2024 IEEE International Conference on Robotics and Automation (ICRA 2024)
May 13-17, 2024. Yokohama, Japan

A Multifidelity Sim-to-Real Pipeline for Verifiable and Compositional Reinforcement Learning

Cyrus Neary[1], Christian Ellis[2], Aryaman Singh Samyal[1], Craig Lennon[3], and Ufuk Topcu[1]

***Abstract*— We propose and demonstrate a compositional framework for training and verifying reinforcement learning (RL) systems within a multifidelity sim-to-real pipeline, in order to deploy reliable and adaptable RL policies on physical hardware. By decomposing complex robotic tasks into component subtasks and defining mathematical interfaces between them, the framework allows for the independent training and testing of the corresponding *subtask policies*, while simultaneously providing guarantees on the overall behavior that results from their composition. By verifying the performance of these subtask policies using a multifidelity simulation pipeline, the framework not only allows for efficient RL training, but also for a refinement of the subtasks and their interfaces in response to challenges arising from discrepancies between simulation and reality. In an experimental case study, we apply the framework to train and deploy a compositional RL system that successfully pilots a *Warthog* unmanned ground robot.**

I. Introduction

Recent progress in reinforcement learning (RL) shows tremendous promise in enabling data-driven robotic systems that learn to carry out complex tasks in a variety of operating environments [1]–[6]. However, the deployment of RL-trained policies on robotic hardware is challenging. Training RL policies on hardware can be costly, and simulations cannot perfectly capture the true system of interest [5], [7]. This gap between simulation and reality may result in unwanted behaviors when simulation-based policies are deployed on hardware [8]–[10]. Furthermore, it is challenging to verify the adherence of RL policies to user-defined specifications (*e.g., complete the task of interest and avoid an unsafe set of states with a probability of at least* 0.95). However, a robot's ability to satisfy such specifications is often essential to its reliable and safe deployment [8], [11]–[13]. These challenges become exacerbated when we consider complex tasks comprising multiple subtasks over long time horizons.

We propose and demonstrate a compositional framework for training and verifying RL systems within a multifidelity sim-to-real pipeline, in order to deploy reliable and adaptable RL policies on physical hardware. Figure 1 illustrates the proposed framework, which consists of a multi-level abstraction of the decision-making problem itself, as well as the aforementioned multifidelity simulation pipeline.

Building on [14], the framework defines a *high-level model* (HLM), which is used to plan *meta-policies* that select the subtasks necessary to complete the overall task. The subtasks themselves are executed by *subtask policies*, each of which is trained using an RL algorithm in simulation. This multi-level approach might be viewed as a form of hierarchical RL [15]–[22]. As with hierarchical RL algorithms, the proposed compositional approach can improve the sample efficiency of learning by decomposing large sequential decision-making problems into smaller ones. In addition to this benefit, our work explicitly builds a model of the upper levels of the decision-making hierarchy. The model is used for planning, probabilistic verification of compositions of RL subsystems, and to facilitate sim-to-real transfer by iterating between training subtask policies in a low-fidelity simulator and testing them across different levels of simulation fidelity.

In particular, by formulating the HLM as a parametric Markov decision process (pMDP) [23], [24], the framework enables automatic decomposition of task-level specifications into a collection of corresponding subtask specifications (*e.g., reach the subtask's exit conditions with a probability of 0.98 from its entry conditions*) via the solution to a parameter synthesis problem. This decomposition allows for goal-oriented training and testing of the subtask policies. It also enables an iterative procedure in which estimates of the capabilities of the subtask policies are used to update the HLM, which replans accordingly and selects the best subtasks to train and deploy. Finally, it allows for efficient adaptation to changes in the decision-making problem: previously learned subtask policies can be reused as components of new HLMs that solve different tasks in altered environments.

In this work, we integrate a multifidelity sim-to-real pipeline with the iterative and compositional approach to RL described above. In general, discrepancies between simulation and reality can result not only from a lack of accuracy in predicting the robot's physical dynamics, but also from a lack of fidelity in capturing the interactions of the many complex subsystems that are typical of robotics (e.g., asynchronous message passing between sensors and processors, inaccuracies in the execution of actuation commands, uncertain state estimations, and the update frequency of the decision-making loop itself). Some of the latter sources of simulation error can be reduced through software-in-the-loop (SIL) simulations that implement the entire autonomy software stack as well as the dynamics of the robot. However, these autonomy stacks typically cannot run faster than real time, which precludes their use for training RL policies.

