Facilitating Safe Sim-to-Real through Simulator Abstraction and Zero-shot Task Composition

Tamlin Love[1*], Devon Jarvis[1*], Geraud Nangue Tasse[1*], Branden Ingram[1], Steven James[1], Benjamin Rosman[1]

***Abstract*— Simulators are a fundamental part of training robots to solve complex control and navigation tasks. This is due to the speed and safety they offer in comparison to training directly on a physical system, where exploration may drive the system towards dangerous action for itself and its environment. However, simulators have a fundamental drawback known as the "*reality gap*", which describes the discrepancy in performance which occurs when a robot trained in simulation performs the same task in the real world. The reality gap is prohibitive as it means many of the most powerful recent advances in reinforcement learning (RL) cannot be used with robots due to their high sample complexity which makes physical training infeasible. In this work we introduce a framework for applying high sample complexity RL algorithms to robots by leveraging recent advances in hierarchical RL and skill composition. We demonstrate that adapting hierarchical RL techniques allows us to close the reality gap at multiple levels of abstraction. As a result we are able to train a robot to perform combinatorially many tasks within a domain with minimal training on a physical system or steps of error correction. We believe this work provides an important starting framework for applying hierarchical RL to perform sim-to-real generalisation at multiple levels of abstraction.**

I. INTRODUCTION

Reinforcement Learning (RL) algorithms provide a powerful toolkit for learning optimal control or navigation policies for robots, and their adoption in robotics has been spurred by a number of recent advances [1], [2], [3]. These advances, however, have occurred predominantly in non-physical domains such as game worlds [4] and simulators [5], [6]. Since exploration is a key component of RL, it is often infeasible and unsafe to train an agent in anything but a non-physical system, where the cost and damage from exploration can be mitigated or avoided. Thus, deploying an RL agent on a physical system remains an open challenge, made worse by the "reality gap". This describes the fact that no simulator is perfectly able to replicate reality. The consequence is that an agent which performs well in simulation is not guaranteed to perform comparably in the real-world.

Due to the discrepancy between simulators and the real-world, the reality gap is an example of a *domain adaptation* [7], [8] problem in which an agent must train in a source domain but generalise to performing well in another target domain [9]. The degree of discrepancy between the domains depends on the fidelity of the simulator itself, as well as the complexity of the task the agent is being trained to perform. Thus, the goal of domain adaptation is for the agent to learn robust and broadly useful features and policies which are common between the source and target domains. Importantly, this does not necessitate that the source domain be a perfect replica of the target domain. Consequentially, it does not require a simulator as the source domain to be any higher fidelity than is useful for learning a policy which transfers between domains. This is a consideration which we aim to leverage with the goal of providing a safe robot learning framework that also accommodates high sample complexity learning algorithms and compositional generalisation of skills.

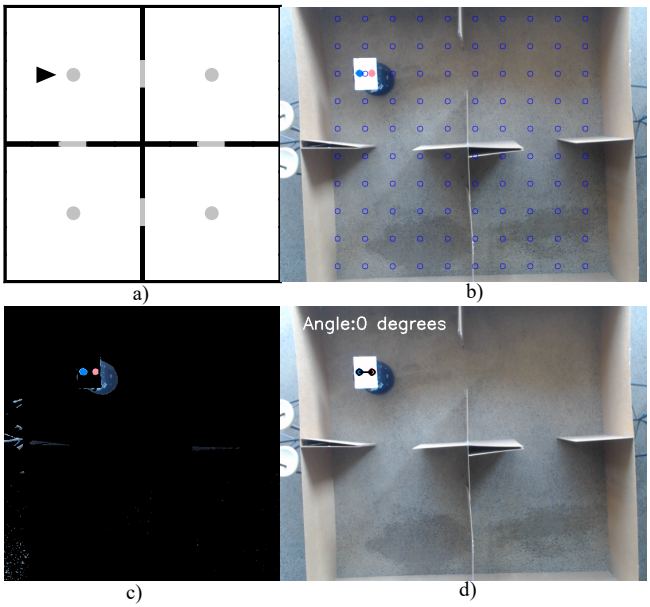


Fig. 1. Visualisation and real world correspondence of the low fidelity simulation of the real four-rooms domain. a) The agent in the abstract skill-level simulator. b) Centres of each grid-state displayed on top of the overhead camera feed. c) Segmentation mask for robot and colour tags identification in the overhead camera feed. d) Localisation of the robot with position and angle in the real world four rooms domain.

