Learning Setup Policies: Reliable Transition Between Locomotion Behaviours

Brendan Tidd, Nicolas Hudson, Akansel Cosgun, *Member, IEEE*, and Jürgen Leitner

***Abstract*—Dynamic platforms that operate over many unique terrain conditions typically require many behaviours. To transition safely, there must be an overlap of states between adjacent controllers. We develop a novel method for training setup policies that bridge the trajectories between pre-trained Deep Reinforcement Learning (DRL) policies. We demonstrate our method with a simulated biped traversing a difficult jump terrain, where a single policy fails to learn the task, and switching between pre-trained policies without setup policies also fails. We perform an ablation of key components of our system, and show that our method outperforms others that learn transition policies. We demonstrate our method with several difficult and diverse terrain types, and show that we can use setup policies as part of a modular control suite to successfully traverse a sequence of complex terrains. We show that using setup policies improves the success rate for traversing a single difficult jump terrain (from 51.3% success rate with the best comparative method to 82.2%), and traversing a random sequence of difficult obstacles (from 1.9% without setup policies to 71.2%).**

***Index Terms*—Humanoid and bipedal locomotion, reinforcement learning, vision-based navigation.**

I. Introduction

BIPEDAL robots have the capability to cover a span of terrains that humans can traverse. This makes bipeds an appealing locomotion option for robots in human environments. It is impractical, however, to design a single controller that works over all required environments, particularly if all conditions are not known a priori. For robots to be able to perform a number of diverse behaviours, it is desirable for a control suite to be modular, where any number of behaviours can be added with minimal retraining.

Deep Reinforcement Learning (DRL) is an appealing method for developing visuo-motor behaviours for legged platforms. However, such DRL policies are usually trained on the terrain they are expected to operate, though it is unlikely that the agent would have access to a complete set of terrain conditions during policy training before deployment. Controllers for dynamic platforms are typically developed and tuned with safety harnesses and human supervisors in order to minimise potential damage to the robot and the environment. These restrictions limit the development of controllers over an exhaustive collection of terrain conditions. Furthermore, modifying controllers that have already been tuned is costly, for example DRL methods that are trained for discrete terrain types typically require retraining if new terrain variations are introduced [1]. We propose training a *setup policy* to transition between two pre-trained policies targeting different terrains.

A modular control suite for a dynamic platform requires careful design. Simply switching between controllers may result in the robot falling over if controllers are switched when the robot is not in a safe region for the subsequent controller. Fig. 1 shows our primary contribution: a setup policy that prepares the robot for the necessary controller to traverse the upcoming terrain obstacle. With our method, we can transition between existing behaviours, allowing pre-trained DRL policies to be combined. Our contributions are as follows:

- Using pre-trained policies, we develop **setup policies** that significantly improve the success rate from transitioning from a default walking policy to a target policy. The setup policy also learns when to switch to the target policy.
- We introduce a novel reward, called **Advantage Weighted Target Value**, guiding the robot towards the target policy.
- We show that we can use setup policies with several difficult terrain types, allowing for a modular control suite, combining behaviours that have been trained separately without any retraining of low level policies.

II. Related Work

Deep reinforcement learning (DRL) has demonstrated impressive results for locomotion tasks in recent works [2], [3], [4], [5]. Typically DRL policies optimise a single reward function, as new environments and behaviours are introduced costly retraining is required. Single policies have demonstrated locomotion over various terrains for simple 2D cases [6], and impressive quadruped behaviours [7], [8], however, for many scenarios multiple policies are often required. Furthermore, developing a single policy to perform multiple behaviours can degrade the performance of each behaviour [9].

Manuscript received 13 March 2022; accepted 25 August 2022. Date of publication 19 September 2022; date of current version 3 October 2022. This letter was recommended for publication by Associate Editor A. Del Prete and Editor A. Kheddar upon evaluation of the reviewers' comments. This work was conducted by the Australian Research Council under Project CE140100016. This work was supported in part by the QUT Centre for Robotics and in part by Robotics and Autonomous Systems Group CSIRO *(Corresponding author: Brendan Tidd.)*

Brendan Tidd is with the Queensland University of Technology (QUT), Australia, and also with Robotics and Autonomous Systems Group, CSIRO Pullenvale, Brisbane QLD 4069, Australia (e-mail: brendan.tidd@hdr.qut.edu.au).

Nicolas Hudson is with Amazon Robotics AI, Seattle 98109 USA (e-mail: nicolashenryhudson@gmail.com).

Akansel Cosgun is with Deakin University Burwood, Melbourne 3125, Australia (e-mail: akansel.cosgun@gmail.com).

Jürgen Leitner is with Monash University Clayton, Melbourne 3800, Australia, and also with LYRO Robotics Pty Ltd, Australia (e-mail: juxi.leitner@monash.edu).

Digital Object Identifier 10.1109/LRA.2022.3207567

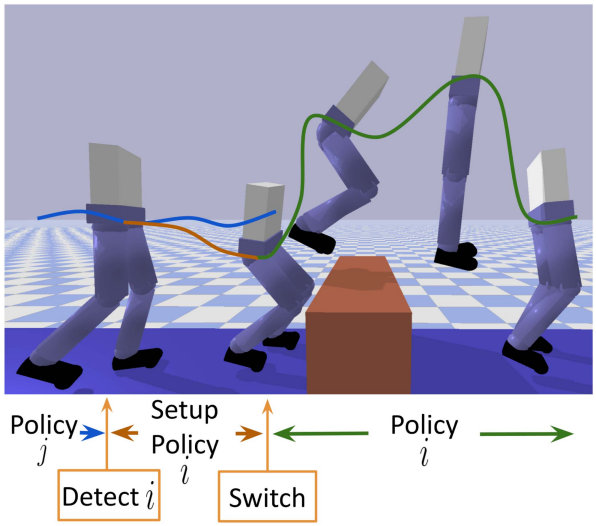

Fig. 1. As the robot approaches a difficult terrain artifact, it must move into position for target policy i. The trajectory of the walking policy j (shown with a blue line), does not intersect with the trajectory of the target policy i (green line). Our setup policy provides a trajectory that prepares the robot for the upcoming target policy (orange line).

