VALUE REFINEMENT NETWORK (VRN)

Anonymous authors
Paper under double-blind review

ABSTRACT

Sparse rewards and long decision horizons make agent navigation tasks difficult to solve via reinforcement learning (RL) such as (deep) Q-learning. Previous work has shown that some of these tasks are efficiently solvable by value-based planning in a state space abstraction, which defines sub-goals for a policy in the original state space. However, value-based planning scales poorly with the number of state space dimensions. Consequently, the planning might not be able to consider all the state information, like other agents' behaviors. Combining the benefits of planning and learning values, we propose the Value Refinement Network (VRN), an architecture that locally refines a plan in a (simpler) state space abstraction, represented by a pre-computed value function, with respect to the full agent state. Training the VRN via RL, it can learn how to correct this initial plan effectively to solve tasks that otherwise would require a prohibitively large abstraction. Evaluating on several simulated agent navigation tasks, we demonstrate the benefits of our VRN: We show that it can successfully refine shortest path plans to match the performance of value iteration in a more complex state space. Furthermore, in vehicle parking tasks where considering all relevant state space dimensions in planning is infeasible, the VRN still enables high task completion rates.

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. Nevertheless, sparse reward signals and long decision horizons still pose an obstacle to achieving high performance levels and data-efficiency. Concepts like 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); Christen et al. (2021) demonstrated that complex sparse reward navigation tasks can be solved successfully by combining planning of sub-goals in some state space abstraction with an RL trained policy operating in the original state space guided by the sub-goals. Planning these sub-goals has the advantage of better generalization to unseen environment layouts and better data-efficiency over learning to generate them, for example, via some (deep) Q-learning. On the contrary, planning does not scale well in the number of state dimensions considered. The cost of planning may be prohibitively large spaces, compute resources, or runtimes. Furthermore, a plan generated once may not consider changes in the environment, like other moving agents. Costly re-planning or decreased performance are the consequences.

However, global planning is often only required for a small subset of the state variables. In case of a navigating agent, it is necessary to plan globally an overall route to the goal location, whereas other state variables like the velocity or orientation of the agent or other moving agents do not have a global relevance. 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 aim at refining an initial, global plan in a simple state space abstraction based on the current full state information. This approach shares similarity with how, for example, a driver would intuitively drive to a shop: First, she would probably obtain an initial route by querying the shortest route from a navigation system. While driving, she would follow this initial route unless some local observation like, for example, a novel construction site

blocking the road imposes a local constraint on the route. According to this latest information, the driver adjusts her route by taking a turn to avoid the situation. In summary, she effectively locally refined the initial route plan using the full amount of information available to her in the very moment.

This work proposes a novel neural network, the Value Refinement Network (VRN), which implements this promising strategy to solve navigation tasks by locally refining an initial, simple (shortest path) plan. 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 navigation, our VRN locally refines this value "prior" by passing a local crop of it and the layout alongside full (continuous) agent 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 to unseen environments and sample-efficiency of planning and the handling of high-dimensional state spaces by learning the local refinement.

In summary, the key contribution of this work is the Value Refinement Network (VRN) architecture. Evaluating on difficult agent navigation tasks, we empirically prove the following hypotheses:

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

H.2: Our VRN can successfully incorporate current state information to maintain high performance in the face of a dynamically changing environment, without the necessity of re-planning.

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, especially value-based planning does 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 some local refinement later on.

The idea of integrating full state information into plans in a lower-dimensional space is also pursued in the sampling-based NEXT planner of Chen et al. (2019). In order to choose the next continuous state for tree expansion, this approach uses an attention-based architecture that maps the state into a discrete latent space, which is then processed by a differential planning module. However, similar to the differentiable planning modules, it can handle dynamically changing environments only by repeated re-planning.

3 PROBLEM STATEMENT

In this work, we consider a distribution of Markov Decision Processes (MDPs) $\mathcal{M}$ that share the same state space $\mathcal{S} \subseteq \mathbb{R}^n$ and action space $\mathcal{A} \subseteq \mathbb{R}^u$. We can sample specific MDPs $m = (\mathcal{S}, \mathcal{A}, \mathcal{P}_m, r_m, \mathcal{S}_{0,m}, \gamma, T)$ that for example represent different (maze) environment layouts. Start states $s_{0,m}$ 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: $r(s, g_m) = \mathbb{I}_{d(s,g_m)\leq\epsilon}$ with goal states g_m sampled from a goal distribution $\mathcal{S}_{g,m} \subseteq \mathcal{S}$ and $d(\cdot,\cdot)$ being some distance measure. $\mathcal{P}_m$ 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. γ is the discount factor and T the time horizon.

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

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

State Space Abstraction: Similar to Wöhlke et al. (2021), we assume that there exists a finite state space $\mathcal{Z}$ that abstracts the original, not necessarily finite, state space $\mathcal{S}$. We assume that $\mathcal{Z}$ is much smaller than $\mathcal{S}$, $|\mathcal{Z}| \lll |\mathcal{S}|$, and that there exists a known mapping $z = f_{\mathcal{Z}}(s)$ that defines the abstraction. Also, we assume that the neighbors $\mathcal{N}(z)$ of any abstract state z are known and that $\mathcal{N}(z) \subset \mathcal{Z}$ contains at least all abstract states directly reachable from z. For this work, we specifically assume $\mathcal{Z}$ to be a regular 2D grid, which fulfills the aforementioned assumptions. In case of continuous state space agent navigation, the state space abstraction $\mathcal{Z}$ may results from discretization of the continuous x- and y-coordinates according to a chosen resolution.

4 MOTIVATING EXAMPLE

As a motivating example of why the approach of locally refining values can be effective, consider the following 25×25 grid-world maze navigation task with discrete states and actions. In addition to the x- and y-coordinate, we augment the agent state by an orientation component that can attain eight different values: 'North', 'North-East', 'East', 'South-East', 'South', 'South-West', 'West', and 'North-West'. Similar to the standard grid-world, the agent's actions correspond to the neighboring tiles. However, 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. Maze layouts are randomly generated. An exemplary generation is shown to the right, in Fig. 1. The generation process ensures that corridors are at least five tiles wide, in order to give the agent the opportunity to turn. Start and goal states are chosen at random from 16 different options uniformly distributed across the maze, which are shown in blue in Fig. 1. The time horizon for navigating the maze is set to $T = 100$.