We introduce a framework which can facilitate a combinatorial explosion of skills within a robotics domain while utilising physical system training sparingly, and only in a safe manner. Our framework is depicted in Figure 2 and our design philosophy can be summarized as follows: the more difficult the task, the less physical system training should be done. To avoid training in a complex continuous

*Equal Contribution

[1]School of Computer Science and Applied Mathematics, University of the Witwatersrand, Johannesburg, South Africa {tamlin.love,devon.jarvis,geraud.tasse}@wits.ac.za

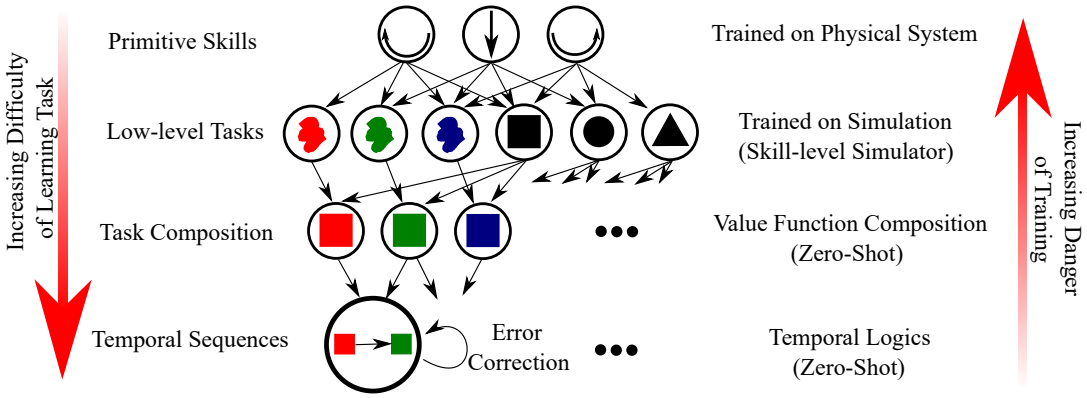


Fig. 2. Hierarchical abstraction of simulation which aims to leverage the benefits of training an agent in the real world while containing the risk of damage. The more dangerous a task is to train the higher the level of abstraction in training, containing the risk of real world training to the most basic skills. Similarly, the more possible tasks there are at a level of abstraction the less time is needed for individual training, leading to zero-shot skill composition and temporal logic.

space simulator at the level of actuators, we instead use an abstract simulator at the level of primitive skills. Learning the primitive skills is the only point at which we train on a physical system. This provides us with an abstraction, allowing us to thereafter operate in a discrete, higher-level of simulation where we train an agent to perform a set of low-level tasks (one step higher in abstraction than primitive skills). This is easy to do in simulation, but importantly due to the abstraction of the simulator there will be less discrepancy when transferring these tasks to the real-world. This is because the abstract simulator would only need to match the real-world in dynamics relative to primitive skills, rather than the far more intricate level of actuator dynamics.

Having learned the low-level tasks, we are then able to easily learn complex behaviours in a simple simulator, leveraging recent advances in RL. This allows us to draw on techniques to generalise to combinatorially many higher-level tasks and tasks requiring temporal logic in a zero-shot manner. Thus, no further training is required for these more difficult tasks as long as they can be composed from a set of low-level tasks in the same domain. The effect of this is that for complex tasks, we can avoid training on a physical system, and instead train in simulation in a way that allows zero-shot generalisation onto the hardware. We describe our entire framework in more detail in Section IV. Prior to this we provide a background on the related works which aim to tackle sim-to-real generalisation in Section II, as well as the necessary background Section III. Finally, we present our results and conclusions in Sections V and VI respectively.

II. RELATED WORK

Due to the potential benefits, addressing the reality gap has received a significant amount of attention in recent years from multiple fields, such as robotics and computer vision. The problem of transferring control policies from simulation to the real world can be viewed as an instance of domain adaptation, where a model trained in a source domain is transferred to a new target domain. One of the key assumptions behind these domain adaption methods is that the different domains share common characteristics such that representations and behaviours learned in one will prove useful for the other [10]. However, most approaches tend to either increase the sample complexity of training or rely on a fine-tuned adaptation to a simulator which may not generalise in its own right. For the first set of approaches which increase sample complexity, the most promising are domain randomisation [11] and dynamics randomisation [10]. This entails adding noise to some aspect of the simulator such that a model cannot learn to exploit idiosyncrasies in the simulator to artificially improve performance. Additionally, if by adding noise the simulator is randomly pushed towards being more reflective of the real world then the model will generalise significantly better. Thus, these methods rely on the real-world dynamics being within the sample space of the simulator dynamics. In general the noise is often added to the low-level dynamics of the simulator, but can also be added to the hyper-parameters of the simulator. Our approach avoids utilising noise due to the nature of our abstraction, where we can be certain skill-level dynamics of the simulator overlaps with that of the real world.