Hierarchical reinforcement learning (HRL) offers flexibility through architecture, by training all segments of the hierarchy concurrently [3], [10], [11], [12], or training parts of the hierarchy separately [13], [14]. When trained together, HRL can improve task level outcomes, such as steering and object tracking [3], or improve learning efficiency by reusing low level skills across multiple high level tasks [12]. When trained separately, low level controllers can be refined efficiently using prior knowledge, such as by utilising motion capture data [4], [13], [15] or behavioural cloning [16]. For robotics applications it may be difficult to develop controllers for multiple behaviours simultaneously. Using pre-trained primitives can break up a large difficult problem into smaller solvable sub-problems [17].

Using pre-trained policies requires suitable handling of transitions between behaviours. Faloutsos et al. [18] learn pre and post-conditions for each controller, such that switching only occurs when these conditions have been satisfied. DeepMimic by Peng et al. [4] combines pre-trained behaviours learned from motion capture, with reliance on a phase variable to determine when one behaviour has been completed. Policy sketches introduce a hierarchical method that uses task specific policies, with each task performed in sequence [19]. CompILE uses soft boundaries between task segments [20]. Work by Peng et al. [21] trains several actor-critic control policies, modulated by the highest critic value in a given state. For these examples there must be a reasonable state overlap between controllers.

Other methods that combine pre-trained behaviours learn latent representations of skills [22] or primitives [23], enabling interpolation in the latent space. From an offline dataset of experience, Pertsch et al. [22] were able to combine low level controllers in manipulation tasks and locomotion for a multi-legged agent. In our previous work [24], we learn when to switch between low level primitives for a simulated biped using data collected by randomly switching between behaviours. Ha et al. [23] utilise motion capture to learn latent representations of primitives, then use model predictive control to navigate with a high dimensional humanoid. The FeUdal approach learns a master policy that modulates low-level policies using a learned goal signal [25]. Interpolation between behaviours yields natural transitions [26]. The work by Peng et al. [15] combines pre-trained policies using a gating function that learns a multiplicative combination of behaviours to generate smooth actions. Yang et al. [27] learn a gating neural network to blend several separate expert neural network policies to perform trotting, steering, and fall recovery in real-world experiments with a quadruped. Da et al. [28] use supervised learning to train a control policy for a biped from several manually derived controllers to perform periodic and transitional gaits on flat and sloped ground. In each of these approaches, experience from all behaviours must be available during training.

For dynamic platforms, where separate controllers occupy different subsets of the state, changing behaviours may result in instability if there is no overlap between controllers. Lee et al. [14] learn a proximity predictor to train a transition policy to guide an agent to the initial state required by the next controller. Locomotion experiments are demonstrated with a 2D planar walker, where learning occurs by training with all terrain conditions. We show that our method performs more reliably (with a higher success rate), despite training with experience from a single terrain type. Our setup policies learn to transition between complex behaviours with a 3D biped in a simulation environment using a depth image to perceive the terrain.

III. Method

In this section we describe the problem we are solving and our method for training setup policies.

A. Problem Description

We investigate composing controllers for a dynamic platform with narrow or no state overlap between adjacent behaviours. An example of where this occurs is when a bipedal robot performs a jump over a large block. In this scenario, the robot walks up to the block using a walking policy and then performs a jump with a terrain-specific target policy, where the behaviour involves jumping with both feet. Fig. 1 shows that the trajectory of the walk policy does not intersect with the trajectory of the target policy, therefore we require an intermediate policy. While it may be possible to improve the robustness of the target policies after they have been trained (i.e. to include states visited by the walk policy), we consider the case where target behaviours and value functions are provided as a black box and are therefore immutable. Our agent has 12 torque controlled actuators, simulated in PyBullet [29].

We select which terrain module to employ using an oracle terrain classifier (Fig. 2(a)). From a walking policy, we utilise a setup policy that prepares the robot for the target policy (Fig. 2(b)). When to switch from the setup policy to the target policy is also a learned output of the setup policy. We first study