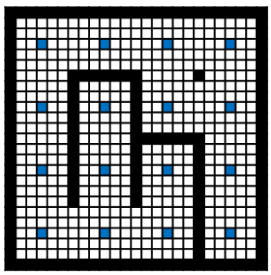


Figure 1: Exemplary maze layout generation; potential starts and goals shown in blue; reaching the goal tile results in a sparse reward of one

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 we can apply value iteration (VI) to the navigation, which we denote "VI 3D". Table 1 shows the success rate of navigating the maze with a greedy policy using the respective value iteration variant. Without any surprise, "VI 3D" results in an optimal policy for the task. However, 3D VI is costly memory- and compute-wise. We can also ignore the agent orientation and perform value iteration only with respect to the agent's position ("VI 2D"). As the underlying transition model neglects the motion constraints imposed by the orientation, the success rate is as low as $25.7\,\%$.

In the following, we will show that these sub-optimal but simple and cheap to obtain 2D plans only need slight local adjustments, accounting for the motion constraints, to become effective. First, observe that we can use the value function 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 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, as expected.

Table 1: Local value refinement example

Method	**Success rate** (mean $\pm$ std; 10 seeds)
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 + 10 iters 3D	0.995 ± 0.005
VI 2D + 100 iters 3D	1.000 ± 0.000
VI 3D	1.000 ± 0.000

Of course, performing exact value iteration steps will hardly be possible in practical applications because the transition model might be unknown or because the task features continuous state components. 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.

5 Value Refinement Network (VRN)

We will now propose a neural network architecture, the Value Refinement Network (VRN), which is a learned refinement operator taking the place of applying a few value iterations with the exact transition model to the initial value function "prior". First, we present the architecture of our VRN in Sec. 5.1, then explain how to employ it in a (hierarchical) policy architecture in Sec. 5.2, and finally shed light on the training procedure in Sec. 5.3.

As a prerequisite of our value refinement, we need access to an initial (2D) value function "prior" $V_p(s) = V_{\mathcal{Z}}(f_{\mathcal{Z}}(s))$ in the abstract state space $\mathcal{Z}$. In practice, we obtain it via value iteration with an optimistic transition model, similar as in the VI-RL OM approach in Wöhlke et al. (2021). Using our VRN, we refine this "prior" to state-action values $Q(s, a)$. Please note that both, the value "prior" as well as the refined values, depend on the MDP goal state g_m: $V_p(s, g_m)$ and $Q(s, a, g_m)$. For simplicity of notation, we omit this goal-dependency throughout the rest of this work.

5.1 Architecture

The value refinement network (VRN) is implemented as a convolutional neural network architecture consisting of convolutional and fully-connected layers with a distinct input representation $\mathcal{I}$. With the network parameters denoting as ψ, the VRN implements the function $Q(s, \cdot) = f_\psi(\mathcal{I}) = f_\psi\left(\widehat{V}_p^z, s, \left(\widehat{\Phi}^z\right)\right)$, outputting a vector of refined values for all actions a. 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" V_p, centered on the current abstract state $z = f_{\mathcal{Z}}(s)$.
- One additional input channel each for potentially any component (or only a subset thereof) of the current state vector s. The respective numerical value of state component 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 layout map Φ of the environment is similar to the assumed prior knowledge in other works (Tamar et al., 2016; Nardelli et al., 2019; Christen et al., 2021; Wöhlke et al., 2021).

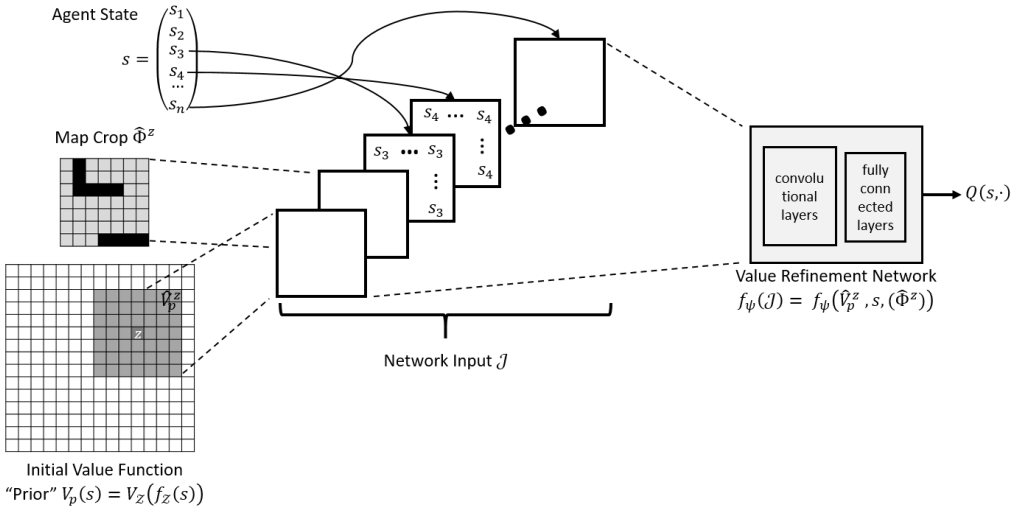


Figure 2: VRN architecture and input representation example (agent navigation): The first input channel is a $k_c \times k_c$ (with $k_c = 7$) crop $\widehat{V}_p^z$ of the value "prior" V_p, centered on the current abstract (high-level) state z of the agent. Given a discretized (x, y) map Φ of the environment, indicating (static) obstacles, a $k_c \times k_c$ crop $\widehat{\Phi}^z$, centered on z, forms the second input channel. Additional input channels are formed by selecting components s_i of the full (continuous) agent state s and inserting the value into every cell of the corresponding input channel (exemplarily implied for s_3 and s_4). Such state components s_i are, for example, the velocity of the agent in x and y and/or the sine and cosine of the agent orientation. Together, the input channels form the network input $\mathcal{I}$ provided to the VRN. The Value Refinement Network is implemented as a convolutional neural network (CNN) composed of some convolutional layers followed up by some fully-connected layers to compute the network output $Q(s, \cdot)$, a vector of refined state-action values.

Figure 2 shows an exemplary realization of the VRN and its input representation $\mathcal{I}$ as it could be used, for example, in an agent navigation task.

