S},\mathcal{A}}\left(s'|s,a\right)$ to this initialization will result in convergence to the exact 3D value function. Yet, let us observe the performance of plans resulting from the first k iterations of the 3D value iterations in Tab 1. Applying few iterations of 3D VI corresponds to a local refinement of the 2D value "prior", looking k steps ahead with the true transition dynamics. The results show that already very few 3D VI iterations result in a close to optimal 3D value function, achieving $96.5\,\%$ success rate with $k = 3$. Increasing k towards 100, which is the set maximum for "VI 3D", the refinement converges to $100\,\%$ success rate.

Method	Success rate (mean $\pm$ std)
VI 2D	0.257 ± 0.057
VI 2D + 1 iter 3D	0.751 ± 0.037
VI 2D + 2 iters 3D	0.909 ± 0.013
VI 2D + 3 iters 3D	0.965 ± 0.018
VI 2D + 5 iters 3D	0.989 ± 0.012
VI 2D + 100 iters 3D	1.000 ± 0.000
VI 3D	1.000 ± 0.000

Table 1: Local value refinement example (10 seed evaluation)

Of course, performing exact value iteration steps will hardly be possible in practical applications because the transition model $\mathcal{P}_m^{\mathcal{S},\mathcal{A}}\left(s'|s,a\right)$ is unknown in an RL setting and only experienced through interaction. However, this example demonstrates that local refinements of a value function of a simplified state space representation can sufficiently solve a task. In the following section, we will utilize this observation and introduce a novel neural network that can learn to perform this refinement from experience.

4.3 Value Refinement Network (VRN)

We will now propose a neural network architecture, the Value Refinement Network (VRN), which is a refinement operator learned to match the effect of applying value iterations with the exact transition model to the value function "prior".

Assumptions

We assume access to a (sub-optimal) value function "prior" $\widetilde{V}_p^{\mathcal{Z}}$ in the abstract state space $\mathcal{Z}$. Reasons for sub-optimality with respect to the original problem are for example that

- it works on a sub-set of the original state dimensions.
- it has to deal with discretization artifacts.
- approximated transition dynamics (since $\mathcal{P}_m^{\mathcal{Z},\mathcal{O}}\left(z'|z,o\right)$ is unknown) are used during generation.

The VRN is designed to learn to correct for all of these.

Architecture

The value refinement network (VRN) is implemented as a convolutional neural network (CNN) architecture with a distinct input representation $\mathcal{I}$. Figure 2 exemplarily shows a VRN design for a spatial navigation task.

With network parameters ψ, the VRN implements the function $Q\left(s,\cdot\right) = f_\psi\left(\mathcal{I}\right)$, outputting a vector of refined values for all (abstract) actions $o \in \mathcal{O}$. The network input $\mathcal{I}$ consists of the following components:

- $\widehat{V}_p^z$, a local $(k_c \times k_c)$ crop of the value "prior" $\widetilde{V}_p^{\mathcal{Z}}$, centered on the current abstract state $z = f_{\mathcal{Z}}\left(s\right)$.

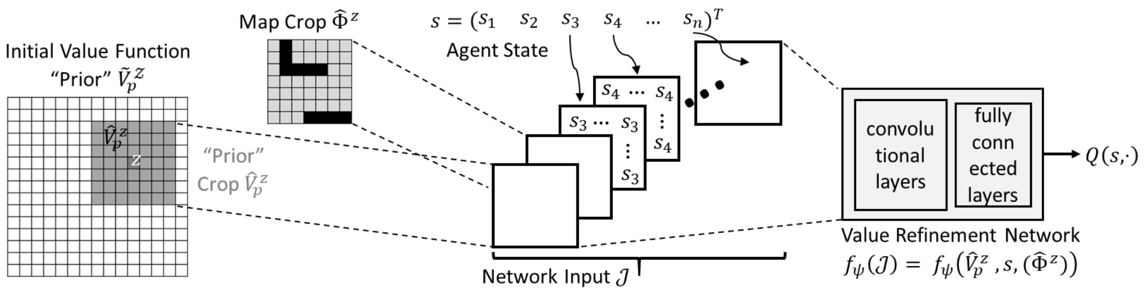

Initial Value Function
"Prior" $\tilde{V}_p^{\mathcal{Z}}$
$\hat{V}_p^z$
z
"Prior" Crop $\hat{V}_p^z$
Map Crop $\hat{\Phi}^z$
$s = (s_1 \; s_2 \; s_3 \; s_4 \; \cdots \; s_n)^T$
Agent State
$s_3 \cdots s_3$
$s_4 \cdots s_4$
Network Input $\mathcal{I}$
convolutional layers
fully connected layers
$Q(s,\cdot)$
Value Refinement Network
$f_\psi(\mathcal{I}) = f_\psi(\hat{V}_p^z, s, (\hat{\Phi}^z))$