Other methods have aimed to learn a network which adapts a simulator policy to a real-world domain [12], [13] and generally aim to push the model to learn invariant features that are common between source and target domain [14], [15], [16]. This latter group of methods includes learning to adjust a simulator in a manner which mitigates the discrepancies between the simulator and real-world domains. For example the Neural-Augmented Simulation (NAS) trains a recurrent neural network to predict the discrepancies between the simulator and reality and then uses the network to augment and adapt the simulator to have more realistic dynamics [13]. This approach improves upon the even more tailored approaches of learning a forward model of the real-world dynamics [17], [18], [19]. Unfortunately, due to the accumulation of errors over time, unless any of these learned-simulator based approaches are nearly perfect they will be limited to short time horizons, which is prohibitive for training RL agents. Thus, there is still a need to explore new

ideas to improving sim-to-real generalisation. One recent idea of interest is the use of a small amount of real-world data at the beginning of a training pipeline which serves to improve the simulator's accuracy [13], [20]. Specifically, this small amount of real-world data is used to train the simulator error prediction or an inverse-RL model. In this work we begin similarly with a small amount of real-world training. However, to the best of our knowledge, addressing the reality gap by simulating the environment at a more abstract level and employing hierarchical RL has not been explored. Unlike these approaches we leverage the fact that it is advantageous to utilise an abstracted simulator using skill-level dynamics rather than actuator-level dynamics.

III. BACKGROUND

We use the standard RL formulation where the tasks to be solved by an agent are modeled as Markov decision processes (MDPs). An MDP is a tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$ where $\mathcal{S}$ is the set of states an agent can be in, $\mathcal{A}$ is the set of actions the agent can take in each each state, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$ is a transition function that gives the probability of an agent moving to a new state after taking an action in its current state, $R : \mathcal{S} \times \mathcal{A} \mapsto [R_{\text{MIN}}, R_{\text{MAX}}] \subset \mathbb{R}$ is the bounded reward function for the task, and γ is a discount factor that represents how much the agent should value recent rewards over later ones.

The goal of the agent is to learn a policy $\pi : \mathcal{S} \mapsto \mathcal{A}$ that maximises the rewards it receives in the environment. To this end, each policy is often associated with a state-value function $V^\pi(s) = \mathbb{E}^\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right]$ and an action-state-value function $Q^\pi(s, a) = \mathbb{E}^\pi \left[R(s_0, a_0) + \gamma V^\pi(s_1)\right]$. The optimal policy π^* is then the policy that maximises these value functions for all states and actions. A popular method for learning these policies is by using Q-learning [21]. Here, the agent iteratively learns the optimal action-value function $Q^* = \max_\pi Q^\pi(s, a)$ by using an ϵ-greedy behaviour policy and updating its Q-function using its estimate of next state values:

$$Q(s, a) \overset{\alpha}{\leftarrow} \left[Q(s, a) - \left(R(s, a) + \gamma \max_{a'} Q(s', a')\right)\right] \quad (1)$$

where α is the learning rate and s' is the next state after taking action a in state s. Given an (approximately) optimal Q-function, the agent can always recover the (approximately) optimal policy by acting greedily over its Q-function: $\pi^*(s) = \arg\max_a Q^*(s, a)$.

A. Logical Skill Composition

One of the main drawbacks of RL is the high sample complexity of learning policies, even in simulation—when the tasks are very complex and their rewards are sparse. Skill composition is an approach that has gained traction in recent years [22], [23], [24], [25] to reduce this sample complexity problem by learning low-level skills and composing them to obtain a wide range of novel skills without further learning. Recently, a Boolean task algebra which enables tasks to be specified as Boolean combinations of a set of low-level (base) ones was introduced [26]. This enables their policies to be obtained by composing the value functions of those low-level tasks accordingly. To achieve this, they first learn goal based value functions $\bar{Q}(s, g, a)$ for each low-level task using extended reward functions:

$$\bar{R}(s, g, a, s') \coloneqq \begin{cases} \bar{R}_{\text{MIN}} & \text{if } g \neq s \text{ and } s' \text{ is terminal} \\ R(s, a) & \text{otherwise,} \end{cases}$$

