

Robotic Control Using Model Based Meta Adaption

Karam Daaboul*, Joel Ikels* and J. Marius Zöllner

Abstract—In machine learning, meta-learning methods aim for fast adaptability to unknown tasks using prior knowledge. Model-based meta-reinforcement learning combines reinforcement learning via world models with Meta Reinforcement Learning (MRL) for increased sample efficiency. However, adaption to unknown tasks does not always result in preferable agent behavior. This paper introduces a new Meta Adaptation Controller (MAC) that employs MRL to apply a preferred robot behavior from one task to many similar tasks. To do this, MAC aims to find actions an agent has to take in a new task to reach a similar outcome as in a learned task. As a result, the agent will adapt quickly to the change in the dynamic and behave appropriately without the need to construct a reward function that enforces the preferred behavior.

I. Introduction

Adaptive behavior lies in the very nature of life as we know it. By forming a variety of behaviors, the animal brain enables its host to adapt to environmental changes continuously [1]. Toddlers, for example, can learn how to walk in the sand in several moments, whereas robots often struggle to adapt fast and show rigid behavior encountering a task not seen before. Fast adaption is possible because animals do not learn from scratch and leverage prior knowledge to solve a new task. In machine learning, the domain of meta-learning takes inspiration from this phenomenon by enabling a learning machine to develop a hypothesis on how to solve a new task using information from prior hypotheses of similar tasks [2]. Thus, it aims to learn models that are quickly adaptable to new tasks and can be described as a set of methods that apply a learned prior of common task structure to make a generalized inference with small amounts of data [2], [3].

The domain of model-based reinforcement learning (MBRL) comprises methods that enable a Reinforcement Learning (RL) agent to successfully master complex behaviors using a deep neural network as a model of a tasks system dynamics [4]. To solve an RL task, this dynamics model is utilized to optimize a sequence of actions (e.g., with model predictive control) or to optimize a policy, making MBRL more sample efficient than model-free reinforcement learning (MFRL) [5], [6], [7]. Even though MBRL methods show improved sample efficiency compared to MFRL approaches, the amount of training data needed to reach "good" performance scales

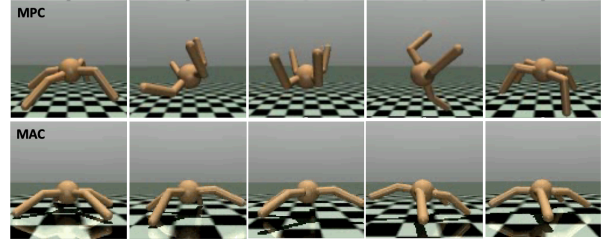


Fig. 1. Action sequences of an ant robot during meta-testing. The test task is to adapt to the gravity of $5 m/s^2$. A model-based meta-reinforcement learning approach with MPC results in an undesired robot behavior (row 1). MAC finds a behavior similar to the one learned at its reference task (row 2).

exponentially with the dimensionality of the input state-action space of the dynamics model [8]. Additionally, data scarcity is even more challenging when a system has to adapt online while executing a task. A robot, for example, might encounter sudden changes in system dynamics (e.g., damaged joints) or changes in environmental dynamics (e.g., new terrain conditions) that require fast online adaption. By combining meta-learning and MBRL, robots can learn how to quickly form new behaviors when the environment- or system-dynamics change [9], [10], [11], [12]. However, newly formed behavior might be undesirable even if the underlying task is mastered correctly according to the environment's reward function. For example, as seen in figure 1, a robot Ant, trained to walk as fast as possible, will start to jump or roll if the gravity of its environment is very low. In a real-world setting, such a situation might damage the robot. Therefore, the RL agent requires a tailored reward function to form behavior that does no damage. Nevertheless, designing a reward function is challenging since it is time-consuming and challenging to master, especially for various tasks. This paper introduces a controller for MBRL that employs meta-learning to apply a selected robot behavior from one robotic task to a range of similar tasks. In other words, it aims to find actions the robot has to take in a new task to reach a similar outcome as in a learned task. Thus, it alleviates the need to construct a reward function that enforces preferred behavior. It builds on top of the FAMLE algorithm by Kaushik et al. [11] making use of an Embedding Neural Network (ENN) for quick adaption to new tasks through task embeddings as learned priors. By combining an ENN with an RL policy of a reference task, the controller predicts which actions need to be taken in unseen tasks to mimic the behavior