Figure 2: VRN architecture and input representation example (agent navigation): The first input channel is a $k_c \times k_c$ (with $k_c = 7$) crop $\widehat{V}_p^z$ of the value "prior" $\widetilde{V}_p^{\mathcal{Z}}$, centered on the current abstract agent state z. Given a discretized environment map Φ, indicating (static) obstacles, a $k_c \times k_c$ crop $\widehat{\Phi}^z$, centered on z, forms the second input channel. Additional input channels emerge by selecting components s_i of the full (continuous) agent state s and broadcasting the value across the corresponding input channel (exemplarily implied for s_3 and s_4). Such state components s_i are, for example, the agent x- and y- velocities and/or (the sine and cosine of the) orientation. Together, the input channels form the network input $\mathcal{I}$ of the VRN, which is implemented as a CNN. The network output $Q(s,\cdot)$ is a vector of refined state-action values.

- One additional input channel for each or a subset of the current state s's components s_i. The numerical value of s_i is broadcasted across the entire corresponding input channel.
- Optionally, a local ($k_c \times k_c$) crop $\widehat{\Phi}^z$ of a layout map for the abstract state space $\mathcal{Z}$, e.g. indicating obstacles, centered on the current abstract state z. This discrete environment map Φ is similar to the prior knowledge in other works [Tamar *et al.*, 2016; Nardelli *et al.*, 2019; Christen *et al.*, 2021; Wöhlke *et al.*, 2021].

In our experiments, we implement the VRN as a CNN with 2 convolutional layers (no padding, stride one) with 3×3 kernels and 16 and 32 feature maps, respectively. The 3×3 feature maps resulting from convolving the 7×7 input ($k_c = 7$), are flattened and passed through 2 fully-connected layers with 64 hidden neurons each. A linear output layer outputs the refined Q-values. We use ReLU activation.

Policy using a VRN

Based on the refined values, we can select (abstract) actions o using an (ϵ-)greedy policy:

$$\omega_\psi\left(o|s, \widetilde{V}_p^{\mathcal{Z}}\right) := \arg\max_o Q(s,o) \tag{3}$$

$$= \arg\max_o f_\psi\left(\widehat{V}_p^z, s, \left(\widehat{\Phi}^z\right)\right) \tag{4}$$

In continuous (state and) action spaces, o may represent a sub-task to be solved by some (learned) controller $\pi(a|s,o)$.

Training

Overall, we want to optimize the objective shown in Eq. 1 with respect to the parameters ψ of the VRN (and optionally the parameters θ of a potential low-level controller):

$$\max_{\psi,(\theta)} \mathbb{E}_{m\sim\mathcal{M}, s_{0,m}\sim\mathcal{S}_{0,m}, g_m\sim\mathcal{S}_{g,m}, \mathcal{P}_m}\left[\sum_{t=0}^{T}\gamma^t r(s_t, g_m)\right] \tag{5}$$

The VRN parameters ψ are optimized via double DQN [Van Hasselt *et al.*, 2016] using the Adam optimizer [Kingma and Ba, 2014] with a learning rate of 1×10^{-4}, a batch size of 128, and gradient clipping to $[-1, 1]$. The memory capacity is 160000. The target network update frequency is 1000 (5000 for Reacher). The discount factor γ is 0.99 (0.98 for Reacher). We furthermore use Hindsight Experience Replay (HER) [Andrychowicz *et al.*, 2017] with, depending on the environment, the 'final' or the 'future' strategy.

For additional details see appendix Sec. B.1.

