Towards real-world navigation with deep differentiable planners

Appendix A.

A.1. Implicit assumptions of the VIN architecture

Differences between VIN and the idealised VI on a grid. Comparing eq. 1 in sec. 3 to eq. 2 in sec. 3.1, we note several differences:

1. The VIN value estimate V takes the maximum action-value Q across *all possible actions* $\mathcal{A}$, even illegal ones (e.g. moving into an obstacle). Similarly, the Q estimate is also updated even for illegal actions. In contrast, VI only considers legal actions for each state (cell), i.e. $\mathcal{A}(i, j)$.
2. The VIN reward $\widehat{R}$ is assumed independent of the action. This means that, for example, a transition between two states cannot be penalised directly; a penalty must be assigned to one of the states (regardless of the action taken to enter it).
3. The VIN transition probability is expanded into 2 terms, P^R which affects the reward and P^V which affects the estimated values. This decoupling means that they do not enjoy the physical interpretability of VI's P (i.e. probability of state transitions), and rewards and values can undergo very different transition dynamics.
4. Unlike the VI, the VIN considers the state transition translation-invariant. This means that it cannot model obstacles (illegal transitions) using P, and must rely on assigning a high penalty to the reward $\widehat{R}$ of those states instead.

A.2. Implementation details

A.2.1 Embodied pose states

One of our contributions is extending the VIN framework to accommodate embodied pose states, i.e. states which encode both position and orientation. We achieve this by augmenting the 2D state-space with an extra dimension for orientations. Table 1 shows correspondences between tensor dimensions of the positional method and the embodied method for each component of the architecture. X and Y are the size of the internal spatial discretisation of the environment, M is the internal discretisation of the orientation, A is the number of actions, and K is the kernel dimension for spatial locality.

Note that the value iteration step in CALVIN performs a 2D convolution of $\widehat{P}$ over a 2D value map in the case of positional states and a 3D convolution over a 3D value map with orientation in the case of embodied pose states. In the embodied case, the second dimension of $\widehat{P}$ corresponds to the orientation of the current state, and the third dimension corresponds to that of the next state.

Table 1. Comparison of individual components in the implementation of CALVIN for positional states and for embodied pose states.

	Positional	Embodied
State s	(x, y)	(θ, x, y)
VI step	Conv2d	Conv3d
$V(s)$	$X \times Y$	$M \times X \times Y$
$Q(s, a)$	$A \times X \times Y$	$A \times M \times X \times Y$
$\widehat{A}(s, a)$	$A \times X \times Y$	$A \times M \times X \times Y$
$\widehat{P}$	$A \times K \times K$	$A \times M \times M \times K \times K$
$\widehat{R}$	$A \times K \times K$	$A \times M \times M \times K \times K$

A.2.2 3D embeddings for geometric reasoning

Since the learnable functions ($\widehat{P}$, $\widehat{A}$ and $\widehat{R}$) in our proposed method (and other VIN-based methods) are 2D CNNs, their natural input is a 2D grid of m-dimensional embeddings, denoted $e_{tij} \in \mathbb{R}^m$, for time t and discrete world-space coordinates (i, j). This can be interpreted as a spatio-temporal map tensor. We then wish to project and aggregate useful semantic information from an image I_t, extracted by a CNN ϕ, into this tensor. This requires both knowledge of the camera position c_t and rotation matrix R_t, which we assume following previous work [4, 8, 11] (and which can be estimated from monocular vision [10]). Spatial projection also requires knowing (or estimating) the depths $d_t(p)$ of each pixel p in I_t (either with a RGBD camera as in our experiments, or monocular depth estimation [3]). We can then write the homogenous 3D coordinates of each pixel p in the absolute reference frame using projective geometry [6]:

$$[x_t(p),\, y_t(p),\, z_t(p),\, 1] = c_t + R_t K\, [p_1, p_2, d_t(p), 1]^\top, \tag{1}$$

where K is the camera's intrinsics matrix. Given these absolute coordinates of pixel p, we can calculate the closest map embedding e_{tij} to it, and thus aggregate the CNN embeddings $\phi(I_t)$ associated with all pixels close to a map cell. Inspired by PointNet [2], we choose mean-pooling for aggregation. Since we have spatial aggregation, we can easily