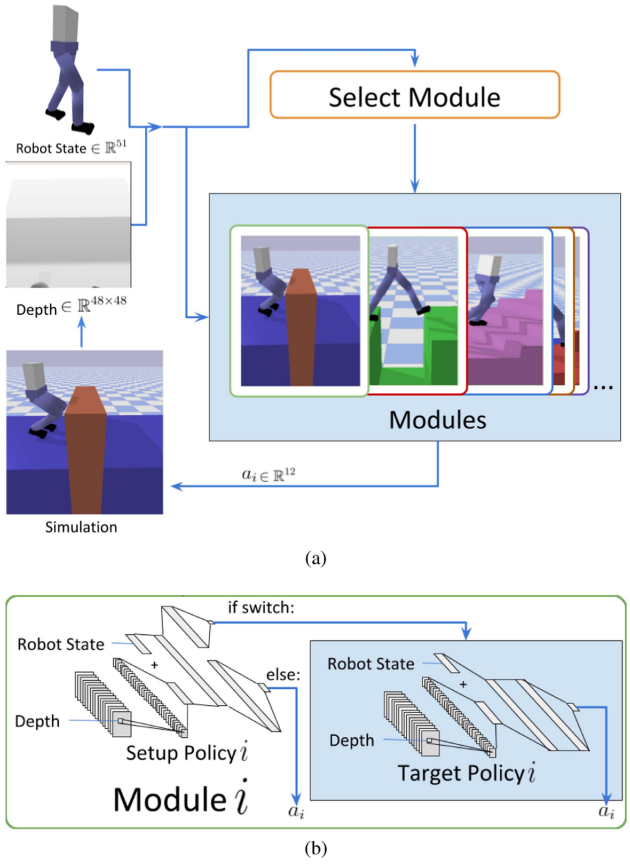

Fig. 2. (a) Shows the modular pipeline. A terrain classifier selects which module should be utilised, and the torque from the output of the selected module is applied to the robot, returning robot state and depth image. (b) Each module is pre-trained on a single terrain type. The target policy has been designed to traverse a specific terrain condition. The setup policy guides the trajectory of the robot from a default walking controller to the target policy, also learning when to switch to the target policy.

the traversal of a single difficult terrain artifact that requires walking on a flat surface and performing a jump. We then consider several diverse terrain types (gaps, hurdles, stairs, and stepping stones). The state provided to each policy is $s_t = [rs_t, I_t]$, where rs_t is the robot state and I_t is the perception input at time t. The terrain oracle (ground-truth terrain position), simplifies the behaviour switching logic, where terrain type and robot body position (yaw and height) influence the initial detection of the terrain from the depth sensor.

Robot state: $rs_t = [J_t, Jv_t, c_t, c_{t-1}, v_{body,t}, \omega_{body,t}, \zeta_{body,t}, \psi_{body,t}, h_{body,t}, s_{right}, s_{left}]$, where J_t are the joint positions in radians, Jv_t are the joint velocities in rad/s, c_t and c_{t-1} are the current and previous contact information of each foot (four Boolean contact points per foot), $v_{body,t}$ and $\omega_{body,t}$ are the linear and angular velocities of the robot body, $\zeta_{body,t}$ and $\psi_{body,t}$ are the pitch and roll angles of the robot body and $h_{body,t}$ is the height of the robot from the walking surface. The sequence of Euler angles used is yaw $>$ pitch $>$ roll. s_{right} and s_{left} are Boolean indicators of which foot is in the swing phase, and are updated when the current swing foot makes contact with the ground. Robot body rotations are provided as Euler angles. In total there are 51 elements to the robot state, which is normalised by subtracting the mean and dividing by the standard deviation for each variable (statistics are collected as an aggregate during training).

Perception: Perception is a depth sensor mounted to the robot base, with a resolution of [48,48,1], and field of view of 60 degrees. Each pixel is a continuous value scaled between $0-1$, measuring a distance between 0.25 and 2 m from the robot base. The sensor moves with the body in translation and yaw, we provide an artificial gimbal to keep the sensor fixed in roll and pitch. The sensor is pointed at the robot's feet (downwards 60 degrees so the feet are visible) and covers at least two steps in front of the robot [30]. We reduce the sampling rate of the perception to 20 Hz (reducing computational complexity), where the simulator operates at 120 Hz.

B. Deep Reinforcement Learning for Continuous Control

We consider our task to be a Markov Decision Process MDP, defined by tuple $\{\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma\}$ where $s_t \in \mathcal{S}, a_t \in \mathcal{A}, r_t \in \mathcal{R}$ are state, action and reward observed at time t, $\mathcal{P}$ is an unknown transition probability matrix from s_t to s_{t+1} taking action a_t, and applying discount factor γ.

The goal of reinforcement learning is to maximise the sum of future rewards $R = \sum_{t=0}^{T} \gamma^t r_t$, where r_t is provided by the environment at time t. Actions are sampled from a deep neural network policy $a_t \sim \pi_\theta(s_t)$, where a_t is the joint commands. Each policy is trained with Proximal Policy Optimisation (PPO) [31]. We use the implementation of PPO from OpenAI Baselines [32].

C. Training Setup Policies

For each target terrain type we train a setup policy with the purpose of bridging the trajectories of a default walking policy and the behaviour for traversing the upcoming terrain. The algorithm for training each setup policy is shown in Algorithm 1.

Default Policy: The default policy is a pre-trained policy designed to walk on a flat surface. When training a setup policy we always begin with the default policy until the terrain artifact is detected by an oracle. Once the terrain has been traversed, we switch back to the default policy when a termination criteria is reached (τ_θ, defined below). We initialise the setup policy from the default walking policy such that the robot continues to walk towards the obstacle while learning how to prepare for the target policy.

Setup Policy: While training setup policies, we are given access to an oracle terrain detector that indicates when the robot is at a fixed distance from the terrain artifact. When terrain type i is detected, we switch immediately from the default policy to the setup policy for terrain type i, denoted as π^i_ϕ. The setup policy outputs joint torques a_i, and switch condition τ_ϕ. If the switch condition is met, we immediately switch to the target policy.

Target Policy: The target policy for terrain type i, denoted as π^i_θ, is trained with curriculum learning following the method outlined in prior work [33]. We also define a termination condition τ_θ that denotes switching from the target policy back to

the default policy. For the termination condition to be reached, the robot must successfully traverse the terrain obstacle and have both feet in contact with the ground. This condition is the same for all artifact types.