We accordingly propose a three-layer simulation pipeline. Initially, a *low-fidelity* simulation—which implements only the robot's physical dynamics—is used to train and verify subsystem policies. This low-fidelity simulation makes a

[1]The University of Texas at Austin {cneary, aryamansinghsamyal, utopcu}@utexas.edu
[2]The University of Massachusetts Dartmouth cellis3@umassd.edu
[3]U.S. Army Research Lab craig.t.lennon.civ@army.mil

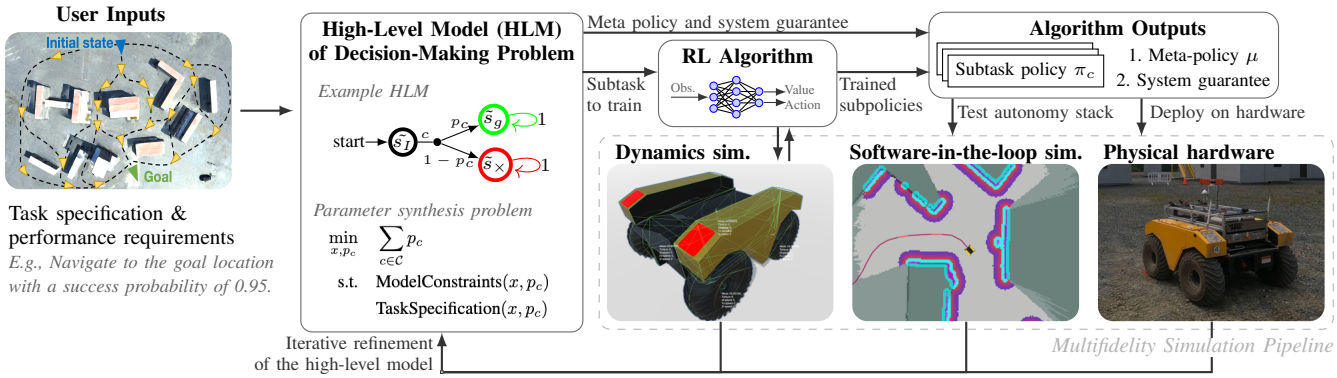


Fig. 1: The proposed framework provides a compositional approach to training and verifying reinforcement learning (RL) policies in a multifidelity simulation pipeline, before deploying the trained policies on robot hardware. It constructs a *high-level model* (HLM) of the robot's task, which it uses to: plan *meta-policies* dictating which subtasks will be composed to complete the overall task, decompose task specifications into subtask specifications, select the subtasks whose RL-based policies require further training, assess the capabilities of the trained subtask policies, and use these assessments to place probabilistic guarantees on the success of the composite policy. A *low-fidelity* simulation of the robot dynamics is initially used to train and empirically verify the subtask policies. A *high-fidelity* software-in-the-loop simulation is used to test the integration of the trained policies with the existing autonomy software stack, before deploying the policies on the target robotic hardware. Feedback from both simulators and from the hardware tests is used to update the HLM, in order to adapt and retrain subtasks as necessary. Videos of experiments are available at `https://tinyurl.com/44j8s8nz`.

number of simplifying assumptions, e.g., that the robot perfectly observes its own state. Next, the resulting compositional RL policies are integrated into the software stack that will be deployed on the true robot, before being tested in a *high-fidelity* SIL simulation that relaxes the assumptions made by the low-fidelity simulation, e.g., state observations are now given by odometry-based estimates of the robot's location and pose. Finally, the policies are deployed on the target robotic system of interest. At every level of fidelity in this simulation pipeline, the performances of the subtask policies are assessed and used to close the iterative compositional RL loop, i.e., to update the HLM.

We apply the proposed framework to train and deploy a compositional RL system that pilots a *Warthog* unmanned ground robot. We implement the low-level simulation using the *Unity* video game engine [25] and photogrammetry data of the target operating environment in the real world. We develop software to integrate the trained RL policies with the *Army Research Lab's* autonomy stack, named *Phoenix*, and test the integration in a SIL simulation before successfully deploying the policies on the ground robot.

Through this case study, we observe that the proposed framework not only facilitates the process of training an RL-based system to complete the robot's navigation task, but it also enables efficient adaptation to changes in the environment and it allows the user to isolate and address challenging portions of the sim-to-real transfer. In one instance, a subtask was failing to satisfy its subtask specification when deployed in the real world. Instead of changing the overall learning objective and restarting the training process for the entire task, the compositional framework allowed us to isolate and retrain only the individual subtask policy responsible for the failure. The end result was a successful compositional policy that consistently completes the robot's task.

II. Preliminaries

We model the robot's control task using a partially observable Markov decision process (POMDP). A POMDP is a tuple $M = (S, A, P, Z, \mathcal{O})$ where S is a set of states, A is a set of actions, $P : S \times A \times S \to [0, 1]$ is a transition probability function, Z is a set of possible observations, and $\mathcal{O} : S \times Z \to [0, 1]$ is an observation probability function.

In general, a policy within the POMDP is a function $\pi : (Z \times A)^* \times Z \times A \to [0, 1]$ that maps histories of observations and actions $z_0 a_0 \dots z_t \in (Z \times A)^* \times Z$ to distributions over actions $a \in A$. Implementing policies over histories of arbitrary length is impractical. Instead, it is common to define policies that map fixed-length histories of observations to distributions over actions. We use RL algorithms to learn such policies π in M [26], [27].

III. Compositional Reinforcement Learning

Instead of taking a monolithic approach that searches for a single policy π accomplishing some objective in M, we decompose the overall task of interest into subtasks and we define a *high-level model* (HLM) to reason over *compositions* of subtasks. For the sake of completeness, below we briefly present definitions of tasks, subtasks, HLMs, and subtask compositions that are adapted from [14]. However, for a more detailed discussion surrounding the motivation and intuition behind these definitions, we refer the reader to [14].

