A Environment Details

We build on the Mujoco [1] environments used in [2]. The rewards in all experiments are sparse, i.e., 0 for reaching the goal and -1 otherwise. We consider the goal reached if $|s-g|_{\max} < 1$. All environments use $dt = 0.02$. Each episode in the simple maze navigation experiment (Section **??**) is terminated after 500 steps and after 800 steps for the complex maze experiment (Section **??**). In the robot manipulation experiment (Section **??**), we terminate after 800 steps and in the reacher experiments (Section **??**) after 200 steps.

A.1 Agents

Our ant agent is equivalent to the one in [3]. In other words, the ant from Rllab [4] with gear power of 16 instead of 150 and 10 frame skip instead of 5. Our ball agent is the PointMass agent from DM Control Suite [5]. We changed the joints so that the ball rolls instead of sliding. Furthermore, we resize the motor gear and the ball itself to match the maze size. For the manipulation robot, we slightly adapt the "Push" task from OpenAI gym [6]. The original environment uses an inverse kinematic controller to steer the robot, whereas joint positions are enforced and realistic physics are ignored. This can cause unwanted behavior, such as penetration through objects. Hence, we change the control inputs to motor torques for the joints. For the robot reacher task, we also adapt the version of the "Reacher" task from OpenAI [6]. Instead of being provided with euclidean position goals, the agents needs to reach goals in the robot's joint angle configuration space.

A.2 Environments

A.2.1 Navigation Mazes

All navigation mazes are modelled by immovable blocks of size $4 \times 4 \times 4$. [2] uses blocks of $8 \times 8 \times 8$. The environment shapes are clearly depicted in Figure 1. For the randomly generated maze, we sample each block with probability being empty $p = 0.7$. The start and goal positions are also sampled randomly at uniform with a minimum of 5 blocks distance apart. Mazes where start and goal positions are adjacent or where the goal is not reachable are discarded. For evaluation, we generated 500 of such environments and reused them (one per episode) for all experiments. We will provide the random environments along with the code.

A.2.2 Manipulation Environments

The manipulation environments differ from the navigation mazes in scale. Each wall is of size $0.05 \times 0.05 \times 0.03$. We use a layout of 9×9 blocks. The object position is the position used for the planning layer. When the object escapes the top-down view range, the episodes are terminated. The random layouts were generated using the same methodology as for the navigation mazes.

A.2.3 Robot Reacher Environments

The robot reacher environment is an adapted version of the OpenAI gym [6] implementation. We add an additional degree of freedom to extend it to 3D space. The joint angles are projected onto a map of size $12 \times 12 \times 12$, where the axes correspond to the joint angles of the robot links. The goals are randomly sampled points in the robot's configuration space.

B Implementation Details

Our PyTorch [7] implementation will be available on the project website[1].

B.1 Baseline Experiments

For HIRO, HIRO-LR and HAC we used the authors' original implementations[2][3]. To improve the performance of HAC, we modified their Hindsight Experience Replay [8] implementation so that

[1]HiDe: `https://sites.google.com/view/hide-rl`

[2]HIRO: `https://github.com/tensorflow/models/tree/master/research/efficient-hrl`

[3]HAC: `https://github.com/andrew-j-levy/Hierarchical-Actor-Critc-HAC-`

they use the FUTURE strategy. More importantly, we also added target networks to both the actor and critic to improve the performance. For HIRO, we ran the hiro_xy variant, which uses only position coordinates for subgoals instead of all joint positions to have a fair comparison with our method. For HIRO-LR, we provide top-down view images of size 5x5x3 as in the original implementation. We train both HIRO and HIRO-LR for 10 million steps as reported in [9]. For RRT+LL, we adapted the RRT python implementation[4] from [10] to our problem. We trained a goal-conditioned low-level control policy in an empty environment with DDPG. During testing, we provide the low-level control policies with subgoals from the RRT planner. We used OpenAI's baselines [11] for the DDPG+HER implementation. When pretraining for domain transfer, we made the achieved goals relative before feeding them into the network. For a better overview of the features the algorithms use, see Table 1.