Setup Policy Reward: We design a reward function to motivate the setup policy to transition the robot from states visited by the default policy to states required by the target policy. We note the value function of the target policy, $V^{\pi^i_\theta}(s_t) \approx \sum_{t=0}^{T} \gamma^t r_t$, (where r_t is the original reward afforded by the environment during training of the target policy), provides an estimate of what return can be expected if we run the target policy π^i_θ from s_t. However, value functions are notoriously over-optimistic for states that have not been visited [34] (such as those we might experience by running the default policy). The discrepancy in actual versus expected return is called the *advantage*.

The *advantage* is a zero centered calculation of how much better or worse policy π performs after taking action a_t compared to the value function prediction $V^\pi(s_t)$:

$$A^\pi(s_t, a_t) = R_t - V^\pi(s_t) \tag{1}$$

where R_t is sum of future rewards from time t, taking action a_t, and from there running policy π.

We estimate the advantage using the temporal difference (TD) error with the value function of the target policy π^i_θ:

$$\hat{A}^{\pi^i_\theta}(s_t, a_t) = r_t + \gamma V^{\pi^i_\theta}(s_{t+1}) - V^{\pi^i_\theta}(s_t) \tag{2}$$

where r_t is the target policy reward.

Using advantage $\hat{A}^{\pi^i_\theta}(s_t, a_t)$ as an indication of the accuracy of $V^{\pi^i_\theta}(s_t)$, we define the reward **Advantage Weighted Target Value**:

$$\hat{r}_t = \left(1 - \min\left(\alpha \hat{A}^{\pi^i_\theta}(s_t, a_t)^2, 1\right)\right) \cdot \beta \hat{V}^{\pi^i_\theta}(s_t) \tag{3}$$

where the target policy value is weighted by the estimated advantage and α and β are scaling factors tuned empirically (set to 0.15 and 0.01 respectively).

The target policy value function $V^{\pi^i_\theta}(s_t)$ has learned the value for states where the target policy has collected experience. Outside of these states, for example when the robot is activating a different policy, we expect the reward provided by the environment r_t, and the next state s_{t+1} will not be consistent with what is expected by the target policy value function. Therefore, the advantage $\hat{A}^{\pi^i_\theta}(s_t, a_t)$ will not be close to zero, and $\hat{r}_t$ becomes small. Intuitively, for states where the target policy value function $V^{\pi^i_\theta}(s_t)$ is accurate, the target policy advantage $\hat{A}^{\pi^i_\theta}$ will be close to zero, thus the setup policy will learn to maximise $V^{\pi^i_\theta}$. State-action pairs where $\hat{A}^{\pi^i_\theta}(s_t, a_t)$ is far from zero will reduce the effect of an overconfident $V^{\pi^i_\theta}(s_t)$.

Extended Reward: One issue with training setup policies is the delayed effect of actions, where once switched to the target policy, the setup policy is no longer acting on the environment or receiving a reward. Once the setup policy has transitioned to the target policy, we must provide the setup policy with information regarding the performance of the robot since switching. Our method for solving this problem is to provide the additional reward obtained *after* switching, by including an extended reward in the rollout buffer of the setup policy. We add this extended reward to the last reward received running the setup policy:

$$\hat{r}_{final} = \hat{r}_{final\_prev} + \hat{r}_t \tag{4}$$

where $\hat{r}_{final\_prev}$ is the final reward entry received by the setup policy before switching, and $\hat{r}_t$ is the reward received at time t, after the setup policy has transitioned to the target policy. For algorithmic stability, we clear all but the last entry of the buffer after each training update. The last reward entry of the buffer becomes $\hat{r}_{final}$ for environment steps that continue after training. The procedure for training setup policies is provided in Algorithm 1. Note that the termination variable τ_t (line 7) is either τ_ϕ if the current policy is the setup policy, or τ_θ if the current policy is the target policy.

Algorithm 1: Setup Before Switching.

1: Load Target Policy π^i_θ. Load Default Policy π^j_θ
2: Initialise Setup Policy π^i_ϕ from the Default Policy
3: Initialise Buffer
4: Current Policy $\leftarrow$ Default Policy
5: **for** $iteration = 1, 2, \ldots$ **do**
6: **while** not $done$ **do**
7: $a_t, \tau_t \sim$ Current Policy(s_t) $\triangleright$ τ is τ_ϕ or τ_θ
8: $s_{t+1}, r_t, done = env.step(a_t)$
9: **if** Terrain detected **then**
10: $\hat{A}^{\pi^i_\theta}(s_t, a_t) = r_t + \gamma V^{\pi^i_\theta}(s_{t+1}) - V^{\pi^i_\theta}(s_t)$
11: $\hat{r}_t = (1 - \min(\alpha \hat{A}^{\pi^i_\theta}(s_t, a_t)^2, 1)) \cdot \beta V^{\pi^i_\theta}(s_t)$
12: **if** Current Policy == Default Policy **then**
13: Current Policy $\leftarrow$ Setup Policy π^i_ϕ
14: $\tau_t \leftarrow \tau_\phi$
15: **else if** Current Policy == Setup Policy **then**
16: Store s_t, a_t, $\hat{r}_t$, τ_ϕ into Buffer
17: **if** Buffer Full **then**
18: PPO Training Update
19: Clear buffer (except last entry)
20: **end if**
21: **if** τ_ϕ **then**
22: Current Policy $\leftarrow$ Target Policy π^i_θ
23: $\tau_t \leftarrow \tau_\theta$
24: **end if**
25: **else if** τ_θ **then**
26: Current Policy $\leftarrow$ Default Policy π^j_θ
27: **end if**
28: **if** Current Policy != Setup Policy **then**
29: $\hat{r}_{final} = \hat{r}_{final\_prev} + \hat{r}_t$
30: **end if**
31: **end if**
32: $s_t = s_{t+1}$
33: **end while**
34: **end for**