*Equal contributions

Karlsruhe Institute of Technology, Kaiserstr. 12, 76131 Karlsruhe, Germany {daaboul, marius.zoellner}@kit.edu, joel.ikels@student.kit.edu

of the reference task. While being initialized with the most likely embedding, a trained meta is adapted to approximate future environment states and compare them to the preferred states of the reference task. Actions leading to states in the unseen task that are very similar to those reached by the RL policy in the reference task are then chosen to be executed in the environment. To account for the usage and adaption of a meta-model during planning, we call our approach Meta Adaptation Controller (MAC).

First, we introduce related work and preliminaries. Next, the challenge and our approach to solving it are described. Finally, experiment results compare MAC with model predictive control (MPC) employing different meta-learning methods are presented. The source code for all our experiments are available at <https://github.com/jikels/mac>.

II. Related Work

In recent years, robotics has achieved remarkable success with model-based RL approaches [13], [14], [15]. The agent can choose optimal actions by utilizing the experiences generated by the model [7]. As a result, the data required for model-based methods are typically much smaller than their model-free counterparts, making these algorithms more attractive for robotic applications. One drawback in many of these works is the assumption that the environment is stationary. In real robot applications, however, many uncertainties are difficult to model or predict, some of which are internal (e.g., malfunctions [9]) and others external (e.g., wind [12]). These uncertainties make the stationary assumption impractical. That can lead to suboptimal behavior or even catastrophic failure. Therefore, a quick adaptation of the learned model is critical.

”Gradient-based meta-learning methods leverage gradient descent to learn the commonalities among various tasks” [16, p. 1]. One such method introduced by Finn et al. [17] is Model-Agnostic Meta-Learning (MAML). The key idea of MAML is to tune a model’s initial parameters such that the model has maximal performance on a new task. Here, meta-learning is achieved with bi-level optimization, a model task-specific optimization, and a task-agnostic meta-optimization. Instantiated for MFRL, MAML uses policy gradients of a neural network model, whereas, in MBRL, MAML is used to train a dynamics model. REPTILE by Nicol et al. [18] is the first-order implementation of MAML. In contrast to MAML, task-specific gradients do not need to be differentiated through the optimization process. This makes REPTILE more computationally efficient with similar performance.

A model-based approach using gradient-based MRL was presented in the work of Nagabandi et al. [9] and targets the online adaption of a robotic system that encounters different system dynamics in real-world environments. In this context, Kaushik et al. [11] point

out that in an MRL setup where situations do not possess strong global similarity, finding a single set of initial parameters is often not sufficient to learn quickly. One potential solution would be to find several initial sets of model parameters during meta-training and, when encountering a new task, use the most similar one so that an agent can adapt through several gradient steps. Their work Fast Adaptation through Meta-Learning Embeddings (FAMLE) approaches this solution by extending a dynamical model input with a learnable d -dimensional vector describing a task. Similarly, Belkhale et al. [12] introduce a meta-learning approach that enables a quadcopter to adapt online to various physical properties of payloads (e.g., mass, tether length) using variational inference. Intuitively each payload causes different system dynamics and therefore defines a task to be learned. Since it is unlikely to model such dynamics by hand accurately and it is not realistic to know every payload’s properties value beforehand, the meta-learning goal is the rapid adaption to unknown payloads without prior knowledge of the payload’s physical properties. That is why a probabilistic encoder network finds a task-specific latent vector fed into a dynamics network as an auxiliary network. Using the latent vector, the dynamics network learns to model the factors of variation that affect the payload’s dynamics and are not present in the current state. All these algorithms use MPC during online adaption. Our work introduces a new controller for online adaption in a model-based meta-reinforcement learning setting.