5.2 (Hierarchical) Policy using a VRN

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

$$\omega_\psi(a|s, V_p) := \arg\max_a Q(s, a) = \arg\max_a f_\psi\left(\widehat{V}_p^z, s, \left(\widehat{\Phi}^z\right)\right) \tag{2}$$

For continuous (state and) action spaces, we can employ our VRN as part of the high-level sub-goal planning of a hierarchical policy architecture similar to the VI-RL architecture proposed by Wöhlke et al. (2021). In this case, we have $\omega_\psi(z_g|s, V_p)$ with z_g being high-level sub-goals: The abstract (high-level) state $z_g \in \mathcal{N}(z)$ with the highest refined value serves as sub-goal for a low-level policy $\pi_\theta(a|s, f_\tau(z_g, s))$. We assume to have access to a function $f_\tau(z_g, s)$ that transforms the abstract sub-goal z_g into a (low-level) continuous target vector τ with respect to the current state s.

5.3 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 policy in a hierarchical policy architecture):

$$\max_{\psi,(\theta)} \mathbb{E}_{m\sim\mathcal{M}, s_{0,m}\sim\mathcal{S}_{0,m}, g_m\sim\mathcal{S}_{g,m}, \mathcal{P}_m}\left[\sum_{t=0}^{T} \gamma^t r(s_t, g_m)\right] \tag{3}$$

The VRN parameters ψ are optimized via double DQN (Van Hasselt et al., 2016) using Hindsight Experience Replay (HER) (Andrychowicz et al., 2017) with, depending on the environment, the 'final' or the 'future' strategy.

Algorithm 1 shows the algorithmic steps for training a VRN in a discrete action space scenario, like the one described in Sec. 4. See Alg. 2 in Sec. A.2.1 in the appendix for training a VRN as part of a hierarchical policy architecture for sub-goal selection.

Algorithm 1: VRN training

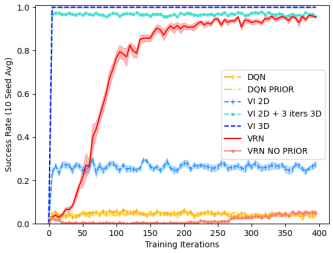
$i = 0, \mathcal{D} \leftarrow \emptyset, \psi_i = \psi_i^- \leftarrow$ `// initialize memory and VRN parameters`
while $i < i_{max}$ **do**
  $m \sim \mathcal{M}, s_{0,m} \sim \mathcal{S}_{0,m}, g_m \sim \mathcal{S}_{g,m}$ `// sample MDP` m`, start, and goal`
  $V_p\left(s, g_m\right) \leftarrow \mathrm{VI}_{\text{optimistic}}\left(\Phi_m, g_m\right)$ `// obtain value prior`
  $t = 0, s_t = s_{0,m}, d_t = \text{False}, D_{\mathcal{I}}, D_a, D_r, D_d, D_z \leftarrow \emptyset$ `// start episode in` m
  while $t < T$ *and* $\neg d_t$ **do**
    $\mathcal{I}_t \leftarrow V_p, s_t, \Phi_m$
    $Q\left(s_t, \cdot\right) = f_{\psi_i}\left(\mathcal{I}_t\right)$ `// VRN refines values`
    $a_t \leftarrow \text{epsilon-greedy}\left(Q\left(s_t, \cdot\right)\right)$ `// select action`
    $s_{t+1}, r_t, d_t \leftarrow \text{step}\left(s_t, a_t, \mathcal{P}_m, r_m\right)$ `// take step in the environment`
    $D_{\mathcal{I}} \leftarrow \mathcal{I}_t, D_a \leftarrow a_t, D_r \leftarrow r_t, D_d \leftarrow d_t, D_z \leftarrow f_{\mathcal{Z}}\left(s_t\right)$
    $s_t = s_{t+1}, t \leftarrow t + 1$
  $D_{\mathcal{I}} \leftarrow \mathcal{I}_t \leftarrow V_p, s_t, \Phi_m, D_z \leftarrow f_{\mathcal{Z}}\left(s_t\right)$
  for $j \leftarrow 0$ **to** $|D_a| - 1$ **do**
    $\mathcal{D} \leftarrow \left(\mathcal{I}_j, a_j, \mathcal{I}_{j+1}, r_j, d_j\right)$ `// add transitions to memory`
    $\mathcal{D} \leftarrow \mathrm{HER}_{\text{strategy}}\left(\mathcal{I}_j, a_j, \mathcal{I}_{j+1}, D_z, \Phi_m\right)$ `// generate HER transitions`
  for $j \leftarrow 0$ **to** $|D_a| - 1$ **do**
    $\boldsymbol{\mathcal{I}}^{\mathrm{b}}, \boldsymbol{a}^{\mathrm{b}}, \boldsymbol{\mathcal{I}}^{\mathrm{b}}_{\text{next}}, \boldsymbol{r}^{\mathrm{b}}, \boldsymbol{d}^{\mathrm{b}} \leftarrow \mathcal{D}$ `// sample batch from memory`
    $\mathcal{L} =$
    $\mathrm{MSE}\left(\boldsymbol{r}^{\mathrm{b}} + \left(1 - \boldsymbol{d}^{\mathrm{b}}\right) \gamma Q\left(\boldsymbol{\mathcal{I}}^{\mathrm{b}}_{\text{next}}, \arg\max_a Q\left(\boldsymbol{\mathcal{I}}^{\mathrm{b}}_{\text{next}}, \boldsymbol{a}^{\mathrm{b}}; \psi_i\right); \psi_i^-\right), Q\left(\boldsymbol{\mathcal{I}}^{\mathrm{b}}, \boldsymbol{a}^{\mathrm{b}}; \psi_i\right)\right)$
    $\psi_{i+1} \leftarrow \text{update}\left(\mathcal{L}, \psi_i\right)$ `// take a gradient step on VRN parameters`
    $i \leftarrow i + 1$
    if $i \mod$ *target network update frequency* $= 0$ **then**
      $\psi_i^- = \psi_i$

6 Experimental Evaluation

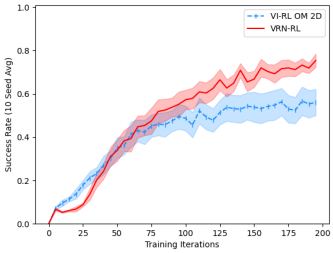
In the previous sections, we motivated the local refinement of an initial value function (in a simpler state space abstraction) and presented a neural network architecture, the VRN, which we claim is capable of learning such a transformation. We will now empirically evaluate our VRN on challenging agent navigation tasks, investigating hypotheses *H.1* and *H.2*. Please note that for the continuous state and action space navigation tasks, we employ our VRN within a hierarchical policy architecture (VRN-RL) similar to the one presented in Wöhlke et al. (2021) (further details in appendix Sec. A.2.1). The VRN is then part of the high-level sub-goal generation. By locally refining "prior" 2D shortest-path value functions with respect to the full (continuous) 3D or 6D agent state, it allows to efficiently obtain good values for sub-goal selection, even in the face of a dynamically changing environment. At the same time, costly planning operations are restricted to 2D, whereas without the refinement more (at least three) state dimensions need to be considered for sub-goal planning.