Features	HiDe	RRT+LL	HIRO-LR	HIRO	HAC	DDPG+HER
Images	✓	✓	✓	x	x	x
Random start pos	x	x	x	x	✓	✓
Random end pos	x	✓	✓	✓	✓	✓
Agent position	✓	✓	x	✓	✓	✓
Shaped reward	x	x	✓	✓	x	x
Agent transfer	✓	✓	x	x	x	x

Table 1: Overview of related work and our method with their respective features. Features marked with a tick are included in the algorithm whereas features marked with a cross are not. See glossary below for a detailed description of the features.

Glossary:

Images:	If the state space has access to images.
Random start pos:	If the starting position is randomized during training.
Random end pos:	If the goal position is randomized during training.
Agent position:	If the state space has access to the agent's position.
Shaped reward:	If the algorithm learns using a shaped reward.
Agent transfer:	Whether transfer of layers between agents is possible without retraining.

B.2 Ablation Baselines

We compare HiDe against several different versions to justify our design choices. In HiDe-A(bsolute), we use absolute positions for the goal and the agent's position. In HiDe-3x3, HiDe-5x5, HiDe-9x9, we compare our learned attention window against fixed window sizes for selecting subgoals. In HiDe-RRT, we use RRT planning [12] for the top-layer, while the lower layer is trained as in HiDe. This is to show that our learned planner improves the overall performance.

B.3 Evaluation Details

For evaluation in the maze navigation experiment of Section **??**, we trained 5 seeds each for 2.5M steps for the simple maze navigation and 10M steps for the complex maze navigation "Training" environments. We performed continuous evaluation (every 100 episodes for 100 episodes). After training, we selected the best checkpoint based on the continuous evaluation of each seed. Then, we tested the learned policies for 500 episodes and reported the average success rate. Although the agent and goal positions are fixed, the initial joint positions and velocities are sampled from uniform distribution as is standard in OpenAI Gym environments [6]. Therefore, the tables in the results (cf. Table **??**) contain means and standard deviations across 5 seeds.

B.4 Network Structure

B.4.1 Planning Layer

Input images for the planning layer were binarized in the following way: each pixel corresponds to one block (0 if it was a wall or 1 if it was a corridor). In our planning layer, we process the input

[4]RRT: `https://github.com/AtsushiSakai/PythonRobotics`

image via two convolutional layers with $3 \times 3 \times 3$ for the 3D case and 3×3 kernels for the 2D case. Both layers have only 1 input and output channel and are padded so that the output size is the same as the input size. We propagate the value through the value map as in [13] $K = 35$ times using a $3 \times 3 \times 3$ max pooling layer (3×3 for 2D). Finally, the value map and position image is processed by 3 convolutions with 32 output channels and $3 \times 3 \times 3$ filter window (3×3 in 2D) interleaved by $2 \times 2 \times 2$ max pool (2×2 for 2D) with ReLU activation functions and zero padding. The final result is flattened and processed by two fully connected layers with 64 neurons, each producing outputs: $\sigma_1, \sigma_2, \sigma_3$ and the respective pairwise correlation coefficients ρ. We use $\mathrm{softplus}$ activation functions for the σ values and $\tanh$ activation functions for the correlation coefficients. The final covariance matrix Σ is given by

$$\Sigma = \begin{pmatrix} \sigma_1^2 & \rho_{1,2}\,\sigma_1\sigma_2 & \rho_{1,3}\,\sigma_1\sigma_3 \\ \rho_{1,2}\,\sigma_1\sigma_2 & \sigma_2^2 & \rho_{2,3}\,\sigma_2\sigma_3 \\ \rho_{1,3}\,\sigma_1\sigma_3 & \rho_{2,3}\,\sigma_2\sigma_3 & \sigma_3^2 \end{pmatrix}$$

so that the matrix is always symmetric and positive definite. For numerical reasons, we multiply by the binarized kernel mask instead of the actual Gaussian densities. We set the values greater than the mean to 1 and the others to zeros.

B.4.2 Control Layer

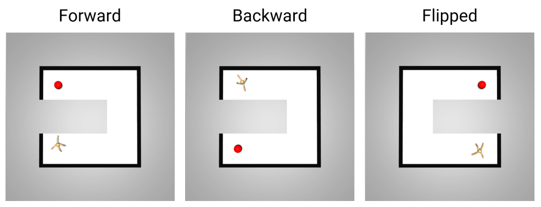
We use the same network architecture for the lower layer as proposed by [3], i.e. we use 3 times a fully connected layer with ReLU activation function. The control layer is activated with $\tanh$, which is then scaled to the action range.