Properties

For a finite $\mathcal{S}$ and $\mathcal{O} = \mathcal{A}$, we compare VRN's properties on the abstract state space $|\mathcal{Z}| \ll |\mathcal{S}|$ to value iteration (VI) and (Dijkstra) shortest path planning (SP) on the full $\mathcal{S}$, in Tab. 2. Please note regarding the computational complexities: VI does not scale quadratically but linearly in the state space size since we use local transition models that only allow transitions to neighboring states $s' \in \mathcal{N}(s)$. In case of our VRN, the refinement is one neural network forward pass ($O(1)$). Over the course of an episode, we need to refine the initial value function "prior" up to T times: $O(T)$. Additionally, to generate the "prior" we also use VI, but in the smaller abstract state space $\mathcal{Z}$. If the individual state components are of somewhat similar size ν, then the (abstract) state space size roughly scales exponentially in the number of state components ($\approx |\nu^n|$). Therefore, a reduction by a few components already has a significant impact. For a detailed discussion, please consult appendix Sec. C.

5 Experimental Evaluation

In the previous sections, we motivated the local refinement of an initial value function and presented a neural network architecture, the VRN, which we claim is capable of learning such a transformation. We will now empirically evaluate our VRN on challenging simulated robotic tasks (in depth details in appendix Sec. A), investigating hypotheses *H.1* and *H.2*.

We obtain the 2D value "prior" $\widetilde{V}_p^{\mathcal{Z}}$ via value iteration assuming an optimistic transition model (assuming abstract actions o always successful unless obstacle), which results in shortest path plans. For the continuous state and action space

Method	Stochastic Environments	Exact Model Required	Computational Complexity
VI	yes	yes	$O(\lvert\mathcal{N}\rvert\lvert\mathcal{S}\rvert\lvert\mathcal{A}\rvert T)$
SP	no	yes	$O(\lvert\mathcal{S}\rvert\lvert\mathcal{A}\rvert + \lvert\mathcal{S}\rvert \log \lvert\mathcal{S}\rvert)$
VRN	yes	no	$O(T + \lvert\mathcal{N}\rvert\lvert\mathcal{Z}\rvert\lvert\mathcal{A}\rvert T)$

Table 2: VRN Properties

tasks, we employ our VRN as well as the baseline abstract planning approaches (SP PLAN BSL, DQN) within a hierarchical policy architecture similar to the one presented in [Wöhlke *et al.*, 2021] (see appendix Sec. B). All hierarchical approaches feature the same sub-goal guided TRPO [Schulman *et al.*, 2015] policy.

5.1 Motivating Example Revisited

In order to empirically demonstrate that our VRN can learn the local refinement procedure presented in the motivating example in Sec. 4.2, we train it on the same environment. Our VRN refines the same initial 2D value "prior" with respect to the agent orientation (sine and cosine provided as additional input channels). Choosing a crop size of 7×7 ($k_c = 7$), corresponding to a three step look ahead, we would expect a similar performance as "VI 2D + 3 iters 3D".

We also conduct a small ablation to analyze the success factors of our VRN: By training "VRN NO PRIOR" without the initial value function, we investigate the importance of this "prior" information. Comparing to a "DQN (PRIOR)" baseline that receives the entire maze (and value "prior"), we study the influence of the specific VRN architecture.

Figure 3 shows the success rate over the course of the training.[2] Indeed, our VRN ultimately achieves a similar success rate as "VI 2D + 3 iters 3D" and is therefore capable of learning the local refinement (*H.1*). The poor performance of "VRN NO PRIOR" emphasizes the reliance on a "prior" for refinement. "DQN (PRIOR)", which uses the same information as the VRN, but globally, performs significantly worse on the task.[3] Hence, we conclude that the specific VRN architecture is key to utilizing the available value "prior".

5.2 Continuous Stick Robot Maze Navigation

We evaluate the absolute performance of our VRN on continuous state and action spaces by comparing to sampling-based planners that additionally require and exploit an available dynamics model. For this, we chose the "3D" (x, y, orientation) stick robot maze navigation task introduced and made available by Chen *et al.* [2019]. It features 2000 different training and 1000 different test problems (layouts). We provide a sparse reward of one for successful navigation. The abstract state space $\mathcal{Z}$ for initial planning emerges from discretizing the continuous x, y-coordinates into 30×30 tiles. Our VRN receives all three continuous state components in additional input channels besides the value "prior" and map crops.

[2]For easy comparison, one "training iteration" means collecting a "batch size" (3200 here) of data from multiple episodes. The VRN parameters are updated multiple times per "training iteration".