where $\bar{R}_{\text{MIN}}$ is a constant large negative penalty that the agent gives itself for achieving the wrong goals—we will use the simple penalty $\bar{R}_{\text{MIN}} = R_{\text{MIN}}$ used in prior work [27]. These goal based value functions are referred to as *world value functions* (WVFs) owing to the fact that they learn how to achieve all goals in the environment irrespective of the specific task rewards [28]. Similarly to regular value functions, WVFs can be learned using a goal oriented version of Q-learning where the agent: keeps track of the goals reached at the end of each episode in a goal buffer, samples goals from that buffer to reach in each episode, and updates its Q-values for each goal using the Q-learning update rule:

$$\bar{Q}(s, g, a) \overset{\alpha}{\leftarrow} \left[\bar{Q}(s, g, a) - \left(R(s, g, a) + \gamma \max_{a'} \bar{Q}(s', g, a')\right)\right]$$

Once the WVF for each low-level task is learned, they can be composed zero-shot [26] to obtain their disjunction ($\vee$), conjunction ($\wedge$), and negations ($\neg$) as follows:

$$\bar{Q}^*_{A \vee B} = \bar{Q}^*_A \vee \bar{Q}^*_B \coloneqq \max\{\bar{Q}^*_A, \bar{Q}^*_B\}$$

$$\bar{Q}^*_{A \wedge B} = \bar{Q}^*_A \wedge \bar{Q}^*_B \coloneqq \min\{\bar{Q}^*_A, \bar{Q}^*_B\}$$

$$\bar{Q}^*_{\neg A} = \neg \bar{Q}^*_A \coloneqq \left(\bar{Q}^*_{SUP} + \bar{Q}^*_{INF}\right) - \bar{Q}^*_A,$$

where $\bar{Q}^*_{SUP}$ and $\bar{Q}^*_{INF}$ are respectively the WVFs for the tasks where every goal is desirable and undesirable.

B. Reward Machines

One difficulty with the standard MDP formulation is that the agent is often required to solve a complex long-horizon task using only a scalar reward signal as feedback from which to learn. To overcome this, *reward machines* (RMs) have been proposed [29], which provide structured feedback to the agent in the form of a finite state machine. RMs encode a reward function using a set of propositional symbols $\mathcal{P}$ that represent abstract environment features. They consist of a finite set of states U, each of which represents a set of propositions that are true at the given environment state. Transitions between RM states are governed by δ_u, and the RM emits a reward function according to δ_r. A particular instantiation of an RM that is used in practice is a *simple reward machine* (SRM), which restricts the form of the state-reward function to be $\delta_r : U \times 2^{\mathcal{P}} \to \mathcal{R}$ [29]. In other words, when a transition between $u, u' \in U$ is made, the SRM emits a single scalar instead of a function (as in the case of RMs).

To incorporate RMs into the RL framework, the agent must be able to determine which abstract propositions are

true at any given state. To achieve this, the agent is equipped with a labelling function $L : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow 2^{\mathcal{P}}$ that assigns truth values to the propositions based on the agent's interaction with its environment. The agent can then learn a policy in a new decision process where the reward function in the original MDP is replaced with the RM, which is defined by the tuple $\langle \mathcal{S}, \mathcal{A}, P, \gamma, \mathcal{P}, L, U, u_0, F, \delta_u, \delta_r \rangle$. The agent's aim is now to learn a policy over the joint MDP and RM state space $\pi : \mathcal{S} \times U \rightarrow \mathcal{A}$, which can be achieved with standard algorithms such as Q-learning [29]. An example of a Reward Machine to navigate the four rooms domain is shown in Figure 3.

IV. METHODS

Our approach to mitigating the reality gap can be summarized in three steps, and shown graphically in Figure 2. These steps involve firstly abstracting the simulator into what we refer to as a "skill-level simulator", from which low-level tasks are learned. Secondly, zero-shot value function composition is utilised to compose tasks in order to solve more complex problems. Thirdly, we utilise temporal logics in order to generate task sequences.