B.4.3 Training Parameters

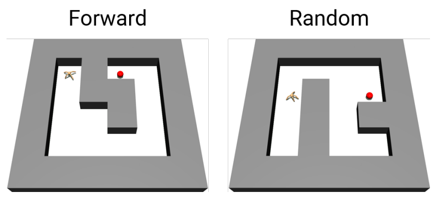
- Discount $\gamma = 0.98$ for all agents.
- Adam optimizer. Learning rate 0.001 for all actors and critics.
- Soft updates using moving average; $\tau = 0.05$ for all controllers.
- Replay buffer size was designed to store 500 episodes, similarly as in [3]
- We performed 40 updates after each epoch on each layer, after the replay buffer contained at least 256 transitions.
- Batch size 1024.
- No gradient clipping
- Rewards 0 and -1 without any normalization.
- Observations also were not normalized.
- 2 HER transitions per transition using the FUTURE strategy [8].
- Exploration noise: 0.05 and 0.1 for the planning and control layer respectively.

B.5 Computational Infrastructure

All HiDe, HAC and HIRO experiments were trained on 1 GPU (GTX 1080). OpenAI DDPG+HER baselines were trained on 19 CPUs using the baseline repository [11].



(a) Simple Maze Navigation



(b) Complex Maze Navigation

Figure 1: Maze environments for the tasks reported in a) Section **??** and **??**, b) Section **??**. The red sphere indicates the goal, the pink cubes represent the subgoals. Agents are trained in only *Training* and tested in *Backward*, *Flipped*, and *Random* environments.

C Additional Results

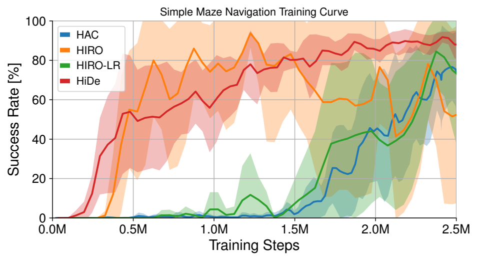
C.1 Maze Navigation

Table 2: Success rates of the individual seeds for achieving a goal in the maze navigation tasks.

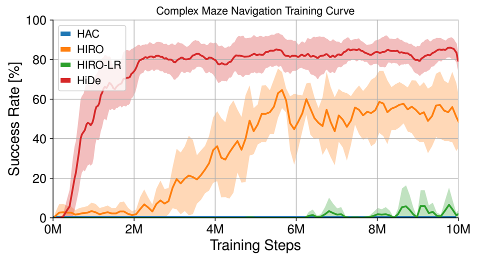
	Simple Maze			Complex Maze			
Method	Training	Backward	Flipped	Training	Backward	Flipped	Random
HAC 1	96.4	00.0	00.0	00.0	00.0	00.0	00.0
HAC 2	82.0	00.0	00.0	00.0	00.0	00.0	00.0
HAC 3	85.6	00.4	00.0	00.0	00.0	00.0	00.0
HAC 4	92.8	00.0	00.0	00.0	00.0	00.0	00.0
HAC 5	55.6	00.0	00.0	00.0	00.0	00.0	00.0
HIRO 1	89.0	00.0	00.0	68.0	00.0	00.0	44.6
HIRO 2	89.2	00.0	00.0	64.0	00.0	00.0	36.2
HIRO 3	94.0	00.0	00.0	80.0	00.0	00.0	33.6
HIRO 4	91.3	00.0	00.0	61.0	00.0	00.0	32.8
HIRO 5	90.8	00.0	00.0	81.0	00.0	00.0	34.4
RRT+LL 1	13.8	24.0	19.6	4.8	5.6	1.2	43.6
RRT+LL 2	40.2	6.2	45.2	6.8	4.6	13.6	50.6
RRT+LL 3	20.2	23.8	29.0	17.0	8.4	0.4	47.0
RRT+LL 4	37.2	18.6	30.0	25.6	2.2	3.4	47.2
RRT+LL 5	15.0	38.2	23.4	9.4	13.2	5.8	50.6
HIRO-LR 1	84.2	00.0	00.0	00.0	00.0	00.0	14.4
HIRO-LR 2	80	00.0	00.0	00.0	00.0	00.0	10.4
HIRO-LR 3	79.8	00.0	00.0	48.2	00.0	00.0	16.2
HIRO-LR 4	76	00.0	00.0	20.4	00.0	00.0	26.6
HIRO-LR 5	96.8	00.0	00.0	33.8	00.0	00.0	9.6
HiDe 1	93.4	90.2	94.0	86.4	82.6	87.4	88.6
HiDe 2	90.8	68.2	91.8	83.4	66.0	66.0	81.6
HiDe 3	94.8	91.4	96.2	87.6	84.6	91.0	85.4
HiDe 4	94.0	85.2	92.6	87.2	76.6	77.6	83.2
HiDe 5	96.2	87.8	92.2	89.0	87.4	77.4	87.6