[3]We tuned hyper-parameters on a simpler layout maze achieving reasonable success rates (see appendix Sec. A.1 and Fig. 1b).

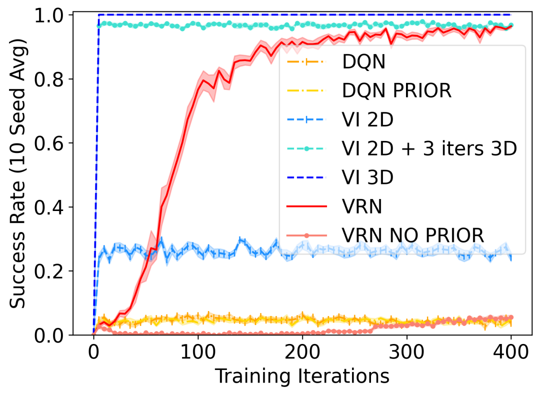


Figure 3: Grid-World Maze Navigation

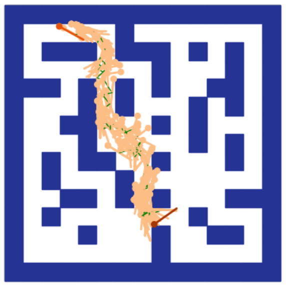
Figure 4c shows the success rates across the training. We compare to a "SP PLAN BSL", which is basically the initial 2D shortest path value function without refinement. Furthermore, we plot (tagged with (x)) the success rates of NEXT and several baselines as reported in [Chen *et al.*, 2019].

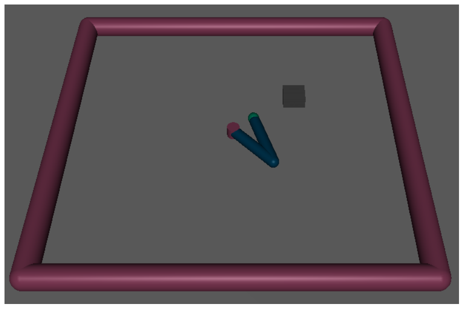
Looking at the results, our VRN achieves higher success rates than SP PLAN BSL, and hence successfully refines the value "prior". Compared to the NEXT planners of Chen *et al.* [2019], with $84\,\%$ and $94\,\%$, our VRN performs on average a bit worse. Still, the VRN achieves up to $88\,\%$ success rate.[4] It should be taken into account that our VRN does not have access to the simulation (apart from the static map), which NEXT needs to globally sample states or re-wire the tree (of the underlying RRT). Furthermore, our VRN learns from environment interaction without NEXT's safety net of a fallback RRT planner guaranteeing to eventually solve the task and acquire successful training episodes.

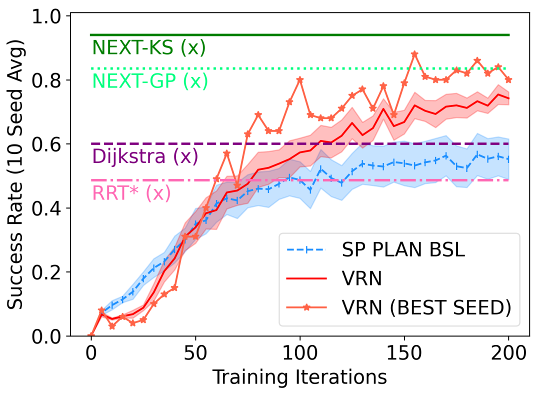
5.3 Robotic Manipulation with Random Obstacle

Apart from robotic navigation tasks, our VRN is also applicable to continuous state and action space robotic manipulation tasks. We therefore evaluate on a (modified) MuJoCo [Todorov *et al.*, 2012] "Reacher" task (details in appendix Sec. A.3.), where a two link robotic arm has to reach a goal. Instead of the standard dense distance-based reward, we use a sparse reward of one for success. Furthermore, we randomly place an obstacle cube in the quadrant of the goal location (see Fig. 4b). As opposed to the previous experiments, we do not plan and refine in the workspace but in the (2D) joint space, without access to a joint space "layout map". For the state abstraction $\mathcal{Z}$, we tile the joint angles into $10°$ segments.