III. Preliminaries

A. Meta Learning

Quick online adaption to new tasks can be viewed in the light of a few-shot learning setting where the goal of meta-learning is to adapt a model f_θ to an unseen task \mathcal{M}_j of a task distribution $p(\mathcal{M})$ with a small amount of k data samples [17]. The meta-learning procedure usually is divided into meta-training with n meta-learning tasks \mathcal{M}_i and meta-testing with y meta-test tasks \mathcal{M}_j both drawn from $p(\mathcal{M})$ without replacement [3]. During meta-training, task data may be split into train and test sets usually representing k data points of a task $\mathcal{D}^{\text{meta-train}} = \{(\mathcal{D}_{i=1}^{\text{tr}}, \mathcal{D}_{i=1}^{\text{ts}}), \dots, (\mathcal{D}_{i=n}^{\text{tr}}, \mathcal{D}_{i=n}^{\text{ts}})\}$. Meta-testing task data $\mathcal{D}^{\text{meta-test}} = (\mathcal{D}_{j=1}^{\text{meta-test}}, \dots, \mathcal{D}_{j=y}^{\text{meta-test}})$ is hold out during meta-training [3]. Meta-training is then performed with $\mathcal{D}^{\text{meta-train}}$ and can be viewed as bi-level learning of optimal meta parameters θ^* , as introduced in [19]:

$$\theta^* = \arg \min_{\theta} \overbrace{\sum_{i=1}^n \mathcal{L}_{\mathcal{D}_i \sim \mathcal{M}_i}(\phi_i)}^{\text{outer-level}} \quad (1)$$

where $\underbrace{\phi_i = \text{Alg}_{\mathcal{D}_i \sim \mathcal{M}_i}^{\psi}(\theta)}_{\text{inner-level}}$

In the inner-level, an update algorithm Alg with hyper-parameters ψ must find task-specific parameters ϕ_i by

adjusting meta-parameters θ . In the outer-level, θ must be adjusted to minimize the cumulative loss of all ϕ_i across all learning tasks by finding common characteristics of different tasks through meta parameters θ^* .

Once θ^* is found, it can be used during meta-testing for quick adaption: $\phi_j = \text{Alg}^\psi(\theta^*, \mathcal{D}_j)$

B. Model-based Reinforcement Learning

In RL, a task can be described as a Markov Decision Process (MDP) $\mathcal{M} = \{S, A, p(s_{t+1} | s_t, a_t), r, T\}$ with a set of states S , a set of actions A , a reward function $r : S \times A \mapsto \mathbb{R}$, an initial state distribution $p(s_{t=0})$, a transition probability distribution $p(s_{t+1} | s_t, a_t)$, and a discrete-time finite or continuous-time infinite horizon T . MBRL methods sample ground truth data $\mathcal{D}_i = \{(s_0, a_0, s_1), (s_1, a_1, s_2), \dots\}$ from a specific task \mathcal{M}_i and use this data to train a dynamics model $p_\theta(s_{t+1} | s_t, a_t)$ that estimates the underlying dynamics of the task to approximate which state follows which action. This is done by optimizing the weights θ to maximize the log-likelihood of the observed data:

$$\begin{aligned} \theta^* &= \arg \max_{\theta} p(\mathcal{D}_i | \theta) \\ &= \arg \max_{\theta} \sum_{(s_t, a_t, s_{t+1}) \in \mathcal{D}_i} \log p_\theta(s_{t+1} | s_t, a_t) \end{aligned} \quad (2)$$

The learned dynamics model is then utilized to optimize a sequence of actions (e.g., with model predictive control) or to optimize a policy [6], [7].