Table 3: Results of the simple maze navigation for HAC and HIRO when provided with an image. Both methods fail to solve the task, as the state space complexity is too high.

	Training	Backward	Flipped
HAC with Image	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
HIRO with Image	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0



(a)



(b)

Figure 2: Learning curves with success rates for the training tasks averaged over 5 seeds for a) the simple maze experiment from Section **??**, b) the complex maze experiment from Section **??**. HiDe matches convergence properties of HIRO in the simple maze (left), albeit having a much larger state space in the planning layer. In the more complex maze (right), HiDe shows similar convergence, while the convergence of HIRO slows.

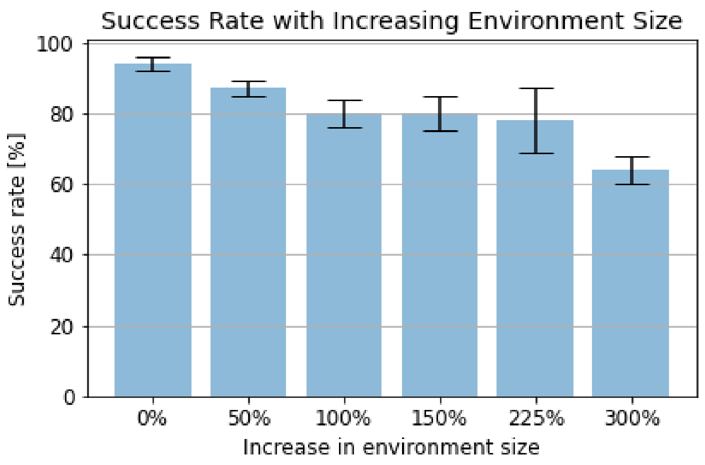


Figure 3: Success rates for the training task when gradually increasing the environment size and number of obstacles. At a 300% increase, HiDe's performance drops to 64%. See Table 4 for the more detailed results.

Table 4: Results for individual seeds of the HiDe experiments with gradually increasing number of obstacles and environment size.

Maze Training	Seed 1	Seed 2	Seed 3	Seed 4	Seed 5	Averaged
HiDe env + 100%	84.0	83.4	82.5	75.8	75.8	80.3 ± 4
HiDe env + 150%	74.2	85.4	81.4	75.6	84.8	80.3 ± 5
HiDe env + 225%	78.0	69.6	70.8	82.2	91.4	78.4 ± 9
HiDe env + 300%	62.8	58.8	69.8	66.4	64.0	64.4 ± 4

C.2 Ablation Studies

Experiment	Training	Backward	Flipped
HiDe-A 1	88.2	5.4	49.4
HiDe-A 2	89.2	28.4	53.2
HiDe-A 3	84.8	35.8	29.6
HiDe-A 4	91.2	13.6	32.2
HiDe-A 5	88.2	00.0	14.4
HiDe-RRT 1	82.0	35.2	78.8
HiDe-RRT 2	80.0	72.4	75.0
HiDe-RRT 3	73.6	53.6	69.4
HiDe-RRT 4	63.4	50.6	63.4
HiDe-RRT 5	85.6	54.0	74.2
HiDe 3x3 1	28.4	0.0	4.4
HiDe 3x3 2	67.6	0.6	44.0
HiDe 3x3 3	59.6	0.4	36.4
HiDe 3x3 4	0.0	0.0	0.0
HiDe 3x3 5	76.8	6.6	67.8
HiDe 5x5 1	91.0	82.8	97.0
HiDe 5x5 2	94.6	10.6	89.6
HiDe 5x5 3	88.0	72.4	87.2
HiDe 5x5 4	91.2	14.4	83.6
HiDe 5x5 5	97.6	83.6	50.6
HiDe 9x9 1	90.6	9.4	81.2
HiDe 9x9 2	92.4	64.4	79.4
HiDe 9x9 3	89.8	1.2	85.6
HiDe 9x9 4	99.0	4	83.0
HiDe 9x9 5	94.6	3	68.2

Table 5: Individual seed results of the ablation study for the simple maze navigation task.