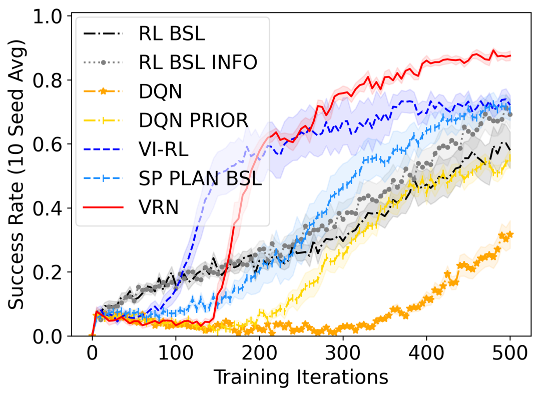
[4]Note that Chen *et al.* [2019] report a single training run.


(a) Stick Robot Maze Task Example ($T = 500$, $H = 5$)


(b) Reacher with Randomly Placed Obstacle ($T = 100$, $H = 5$)



(c) Stick Robot Maze Navigation



(d) Robotic Manipulation with Obstacle

Figure 4: Plots for robotic tasks. Success rate plots are: mean (solid line) $\pm$ standard error (shaded area). Note that the reported planner performances tagged with (x) had, compared to the other approaches, additional access to the simulator, which is not given in an RL setting.

The VRN receives the x, y obstacle location in the workspace along the continuous joint angle (sine and cosine) values and the joint angle velocity values in additional input channels for refining the initial (joint space) value map crop.

Fig. 4d shows the success rates. Our VRN successfully manages to refine the joint space shortest path value "prior", increasing the success rate compared to the SP PLAN BSL and achieving roughly $90\,\%$. We do not check if the obstacle renders the goal unreachable which might explain a success rate below $100\,\%$. The vanilla RL BSL (TRPO) even with information (INFO) about the obstacle position does not match this performance with reasonable amounts of training data. While quickly improving the success rate in the beginning, the hierarchical VI-RL [Wöhlke *et al.*, 2021] baseline does not match the asymptotic VRN performance. DQN (PRIOR) struggles with processing the entire abstract state space at once while facing the sparse rewards, whereas our VRN only locally refines the "prior".

5.4 Non-Holonomic Vehicle Parking

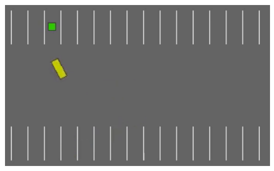
Finally, we evaluate our VRN on two difficult vehicle parking tasks with complex non-holonomic dynamics using *highway-env* [Leurent, 2018]. As shown in Fig. 5a, a yellow vehicle (6D state space) starting in the middle of the parking lot needs to park in the slot marked in green, which is selected at random. The vehicle receives a sparse reward of one for successfully parking. The abstract high-level state space $\mathcal{Z}$ results from tiling the continuous x, y-vehicle position into 24×12 tiles (not including the orientation and speed).

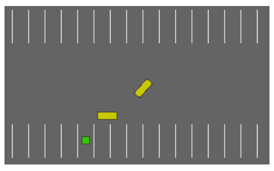
We compare our VRN to a number of sensible baselines: RL BSL (TRPO; used as sub-goal guided policy for the hierarchical approaches), DQN (with and without value "prior"), HIRO [Nachum *et al.*, 2018] (SotA HRL baseline), VI-RL [Wöhlke *et al.*, 2021] (plans in 3D abstract state space including discretized orientation (eight $45°$ segments)), and SP PLAN BSL (uses unrefined 2D value "prior"). For more details see appendix Sec. B.

In this parking task, our VRN refines the initial 2D value function based on the continuous x- and y-vehicle velocity as well as the sine and cosine of the orientation. The results are depicted in Fig. 6a. Only our VRN and VI-RL are able to efficiently learn to solve the parking task achieving roughly $90\,\%$ success rate. However, VI-RL needs to apply costly value iteration in a 3D state space abstraction whereas the local VRN refinement enables VRN to match the performance while only requiring "prior" value planning in a lower dimensional 2D space (*H.1*). SP PLAN BSL without the VRN refinement only achieves up to $30\,\%$ success rate. DQN, even with the 2D value "prior", does not match the performance of VRN, although showing some learning progress.