A. Skill-level Simulator

To avoid the difficulties associated with using a high fidelity simulator for training we instead work with an abstract simulator which captures only the necessary structure in the source domain which is common with the target domain. This is achieved by training a set of primitive skills on a physical system. It is key that these primitive skills be small and easy enough to learn that training is safe and quick, avoiding the usual pitfalls of training on a physical system. Additionally, the use of early training on the physical system has been used to make downstream sim-to-real easier in prior work [13][20][30][31]. This abstracts the simulator away from the level of actuators and instead simulates the environment in terms of discrete primitive skills. By learning in the skill-level simulator we are then able to train policies for low-level tasks (sequence of skills). Figure 5 shows the returns obtained during training of the WVFs and VFs of said tasks. Similarly to previous work [32], we observe that learning WVFs has the additional benefit of faster training than regular VFs—since the WVF learns how to achieve all goals leading to better goal-directed exploration during learning.

B. Value Function Composition

Our second step is to leverage the recent RL advancement of zero-shot value function composition [26], described in Section III-A, which provides super-exponential growth in the number of possible tasks an agent can perform just from learning low-level tasks in the skill-level simulator. This alone speeds up learning significantly in our framework. However, due to the discrete nature of the skill-level simulator and resulting value functions it is necessary to perform error correction when transferring to the continuous real-world. Error correction would then be performed after every primitive skill which is time-consuming and a potential drawback of our framework. Thus, we leverage another level of abstraction introduced in the RL literature: temporal logic of task composition [29].

C. Temporal Logics

Temporal logic of task composition defines a problem as a sequence of steps to be completed by modelling the sequence with an RM. Thus the task of making a cup of coffee would be split into adding coffee granules to a cup, adding sugar, pouring hot water, etc. This is beneficial as we can use the RM to perform error correction only at the level of transitions between RM states as opposed to after transitions in the skill-level states (after each skill is performed). Thus, we have multiple levels of abstraction, each serving a purpose in minimizing physical risks, speeding up learning and reducing the amount of error correction needed when switching from sim-to-real respectively. The aim of this work is to provide a helpful framework for robot sim-to-real generalisation by applying hierarchical and compositional techniques from RL to split a learning problem into multiple levels of abstraction. The benefit is that by training a very few easy and safe primitive skills on a physical system the framework can facilitate a combinatorial explosion of skills within a domain, culminating in zero-shot generalisation to temporal logic sequences of complex tasks.

V. RESULTS

In order to test[1] the benefits of our approach, we conducted a number of experiments in a real-world "four rooms" domain on a Kobuki TurtleBot2. The "four rooms" domain consists of four square rooms, connected by doors which can be open or closed. In the skill-level simulation, the domain is represented by a discrete grid of cells connected by a move-forward skill (each room being 5×5 cells), with goal states in the centres of rooms. Three primitive actions (move forward one cell, turn left 90°, turn right 90°) were hand-trained on the robot. The policy of the robot was trained using the three levels of complexity described in Section III, namely naive Q-learning, value function composition from naive Q-learning with low-level tasks and lastly with temporal logics.

The error-correction procedure was designed to be a simple and naïve approach to match the robot's state to that of the simulated agent. We suspended a webcam above the domain and placed colour markers on the robot to allow for its location and orientation to be tracked (shown in Figure 1b)-d)). A grid of coordinates was superimposed on the webcam feed to map the simulated gridworld to the real world. The error-correcting procedure attempts to move the robot to the corresponding discretized position of the simulated agent following the same policy. As such, an error-correcting step may consist of a rotation or forward movement towards the coordinates of the grid cell.

In the first set of experiments, we consider the problem of moving from the bottom-left room to the top-right. For

[1]Code is available at: https://github.com/tamlinlove/kuricomposition

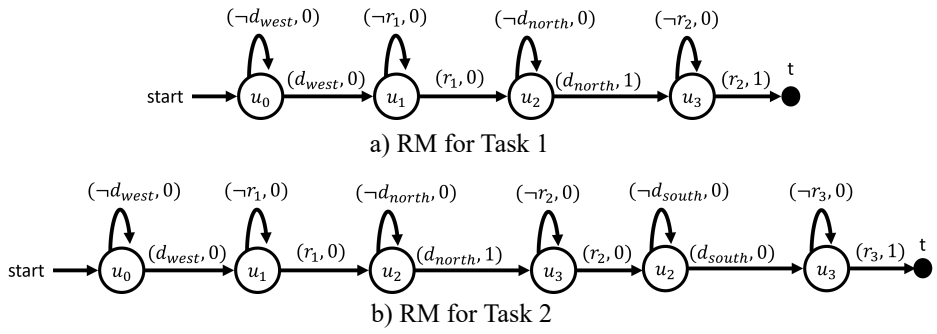


Fig. 3. Example of an RM used to complete task 2 in the four rooms domains. The agent will navigate from the bottom left to top left and finally to the top right. $d_{west}, d_{north}, d_{south}, r_1$, r_2 and r_3 are respectively propositions that are true when the agent is in front of the west door, in front of the north door, in front of the south door, in the middle of the top-left room, in the middle of the top-right room in the middle of the bottom-right room. The U_i symbols represent state which track the sequence of propositions that are used and correspond to value functions which are relevant to the state.