A. Tasks, Subtasks, and (Sub)Task Specifications

We define a task in POMDP M as the tuple (s_I, S_{targ}) that consists of an initial state $s_I \in S$ and a target set of

states $S_{targ} \subseteq S$ that the robot must reach. Furthermore, let $\mathbb{P}^{s_I}_M(\Diamond S_{targ}|\pi)$ denote the probability of reaching the target set from s_I under a particular policy π. We then define a *task specification* as the requirement that $\mathbb{P}^{s_I}_M(\Diamond S_{targ}|\pi) \geq 1-\delta$ for some allowable probability of failure $\delta \in [0,1]$.

We similarly define a *subtask* as a tuple $c = (\mathcal{I}_c, \mathcal{F}_c)$, where $\mathcal{I}_c \subseteq S$ is a set defining the subtask's *entry conditions* and $\mathcal{F}_c \subseteq S$ is a set defining the subtask's *exit conditions*. The objective of a *subtask policy* π_c is to reach an exit condition $s' \in \mathcal{F}$ from any one of the subtask entry conditions $s \in \mathcal{I}$. We note that this definition is similar to the popular *options* framework, defined by [15]. A *subtask specification* is then defined as the requirement that $\mathbb{P}^{s}_M(\Diamond \mathcal{F}_c|\pi_c) \geq p_c$ for every $s \in \mathcal{I}_c$ and for some $p_c \in [0,1]$. In words, for subtask policy π_c to satisfy the subtask specification, it must reach the target set with a minimum probability of p_c from any state in the subtask's set of entry conditions.

We define a collection $\mathcal{C} = \{c_1, c_2, ..., c_k\}$ of subtasks to be *composable*, if and only if for every $i, j \in \{1, 2, \ldots, k\}$, either $\mathcal{F}_{c_i} \subseteq \mathcal{I}_{c_j}$ or $\mathcal{F}_{c_i} \cap \mathcal{I}_{c_j} = \emptyset$. We define a collection $\mathcal{C}$ of subtasks to be *compatible* with an overall task (s_I, S_{targ}) if and only if the following three conditions hold: 1) there exists at least one $i \in \{1, \ldots, k\}$ such that $s_I \in \mathcal{I}_{c_i}$, 2) there exists at least one $i \in \{1, \ldots, k\}$ such that $\mathcal{F}_{c_i} = S_{targ}$, 3) for every $i \in \{1, \ldots, k\}$ either $\mathcal{F}_{c_i} = S_{targ}$ or $\mathcal{F}_{c_i} \cap S_{targ} = \emptyset$.

B. The High-Level Model and Compositions of Subtasks

Given a composable collection $\mathcal{C} = \{c_1, c_2, \ldots, c_k\}$ of user-defined subtasks that are compatible with a task (s_I, S_{targ}), we define a *high-level state abstraction* as the equivalence relation $R \subseteq S \times S$. Two states s and s' are related by R, denoted $(s, s') \in R$, if and only if the following two conditions hold.

1. For every $c \in \mathcal{C}$, $s \in \mathcal{I}_c$ if and only if $s' \in \mathcal{I}_c$, and,
2. $s \in \mathcal{F}_{targ}$ if and only if $s' \in \mathcal{F}_{targ}$.

Given the definition of a high-level state abstraction, we now define the high-level model (HLM) as a parametric MDP (pMDP) $\tilde{M} = (\tilde{S}, \tilde{s_I}, \tilde{s}_g, \tilde{s}_\times, \mathcal{C}, \tilde{P})$. Here, $\tilde{S}$ is a set of high-level states defined as the collection of all equivalence classes $[s]_R = \{s' \in S | (s, s') \in R\}$ induced by R, $\tilde{s_I} \in \tilde{S}$ is an initial high-level state defined as the equivalence class $[s_I]_R$ containing $s_I \in S$, $\tilde{s}_g \in \tilde{S}$ is a high-level goal state defined as the equivalence class $[s]_R$ such that $s \in S_{targ}$, and $\tilde{s}_\times \in \tilde{S}$ is an additional high-level state that is only transitioned to when the task is failed. Meanwhile, $\tilde{P} : \tilde{S} \times \mathcal{C} \times \tilde{S} \to [0,1]$ is a parametric transition probability function defined as,

$$\tilde{P}(\tilde{s}, c, \tilde{s}') = \begin{cases} p_c, & if \;\; c \in \mathcal{C}(\tilde{s}), \;\; \tilde{s}' = succ(c) \\ 1 - p_c, & if \;\; c \in \mathcal{C}(\tilde{s}), \;\; \tilde{s}' = \tilde{s}_\times \\ 0, & \text{otherwise,} \end{cases}$$

where $\mathcal{C}(\tilde{s}) = \{c \in \mathcal{C} | s \in \mathcal{I}_c \text{ for all } s \in \tilde{s}\}$ denotes the set of *available subtasks* from high-level state $\tilde{s}$, $succ(c)$ denotes the unique high-level successor state of subtask c (i.e. $succ(c) = [s]_R$ such that $s \in \mathcal{F}_c$), and $p_c \in [0,1]$ are parameters associated with the separate subtasks $c \in \mathcal{C}$.