6.1 Discrete Maze Navigation Revisited

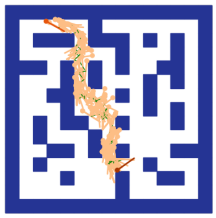
In our motivating example, in Sec. 4, we introduced a discrete state and action space maze navigation task featuring the agent orientation in the state space. We showed that an initial, sub-optimal 2D value function ("prior"), ignoring the orientation, can be refined effectively to an approximately optimal 3D value function by only applying few iterations of the computationally more costly 3D value iteration. In order to empirically prove that our VRN is capable of learning this local refinement procedure, we train it in the same environment having access to the same initial 2D value function "prior". Choosing a crop size of 7×7, corresponding to a three step look ahead, we would expect a similar performance as "VI 2D + 3 iters 3D". We furthermore conduct a small ablation to analyze



(a) Discrete Random Rooms



(b) Stick Robot Maze Navigation



(c) Stick Robot Example

Figure 3: Plots for maze navigation tasks: mean (solid line) ± standard error (shaded area)

the success factors of our VRN: The importance of the "prior" value information is investigated by training "VRN NO PRIOR" without this additional information. Furthermore, by training a "DQN (PRIOR)" baseline that globally receives maze and optionally also value "prior" information, we want to study the influence of the specific VRN architecture.

Figure 3a shows the success rate over the course of the training [1]. 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 [2]. Hence, we can conclude that the specific local refinement architecture of VRN is key to utilizing the available value "prior".

6.2 Continuous Stick Robot Maze Navigation

We increase the difficulty by evaluating our VRN on the continuous state and action space "3D" stick robot maze navigation task introduced and made available by (Chen et al., 2019). The state space consists of the x- and y-position as well as the orientation of the stick robot, which features very simple dynamics: $s' = s + a$ if the movement is collision free and $s' = s$ otherwise. Since this environment was originally proposed to evaluate (sampling-based) planners, we had to make slight modifications to use it for episodic RL. However, we keep the same time horizon $T = 500$. For the abstract high-level state space $\mathcal{Z}$ used for initial (2D) planning, the continuous x, y-coordinates are discretized into 30×30 tiles. Our VRN receives all three continuous state components as individual input channels (additionally to layout and value "prior" crop). Successfully reaching the goal configuration results in a reward of one. Otherwise, it is zero.

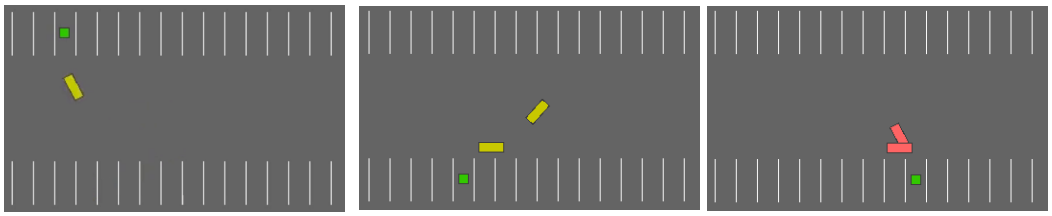
Table 2: Full test set evaluation

Method (training iteration)	**Success rate** (mean ± std; 10 seeds)
VI-RL OM 2D (100)	0.48 ± 0.17
VRN-RL (100)	0.57 ± 0.15
VI-RL OM 2D (200)	0.56 ± 0.20
VRN-RL (200)	0.74 ± 0.09

Figure 3b shows the success rate of VRN-RL over the course of the training. We also train a hierarchical VI-RL OM 2D (Wöhlke et al., 2021) baseline, which is basically VRN-RL without VRN refinement of the initial 2D value function, which reflects shortest paths. During training, maze layouts are repeatedly randomly sampled from the 2000 training layouts. For approximating the success rate, 100 layouts are randomly sampled from the 1000 test layouts. Looking at the results, VRN-RL is able to achieve higher success rates than VI-RL OM 2D, and hence successfully refines the value "prior". In addition to these training plots, we evaluated the VRN-RL and VI-RL OM 2D models after half and the entire training across the full 1000 test layouts. The results are shown in Tab. 2. VRN-RL performs consistently better than VI-RL OM 2D.

[1] Please note that for easy comparability with the other (hierarchical) approaches, later, one "training iteration" corresponds to collecting data from multiple episodes with respect to a chosen "batch size" (3200 here). Hence, within one "training iteration" the VRN parameters are updated multiple times.

[2] We tuned the hyper-parameters of DQN (as well as VRN) on a simpler layout of this maze navigation task and achieved reasonable success rates (see appendix Sec. A.1.1 and Fig. 6b)



(a) Parking Environment (b) Parking with Moving Vehicle (c) Parking Collision

Figure 4: Exemplary visualization of the vehicle parking tasks

Compared to the results presented for the NEXT planner in Chen et al. (2019), of $84\,\%$ and $94\,\%$ for their variants, respectively, VRN-RL performs a bit worse. Still, VRN-RL achieves up to $81\,\%$ success rate for the best seed in the full test set evaluation [3]. When looking at these numbers, it should be taken into account that our VRN-RL can be used without access to the simulation (apart from the static map), which NEXT needs to globally sample states or re-wire the tree (as part of the underlying RRT). Furthermore, our VRN-RL is able to learn the navigation from environment interaction without NEXT's safety net of a fallback RRT planner that guarantees to eventually solve the task and acquire successful training episodes.