C. Gradient-based Reinforcement Learning with REPTILE

REPTILE from Nichol et al. [18] is a first-order implementation of MAML. During meta-training, task data in the form of \mathcal{K} trajectories $\mathcal{D}_i = \{(s_1, a_1, r_1, \dots, s_H), \dots, \mathcal{K}\}$ is sampled with roll-outs from f_θ or by taking random actions. A single task \mathcal{M}_i is sampled from $p(\mathcal{M})$ without replacement for each training-iteration that passes the inner-level and outer-level once. In the inner-level, task-specific parameters ϕ_i are generated by adapting f_θ with $g > 1$ steps of stochastic gradient descent as Alg and its learning rate $\psi = \beta$:

$$\phi_i = \theta - \beta \nabla_{\theta} \mathcal{L}_{\mathcal{D}_i}(\theta) \quad (3)$$

In the outer-level the parameters θ are then being adjusted to minimize the euclidean distance between θ and ϕ_i with learning rate α :

$$\theta \leftarrow \theta + \alpha (\phi_i - \theta) \quad (4)$$

D. Model-based Meta-Reinforcement Learning using task embeddings

Each dynamic encountered by an agent can be represented by an MDP and therefore interpreted as an RL task \mathcal{M}_i . Since, in real-world applications, new dynamics

can appear at any time (e.g., a malfunctioning robot leg), a new task could appear at any time. Hence, a task can be understood as an arbitrary trajectory segment of k timesteps under a specific dynamic. A meta-learner is trained to adapt to the distribution of these temporal fragments based on o recent observations [9], [11]. FAMLE, proposed by Kaushik et al. [11], extends the input of a dynamics model with an additional d-dimensional vector, denoted as h , which describes a task \mathcal{M}_i . Through a meta-training process that jointly optimizes model parameters θ and task embeddings h , FAMLE learns several sets of initial model parameters, each conditioned on a specific task represented by h_i . This results in a task-conditioned dynamics model $p_\theta(s_{t+1} | s_t, a_t, h)$ that can quickly adapt to unseen tasks. If an unseen task \mathcal{M}_j appears, its similarity to prior tasks is measured and the most probable task embedding, denoted as h^{likely} , is utilized to condition the model parameters, facilitating a quicker adaptation to the new task. The meta-training process for FAMLE is described in Algorithm 1. Before the meta-training process, n training tasks are randomly sampled from the distribution $p(\mathcal{M})$. For each training task \mathcal{M}_i , a set of training data $\mathcal{D}_i = \{(s_t, a_t, s_{t+1}) | t = 0, \dots, T\}$ is generated by a simulated robot randomly taking T actions. Additionally, initial embeddings $\mathcal{H} = \{h_i | i = 1 \dots n\}$ corresponding to each training task are initialized. The goal of the meta-training process is to find the initial model parameters θ^* and the n task embeddings \mathcal{H}^* that minimize the loss for any task \mathcal{M} sampled from $p(\mathcal{M})$. During the meta-training process, the algorithm updates the model parameters and the task embeddings based on the sampled tasks and their corresponding training data, aiming to achieve this goal.

$$\begin{aligned} \theta^*, \mathcal{H}^* &= \arg \min_{\theta, h_i} \sum_{i=1}^n \mathcal{L}_{\mathcal{D}_i}(\phi_i, h_i) \\ &\text{where } \phi_i, h_i = \text{Alg}_{\mathcal{D}_i}^\psi(\theta, h) \end{aligned} \quad (5)$$

The loss of a task-conditioned dynamical model for a specific training task $\sim \mathcal{M}_i$ be as follows:

$$\mathcal{L}_{\mathcal{D}_i}(\phi_i, h_i) = -\frac{1}{K} \sum_{t=1}^{t=K} \log p_{\phi_i}(s_{t+1} | s_t, a_t, h_i) \quad (6)$$

Following the REPTILE algorithm, bi-level optimization is achieved by making a gradient-based, task-specific update of θ and h in the inner level with a fixed $\psi = \beta$:

$$\begin{aligned} \phi_i &= \theta - \beta \nabla_{\theta} \mathcal{L}_{\mathcal{D}_i}(\theta, h) \\ h'_i &= h - \beta \nabla_h \mathcal{L}_{\mathcal{D}_i}(\theta, h) \end{aligned} \quad (7)$$

and simultaneously updating θ and h towards their task-specific counterparts in the outer level:

$$\begin{aligned} \theta &\leftarrow \theta + \alpha (\phi_i - \theta) \\ h_i &\leftarrow h_i + \alpha (h'_i - h_i) \end{aligned} \quad (8)$$