IV. Experiments

We evaluate our method with a biped in simulation. Initial evaluations are performed with a single difficult terrain (Section IV-A, Section IV-B, Section IV-C), before adding more behaviours in Section IV-D. The difficult terrain is a sample of

TABLE I
ABLATION STUDY FOR INITIALISING SETUP POLICIES FROM A DEFAULT WALK POLICY, AND RECEIVING AN EXTENDED REWARD, FOR SWITCHING FROM A WALKING POLICY TO A JUMP POLICY ON A SINGLE TERRAIN SAMPLE

	Success %	Distance %
Without Initialisation	38.8	73.2
Without Extended Reward	12.3	58.3
Full Method	**82.2**	**96.1**

flat terrain with a single block 50 cm high, and 30 cm in length (in the forward direction of the robot), an example is shown in Fig. 1. We perform training and evaluation on a single terrain sample, starting the robot from a random x (forward direction) uniformly sampled from (0.0, 2.2) m before the terrain artifact, y (strafe) uniformly sampled from $(-0.6, 0.6)$ m, and $heading$ (yaw) uniformly sampled from $(-0.3, 0.3)$ rad. Setup policies are trained for a total of 10 million environment steps collected from 16 worker nodes, each performing a policy update every 2048 steps with averaged gradients. All experiments using a single terrain type are evaluated by pausing training and performing 100 evaluation runs, and recording both the Success percentage and Distance percentage. Success % is defined as the average success rate to reach a goal location several meters past the terrain artifact. Distance % is the average percentage of total terrain distance the agent successfully covered.

A. *Effect of Using Initialisation and Extended Reward*

We perform an ablation to study the effect of initialising the setup policy from the walking policy, and the effect of extending the reward after the setup policy has transitioned to the target policy. In all experiments the default policy is used until the terrain is detected, then the setup policy is activated until the termination signal τ_ϕ indicates switching to the target policy, and τ_θ then results in switching back to the default policy.

- **Without Initialisation:** We evaluate our method without initialising the setup policy from the default walking policy, i.e. the setup policy is randomly initialised.
- **Without Extended Reward:** We evaluate our method without including the reward after the setup policy has switched to the target policy.
- **Full Method:** Our method uses a setup policy initialised from the default walk policy, and receives reward signal after the setup policy has transitioned to the target policy.

We can see that initialising the setup policy with the default walking policy and extending the reward improves learning outcomes as shown in Table I, where our method achieves 82.2% success compared to 38.8% without initialisation from the default walking policy and 12.3% without receiving the extended reward.

B. *Setup Policy Reward Choice*

For training the setup policy, we investigate several options for the reward provided at each timestep. For each reward type we evaluate with three different random seeds, results are shown in Fig. 3.

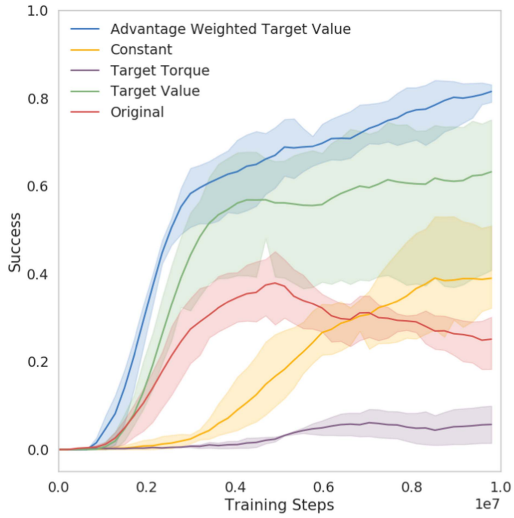

Fig. 3. We investigate several options for the reward function used to train a setup policy on the difficult jump terrain. We show the success rate (robot reaches several meters after the jump) during training for three different random seeds.

- **Original:** The reward afforded by the environment that is used to train the target policy: the reward encourages following an expert motion.
- **Constant:** At each timestep a constant reward of 1.5 is given.
- **Target Torque:** We encourage the setup policy to match the output of the target policy by minimising the error: $\exp[-2.0 \cdot (\pi_\phi(s_t) - \pi_\theta(s_t))^2]$.
- **Target Value:** We evaluate the effect of using the target policy value function as the reward for the setup policy. This is not the same as the original reward, as it is not provided by the environment, but by $\beta V^{\pi^i_\theta}(s_t)$, where β is scaling factor 0.01.
- **Advantage Weighted Target Value:** We evaluate the reward introduced in (3).

It can be seen from Fig. 3 that a reward using **Advantage Weighted Target Value** performs better than other reward choices, achieving the highest average success rate (78.7%). Using the target value reaches a success rate of 63.9%. This disparity, and the large variation in the target value (green line), shows the effect of over estimation bias often seen with value functions, validating our idea that weighting the value by the advantage reduces the effect of this bias when training setup policies.

C. *Comparisons*

We compare our method with several others, including end to end methods, without using setup policies, and a method that uses proximity prediction to train a transition policy [14].

- **Single Policy:** We train a single end to end policy on the terrain sample.