6.3 Non-Holonomic Vehicle Parking

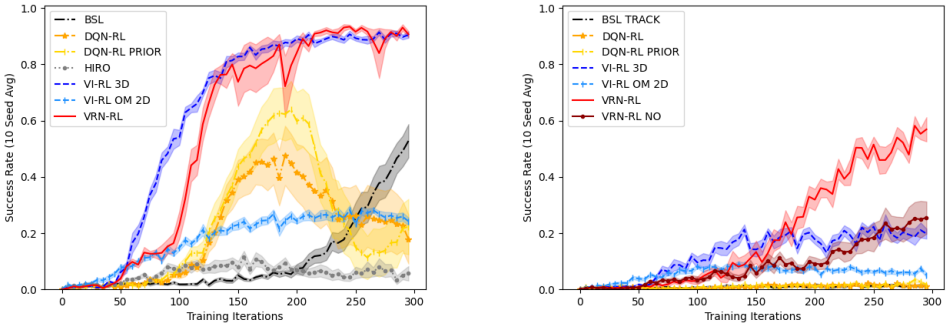
Finally, we evaluate our VRN on two difficult vehicle parking tasks with complex non-holonomic dynamics using *highway-env* (Leurent, 2018). We start with the parking task used in (Wöhlke et al., 2021), but only allow the vehicle to drive forward (to resolve orientation ambiguity in planning of the VI-RL 3D baseline). As shown in Fig. 4a, a yellow vehicle (6D state) 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 and zero otherwise. For the abstract high-level state space $\mathcal{Z}$ used for 2D value "prior" planning, we similar to Wöhlke et al. (2021) tile the x, y-position into 24×12 tiles, but do not include the orientation. Using VI with an optimistic transition model, we obtain initial 2D value functions, which reflect shortest paths, ignoring the non-holonomic motion constraints.

We compare our VRN(-RL) to a number of sensible baselines (more details in appendix Sec. A.2):

- **BSL**: A vanilla TRPO policy. Used as low-level policy for the hierarchical approaches.
- **DQN-RL (PRIOR)**: A DQN policy operating in the high-level state space abstraction $\mathcal{Z}$ to set sub-goals for low-level TRPO. Optionally, receives the initial 2D value function "prior".
- **HIRO** (Nachum et al., 2018): State-of-the-art hierarchical RL approach. We use the same implementation as in Wöhlke et al. (2021).
- **VI-RL 3D**: The hierarchical VI-RL approach as presented in Wöhlke et al. (2021). Performs value iteration with a learned transition model in a 3D high-level state space abstraction explicitly including the vehicle orientation to set sub-goals for low-level RL (TRPO).
- **VI-RL OM 2D**: VRN-RL without using the VRN to refine the initial value function. Directly selects high-level sub-goals based on the 2D value "prior", ignoring the orientation.

In this parking task, our VRN refines the initial 2D value function based on the continuous x- and y-velocity of the vehicle as well as the sine and cosine of the vehicle orientation. The results are depicted in Fig. 5a. Only our VRN-RL and VI-RL 3D are able to efficiently learn to convincingly solve the parking task achieving roughly $90\,\%$ success rate. However, VI-RL 3D needs to apply costly planning operations in a 3D state space abstraction whereas the local VRN refinement enables VRN-RL to match the performance while only requiring "prior" value planning in a lower dimensional 2D space (*H.1*). VI-RL OM 2D without the VRN refinement only achieves up to $30\,\%$ success rate. DQN-RL, even with the 2D value "prior", is not able to match the performance level of VRN-RL although showing some learning progress.

[3] Note that Chen et al. (2019) report a single run.



(a) Standard Parking Task

(b) Parking with Moving Vehicle

Figure 5: Plots for vehicle parking tasks: mean (solid line) $\pm$ standard error (shaded area)

Parking with Moving Vehicle: In order to showcase the ability of our VRN to consider latest state information (sensor measurements) during refinement to handle dynamically changing environments, we modify the parking task. As shown in Fig. 4b, we add a second vehicle, which considerably increases the difficulty of successful parking. The other vehicle drives horizontally in front of the parking slot row from which the goal is selected. The constant velocity is randomly sampled from the interval $[1.5, 3.5]\,\mathrm{m/s}$. Upon a collision of the two vehicles, the velocity of the controlled "ego" vehicle is set to zero until the collision is resolved.

Our VRN(-RL) 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-RL (PRIOR) receives entries in the map input. We furthermore adjust the vanilla TRPO BASELINE TRACK by providing additional inputs containing a positional difference vector to the other vehicle as well as its velocity and orientation. VI-RL 3D could in principle account for the other vehicle by repeated high-level re-planning with updated map information. However, this is computationally prohibitive, increasing the training time by roughly a factor ten. On the contrary, our VRN(-RL) only requires a single network forward pass at each time step to account for the new situation.

Figure 5b shows the success rates for parking with the moving vehicle. Our VRN(-RL), by locally incorporating the sensing information about the other vehicle into its refinement, is the only approach to convincingly 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-RL NO without the obstacle information performs clearly worse only achieving a bit over $20\,\%$ success rate, similar to VI-RL 3D. The DQN and TRPO baselines do not perform well at all in this dynamic setting.

7 Conclusion

In this work, we propose the Value Refinement Network (VRN), a network architecture to locally modify an initial plan in a simple 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 initial plan to solve tasks that otherwise would require a prohibitively large state space abstraction for planning or computationally costly repeated re-planning.

Evaluating our VRN on different agent navigation tasks, we arrive at the following conclusion regarding our research hypotheses: Our VRN can successfully refine initial (shortest path) plans, represented by a value function, matching the performance of directly planning in a more complex 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.

Reproducibility Statement We provide details on the algorithms and benchmark environments used for the empirical evaluation, in the appendix (see Sec. A). After potential acceptance of this work, we plan to open source the code.

REFERENCES

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*, pp. 5048–5058, 2017.

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

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

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 *International Conference on Learning Representations*, 2019.

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. doi: 10.1109/LRA.2021.3060403.

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

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.

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*, pp. 8388–8399, 2018.

Brian Ichter, James Harrison, and Marco Pavone. Learning sampling distributions for robot motion planning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7087–7094. IEEE, 2018.

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

Lisa Lee, Emilio Parisotto, Devendra Singh Chaplot, Eric Xing, and Ruslan Salakhutdinov. Gated path planning networks. In *International Conference on Machine Learning*, pp. 2947–2955, 2018.

Edouard Leurent. An environment for autonomous driving decision-making. `https://github.com/eleurent/highway-env`, 2018.

Andrew Levy, George Konidaris, Robert Platt, and Kate Saenko. Learning multi-level hierarchies with hindsight. In *International Conference on Learning Representations*, 2019.

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.

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

Nantas Nardelli, Gabriel Synnaeve, Zeming Lin, Pushmeet Kohli, Philip HS Torr, and Nicolas Usunier. Value propagation networks. In *International Conference on Learning Representations*, 2019.

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

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pp. 1889–1897, 2015.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *International Conference on Learning Representations*, 2016.

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.

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

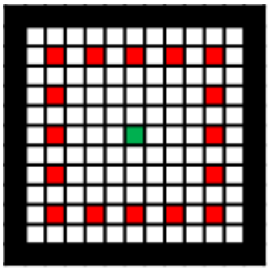
Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. FeUdal networks for hierarchical reinforcement learning. In *International Conference on Machine Learning*, pp. 3540–3549, 2017.