	Successful Runs	**Total Time (s)**		**Correction Time (%)**		**Distance from Goal (pixels)**		**Number of Corrections**	
		Average	STD	Average	STD	Average	STD	Average	STD
Transferred Policy, No Correction	3/5	12.80	0.01	0	0	38.78	13.49	0	0
Transferred Policy + Correction	5/5	199.83	34.98	93.50	1.44	5.59	2.76	175	32.81
VFC Policy + Correction	5/5	205.72	29.20	93.53	0.82	4.35	2.98	179.4	28.32
Temporal Logics + Correction	3/3	49.55	2.27	73.98	1.13	10.21	3.73	34.33	2.05

TABLE I

THE RESULTS FOR THE FIRST SET OF EXPERIMENTS (GOAL STATE: TOP-RIGHT, ALL DOORS OPEN).

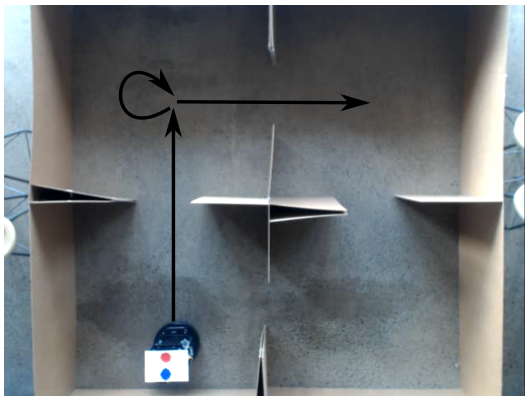
	Successful Runs	**Total Time (s)**		**Correction Time (%)**		**Distance from Goal (pixels)**		**Number of Corrections**	
		Average	STD	Average	STD	Average	STD	Average	STD
Transferred Policy, No Correction	0/3	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
VFC Policy + Correction	3/3	304.03	6.13	93.57	0.14	3.46	1.42	265.67	6.18
Temporal Logics + Correction	4/4	154.04	29.65	86.26	2.66	6.17	1.71	125.25	27.84

TABLE II

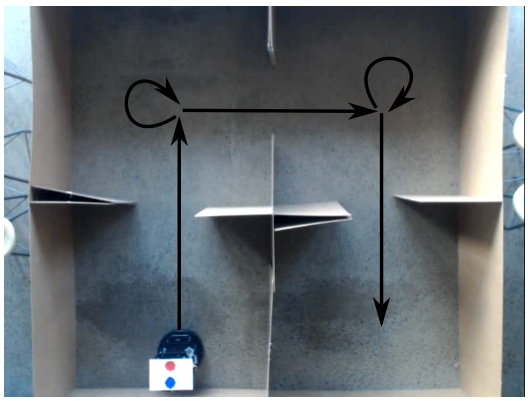
THE RESULTS FOR THE SECOND SET OF EXPERIMENTS (GOAL STATE: BOTTOM-RIGHT, BOTTOM DOOR CLOSED).

naive Q-learning this is learned directly as part of the broader training of navigation the entire domain, where the robot is trained to move from any source and to any target room where each combination is treated as its own task. For the value function composition the low-level tasks involve the agent learning to move to any two target rooms. To get to one specific target room, the intersection between two low-level task rooms is used. Finally, for the temporal logics instead of learning to move between a sequence of rooms to reach the target, the transition between rooms is modelled with an RM and so the robot is only required to learn how to move between adjacent rooms with the composed value functions. For example, the RM used to perform task 1 is shown in Figure 3a). Thus, the full set of methods we compare are: a simulated naive Q-learning policy transferred directly to the robot (no error-correction), the same policy with error-correction after every discrete action, a composed value-function policy with error-correction (which we call the VFC Policy) and a policy obtained using temporal logics with error correction after each RM state (so that error correction only happens at doorways and room centres). Each method is evaluated using the metrics of "successful runs" (the number of successful runs, where a run is considered failed if the robot crashes into a wall), "total time" (the total time in seconds to perform a run), "correction time" (the percentage of time spent correcting the robot's position), "distance from goal" (the distance, in pixels on the webcam feed, from the final goal at the end of the run), and "number of corrections" (the number of error-correction steps performed in a run). Results are averaged over the number of successful runs and detailed in Table I.