Intuitively, the states of HLM $\tilde{M}$ correspond to sets of states in POMDP M from which the same collection of subtasks may be initiated. If a subtask c is initiated from a given high-level state $\tilde{s}$, then with probability p_c the HLM transitions to a successor state that corresponds to the successful completion of the subtask, and with probability $1 - p_c$ it transitions to the failure state $\tilde{s}_\times$. We note that the parameter p_c may thus be interpreted as the probability of subtask c being successfully completed, given that it is initiated from a state within its set of entry conditions.

A composition of subtasks is defined by a *meta-policy* $\mu : \tilde{S} \times \mathcal{C} \to [0,1]$ that maps high-level states to distributions over subtasks. The execution of a composition of subtask policies thus proceeds as follows. From initial state s_I, the meta-policy conditions on the corresponding high-level initial state $\tilde{s_I}$ to select a subtask c to execute. The corresponding subtask policy π_c is executed in POMDP M until it reaches an exit condition $\mathcal{F}_c$, at which point it relinquishes control. The meta-policy then uses the corresponding high-level state $\tilde{s} = succ(c)$ to select the next subtask to execute and the process continues. If the subtask policy fails to reach an exit condition (e.g., it does not relinquish control within some user-defined maximum allowable time), then the task is considered failed and the HLM transitions to $\tilde{s}_\times$.

Let $\mathbb{P}^{s_I}_M(\Diamond S_{targ}|\mu, \pi_{c_1}, \ldots, \pi_{c_k})$ denote the probability that such a composition of subtask policies reaches the target set S_{targ} from initial state s_I within POMDP M. Given an allowable failure probability $\delta \in [0,1]$, our objective is to find a collection of subtask policies $\{\pi_{c_1}, \ldots, \pi_{c_k}\}$ and a meta-policy μ such that the composition satisfies the task specification $\mathbb{P}^{s_I}_M(\Diamond S_{targ}|\mu, \pi_{c_1}, \ldots, \pi_{c_k}) \geq 1 - \delta$.

C. Automatic Decomposition of Task Specifications

We note that the meta-policy may be viewed as a Markovian policy within the HLM [28]. We may thus define the probability $\mathbb{P}^{\tilde{s_I}}_{\tilde{M}}(\Diamond \tilde{s}_g|\mu, p_{c_1}, \ldots, p_{c_k})$ of μ reaching the high-level goal state $\tilde{s}_g$ from the high-level initial state $\tilde{s_I}$, given values for the transition parameters $p_{c_1}, \ldots, p_{c_k}$.

We briefly recall the conclusion of Theorem 1 from [14], which states that if every subtask policy π_c satisfies the subtask specification defined by parameter p_c, i.e., $\mathbb{P}_M(\Diamond \mathcal{F}_c|\pi_c, s) \geq p_c$ for every $s \in \mathcal{I}_c$, then

$$\mathbb{P}^{s_I}_M(\Diamond S_{targ}|\mu, \pi_{c_1}, \ldots, \pi_{c_k}) \geq \mathbb{P}^{\tilde{s_I}}_{\tilde{M}}(\Diamond \tilde{s}_g|\mu, p_{c_1}, \ldots, p_{c_k}).$$

So, if we pick values for parameters $p_{c_1}, \ldots, p_{c_k}$ and simultaneously find a meta-policy μ such that $\mathbb{P}^{\tilde{s_I}}_{\tilde{M}}(\Diamond \tilde{s}_g|\mu, p_{c_1}, \ldots, p_{c_k}) \geq 1-\delta$, then we may conclude that the composition of subtask policies defined by μ will satisfy the task specification of interest. A parameter synthesis problem that simultaneously solves for a meta-policy and for a collection of values $p_{c_1}, \ldots, p_{c_k}$ that minimize $\sum_{i=1}^{k} p_{c_i}$ while ensuring this condition holds may be formulated as a bilinear program. We exclude a full statement of this optimization problem due to space constraints, however, a detailed description of the problem is available in [14].

IV. The Multifidelity Sim-to-Real Pipeline

We integrate the above framework for compositional RL with a multifidelity simulation of a wheeled ground robot and its operating environment. This pipeline allows for efficient training and testing of subtask policies in a *low-fidelity* simulation that can be run faster than real time. However, it also allows for the nuanced interactions between the trained policies and the existing software stack to be tested in a *high-fidelity* simulation before deployment.

We formulate states, actions, observations, and rewards that are specific to wheeled robot navigation tasks. However, we emphasize that the presented framework may easily be adapted to different robotic systems, or to handle different task objectives and decision-making considerations. We also remark that while we develop the proposed framework with RL algorithms in mind, both the HLM and the multifidelity simulation pipeline may be used to develop and test any collection of subtask policies π_c, so long as they all satisfy their subtask specifications.