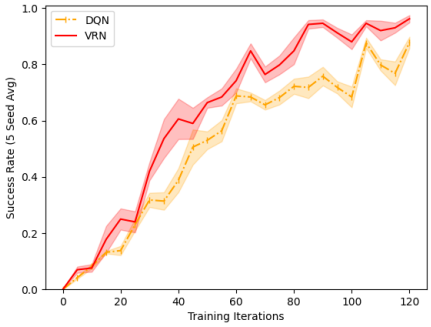
Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pp. 1995–2003. PMLR, 2016.

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

Jan Wöhlke, Felix Schmitt, and Herke van Hoof. Hierarchies of planning and reinforcement learning for robot navigation. In *IEEE International Conference on Robotics and Automation*, 2021.



(a) Layout



(b) Results

Figure 6: Empty Room Discrete Maze Navigation

A Appendix

A.1 Additional Environment Details

A.1.1 Discrete Maze Navigation

Wide Corridor Maze Generation: We first generate 9×9 random mazes. Then, we expand every other row and column (the corridors) so that one tile is replaced by five tiles of the same value, resulting in the wide corridors.

State Space $\mathcal{S}$: $s = (x, y, \vartheta)^T$ with $x \in \{0, ..., 24\}$, $y \in \{0, ..., 24\}$, and $\vartheta \in$ {'North', 'North-East', 'East', 'South-East', 'South', 'South-West', 'West', and 'North-West'}

State Space Abstraction $\mathcal{Z}$: $z = (x, y)^T$

Hindsight Experience Replay (HER) (Andrychowicz et al., 2017): For the discrete maze navigation, we generate four HER transitions per transition using the 'future' HER strategy, when training VRN or DQN. 'Future' means that we randomly sample four abstract states without replacement from the sequence of abstract states traversed during the episode that appear in the episode after the current abstract state z underlying the transition at hand. Each of the sampled states serves as the HER goal $z_{g,\text{HER}}$ for one corresponding HER transition that is added to the memory alongside the normal transition. If the abstract state z equals the HER goal $z_{g,\text{HER}}$, the reward of the HER transition is set to one and the done flag d to 'True'. The network inputs $\mathcal{I}$ and $\mathcal{I}'$ of the HER transitions are modified according to the HER goal $z_{g,\text{HER}}$: For DQN, the 'goal state channel' is adjusted. Furthermore, the value function 'prior' for VRN (and DQN) in $\mathcal{Z}$ is re-computed with the new goal $z_{g,\text{HER}}$.

Parameter Tuning on Simple Layout: We tuned the parameters of the double DQN (Van Hasselt et al., 2016) trainer of our VRN and the DQN baseline on a smaller, single-layout version, shown in Fig. 6a, of the discrete maze navigation task. With a size of only 13×13 position tiles, the horizon is set to $T = 20$. The agent always starts in the green tile in the middle with a randomly sampled orientation. The goal tile is uniformly sampled from the tiles shown in red when starting an episode. The dynamics are the same as for the bigger, random-layout version. The performance of our VRN and the DQN baseline for this simpler discrete maze navigation task is shown in Fig. 6b. Both are able to achieve a high success rate, which means that we found suitable hyper-parameters.

A.1.2 Continuous Stick Robot Maze Navigation

Mazes: We use the same sets of 2000 dedicated training and 1000 dedicated test maze navigation problems, respectively. Each problem consists of a 15×15 maze layout (example shown in Fig. 3c) with a specific start and a specific goal configuration for the stick robot.

State Space $\mathcal{S}$: $s = (x, y, \vartheta)^T$ with $x \in [-1, 1]$, $y \in [-1, 1]$, and $\vartheta \in [-0.4, 0.4]$ (Chen et al. (2019) normalized the angle interval $[-\pi, \pi]$ to this arbitrary interval)

State Space Abstraction $\mathcal{Z}$: $z = (\bar{x}, \bar{y})^T$ with $\bar{x} \in \{0, ..., 29\}$, $\bar{y} \in \{0, ..., 29\}$, regularly tiling the continuous x- and y-coordinate

Hindsight Experience Replay (HER) (Andrychowicz et al., 2017): We use the same 'future' strategy with four HER transitions as for the discrete maze navigation above.

A.1.3 Non-Holonomic Vehicle Parking

We allow only for forward driving by not allowing negative velocities in direction of the heading of the vehicle: $v' = \max(0, v + a \cdot \Delta t)$. If the (ego) vehicle leaves the parking lot (see x- and y-limits below), the episode terminates.

State Space $\mathcal{S}$: $s = (x, y, v_x, v_y, \cos(\vartheta), \sin(\vartheta))^T$ with $x \in [-0.48, 0.48]$, $y \in [-0.24, 0.24]$, and $\vartheta \in [-\pi, \pi]$

State Space Abstraction $\mathcal{Z}$ (for DQN-RL (PRIOR), VI-RL OM 2D, and VRN-RL (NO)): $z = (\bar{x}, \bar{y})^T$ with $\bar{x} \in \{0, ..., 24\}$, $\bar{y} \in \{0, ..., 12\}$, regularly tiling the continuous x- and y-coordinate

3D State Space Abstraction $\mathcal{Z}_{\text{3D}}$ (for VI-RL 3D): $z = \left(\bar{x}, \bar{y}, \bar{\vartheta}\right)^T$ with $\bar{\vartheta} \in \{0, ..., 7\}$

Hindsight Experience Replay (HER) (Andrychowicz et al., 2017): Due to better performance, we use the 'final' HER strategy for the non-holonomic vehicle parking tasks: The final abstract state reached in the episode serves as the only HER goal $z_{g,\text{HER}}$. Similarly to the 'future' strategy, a HER transition is added to the memory using $z_{g,\text{HER}}$ as goal and adjusting the transition accordingly.

Moving Vehicle: We initialize the moving vehicle differently, depending on whether the selected goal $g = (x_g, y_g, 0, 0, \cos(\vartheta_g), \sin(\vartheta_g))^T$ (parking slot) is in the upper or lower row. If it is in the upper row ($y_g < 0$), we initialize the moving vehicle at position $(x_g - 0.08, -0.08)$ with orientation $\vartheta = 0$. If the goal is in the lower row ($y_g > 0$), we initialize the moving vehicle at position $(x_g + 0.08, 0.08)$ with orientation $\vartheta = -\pi$. This specific initialization of the moving vehicle increases the likelihood of the ego vehicle having to deal with an encounter during parking. The "forward" velocity is uniformly sampled from the interval $[1.5, 3.5]$.