While the directly transferred policy executes quickly, it has a poor success rate, and even the runs that do succeed end up relatively far from the centre of the room. Applying error-correction after every step of the policy greatly improves the success rate and final position accuracy, at the cost of time. The benefits of the composed value functions is not seen on the real world generalisation but significantly speeds up the training time of robots using our framework. Finally, the temporal logics method strikes a balance between speed and safety, achieving a comparable success rate to the error-correction after ever step but using less error corrections and time to complete the task. In the second set of experiments, the task is to move from the bottom-left room to the bottom-right room. However, the door between the two rooms is closed, forcing the robot to take a less-direct path. With a greater distance to traverse and more doorways to pass through, this represents a harder task for the robot. The same methods are evaluated using the same metrics as before, tabulated in Table II. From these results we may draw the



a) Task 1 Trajectory



b) Task 2 Trajectory

Fig. 4. Desired trajectories for the experiment 1 (shown in a) and experiment 2 (shown in b). Note that for experiment 2 the door between the bottom left (source) and bottom right (target) is closed forcing the robot to take the longer path through the rooms. The RM and temporal logics allows for zero-shot generalization to such changes in the environment, unlike naive Q-learning.

Fig. 5. Average returns per episode obtained when learning WVFs and regular VFs in the four-rooms simulation. The shaded regions represent 1 standard error over the returns obtained when training the following 10 tasks: Navigate to the "bottom-left room", "bottom-right room", "top-right room", "top-left room", "front of left door", "front of top door", "front of right door" and "front of bottom door".

same conclusions. Naive Q-learning without error correction is not a reliable procedure for completing the task, and does significantly worse on this more challenging second task which requires a longer sequence of accurate decisions. The value function composition method (we omit naive Q-learning with error correction since it is equivalent at test time to the value function composition but trains significantly slower) is safe and reliably completes the task but is very slow. Similarly to the first task, temporal logics with state-machine error corrections is both reliable and efficient at completing the task, demonstrating the benefits of the final level of abstraction.

VI. CONCLUSION

To date the application of RL in robotics has been limited due to the danger of training real world systems. This has meant training has traditionally required the use of a simulator to avoid the risk of damaging expensive equipment. However, transferring a policy learned in simulation to the real world is not trivial due to the reality gap. We demonstrate a novel framework for solving this problem by training a robot in an abstracted simulator we dubbed the "skill-level simulator" and leveraging value function composition with temporal logics from prior work in RL. This framework allowed a robot to learn a set of primitive actions which can then be used to learn low-level tasks in the skill-level simulator. These low-level task value functions can then be composed in a combinatorial and sequential fashion without the need for additional training. This approach allowed us to solve complex problems while only needing to train a small set of primitive skills in the real world and yet still outperformed comparative baselines. The abstracted simulator and error correcting mechanisms also alleviated the risks associated with traditional sim-to-real systems. We believe this work provides a powerful first framework for training complex robot policies which balance safety, training efficiency and reliability considerations. Importantly, many aspects of the framework are self-contained, such as the error correction, which is left purposefully basic in this work, or method of determining the value function used to navigate the domain. Thus, there is room for future work to improve upon pieces of the framework while still utilizing the demonstarted benefits of the levels of abstraction.

REFERENCES

[1] M. Deisenroth and C. E. Rasmussen, "Pilco: A model-based and data-efficient approach to policy search," in *Proceedings of the 28th International Conference on machine learning (ICML-11)*. Citeseer, 2011, pp. 465–472.

[2] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 3357–3364.

[3] T. Johannink, S. Bahl, A. Nair, J. Luo, A. Kumar, M. Loskyll, J. A. Ojea, E. Solowjow, and S. Levine, "Residual reinforcement learning for robot control," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 6023–6029.

[4] D. Arumugam, J. K. Lee, S. Saskin, and M. L. Littman, "Deep reinforcement learning from policy-dependent human feedback," *arXiv preprint arXiv:1902.04257*, 2019.

[5] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-real transfer in deep reinforcement learning for robotics: a survey," in *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2020, pp. 737–744.

[6] B. Osiński, A. Jakubowski, P. Ziecina, P. Miłoś, C. Galias, S. Homoceanu, and H. Michalewski, "Simulation-based reinforcement learning for real-world autonomous driving," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 6411–6418.