A. Modeling the Robotic System

We use the POMDP $M_{true} = (S,\ A,\ P_{true},\ Z,\ \mathcal{O}_{true})$ to represent the true physical system of interest. The robot's state s describes its position $x \in \mathbb{R}^3$, velocity $v_x \in \mathbb{R}^3$, orientation $\omega \in \mathbb{H}$, and angular velocities $v_\omega \in \mathbb{R}^3$. So, the set of all possible states is given by $S = \mathbb{R}^9 \times \mathbb{H}$, where $\mathbb{H}$ denotes the quaternions. The robot's permissible actions correspond to linear and angular velocity commands, i.e., $a = [v_x, v_\omega] \in A$, where v_x corresponds to velocity along the forward-backward axis and v_ω corresponds to angular velocity about the axis perpendicular to the ground plane. The action set itself is given by a bounded rectangle in $\mathbb{R}^2$, i.e., $A = [v_x^{min}, v_x^{max}] \times [v_\omega^{min}, v_\omega^{max}] \subseteq \mathbb{R}^2$, where v_x^{min}, v_x^{max}, v_ω^{min}, and v_ω^{max} correspond to the minimum and maximum allowable linear and angular velocity commands. The robot's observations $z \in Z$ contain information related to its current state, although these observations may be noisy and may additionally include other modes of information (e.g., lidar or camera data). Meanwhile, the observation probability function $z \sim \mathcal{O}_{true}(\cdot|s)$ is unknown. The specific observation $z \in Z$ that is sampled will depend not only on the measurement noise in the robot's sensors, but also on the software that the robot implements to process these measurements and to estimate its state. Similarly, the transition probability function $s' \sim P_{true}(\cdot|s, a)$ is unknown and depends on the physical dynamics of the robot, as well as on the software that implements the velocity commands (e.g., velocity commands are typically used to compute setpoints for the feedback controllers of the individual wheels).

B. The Low-Fidelity Dynamics-Only Simulator

The low-fidelity simulation models only the aspects of the system that are fundamental to the decision-making problem at hand, while making several simplifying assumptions to reduce its required computational effort. In particular, it simulates the physical dynamics of the ground robot, and the configuration of the robot's operating environment (e.g., the locations of the buildings and goal). However, it assumes that sensor measurements and state estimations are perfectly accurate, and it does not implement the majority of the software stack required to operate the robot.

Conceptually, we use POMDP $M_{low} = (S,\ A,\ P_{low},\ Z,\ \mathcal{O}_{low})$ to model the low-fidelity simulation. We note that the sets of states S, actions A, and observations Z are identical to those from M_{true}: this ensures that the subtask entry and exit conditions, as well as the inputs and outputs to the learned subtask policies π_c, are consistent across simulations. By contrast, P_{low} and $\mathcal{O}_{low}$ will both be slightly different than their real-world counterparts. In particular, state transitions $s' \sim P_{low}(\cdot|s, a)$ are governed by a simulator of the robot's dynamics, which is accurate enough to capture the robot's kinematics and the individual feedback controllers outputting motor torques, but which cannot perfectly represent more complex effects like motor nonlinearities or wheel slip. Furthermore, to avoid simulating localization and mapping algorithms, we assume that $\mathcal{O}_{low}$ returns perfect observations of the robot's position and velocity.

C. The High-Fidelity Software-in-the-Loop Simulator

The high-fidelity simulation builds on the low-fidelity dynamics simulation, but also includes the entire software stack that will be deployed to control the physical robot. This software-in-the-loop (SIL) simulation thus relaxes the assumptions that the robot can perfectly observe its own state and that the velocity commands are instantaneously converted into motor-torque setpoints. Instead, $z \sim \mathcal{O}_{high}(\cdot|s)$ will be given by estimates of the robot's global position and velocity that are computed from local sensor measurements with simulated noise. Meanwhile, the SIL simulation also captures the asynchronous message passing between the robot's sensors, post-processing scripts, subtask policies, and actuators. This affects the state transition probabilities $s' \sim P_{high}(\cdot|s, a)$ by changing the update rate of the decision-loop. This asynchronicity between the RL-based policies and the rest of the simulation is closer to reality, but is markedly different from the implementation of the low-fidelity simulation, which sequentially alternates between sampling actions from the policy and advancing individual timesteps in the environment.

D. Verifying the Subtask Policies and Refining the HLM

Both the low-fidelity and high-fidelity simulations, as well as tests on physical hardware, can be used to verify the performance of the compositional RL system. The results of these tests can be used to update the HLM and the definitions of the subtask entry and exit conditions, or to help manually troubleshoot issues with particularly challenging subtasks. The result is an iterative procedure in which subtask policies are trained, their performance is tested in the multifidelity simulation pipeline, and the results of those tests are used to replan which subtasks should be used and trained further. We demonstrate an instance of such simulation-driven verification, as well as of the iterative procedure, in the context of an experimental case study in Section V.