extend this framework to work spatio-temporally, aggregating information from past frames $t' \leq t$. More formally:

$$e_{tij} = \text{avg}_{t' \leq t} \{\phi_p(I_{t'}) : \tau i \leq x_{t'}(p) < \tau(i+1), \\ \tau j \leq y_{t'}(p) < \tau(j+1), \\ \tau k \leq z_{t'}(p) < \tau(k+1), p \in I_{t'}\} \quad (2)$$

where τ is the absolute size of each square grid cell, avg averages the elements of a set, and $\phi_p(I_{t'})$ retrieves the CNN embedding of image $I_{t'}$ for pixel p. Due to the similarity between Eq. 2 and a PointNet embedded on a 2D lattice, we named it Lattice PointNet (LPN). Other than the lattice embedding, there are other major differences from the PointNet: we apply it spatio-temporally with a causal constraint ($t' \leq t$), and the downstream predictors that take it as input ($\widehat{P}(e_t)$, $\widehat{A}(e_t)$ and $\widehat{R}(e_t)$) are 2D CNNs that can reason spatially in the lattice, as opposed to the PointNet's unstructured multi-layer perceptrons [2]. A related proposal for SLAM used spatial max-pooling but more complex LSTMs/GRUs for temporal aggregation [1, 7]. Another related work on end-to-end trainable spatial embeddings uses egospherical memory [9].

A.2.3 Architectural design of Lattice PointNet

The Lattice PointNet described in Appendix A.2.2 consists of three stages: a CNN that extracts embeddings from observations in image-space (image encoder), a spatial aggregation step (eq. 10 in sec. 4.2.2) that performs mean pooling of embeddings for each map cell, and another CNN that refines the map embedding (map encoder). The image encoder consists of two CNN blocks, each consisting of the following layers in order: optional group normalisation, 2D convolution, dropout, ReLU and 2D max pooling. The map encoder consists of 2D convolution, dropout, ReLU, optional group normalisation, and finally, another 2D convolution. The number of channels of each convolutional layer are $(80, 80, 80, 40)$ for MiniWorld and $(40, 40, 40, 20)$ for AVD respectively. The point clouds can consume a significant amount of memory for long trajectories. Hence, we use the most recent 40 frames for the 8×8 MiniWorld maze.

The input to the LPN is a 3-channel RGB image for the MiniWorld experiment, and a 128-channel embedding extracted using the first 2 blocks of ResNet18 pre-trained on ImageNet for the AVD experiment.

A.2.4 Architectural design of the CNN backbone

This CNN backbone is used in a control experiment in Appendix A.4.2 to show the effectiveness of the LPN backbone. In contrast to LPN which performs spatial aggregation of embeddings, the CNN backbone is a direct application of an encoder-decoder architecture that transforms image-space observations into map-space embeddings. Gupta *et al*. [4] employed a similar architecture to obtain their map embeddings. While they use ResNet50 as the encoder network, we used a simple CNN for the MiniWorld experiment to match the result obtained with LPN.

The CNN backbone consists of three stages: a CNN encoder, two fully-connected layers with ReLU to transform embeddings from image-space to map-space, and a CNN decoder. The encoder consists of 3 blocks of batch normalisation, 2D convolution, dropout, ReLU and 2D max pooling, and a final block with just batch normalisation and 2D convolution. The number of channels of each convolutional layer are $(64, 128, 128, 128)$, respectively.

The fully-connected layers take in an input size of $128 \times 5 \times 7$, reduces it to a hidden size of 128, and outputs either $128 \times 5 \times 5$ for the smaller maze or $128 \times 4 \times 4$ for the larger maze, which is then passed to the decoder.

The decoder consists of 3 blocks of batch normalisation, 2D deconvolution, dropout and ReLU, and a final block with just 2D deconvolution. The number of channels of each deconvolution layers are $(128, 128, 64, 20)$, respectively. The output size of the decoder depends on the map resolution, hence we chose appropriate strides, kernel sizes and paddings in the decoder network to match the output sizes of 30×30 and 80×80. This approach is not scalable to maps with high resolution or with arbitrary size, which is one of the drawbacks of this approach.