A.2 Additional Algorithm Details

A.2.1 VRN(-RL)

Network Architecture: The VRN is a CNN with two convolutional layers (no padding, stride one) with 3×3 kernels and 16 and 32 feature maps, respectively. The 3×3 features maps resulting from applying the convolutions to the 7×7 input, are flattened and passed through two fully-connected layers with 64 hidden neurons each. The linear output layer outputs a vector of eight (four in case of Stick Robot Navigation) refined Q-values. We use ReLU activation.

Double DQN (Van Hasselt et al., 2016) training:

- Memory capacity: 160000
- Optimizer: Adam (Kingma & Ba, 2014)
- Learning rate: 1×10^{-4}
- Batch size: 128
- Target update frequency: 1000
- Gradient clipping: $[-1, 1]$
- Discount γ: 0.99

Input representation $\mathcal{I}$ by environment:

- *Discrete Maze Navigation*: Channel containing the layout map crop $\widehat{\Phi}^z$, channel with cosine of orientation, channel with sine of orientation, channel containing the value function "prior" crop $\widehat{V}_p^z$
- *Stick Robot Maze Navigation*: Channel containing the layout map crop $\widehat{\Phi}^z$, channel with x, channel with y, channel with ϑ, channel containing the value function "prior" crop $\widehat{V}_p^z$
- *Non-Holonomic Vehicle Parking*: Channel containing the layout map crop $\widehat{\Phi}^z$ (with moving vehicle indicated as obstacle; marking all cells occupied by part of the moving vehicle), channel with v_x, channel with v_y, channel with $\cos(\vartheta)$, channel with $\sin(\vartheta)$, channel containing the value function "prior" crop $\widehat{V}_p^z$

For the discrete maze navigation, we obtain the "cosine and sine of orientation" the following way: We map the orientations 'North', 'North-East', 'East', etc. to the corresponding orientations: "North" $\vartheta = 0$, "North East" $\vartheta = \frac{\pi}{4}$, and analogously. Based on these orientations ϑ, we compute the cosine and the sine.

Low-level sub-goal / target vector-guided RL policy π_θ (in case of VRN-RL):

Similar to Wöhlke et al. (2021), we train a sub-goal-conditioned low-level policy $\pi_\theta(a|s, f_\tau(z_g, s))$ for low-level control via TRPO (Schulman et al., 2015). It is trained using the "sub-episodes" that arise within the training episodes of horizon T from repeated sub-goal selection of the high-level policy ω. The execution of a sub-goal z_g has a maximum time horizon of $T_{z_g} \ll T$, which is five for the stick robot maze navigation and two for the non-holonomic vehicle parking tasks. Upon reaching the sub-goal z_g / satisfying the target vector τ, the low-level policy receives are sparse intrinsic reward of one and otherwise zero. After reaching T_{z_g} or succeeding in the sub-goal navigation, the "sub-episode" ends and a new high-level sub-goal is selected.

TRPO hyper-parameters:

- Actor network: MLP with three hidden layers of 64 neurons each (ReLU activation)
- Critic network: MLP with three hidden layers of 64 neurons each (ReLU activation)
- GAE advantage estimator with $\lambda = 0.95$
- Max KL: 5×10^{-4}
- Damping: 5×10^{-3}
- Batch Size: 10000
- Discount: $\gamma = 0.99$

For the stick robot maze navigation, the policy output is passed through a Tanh activation multiplied by $\frac{1}{15}$ to limit the output range (twice the minimum corridor width in the maze).

Low-Level Target Vector Generation (in case of VRN-RL):

- *Stick Robot Maze Navigation*: The four refined Q-values correspond to directional targets 'North', 'East', 'South', and 'West'. According to the selected direction, the positional target for the low-level policy are the continuous x- and y-coordinates of the center of the corresponding neighboring high-level tile (i.e. $(\bar{x}, \bar{y}+1)^T$ for 'North'). Furthermore, an orientation target for the low-level policy matching the directional target is set: 0.2, 0, -0.2, and -0.4 for 'North', 'East', 'South', and 'West', respectively. The target vector τ is repeatedly updated with respect to the (low-level) state s: It is computed as the difference vector between the positional and orientational target and the corresponding components of s. In case the directional target corresponds to the abstract state corresponding to the goal configuration, the target vector τ is set to the difference between the goal configuration and the state s.
- *Non-Holonomic Vehicle Parking*: The eight refined Q-values correspond to directional targets 'North', 'North-East', 'East', 'South-East', 'South', 'South-West', 'West', and 'North-West'. According to the selected direction, the positional target for the low-level policy are the continuous x- and y-coordinates of the center of the corresponding neighboring high-level tile (i.e. $(\bar{x}+1, \bar{y}-1)^T$ for 'South-East'). Furthermore, an orientation target for the

low-level policy matching the directional target is set: Cosine and sine of the orientation ϑ matching the direction (i.e. $\vartheta = \frac{\pi}{4}$ for 'North-East', or $\vartheta = \frac{-3\pi}{4}$ for 'South-West'). The target vector τ is repeatedly updated with respect to the (low-level) state s: It is computed as the difference vector between the positional and orientational target and the corresponding components of s.

VRN-RL algorithm sketch: An overview over the algorithmic implementation of VRN-RL is provided in Alg. 2.

Algorithm 2: VRN-RL training (with low-level TRPO policy)