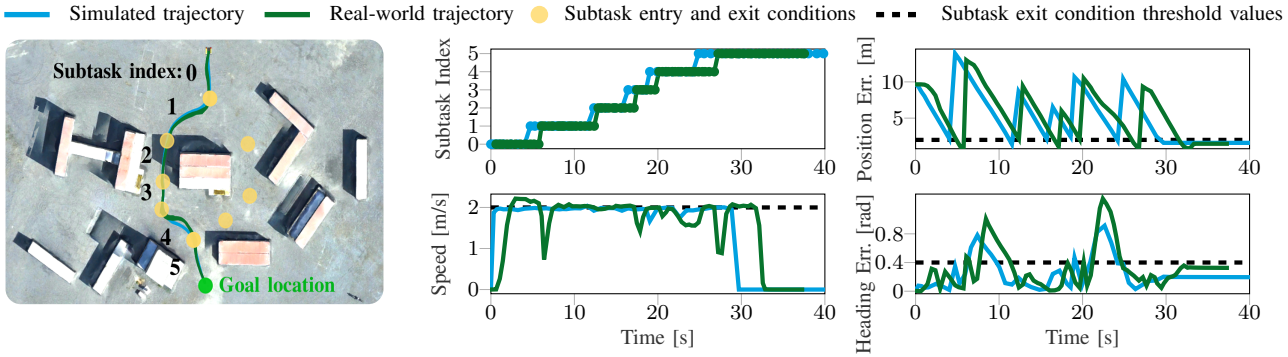


Fig. 2: Left: Robot trajectories generated by a composition of subtask policies in the high-fidelity simulator (blue) and in the real world (green). The exit conditions of the possible subtasks are highlighted in yellow. Right: The index of the active subtask policy, the position and heading errors (w.r.t. the currently active subtask), and the robot's speed. The dashed black lines illustrate the threshold error and speed values that the robot must reach to successfully complete the active subtask.

V. An Experimental Case Study

To demonstrate the proposed framework, we apply it to train compositional policies that control a *Warthog* unmanned ground robot, illustrated in the bottom right of Figure 1.

The autonomous navigation task: We consider the navigation task of reaching the goal location (illustrated on the left in Figure 2) with a probability of at least $\delta = 0.95$. We manually define 10 possible subtasks, allowing the robot to either navigate left or right around the central building. Each of these subtasks is defined by an initial location and heading, as well as a goal location and heading. The subtask entry conditions $\mathcal{I}_c$ are then defined as the collection of states such that the robot is within 3.0 meters of the initial location and within 0.5 radians of the initial heading. Similarly, the exit conditions $\mathcal{F}_c$ are defined such that the robot is within 1.0 meter and 0.4 radians of the goal location and heading.

We define these subtask entry and exit conditions to be overlapping, i.e., the goal location of one subtask will coincide with the initial location of another. This overlapping structure defines a subtask graph (similar to that illustrated on the left in Figure 1), which we use to construct the HLM.

Training the subtask policies: We train each of the possible subtask policies π_c for one million training steps in the low-fidelity simulator. We parametrize each policy, and its corresponding value function, as a neural network with two fully-connected hidden layers of 64 units with $\tanh(\cdot)$ activation functions. We train these networks using the *Stable-Baselines3* implementation of proximal policy optimization (PPO) algorithm with the default hyperparameters [29], [30].

Each training episode for subtask c begins by randomly sampling an initial state from the subtask's entry conditions $s \in \mathcal{I}_c$. The episode terminates with a reward of $+5$ when the robot reaches an exit condition $s \in \mathcal{F}_c$, and it terminates with a reward of -20 whenever the robot collides with an obstacle. At every other timestep, the robot is assigned a negative reward that is a linear combination of its distance to the subtask goal, the magnitude of the difference between its own heading and the goal heading, and its change in heading since the last timestep.

Constructing the low-fidelity simulation: We train the subtask policies in the low-fidelity simulation, which we implement using the video game engine *Unity* [25]. The robot dynamics are modeled using Unity's built-in physics engine, the robot's kinematics, and simulated PID controllers for each of its four wheels. Meanwhile, the simulation of the robot's environment is constructed from photogrammetry data from the real-world test site. At each timestep, the robot observes: its relative position and orientation with respect to the end goal defined by its subtask, its heading relative to the location of this goal, and its linear and angular velocities. The minimum and maximum allowable linear and angular velocity commands are set to $v_x^{min} = 0[\frac{m}{s}]$, $v_x^{max} = 2[\frac{m}{s}]$, $v_\omega^{min} = -1.0[\frac{rad}{s}]$, and $v_\omega^{max} = 1.0[\frac{rad}{s}]$.

Subtask	c_0	c_1	c_2	c_3	c_4	c_5
$\hat{p}_c$	1.00	0.98	1.00	1.00	0.90	0.97
p_c	1.00	0.98	1.00	1.00	0.95	0.97

TABLE I: Top: Empirical estimates of the probability of subtask success. Bottom: Automatically decomposed subtask specification values for the meta-policy in Figure 2.

Verifying the compositional RL systems in simulation: After training each subtask policy π_c, we estimate its probability $\hat{p}_c$ of subtask success by rolling out the learned policy 100 times in the low-fidelity simulator from initial states that are sampled uniformly from its entry conditions $\mathcal{I}_c$.

To decompose the task specification (complete the task with a probability of at least $\delta = 0.95$) into subtask specifications, we solve the HLM parameter synthesis problem discussed in §III-C using *Gurobi* [31]. Recall that the output of this problem is a meta-policy μ and a collection of subtask specifications: acceptable values p_c of the lower-bounds on the probabilities of subtask success.

By comparing the empirical estimates, $\hat{p}_c$, to the subtask specification values p_c, the framework automatically determines which of the subtask policies are underperforming with respect to their requirements. These specific subtask policies are then trained further until they either satisfy their subtask specifications, or a pre-defined maximum training budget is exhausted. In the latter case, the framework will automatically add the constraint $p_c \leq \hat{p}_c$ to the parameter

synthesis problem and re-solve it: alternate subtasks need to be selected by the meta-policy to complete the overall task.