A.3. Experiment setup

A.3.1 Expert trajectory generation

Expert trajectories are generated by running an A* [5] planner from the start state to the target state. We assigned Euclidean costs to every transition in the 2D grid environments, and a cost of 1 per move for the MiniWorld and AVD environments. In the case of MiniWorld, an additional cost is assigned to locations near obstacles to ensure that the trajectories are not in close proximity to the walls.

A.3.2 Hyperparameter choies

Similarly to VIN [11] which uses a 2-layer CNN to predict the reward map, and GPPN [8], which uses a 2-layer CNN to produce inputs to the LSTM, CALVIN uses a 2-layer CNN as an available actions predictor $\widehat{A}(s, a)$. For each experiment, we chose the size of the hidden layer from $\{40, 80, 150\}$. 150 was used for all the grid environments, 80 for MiniWorld and 40 for AVD, partially due to memory constraints.

VIN has an additional hyperparameter for the number of hidden action channels, which we set to 40, which is sufficiently bigger than the number of actual actions in all of our experiments. While the kernel size K for VIN and CALVIN were set to 3 for experiments in the grid environment, it was noted in [8] that GPPN works better with larger kernel size. Therefore, we chose the best kernel size out

of $\{3, 5, 7, 9, 11\}$ for GPPN. For experiments on MiniWorld and AVD, there are state transitions with step size of 2, hence we chose $K = 5$ for VIN and CALVIN.

The number of value iteration steps k was chosen from $\{20, 40, 60, 80, 100\}$. For trajectory reweighting, β was chosen from $\{0.1, 0.25, 0.5, 0.75, 1.0\}$.

A.3.3 Rollout at test time

We test the performance of the model by running navigation trials (rollouts) on a randomly generated environment. At every time step, the model is queried the set of Q-values $\{Q(s, a) : a \in \mathcal{A}\}$ for the current state s, and an action which gives the maximum predicted Q value is taken.

While VIN is trained with $V^{(0)}$ initialised with zeros, in a true Value Iteration algorithm, the value function must converge for an optimal policy to be obtained. To help the value function converge faster under a time and compute budget, we initialise the value function with predicted values from the previous time step at test time with online navigation.

We set a limit to the maximum number of steps taken by the agent, which were 200 for the fully-known 15×15 grid, 500 for the partially known grid, 300 for MiniWorld (3×3), 1000 for MiniWorld (8×8), and 100 for AVD.

A.4. Additional experiments

A.4.1 Ablation study of removing loss components

CALVIN is trained on three additive loss components: a loss term for the predicted Q-values L_Q (sec.4.1.1), a loss term for the transition models L_P (sec.4.1.2), and a loss term for the action availability L_A (sec.4.1.3). We assessed the contribution of each loss component to the overall performance.

We conducted the experiments on the partially observable grid environment (sec. 5.1.2). The results in Tab. 2 indicate that all loss components, in particular the transition model loss, contributes to the robust performance of the network.

Table 2. Navigation success rate of CALVIN in partially observable 2D mazes with loss components removed.

Loss	$L_Q + L_P + L_A$	$L_Q + L_P$	$L_Q + L_A$
Success rate	92.2	84.1	8.3

A.4.2 Comparison of LPN against CNN backbone

We compared our proposed LPN backbone against a typical encoder-decoder CNN backbone as a component that maps observations to map embeddings. We evaluated the performance of the two methods for VIN, GPPN and CALVIN. In Tab. 3, we observe that LPN backbone is highly effective, especially for larger environments where long-term planning based on spatially aggregated embeddings is necessary.

Table 3. Navigation success rate on unseen 3D mazes (MiniWorld). Most methods do not generalise to larger mazes. The proposed LPN demonstrates robust performance in larger unseen mazes.