	Ant 1	Ant 2	Ant 3	Ant 4	Ant 5	Averaged
Training	0.0	78.8	0.0	0.0	0.0	16 ± 35
Random	67.2	88.8	17.6	0.0	0.0	35 ± 41
Backward	0.0	62.2	0.0	0.0	0.0	12 ± 28
Flipped	0.0	79.8	0.0	0.0	0.0	16 ± 36

Table 6: Results for the complex maze navigation task with HiDe-Absolute.

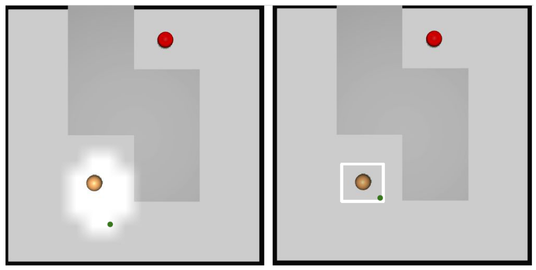


Figure 4: A visual comparison of *(left)* our dynamic attention window with a *(right)* fixed neighborhood. The green dot corresponds to the selected subgoal in this case. Notice how our window is shaped so that it avoids the wall and induces a further subgoal.

C.3 Agent Transfer

	A→B 1	A→B 2	A→B 3	A→B 4	A→B 5	Averaged
Training	99.8	100	100	100	100	100 ± 0
Random	98.4	98.6	98.0	98.6	98.8	98 ± 0
Backward	100	100	99.8	100	100	100 ± 0
Flipped	99.6	100	99.6	100	100	100 ± 0

Table 7: Results of HiDe for ant to ball transfer for individual seeds.

	B→A 1	B→A 2	B→A 3	B→A 4	B→A 5	Averaged
Training	73.4	73.4	71.2	68.2	71.6	72 ± 2
Random	86.2	84.4	83.8	82.6	78.0	83 ± 3
Backward	56.8	48.4	66.4	57.8	58.4	58 ± 6
Flipped	74.4	77.4	80.0	61.6	72.8	73 ± 7

Table 8: Results of HiDe for ball to ant transfer for individual seeds.

C.4 Robotic Arm Manipulation

	Arm 1	Arm 2	Arm 3	Arm 4	Arm 5	Averaged
Random	50	48	47	48	51	49 ± 1

Table 9: Results of the different seeds for the domain transfer experiments.

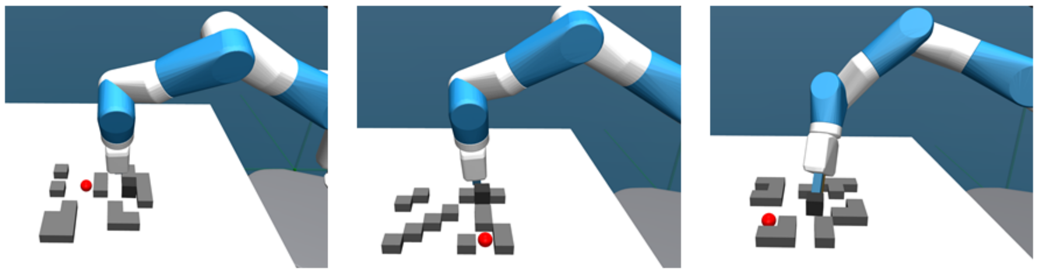


Figure 5: Example of three randomly configured test environments we use to demonstrate the domain transfer of the planning layer from a locomotion domain to a manipulation robot.