5.5 Parking with Moving Vehicle

In order to evaluate the ability of our VRN to consider latest state information (sensor measurements) during refinement to


(a) Parking Environment


(b) With Moving Vehicle

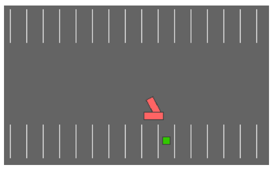
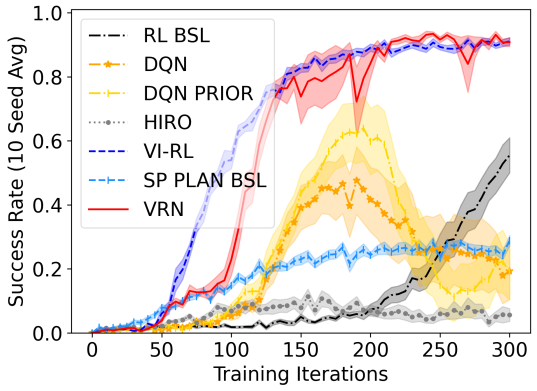
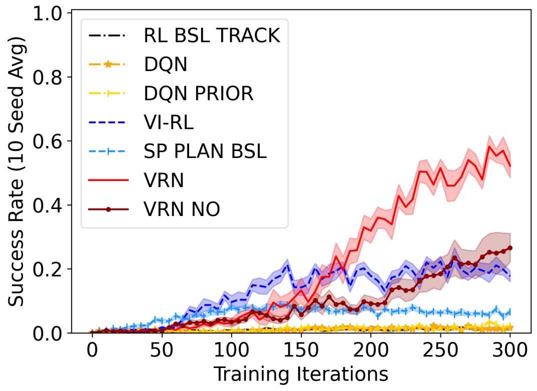

(c) Collision Example

Figure 5: Exemplary visualization of the vehicle parking tasks ($T = 100$, $H = 2$)



(a) Standard Parking Task



(b) Parking with Moving Vehicle

Figure 6: Plots for vehicle parking tasks: mean (solid line) $\pm$ standard error (shaded area)

handle dynamically changing environments, we modify the parking task. As shown in Fig. 5b, we add a second vehicle, which considerably increases the difficulty of successful parking. It drives with a constant velocity randomly drawn from the interval $[1.5, 3.5]\,\mathrm{m/s}$ horizontally in front of the parking slot row containing the goal. Upon collision, the velocity of the "ego" vehicle is set to zero until the collision is resolved.

Our VRN receives "sensory information" about the moving vehicle by marking all tiles occupied by it in the input channel containing the local map crop in case the other vehicle is close enough. Similarly, DQN (PRIOR) receives entries in the map input. We adjust the RL BSL by providing additional inputs containing a positional difference vector to the other vehicle as well as its velocity and orientation. VI-RL could in principle account for the other vehicle by repeated re-planning with updated map information. However, this is computationally prohibitive, increasing the training time by roughly a factor ten (from $\approx 14\,\mathrm{h}$ to $\approx 120\,\mathrm{h}$ per seed). On the contrary, our VRN only requires a single network forward pass at each time step to account for the new situation.

Figure 6b shows the success rates for parking with the moving vehicle. Our VRN, by locally incorporating the sensing information about the other vehicle into its refinement, is the only approach to increase the success rate to about $60\,\%$. This result provides empirical evidence for our hypothesis that the VRN can perform well in dynamically changing environments, without re-planning, by incorporating latest state information (*H.2*). VRN NO without the obstacle information performs clearly worse, only achieving a bit over $20\,\%$ success rate, similar to VI-RL. The DQN and TRPO baselines do not perform well in this dynamic setting.

6 Conclusion

In this work, we propose the Value Refinement Network (VRN), a network architecture to locally modify an initial plan in a simple 2D state space abstraction, represented by a value function, based on the full (continuous) agent state. Training the VRN via reinforcement learning, it learns to effectively refine this "prior" plan to solve tasks that otherwise would require a larger state space abstraction for planning or computationally costly repeated re-planning.

The results of evaluating our VRN on different robotic navigation and manipulation tasks support our research hypotheses: Our VRN successfully refines initial (shortest path) plans, represented by (2D) value functions, matching the performance of directly planning in a more complex state space. Furthermore, in vehicle parking tasks with a dynamically changing environment, where considering all relevant state space dimensions in planning is infeasible, our VRN successfully incorporates latest state information to maintain high performance without the necessity for re-planning.