$i = 0, \mathcal{D}_H, \mathcal{D}_L \leftarrow \emptyset, \psi_i = \psi_i^-, \theta \leftarrow$ `// initialize memories and parameters`
for $n^{iter} \leftarrow 1$ *to* n_{max} **do**
  while $|\mathcal{D}_L| <$ *TRPO batch size* **do**
    $m \sim \mathcal{M}, s_{0,m} \sim \mathcal{S}_{0,m}, g_m \sim \mathcal{S}_{g,m}$ `// sample MDP` m`, start, and goal`
    $V_p(s, g_m) \leftarrow \mathrm{VI}_{\text{optimistic}}(\Phi_m, g_m)$ `// obtain value prior`
    $t = 0, s_t = s_{0,m}, d_t = \text{False}, D_{\mathcal{I}}, D_{z_g}, D_r, D_d, D_z \leftarrow \emptyset, k = 0, t_k = 0$ `// start episode in` m
    $\mathcal{I}_{t_0} \leftarrow V_p, s_t, \Phi_m$
    $Q(s_t, \cdot) = f_{\psi_i}(\mathcal{I}_{t_0})$ `// VRN refines values`
    $z_{g,t_0} \leftarrow \text{epsilon-greedy}(Q(s_t, \cdot))$ `// select sub-goal`
    while $t < T$ *and* $\neg d_t$ **do**
      $a_t \sim \pi_{\theta_{n^{\text{iter}}}}(s_t, f_\tau(s_t, z_{g,t_k}))$
      $s_{t+1}, r_t, d_t \leftarrow \text{step}(s_t, a_t, \mathcal{P}_m, r_m)$ `// take step in the environment`
      $D_z \leftarrow f_{\mathcal{Z}}(s_t)$
      $\mathcal{D}_L \leftarrow (s_t, a_t, r_t, s_{t+1})$
      $s_t = s_{t+1}, t \leftarrow t + 1$
      if $(f_{\mathcal{Z}}(s_t) = z_{g,t_k}) \vee (t - t_k = T_{z_g})$ **then**
        $D_{\mathcal{I}} \leftarrow \mathcal{I}_{t_k}, D_{z_g} \leftarrow z_{g,t_k}, D_r \leftarrow 0, D_d \leftarrow \text{False}$ `// sub-goal reached`
        $k \leftarrow k + 1, t_k = t$
        $\mathcal{I}_{t_k} \leftarrow V_p, s_t, \Phi_m$
        $Q(s_t, \cdot) = f_{\psi_i}(\mathcal{I}_{t_k})$ `// VRN refines values`
        $z_{g,t_k} \leftarrow \text{epsilon-greedy}(Q(s_t, \cdot))$ `// select new sub-goal`
    $D_{\mathcal{I}} \leftarrow \mathcal{I}_{t_k}, D_{z_g} \leftarrow z_{g,t_k}, D_r \leftarrow r_t, D_d \leftarrow d_t$
    $D_{\mathcal{I}} \leftarrow \mathcal{I} \leftarrow V_p, s_t, \Phi_m, D_z \leftarrow f_{\mathcal{Z}}(s_t)$
    for $j \leftarrow 0$ **to** $|D_{z_g}| - 1$ **do**
      $\mathcal{D}_H \leftarrow (\mathcal{I}_j, z_{g,j}, \mathcal{I}_{j+1}, r_j, d_j)$ `// add transitions to memory`
      $\mathcal{D}_H \leftarrow \mathrm{HER}_{\text{strategy}}(\mathcal{I}_j, z_{g,j}, \mathcal{I}_{j+1}, D_z, \Phi_m)$ `// generate HER transitions`
    for $j \leftarrow 0$ **to** $|D_{z_g}| - 1$ **do**
      $\boldsymbol{\mathcal{I}}^{\text{b}}, \boldsymbol{z}_{\boldsymbol{g}}^{\text{b}}, \boldsymbol{\mathcal{I}}_{\text{next}}^{\text{b}}, \boldsymbol{r}^{\text{b}}, \boldsymbol{d}^{\text{b}} \leftarrow \mathcal{D}_H$ `// sample batch from memory`
      $\mathcal{L} =$
      $\mathrm{MSE}\left(\boldsymbol{r}^{\text{b}} + \left(1 - \boldsymbol{d}^{\text{b}}\right)\gamma Q\left(\boldsymbol{\mathcal{I}}_{\text{next}}^{\text{b}}, \arg\max_a Q\left(\boldsymbol{\mathcal{I}}_{\text{next}}^{\text{b}}, \boldsymbol{z}_{\boldsymbol{g}}^{\text{b}}; \psi_i\right); \psi_i^-\right), Q\left(\boldsymbol{\mathcal{I}}^{\text{b}}, \boldsymbol{z}_{\boldsymbol{g}}^{\text{b}}; \psi_i\right)\right)$
      $\psi_{i+1} \leftarrow \text{update}(\mathcal{L}, \psi_i)$ `// take a gradient step on VRN parameters`
      $i \leftarrow i + 1$
      if $i \mod$ *target network update frequency* $= 0$ **then**
        $\psi_i^- = \psi_i$
  $\theta_{n^{\text{iter}}+1} \leftarrow \mathrm{TRPO}(\theta_{n^{\text{iter}}}, \mathcal{D}_L)$ `// update low-level policy parameters`
  $\mathcal{D}_L \leftarrow \emptyset$

A.2.2 VI-RL OM 2D

VI-RL OM 2D works similar to VRN-RL just that no VRN value refinement takes place and the high-level sub-goals are selected based on the value function prior $V_{\mathcal{Z}}\left(f_{\mathcal{Z}}\left(s\right)\right)$ by treating the values in the 3×3 neighborhood of $z = f_{\mathcal{Z}}\left(s\right)$ as Q-values $Q_{\mathcal{Z}}\left(z, z_g\right)$.

A.2.3 DQN(-RL) (PRIOR)

Network Architecture: We use a basic CNN architecture with three convolutional layers with 5×5, 3×3, and 3×3 kernels, respectively, and strides of 2, 2, and 1, respectively. The layers have 16, 32, and 32 feature maps, respectively. The 3×3 features maps resulting from applying the convolutions to the 25×25 input (24×12 map for parking padded to size), are flattened and passed through fully-connected layers. Due to using a dueling DQN architecture (Wang et al., 2016), the flattened feature vector first passes through one fully-connected layer with 64 hidden neurons to then split into the value and the advantage head, which feature another fully-connected layer with 64 hidden neurons and a linear output layer, each. The output of the DQN architecture is a vector of eight Q-values. We also use ReLU activation.

Input representation by environment:

- *Discrete Maze Navigation*: Channel containing the maze layout map Φ (1 for free space, 0 for obstacle), channel with agent position 1 else 0, channel with goal position 1 else 0, channel with cosine of orientation, channel with sine of orientation
- *Non-Holonomic Vehicle Parking*: Channel containing the layout map Φ (with moving vehicle indicated as obstacle (similar to VRN-RL)), channel with agent position 1 else 0, channel with goal position 1 else 0, channel with v_x, channel with v_y, channel with $\cos\left(\vartheta\right)$, channel with $\sin\left(\vartheta\right)$

Additional channel with (non-cropped) value function "prior" in case of PRIOR option.

Training: DQN(-RL) (PRIOR) uses the same double DQN training algorithm as VRN(-RL), the same hindsight strategies, and the same overall training loop (including low-level policy, in case of DQN-RL) as in Alg. 1 and Alg. 2 just that the DQN network is employed instead of the VRN (and value "prior" generation takes only place, in case of PRIOR). It just uses a batch size of 64 instead 128.