TABLE II
SUCCESS AND AVERAGE DISTANCE OF THE TOTAL LENGTH COVERED WHEN SWITCHING FROM A WALKING POLICY TO A JUMP POLICY ON A SINGLE TERRAIN SAMPLE

	Success%	Distance%
Single Policy	0.0	11.3
Single Policy With Curriculum	0.0	3.8
Without Setup Policies	1.5	51.3
Learn When to Switch	0.0	44.2
Proximity Prediction	51.3	82.4
Setup Policy (Ours)	**82.2**	**96.1**

- **Single Policy With Curriculum:** We utilise curriculum learning to train a single end to end policy on the terrain sample (using the method outlined in [33]).
- **Without Setup Policies:** We switch to the target policy from the default policy as soon as the terrain oracle detects the next terrain.
- **Learn When to Switch:** We collect data by switching from the default policy to the target policy at random distances from the Jump. We use supervised learning to learn when the robot is in a suitable state to switch. Details of this method is found in our previous work [24].
- **Proximity Prediction:** We follow the method defined by Lee et al. [14] to train Transition Policies (one for each behaviour) using a proximity predictor function $P(s_t)$. $P(s_t)$ outputs a continuous value that indicates how close the agent is to a configuration that resulted in successful traversal. $P(s_t)$ is trained using supervised learning from success and failure buffers. The reward used to train the transition policy is the dense reward created by $P(s_{t+1}) - P(s_t)$, encouraging the agent to move closer to a configuration that results in successful switching. For accurate comparison we initialise the transition policy with weights from the walk policy, and utilise the terrain oracle for policy selection (the paper refers to a rule-based meta-policy in place of a terrain oracle [14]).
- **Setup Policy (Ours):** We follow Algorithm 1 to train setup policies.

We can see from Table II that our method for developing setup policies performs the best (82.2% success rate), compared to other methods. A single policy was unable to traverse the difficult jump terrain, even with an extended learning time and curriculum learning. The poor performance of **Single Policy With Curriculum** was the result of the robot not progressing through the curriculum, and as a consequence is unable to move without assistive forces during evaluation. In contrast, the single policy trains without assistive forces, so while the single policy still fails to complete the task, it is able to make forward progress when evaluated. These results highlight the difficulty of learning to walk and jump as a single behaviour. We show that setup policies are necessary for this problem, with **Without Setup Policies** unable to successfully traverse the terrain (1.5% success). **Learning When to Switch** also performed poorly as there are very few states that overlap between the default policy and the target policy for the switch estimator to learn. The **Proximity Prediction** method was able to successfully traverse the jump 51.3% of the time. It is worth noting that this method required approximately three times more training episodes than our method to reach the provided results, despite also being initialised from the default policy.

These experiments show that for the difficult jump terrain, we require a specialised target policy dedicated to learning the complex jumping behaviour. We show that this specialised policy has a trajectory that does not overlap with the default policy, and thus setup policies are required. Our method for learning setup policies achieved the highest performance on this difficult task.

TABLE III
SUCCESS RATE AND PERCENTAGE OF THE TOTAL TERRAIN LENGTH TRAVELLED FROM 1000 EPISODES OF ALL 5 TERRAIN TYPES, RANDOMLY SHUFFLED EACH EPISODE

	Success %	Distance %
Without Setup Policies	1.9	36.3
With Setup Policies	**71.2**	**80.2**

TABLE IV
NUMBER OF FAILURES BY TERRAIN TYPE FROM 1000 EPISODES OF ALL 5 TERRAIN TYPES, RANDOMLY SHUFFLED EACH EPISODE

	Jump	Gap	Hurdle	Stairs	Steps
Without Setup Policies	782	52	**36**	**31**	**86**
With Setup Policies	**36**	**48**	43	65	96

D. Multiple Terrain Types

For our final experiment we train setup policies for several additional terrain types. In total we now have modules for jumps, gaps, hurdles, stairs, and stepping stones. Fig. 4 shows a sequence with each terrain type. We follow the pipeline shown in Fig. 2(a), where a terrain oracle determines which module should be selected. On selection, the setup policy is employed until the robot reaches a switch state (τ_ϕ), then switching to the target policy. Termination criteria τ_θ (introduced in Section III) determines when to switch back to the default policy for the jump obstacle only. All other policies resemble the behaviour of the default policy on flat terrain, i.e. we continue using the target policy after the terrain has been traversed for gaps, hurdles, stairs, and stepping stones, until the oracle determines the terrain has changed.

We evaluate with a sequence of each of the 5 terrain types, randomly shuffled before each run. We perform 1000 runs, with a total of 5000 various terrain types to be traversed. In Table III we show the average percentage of total distance travelled and success rate with and without using setup policies. A successful run in this scenario is defined as traversing the last terrain in the sequence. Table IV shows the failure count for each terrain type.

We can see from Table III that setup policies improve the success rate compared to switching policies without the intermediate policy (71.2% success compared to 1.9%). Table IV provides a breakdown of the number of times the robot failed on each terrain. From Table IV we can see that setup policies are

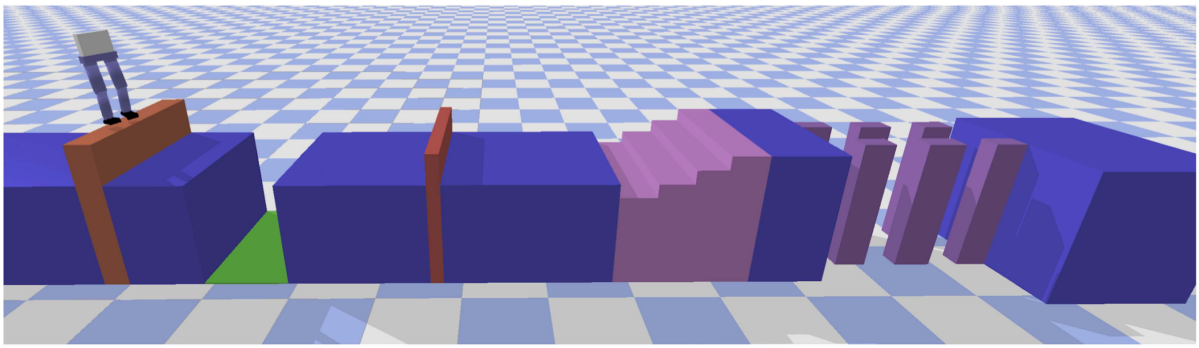

Fig. 4. Setup policies enable a biped to transition between complex visuo-motor behaviours for traversing a sequence of diverse terrains.