A potential limitation of the work in its current form is the focus on refining initial value functions represented by a regular 2D grid, which are especially suited for spatial navigation tasks. Therefore, an interesting avenue for future research is to explore network architectures allowing for refinement of different state space abstractions like, for example, 3D tensors for 3D workspaces or general graph structures.

References

[Andrychowicz *et al.*, 2017] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in neural information processing systems*, pages 5048–5058, 2017.

[Bacon *et al.*, 2017] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *AAAI*, 2017.

[Bellman, 1957] Richard Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[Chen *et al.*, 2019] Binghong Chen, Bo Dai, Qinjie Lin, Guo Ye, Han Liu, and Le Song. Learning to plan in high dimensions via neural exploration-exploitation trees. In *ICLR*, 2019.

[Christen *et al.*, 2021] Sammy Christen, Lukas Jendele, Emre Aksan, and Otmar Hilliges. Learning functionally decomposed hierarchies for continuous control tasks with path planning. *IEEE Robotics and Automation Letters*, 6(2):3623–3630, 2021.

[Florensa *et al.*, 2017] Carlos Florensa, David Held, Markus Wulfmeier, Michael Zhang, and Pieter Abbeel. Reverse curriculum generation for reinforcement learning. In *Conference on Robot Learning*, pages 482–495, 2017.

[Francis *et al.*, 2020] Anthony Francis, Aleksandra Faust, Hao-Tien Chiang, Jasmine Hsu, J. Chase Kew, Marek Fiser, and Tsang-Wei Edward Lee. Long-range indoor navigation with PRM-RL. *IEEE Transactions on Robotics*, 2020.

[Haber *et al.*, 2018] Nick Haber, Damian Mrowca, Stephanie Wang, Li F Fei-Fei, and Daniel L Yamins. Learning to play with intrinsically-motivated, self-aware agents. In *Advances in Neural Information Processing Systems*, pages 8388–8399, 2018.

[Ichter *et al.*, 2018] Brian Ichter, James Harrison, and Marco Pavone. Learning sampling distributions for robot motion planning. In *ICRA*, pages 7087–7094. IEEE, 2018.

[Kingma and Ba, 2014] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[Lee *et al.*, 2018] Lisa Lee, Emilio Parisotto, Devendra Singh Chaplot, Eric Xing, and Ruslan Salakhutdinov. Gated path planning networks. In *ICML*, pages 2947–2955, 2018.

[Leurent, 2018] Edouard Leurent. An environment for autonomous driving decision-making. https://github.com/eleurent/highway-env, 2018. Accessed: 2022-05-21.

[Levy *et al.*, 2019] Andrew Levy, George Konidaris, Robert Platt, and Kate Saenko. Learning multi-level hierarchies with hindsight. In *ICLR*, 2019.

[Mnih *et al.*, 2015] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[Nachum *et al.*, 2018] Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 3303–3313, 2018.

[Nardelli *et al.*, 2019] Nantas Nardelli, Gabriel Synnaeve, Zeming Lin, Pushmeet Kohli, Philip HS Torr, and Nicolas Usunier. Value propagation networks. In *ICLR*, 2019.

[Pathak *et al.*, 2017] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *Computer Vision and Pattern Recognition Workshops*, pages 16–17, 2017.

[Schulman *et al.*, 2015] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *ICML*, pages 1889–1897, 2015.

[Schulman *et al.*, 2016] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *ICLR*, 2016.

[Silver *et al.*, 2018] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419):1140–1144, 2018.

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

[Todorov *et al.*, 2012] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[Van Hasselt *et al.*, 2016] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI*, volume 30, 2016.

[Vezhnevets *et al.*, 2017] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. FeUdal networks for hierarchical reinforcement learning. In *ICML*, pages 3540–3549, 2017.

[Wöhlke *et al.*, 2020] Jan Wöhlke, Felix Schmitt, and Herke van Hoof. A performance-based start state curriculum framework for reinforcement learning. In *Autonomous Agents and MultiAgent Systems*, pages 1503–1511, 2020.

[Wöhlke *et al.*, 2021] Jan Wöhlke, Felix Schmitt, and Herke van Hoof. Hierarchies of planning and reinforcement learning for robot navigation. In *ICRA*, 2021.