	CNN backbone			LPN backbone (ours)		
Size	VIN	GPPN	CALVIN	VIN	GPPN	CALVIN
3×3	89.4	73.1	75.2	90.3	91.3	**97.7**
8×8	0.6	18.3	8.6	41.2	33.3	**69.2**

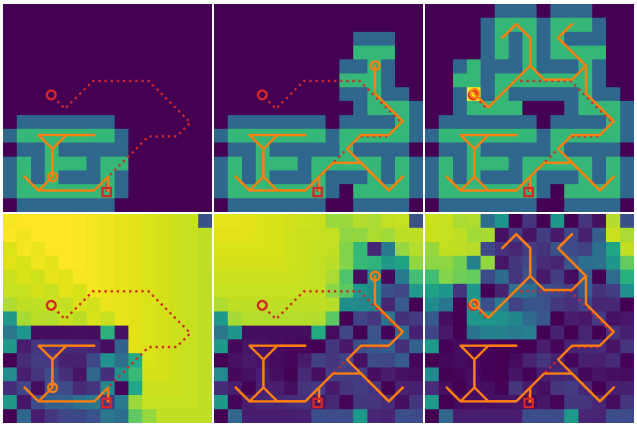

Figure 1. Example rollout of CALVIN after 21 steps (left column), 43 steps (middle column) and 65 steps (right column). CALVIN successfully terminated at 65 steps. **(top row)** Input visualisation: unexplored cells are dark, the discovered target is yellow. The correct trajectory is dashed, the current one is solid. The orange circle shows the position of the agent. **(bottom row)** Predicted values (higher values are brighter). Explored cells have low values, while unexplored cells and the discovered target are assigned high values.

A.5. Example rollout in a partially observable maze

We present an example of a trajectory taken by CALVIN at runtime, with corresponding observation maps and predicted values in Figure 1. At each rollout step, CALVIN performs inference on the best action to take based on its current observation map. No information about the location of the target is given until it is within view of the agent. This makes the problem challenging, since the agent may have to take significantly more steps compared to an optimal route to reach the target. In this example, the agent managed to backtrack every time it encountered a dead end, successfully reaching the target after 65 steps. The model initially assigns high values to all unexplored states. When the target comes into view, the model assigns a high probability to the availability of the "done" action at the corresponding state. The agent learns a sufficiently high reward for a successful termination so that the "done" action is triggered at the target.