TABLE V
SUCCESS RATE AND PERCENTAGE OF THE TOTAL TERRAIN LENGTH TRAVELLED FROM 1000 EPISODES OF 4 TERRAIN TYPES (**WITHOUT THE JUMP TERRAIN**), RANDOMLY SHUFFLED EACH EPISODE

	Success %	Distance %
Without Setup Policies	68.2	78.1
With Setup Policies	**76.7**	**84.6**

TABLE VI
NUMBER OF FAILURES BY TERRAIN TYPE FROM 1000 EPISODES OF 4 TERRAIN TYPES (**WITHOUT THE JUMP TERRAIN**), RANDOMLY SHUFFLED EACH EPISODE

	Gap	Hurdle	Stairs	Steps
Without Setup Policies	74	**41**	64	142
With Setup Policies	**44**	51	**49**	**90**

most critical for traversing the difficult jump terrain (reducing the failures from 782 to 36), though we also see improvements for gap terrain.

To investigate further, we exclude the jump obstacle from the terrain sequence and still observe a significant benefit using setup policies. Table V shows the performance increase from 68.2% to 76.7% of successful traversals for 4 terrain types (excluding the jump obstacle). Common failure modes were seen when the robot veered from the walking line (centre y position) and was unable to recover in time for the next upcoming terrain. This often occurred before the stepping stone obstacle, which had the highest failure rate of the remaining terrain types (Table VI). Setup policies showed a clear benefit for each separate terrain type, however, we can see with the multi-terrain experiments there were cases where not using setup policies performed better (Table IV and VI). For the behaviours that did not require jumping with both feet switching early (**Without Setup Policies**) gave the robot time to correct its walking line before approaching the next terrain.

Despite the clear need for setup policies, we attribute failures to out of distribution states, i.e. the robot visits states in the terrain sequence that it does not experience when training on the individual terrain types. Earlier implementations found that training setup policies from less diverse starting locations resulted in poor performance when evaluated across multiple terrain types. In these experiments, setup policies successfully guided the robot during a set stage of the walking cycle, however were unable to control the robot when in other stages of walking, as were common after traversing different terrain types. Training setup policies for a wider range of states will be investigated in future work.

V. CONCLUSION

It is common practice for legged robots to have separate locomotion policies to traverse different terrain conditions. We propose setup policies that enable smooth transition from the trajectory of one locomotion policy to the trajectory of a target policy for traversing difficult terrain with a dynamic biped. We use deep reinforcement learning to learn the setup policies for which we introduce a novel reward based on the Advantage Weighted Target Value, utilising the scaled value prediction of the target policy as the reward for the setup policy.

In simulation experiments for transitioning from a walking policy to a jumping policy, we show using an in-between setup policy yields a higher success rate compared to using a single policy (0% success rate), or the two policies without the setup policy (from 1.5% success rate without setup policies to 82%). We further show that our method is scalable to other terrain types, and demonstrate the effectiveness of the approach on a sequence of difficult terrain conditions, improving the success rate from 1.9% without setup policies to 71.2%.

A limitation of our method is that setup policies are trained from a default walking policy. We would like to improve the scope of the setup policies such that a wider range of states are funnelled towards the target policy. The idea of utilising the advantage (estimated by the temporal difference (TD) error) as a confidence metric can be applied generally for region of attraction (RoA) expansion, safe reinforcement learning, and also for blending several behaviours for performing complex composite tasks. These ideas will be explored in future work. A significant assumption in this work is that we have access to a terrain oracle, removing this dependency will be required as we consider experiments in the real world.

REFERENCES

[1] X. B. Peng, G. Berseth, and M. van de Panne, "Dynamic terrain traversal skills using reinforcement learning," *ACM Trans. Graph.*, vol. 34, no. 4, pp. 1–11, Jul. 2015. [Online]. Available: https://dl.acm.org/doi/10.1145/2766910