Table I illustrates the values of both $\hat{p}_c$ and p_c for subtasks c_0 to c_5 in the example from Figure 2. In this case, subtask c_4 is the only subtask for which $\hat{p}_{c_4} \leq p_{c_4}$. After further training, however, we empirically observe that $\hat{p}_{c_4} = 1.00$. At this point, all subtask policies satisfy their subtask specification and so we conclude that the compositional policy defined by meta-policy μ will satisfy the overall task specification.

Testing the compositional RL systems using the high-fidelity simulation: Before deploying the compositional subtask policies on the physical hardware, we test their integration with the existing ROS-based software stack. In the high-fidelity simulation, the robot's observations are derived from odometry measurements estimated using a multimodal approach that combines LiDAR and IMU data. These observations are processed into an input format compatible with the subtask policy networks. The outputs of the policy networks are in turn converted into, and published as, ROS *Twist* messages. We remark that the inference time required to evaluate the policy networks is minimal: on the robot's onboard computers, the policy can be evaluated at $80Hz$.

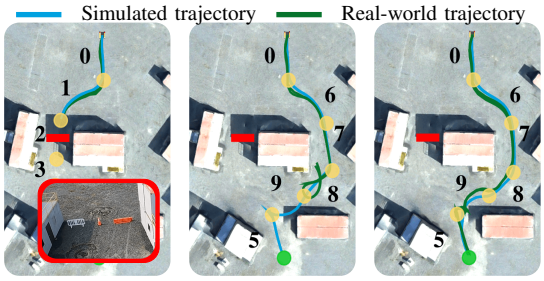
Testing compositions of learned policies in the high-fidelity simulation reveals potential causes of subtask failure before the policies are deployed on hardware, e.g., due to simulation mismatch, or due to practical issues arising from policy integration with the software stack. This information is not only helpful for refining the HLM and the subtask entry and exit conditions, but is also necessary to practically deploy the learned policies while reducing overall engineering effort.

Compositional RL systems trained in simulation lead to successful task completion on hardware: Figure 2 illustrates the result of deploying the trained compositional RL systems both in the high-fidelity simulator as well as on the robot hardware. The left image illustrates four separate runs of the compositional policies on hardware and five separate runs in simulation. These separate trajectories can hardly be told apart. The plots on the right of the figure further detail a representative trajectory from both simulation and hardware.

The framework automatically and efficiently adapts to environment changes: We now move a barricade into the robot's path, as illustrated on the left in Figure 3. We mirror this real-world environmental change by also including the obstacle in the low-fidelity simulator and by re-estimating $\hat{p}_c$, the probabilities of subtask success. As a result, $\hat{p}_{c_2}$ falls to 0.0, and this value does not improve within the maximum allowed training budget for π_{c_2}. So, the meta-policy illustrated in Figure 2 is no longer guaranteed to satisfy the task specification in the low-fidelity simulation.

As described above, the framework thus adds the constraint $\hat{p}_{c_2} \leq 0.0$ to the HLM parameter synthesis problem and re-solves it to obtain a new meta-policy and new subtask specifications. This new meta-policy now selects subtasks that navigate past the other side of the central building.

Instead of restarting the training process for the entire system, the framework reuses policies π_{c_0} and π_{c_5}. It only trains the newly required subtask policies π_{c_6}, π_{c_7}, π_{c_8}, and π_{c_9} until they satisfy their respective subtask specifications. The resulting composition of policies successfully completes its task in 5 consecutive trials within the high-fidelity simulation, illustrated in blue in the middle of Figure 3.



An obstacle (red) blocks the previously selected path.

Automatic replanning fails subtask 8 on physical hardware.

Targeted re-training of specific subtasks leads to task success.

Fig. 3: The compositional framework enables efficient adaptation to changes in the operating environment. It also simplifies the process of resolving sim-to-real errors.

The framework simplifies the process of resolving sim-to-real errors: However, we observe from this same middle figure that while the composition of subtask policies is successful in simulation, it fails to satisfy the task when deployed in the real world. In particular, subtask policy π_{c_8} is never able to reach the heading angle necessary to complete its subtask. This discrepancy between simulation and reality is likely due to errors in the simulated dynamics, particularly when the robot is attempting to turn on loose gravel.

However, this sim-to-real issue only causes a challenge for subtask c_8. We accordingly adjust the heading angles that are used to define its entry $\mathcal{I}_{c_8}$ and exit $\mathcal{F}_{c_8}$ conditions, in order to reduce sharpness and precision of the turns that the robot is required to make in this area. We then retrain only subtask policies π_{c_7}, π_{c_8}, and π_{c_9} (note that by redefining $\mathcal{I}_{c_8}$ and $\mathcal{F}_{c_8}$ we have changed the definitions of $\mathcal{F}_{c_7}$ and $\mathcal{I}_{c_9}$ as well).

The image on the right of Figure 3 illustrates the robot trajectories that result from this updated composition of subtask policies. By reusing the successful subtask policies and retraining only those causing challenges, we are able to efficiently adapt them until their composition consistently completes its task on hardware.