D Algorithm

Algorithm 1 Hierarchical Decompositional Reinforcement Learning (HiDe)

```
Input:
  • Agent position $s_{pos}$, goal position $G$, and projection from environment coordinates to image
    coordinates and its inverse $Proj$,$Proj^{-1}$.
Parameters:
  1. maximum subgoal horizon $H = 40$, subgoal testing frequency $\lambda = 0.3$
Output:
  • $k = 2$ trained actor and critic functions $\pi_0, ..., \pi_{k-1}, Q_0, ..., Q_{k-1}$

for $M$ episodes do                                                     ▷ Train for M episodes
    $s \leftarrow S_{init}, g \leftarrow G_{k-1}$                        ▷ Get initial state and task goal
    $train\_top\_level(s, g)$                                           ▷ Begin training
    Update all actor and critic networks
end for

function $\pi_1(s :: state, g :: goal)$
    $v_{map} \leftarrow MVProp(I, g_1)$         ▷ Run MVProp on top-down view image and goal position
    $\sigma, \rho \leftarrow CNN(v_{map}, Proj(s_{pos}))$               ▷ Predict mask parameters
    $v \leftarrow v_{map} \bigodot \mathcal{N}(\cdot|s_{pos}, \Sigma)$  ▷ Mask the value map
    return $a_1 \leftarrow Proj^{-1}(\arg\max v) - s_{pos}$ ▷ Output relative subgoal corresponding to the max
value pixel
end function

function $\pi_0(s :: joints\_state, g :: relative\_subgoal)$
    return $a_0 \leftarrow MLP(s, g)$                                   ▷ Output actions for actuators
end function

function TRAIN_LEVEL($i :: level, s :: state, g :: goal$)
    $s_i \leftarrow s, g_i \leftarrow g$                                ▷ Set current state and goal for level $i$
    for $H$ attempts or until $g_n, i \leq n < k$ achieved do
        $a_i \leftarrow \pi_i(s_i, g_i)$ + $noise$ (if not subgoal testing)    ▷ Sample (noisy) action from policy
        if $i > 0$ then
            Determine whether to test subgoal $a_i$
            $s_i^{'} \leftarrow train\_level(i - 1, s_i, a_i)$          ▷ Train level $i - 1$ using subgoal $a_i$
        else
            Execute primitive action $a_0$ and observe next state $s_0^{'}$
        end if
                                                                        ▷ Create replay transitions
        if $i > 0$ and $a_i$ not reached then
            if $a_i$ was subgoal tested then                            ▷ Penalize subgoal $a_i$
                $Replay\_Buffer_i \leftarrow [s = s_i, a = a_i, r = Penalty, s^{'} = s_i^{'}, g = g_i, \gamma = 0]$
            end if
            $a_i \leftarrow s_i^{'}$                   ▷ Replace original action with action executed in hindsight
        end if
                                          ▷ Evaluate executed action on current goal and hindsight goals
        $Replay\_Buffer_i \leftarrow [s = s_i, a = a_i, r \in \{-1, 0\}, s^{'} = s_i^{'}, g = g_i, \gamma \in \{\gamma, 0\}]$
        $HER\_Storage_i \leftarrow [s = s_i, a = a_i, r = TBD, s^{'} = s_i^{'}, g = TBD, \gamma = TBD]$
        $s_i \leftarrow s_i^{'}$
    end for
    $Replay\_Buffer_i \leftarrow$ Perform HER using $HER\_Storage_i$ transitions
    return $s_i^{'}$                                                    ▷ Output current state
end function
```

References

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

[2] 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.

[3] Andrew Levy, George Konidaris, Robert Platt, and Kate Saenko. Learning Multi-Level Hierarchies with Hindsight. In *International Conference on Learning Representations*, 2019.

[4] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking Deep Reinforcement Learning for Continuous Control. In *International Conference on Machine Learning*, pages 1329–1338, 2016.

[5] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy Lillicrap, and Martin Riedmiller. DeepMind Control Suite. Technical report, January 2018.

[6] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[7] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. 2017.

[8] 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.

[9] Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. Near-Optimal Representation Learning for Hierarchical Reinforcement Learning. In *International Conference on Learning Representations*, 2019.

[10] Atsushi Sakai, Daniel Ingram, Joseph Dinius, Karan Chawla, Antonin Raffin, and Alexis Paques. Pythonrobotics: a python code collection of robotics algorithms. *CoRR*, abs/1808.10703, 2018.

[11] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines. `https://github.com/openai/baselines`, 2017.

[12] Steven M. Lavalle. Rapidly-exploring random trees: A new tool for path planning. Technical report, 1998.

[13] Nantas Nardelli, Gabriel Synnaeve, Zeming Lin, Pushmeet Kohli, Philip H. S. Torr, and Nicolas Usunier. Value Propagation Networks. In *International Conference on Learning Representations*, 2019.