[2] N. Heess et al., "Emergence of locomotion behaviours in rich environments," Jul. 2017, *arXiv:1707.02286*.
[3] X. B. Peng, G. Berseth, K. Yin, and M. Van De Panne, "DeepLoco: Dynamic locomotion skills using hierarchical deep reinforcement learning," *ACM Trans. Graph.*, vol. 36, no. 4, pp. 1–13, Jul. 2017.
[4] X. B. Peng, P. Abbeel, S. Levine, and M. van de Panne, "DeepMimic: Example-guided deep reinforcement learning of physics-based character skills," *ACM Trans. Graph.*, vol. 37, no. 4, pp. 1–14, Jul. 2018.
[5] Z. Xie, H. Y. Ling, N. H. Kim, and M. v. d. Panne, "ALLSTEPS: Curriculum-driven learning of stepping stone skills," in *Proc. ACM SIGGRAPH / Eurographics Symp. Comput. Animation*, 2020, pp. 1–12.
[6] D. R. Song, C. Yang, C. McGreavy, and Z. Li, "Recurrent deterministic policy gradient method for bipedal locomotion on rough terrain challenge," in *Proc. IEEE 15th Int. Conf. Control, Automat., Robot. Vis.*, Nov. 2018, pp. 311–318. [Online]. Available: http://arxiv.org/abs/1710.02896
[7] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *Sci. Robot.*, vol. 5, no. 47, Oct. 2020, Art. no. eabc5986.
[8] N. Rudin, D. Hoeller, P. Reist, and M. Hutter, "Learning to walk in minutes using massively parallel deep reinforcement learning," in *Proc. Conf. Robot Learn.*, 2021, pp. 91–100.
[9] J. Lee, J. Hwangbo, and M. Hutter, "Robust Recovery Controller for a Quadrupedal Robot Using Deep Reinforcement Learning," Jan. 2019, *arXiv:1901.07517*.
[10] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artif. Intell.*, vol. 112, no. 1-2, pp. 181–211, Aug. 1999.
[11] P.-L. Bacon, J. Harb, and D. Precup, "The option-critic architecture," in *Proc. 31st AAAI Conf. Artif. Intell.*, San Francisco, California, USA, 2017, pp. 1726–1734.
[12] K. Frans, J. Ho, X. Chen, P. Abbeel, and J. Schulman, "Meta learning shared hierarchies," in *Proc. Int. Conf. Learn. Representations*, 2018.
[13] J. Merel et al., "Hierarchical visuomotor control of humanoids," in *Proc. Int. Conf. Learn. Representations*, 2019.
[14] Y. Lee, S.-H. Sun, S. Somasundaram, E. S. Hu, and J. J. Lim, "Composing complex skills by learning transition policies," in *Proc. Int. Conf. Learn. Representations*, 2019.
[15] X. B. Peng, M. Chang, G. Zhang, P. Abbeel, and S. Levine, "MCP: Learning composable hierarchical control with multiplicative compositional policies," in *Proc. 33rd Conf. Neural Inf. Process. Syst.*, Vancouver, Canada, 2019, pp. 3681–3692.
[16] R. Strudel, A. Pashevich, I. Kalevatykh, I. Laptev, J. Sivic, and C. Schmid, "Learning to combine primitive skills: A step towards versatile robotic manipulation," in *Proc. IEEE Int. Conf. Robot. Automat.,*2020, pp. 4637–4643.
[17] S. Schaal*, Dynamic Movement Primitives–A Framework for Motor Control in Humans and Humanoid Robotics.* Tokyo, Japan: Springer International Publishing, 2006.
[18] P. Faloutsos, M. van de Panne, and D. Terzopoulos, "Composable controllers for physics-based character animation," in *Proc. 28th Annu. Conf. Comput. Graph. Interactive Techn. - SIGGRAPH*, 2001, pp. 251–260.
[19] J. Andreas, D. Klein, and S. Levine, "Modular multitask reinforcement learning with policy sketches," in *Proc. 34th Int. Conf. Mach. Learn.*, 2017, pp. 166–175.
[20] T. Kipf et al., "CompILE: Compositional imitation learning and execution," in *Proc. 36th Int. Conf. Mach. Learn.*, 2019, pp. 3418–3428.
[21] X. B. Peng, G. Berseth, and M. van de Panne, "Terrain-adaptive locomotion skills using deep reinforcement learning," *ACM Trans. Graph.*, vol. 35, no. 4, pp. 1–12, Jul. 2016.
[22] K. Pertsch, Y. Lee, and J. J. Lim, "Accelerating reinforcement learning with learned skill priors," in *Proc. Conf. Robot Learn.*, 2020.
[23] J.-S. Ha, Y.-J. Park, H.-J. Chae, S.-S. Park, and H.-L. Choi, "Distilling a hierarchical policy for planning and control via representation and reinforcement learning," Nov. 2020. [Online]. Available: http://arxiv.org/abs/2011.08345
[24] B. Tidd, N. Hudson, A. Cosgun, and J. Leitner, "Learning when to switch: Composing controllers to traverse a sequence of terrain artifacts," in *Proc. IEEE Int. Conf. Intell. Robots Syst.*, 2021, pp. 5144–5150.
[25] A. S. Vezhnevets et al., "FeUdal networks for hierarchical reinforcement learning," in *Proc. 34th Int. Conf. Mach. Learn.*, 2017, pp. 3540–3549.
[26] J. Xu, H. Xu, B. Ni, X. Yang, X. Wang, and T. Darrell, "Hierarchical style-based networks for motion synthesis," in *Proc. Eur. Conf. Comput. Vis.*, 2020, pp. 178–194.
[27] C. Yang, K. Yuan, Q. Zhu, W. Yu, and Z. Li, "Multi-expert learning of adaptive legged locomotion," *Sci. Robot.*, vol. 5, no. 49, 2020, Art. no. eabb2174.
[28] X. Da, R. Hartley, and J. W. Grizzle, "Supervised learning for stabilizing underactuated bipedal robot locomotion, with outdoor experiments on the wave field," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2017, pp. 3476–3483.
[29] E. Coumans and Y. Bai, "PyBullet, a python module for physics simulation for games, robotics and machine learning," 2020, [Online]. Available: http://pybullet.org
[30] P. Zaytsev, S. J. Hasaneini, and A. Ruina, "Two steps is enough: No need to plan far ahead for walking balance," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2015, pp. 6295–6300.
[31] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," Aug. 2017, *arXiv:1707.06347*.
[32] P. Dhariwal et al., *OpenAI Baselines*, 2017. [Online]. Available: https://github.com/openai/baselines
[33] B. Tidd, N. Hudson, and A. Cosgun, "Guided curriculum learning for walking over complex terrain," in *Proc. Australas. Conf. Robot. Automat.*, 2020.
[34] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proc. 35th Int. Conf. Mach. Learn.*, 2018, pp. 1861–1870.