VI. Conclusions

We propose a framework for compositional reinforcement learning (RL) within a multifidelity sim-to-real pipeline in order to facilitate the process of reliably deploying RL-based controllers on robot hardware. We demonstrate the framework's capabilities on an unmanned wheeled ground robot. Future work will study compositional multi-robot systems and vision-based subtask policies, automating the definition of subtasks, and using data from all levels of the multifidelity simulation for training.

Acknowledgments

This work was supported in part by ARL W911NF-20-2-0132, ARL W911NF-19-2-0285, and ARO W911NF2010140.

REFERENCES

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[2] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[3] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel *et al.*, "Mastering atari, go, chess and shogi by planning with a learned model," *Nature*, vol. 588, no. 7839, pp. 604–609, 2020.

[4] P. R. Wurman, S. Barrett, K. Kawamoto, J. MacGlashan, K. Subramanian, T. J. Walsh, R. Capobianco, A. Devlic, F. Eckert, F. Fuchs *et al.*, "Outracing champion gran turismo drivers with deep reinforcement learning," *Nature*, vol. 602, no. 7896, pp. 223–228, 2022.

[5] J. Degrave, F. Felici, J. Buchli, M. Neunert, B. Tracey, F. Carpanese, T. Ewalds, R. Hafner, A. Abdolmaleki, D. de Las Casas *et al.*, "Magnetic control of tokamak plasmas through deep reinforcement learning," *Nature*, vol. 602, no. 7897, pp. 414–419, 2022.

[6] Y. Song, A. Romero, M. Müller, V. Koltun, and D. Scaramuzza, "Reaching the limit in autonomous racing: Optimal control versus reinforcement learning," *Science Robotics*, vol. 8, no. 82, 2023.

[7] B. Balaji, S. Mallya, S. Genc, S. Gupta, L. Dirac, V. Khare, G. Roy, T. Sun, Y. Tao, B. Townsend *et al.*, "Deepracer: Educational autonomous racing platform for experimentation with sim2real reinforcement learning," *arXiv preprint arXiv:1911.01562*, 2019.

[8] G. Dulac-Arnold, D. Mankowitz, and T. Hester, "Challenges of real-world reinforcement learning," *arXiv preprint arXiv:1904.12901*, 2019.

[9] 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.

[10] J. Ibarz, J. Tan, C. Finn, M. Kalakrishnan, P. Pastor, and S. Levine, "How to train your robot with deep reinforcement learning: lessons we have learned," *The International Journal of Robotics Research*, vol. 40, no. 4-5, pp. 698–721, 2021.

[11] C. Baier and J.-P. Katoen, *Principles of model checking*. MIT press, 2008.

[12] J. Garcıa and F. Fernández, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, no. 1, pp. 1437–1480, 2015.

[13] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané, "Concrete problems in ai safety," *arXiv preprint arXiv:1606.06565*, 2016.

[14] C. Neary, C. Verginis, M. Cubuktepe, and U. Topcu, "Verifiable and compositional reinforcement learning systems," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 32, 2022, pp. 615–623.

[15] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artificial Intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[16] A. G. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," *Discrete event dynamic systems*, vol. 13, no. 1, pp. 41–77, 2003.

[17] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum, "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation," in *Advances in Neural Information Processing Systems*, vol. 29, 2016.

[18] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu, "Feudal networks for hierarchical reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2017, pp. 3540–3549.

[19] O. Nachum, S. S. Gu, H. Lee, and S. Levine, "Data-efficient hierarchical reinforcement learning," in *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[20] A. Levy, G. D. Konidaris, R. W. Platt, and K. Saenko, "Learning multi-level hierarchies with hindsight," in *International Conference on Learning Representations*, 2019.

[21] W. Zhu and M. Hayashibe, "A hierarchical deep reinforcement learning framework with high efficiency and generalization for fast and safe navigation," *IEEE Transactions on Industrial Electronics*, vol. 70, no. 5, pp. 4962–4971, 2023.

[22] Y. Ji, Z. Li, Y. Sun, X. B. Peng, S. Levine, G. Berseth, and K. Sreenath, "Hierarchical reinforcement learning for precise soccer shooting skills using a quadrupedal robot," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022, pp. 1479–1486.

[23] M. Cubuktepe, N. Jansen, S. Junges, J.-P. Katoen, and U. Topcu, "Synthesis in pMDPs: A Tale of 1001 Parameters," in *International Symposium on Automated Technology for Verification and Analysis*. Springer, 2018, pp. 160–176.

[24] S. Junges, E. Abraham, C. Hensel, N. Jansen, J.-P. Katoen, T. Quatmann, and M. Volk, "Parameter synthesis for markov models," *arXiv preprint arXiv:1606.06565*, 2019.

[25] A. Juliani, V.-P. Berges, E. Teng, A. Cohen, J. Harper, C. Elion, C. Goy, Y. Gao, H. Henry, M. Mattar *et al.*, "Unity: A general platform for intelligent agents," *arXiv preprint arXiv:1809.02627*, 2018.

[26] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[27] J. Achiam, "Spinning Up in Deep Reinforcement Learning," 2018.

[28] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[29] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[30] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021.

[31] Gurobi Optimization, LLC, "Gurobi optimizer reference manual," https://www.gurobi.com/documentation/9.1/refman/index.html, 2021.