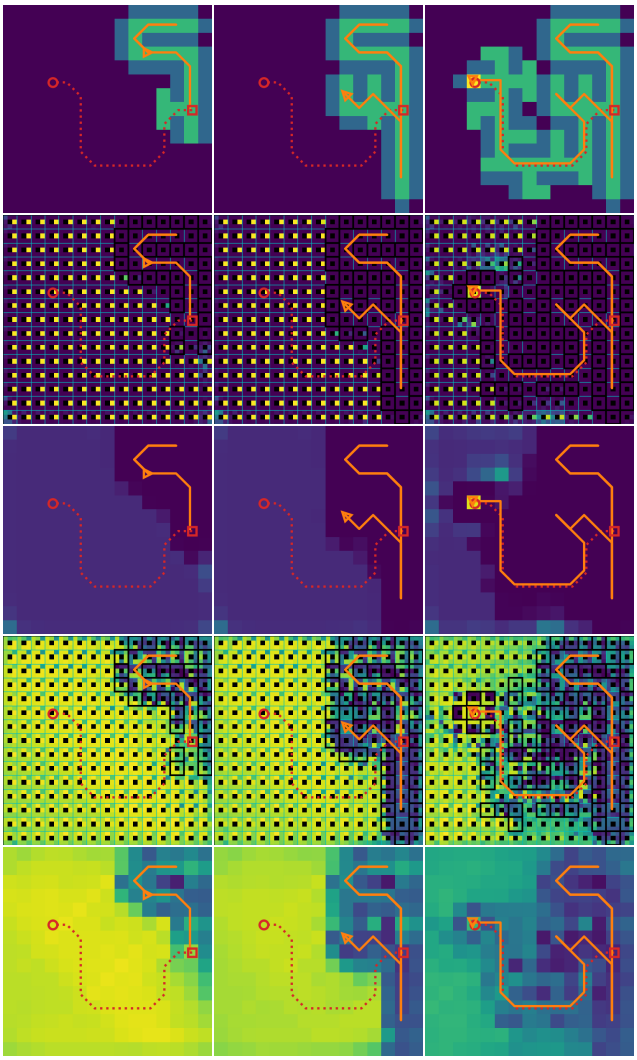

Figure 2. Example rollout of embodied CALVIN after 30 steps (left column), 60 steps (middle column) and 90 steps (right column). CALVIN successfully terminated at 91 steps. **(first row)** Input visualisation: unexplored cells are dark, the discovered target is yellow. The correct trajectory is dashed, the current one is solid. The orange triangle shows the position and the orientation of the agent. **(second row)** Predicted rewards (higher values are brighter). The 3D state-space (position/orientation) is shown, with rewards for the 8 orientations in a radial pattern within each cell (position). Explored cells have low rewards, while unexplored cells and the discovered target are assigned high rewards. **(third row)** Predicted rewards averaged over the 8 orientations. **(fourth row)** Predicted values following the same convention. Values are higher facing the direction of unexplored cells and the target (if discovered). **(fifth row)** Predicted values averaged over the 8 orientations.

A.6. Comparison of embodied navigation

For visual comparison of CALVIN, VIN and GPPN, we generated a maze and performed rollouts using each of the algorithms, assuming partial observability and embodied navigation.

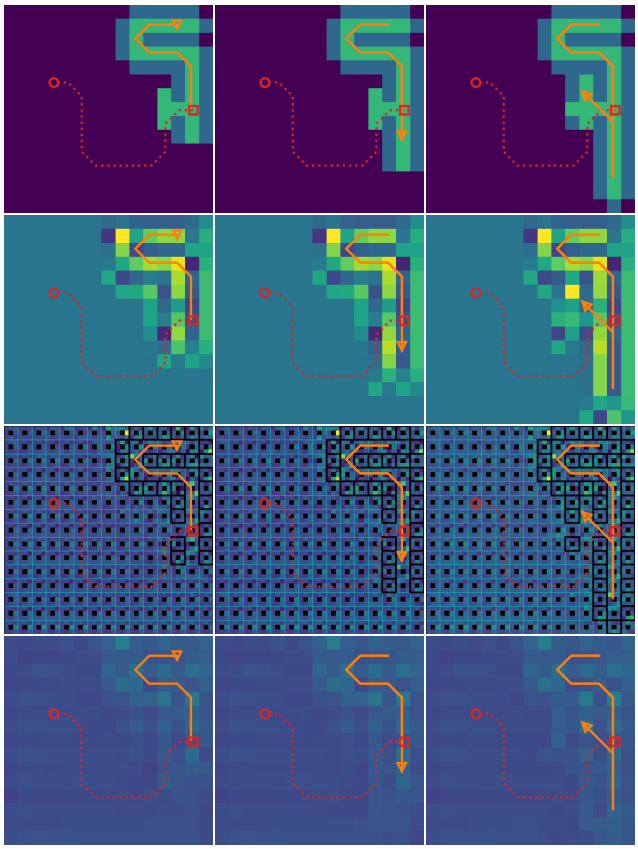

Figure 3. Example rollout of embodied VIN after 20 steps (left column), 40 steps (middle column) and 60 steps (right column). VIN kept oscillating between the same two states after 57 steps. The convention is the same as for Fig. 1, except that a single reward map is shared across all orientations. **(first row)** Input visualisation. **(second row)** Predicted rewards. **(third row)** Predicted rewards averaged over the 8 orientations. **(fourth row)** Predicted values.

A.6.1 Rollout of CALVIN

Figure 2 shows an example of a trajectory taken by CALVIN at runtime, with corresponding observation maps, predicted values and predicted rewards for taking the "done" action. Similarly to Appendix A.5, the agent manages to explore unvisited cells and backtrack upon a dead end until the target is discovered. One key difference is that now the agent learns to predict rewards and values for every discretised orientation as well as the discretised location. Upon closer inspection, we observe that the predicted values are higher facing the direction of unexplored cells and towards the discovered target. Since rotation is a relatively low cost operation, in this training example, the network seems to have learnt to assign high rewards to a particular orientation at unexplored cells, from which high values propagate. Rewards and values averaged over orientations yield a more intuitive visualisation.