[7] M. Wang and W. Deng, "Deep visual domain adaptation: A survey," *Neurocomputing*, vol. 312, pp. 135–153, 2018.

[8] Y. Jiang, T. Zhang, D. Ho, Y. Bai, C. K. Liu, S. Levine, and J. Tan, "Simgan: Hybrid simulator identification for domain adaptation via adversarial reinforcement learning," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 2884–2890.

[9] J. Matas, S. James, and A. J. Davison, "Sim-to-real reinforcement learning for deformable object manipulation," in *Conference on Robot Learning*. PMLR, 2018, pp. 734–743.

[10] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 3803–3810.

[11] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 2017, pp. 23–30.

[12] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky, "Domain-adversarial training of neural networks," *The journal of machine learning research*, vol. 17, no. 1, pp. 2096–2030, 2016.

[13] F. Golemo, A. A. Taiga, A. Courville, and P.-Y. Oudeyer, "Sim-to-real transfer with neural-augmented robot simulation," in *Conference on Robot Learning*. PMLR, 2018, pp. 817–828.

[14] Y. Taigman, A. Polyak, and L. Wolf, "Unsupervised cross-domain image generation," *arXiv preprint arXiv:1611.02200*, 2016.

[15] A. A. Rusu, M. VeVcerík, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell, "Sim-to-real robot learning from pixels with progressive nets," in *Conference on robot learning*. PMLR, 2017, pp. 262–270.

[16] E. Tzeng, J. Hoffman, N. Zhang, K. Saenko, and T. Darrell, "Deep domain confusion: Maximizing for domain invariance," *arXiv preprint arXiv:1412.3474*, 2014.

[17] A. Punjani and P. Abbeel, "Deep learning helicopter dynamics models," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 3223–3230.

[18] J. Fu, S. Levine, and P. Abbeel, "One-shot learning of manipulation skills with online dynamics adaptation and neural network priors," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 4019–4026.

[19] I. Mordatch, N. Mishra, C. Eppner, and P. Abbeel, "Combining model-based policy search with online model learning for control of physical humanoids," in *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2016, pp. 242–248.

[20] R. Jeong, J. Kay, F. Romano, T. Lampe, T. Rothorl, A. Abdolmaleki, T. Erez, Y. Tassa, and F. Nori, "Modelling generalized forces with reinforcement learning for sim-to-real transfer," *arXiv preprint arXiv:1910.09471*, 2019.

[21] C. Watkins, "Learning from delayed rewards," Ph.D. dissertation, King's College, Cambridge, 1989.

[22] E. Todorov, "Compositionality of optimal control laws," in *Advances in Neural Information Processing Systems*, 2009, pp. 1856–1864.

[23] A. Saxe, A. Earle, and B. Rosman, "Hierarchy through composition with multitask LMDPs," *International Conference on Machine Learning*, pp. 3017–3026, 2017.

[24] B. Van Niekerk, S. James, A. Earle, and B. Rosman, "Composing value functions in reinforcement learning," in *International Conference on Machine Learning*, 2019, pp. 6401–6409.

[25] S. Alver and D. Precup, "Constructing a good behavior basis for transfer using generalized policy updates," in *International Conference on Learning Representations*, 2022.

[26] G. Nangue Tasse, S. James, and B. Rosman, "A Boolean task algebra for reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 9497–9507, 2020.

[27] G. Nangue Tasse, S. James, and B. Rosman, "Generalisation in lifelong reinforcement learning through logical composition," in *International Conference on Learning Representations*, 2021.

[28] G. Nangue Tasse, S. James, and B. Rosman, "World value functions: Knowledge representation for multitask reinforcement learning," in *The 5th Multi-disciplinary Conference on Reinforcement Learning and Decision Making (RLDM)*, 2022.

[29] R. T. Icarte, T. Klassen, R. Valenzano, and S. McIlraith, "Using reward machines for high-level task specification and decomposition in reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2018, pp. 2107–2116.

[30] J. P. Hanna and P. Stone, "Grounded action transformation for robot learning in simulation," in *Thirty-first AAAI conference on artificial intelligence*, 2017.

[31] S. Desai, H. Karnan, J. P. Hanna, G. Warnell, and P. Stone, "Stochastic grounded action transformation for robot learning in simulation," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 6106–6111.

[32] G. Nangue Tasse, B. Rosman, and S. James, "World value functions: Knowledge representation for learning and planning," *arXiv preprint arXiv:2206.11940*, 2022.