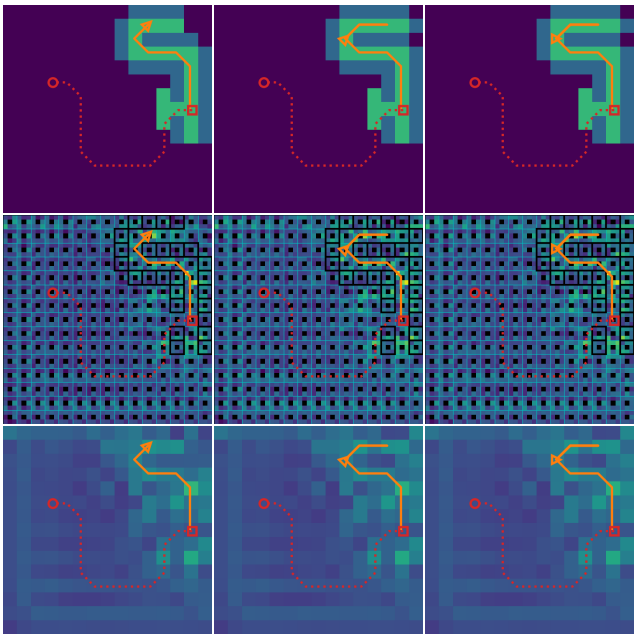

Figure 4. Example rollout of embodied GPPN after 15 steps (left column), 30 steps (middle column) and 45 steps (right column). GPPN revisits the same sequences of states leading to a dead end after 45 steps. The convention is the same as for Fig. 1. **(first row)** Input visualisation. **(second row)** Predicted rewards. **(third row)** Predicted rewards averaged over the 8 orientations.

A.6.2 Rollout of VIN

A corresponding visualisation for VIN is shown in Fig. 3. Unlike CALVIN's implementation of rewards (eq. 5 in sec. 4.1) as a function of discretised states and actions, the "reward map" produced by the VIN does not offer a direct interpretation, as it is shared across all actions as implemented by Tamar *et al*. [11], and is also shared across all orientations in the case of embodied navigation. The values are also not well learnt, with some of the higher values appearing in obstacle cells. The unexplored cells are not assigned sufficiently high values to incentivise exploration by the agent. In this example, the agent gets stuck and starts oscillating between two orientations after 57 steps.

A.6.3 Rollout of GPPN

Finally, a visualisation for GPPN is shown in Fig. 4. Unlike VIN and CALVIN, GPPN does not have an explicit reward map predictor, but performs value propagation using an LSTM before outputting a final Q-value prediction. Similarly to Appendix A.6.2, the values predicted is not very interpretable, and does not incentivise exploration or avoidance of dead ends. In this example, the agent keeps revisiting a dead end that has already been explored in the first 20 steps.

References

[1] Vincent Cartillier, Zhile Ren, Neha Jain, Stefan Lee, Irfan Essa, and Dhruv Batra. Semantic mapnet: Building allocentric semantic maps and representations from egocentric views, 2021. 2

[2] R. Qi Charles, Hao Su, Mo Kaichun, and Leonidas J. Guibas. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 77–85, Honolulu, HI, July 2017. IEEE. 1, 2

[3] Huan Fu, Mingming Gong, Chaohui Wang, Kayhan Batmanghelich, and Dacheng Tao. Deep ordinal regression network for monocular depth estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2002–2011, 2018. 1

[4] Saurabh Gupta, Varun Tolani, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive Mapping and Planning for Visual Navigation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017. arXiv: 1702.03920. 1, 2

[5] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 1968. 2

[6] R Hartley and A Zisserman. Multiple view geometry in computer. *Vision, 2nd ed., New York: Cambridge*, 2003. 1

[7] Joao F. Henriques and Andrea Vedaldi. MapNet: An Allocentric Spatial Memory for Mapping Environments. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8476–8484, June 2018. ISSN: 1063-6919. 2

[8] Lisa Lee, Emilio Parisotto, Devendra Singh Chaplot, Eric Xing, and Ruslan Salakhutdinov. Gated Path Planning Networks. *arXiv:1806.06408 [cs, stat]*, June 2018. arXiv: 1806.06408. 1, 2

[9] Daniel James Lenton, Stephen James, Ronald Clark, and Andrew Davison. End-to-end egospheric spatial memory. In *International Conference on Learning Representations*, 2021. 2

[10] Raul Mur-Artal and Juan D. Tardos. ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, Oct. 2017. arXiv: 1610.06475. 1

[11] Aviv Tamar, YI WU, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value Iteration Networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 2154–2162, 2016. 1, 2, 5