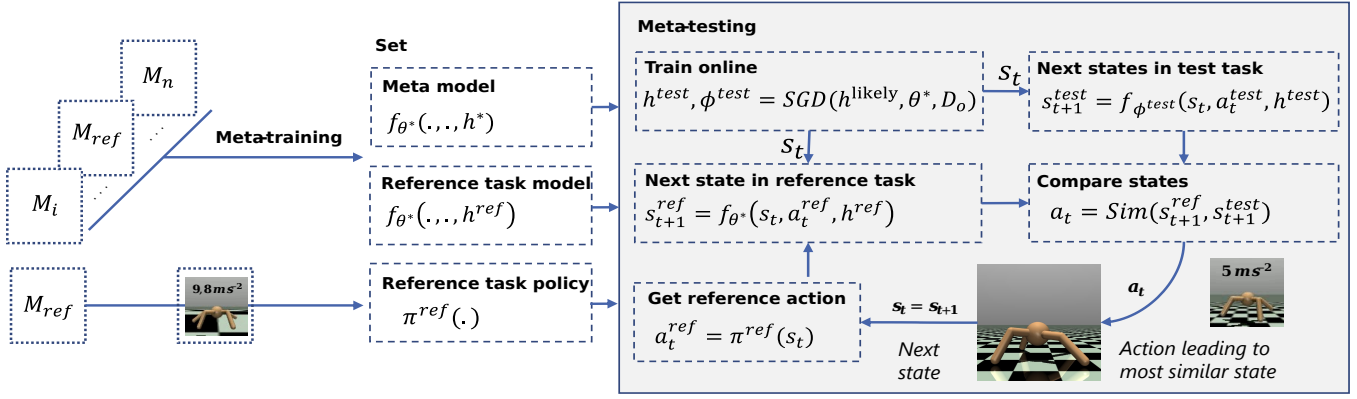


Fig. 2. A high-level overview of the meta adaptation controller. While initializing with the most likely embedding, a trained meta-model is adapted to approximate future environment states and compare them to states of the reference task. Actions leading to states in the unseen task that are very similar to those reached by the RL policy in the reference task are then chosen to be executed in the environment.

Algorithm 1 Meta-training process using REPTILE and an Embedding Neural Network

Require: Distribution $p(\mathcal{M})$ over tasks
 Require: Learning rate outer-level $\alpha \in \mathbb{R}^+$
 Require: Learning rate inner-level $\beta \in \mathbb{R}^+$
 Require: Number of sampled tasks n
 Require: Empty Dataset $\mathcal{D}^{\text{meta-train}} = \{\}$
 Require: $\text{Alg}_{\mathcal{D}_i}^{\psi=\beta}()$ as k steps of stochastic gradient descent
 1: for $i = 1 \dots n$ do
 2: Sample a training task \mathcal{M}_i from $p(\mathcal{M})$
 3: Save the task: $\mathcal{M} \leftarrow \mathcal{M}_i$
 4: Collect task data: $\mathcal{D}_i = \{(s_t, a_t, s_{t+1}) | t = 0, \dots, T\}$
 5: Save task data: $\mathcal{D}^{\text{meta-train}} \leftarrow \mathcal{D}^{\text{meta-train}} \cup \{\mathcal{D}_i\}$
 6: end for
 7: for $x = 0, 1, \dots$ do
 8: Sample task data: $\mathcal{D}_i \sim \mathcal{D}^{\text{meta-train}}$
 9: Perform k steps of SGD with: $\phi_i, h'_i = \text{Alg}_{\mathcal{D}_i}^{\psi=\beta}(\theta, h_i)$
 10: Perform update: $\theta \leftarrow \theta + \alpha(\phi_i - \theta)$
 11: Perform update: $h_i \leftarrow h_i + \alpha(h'_i - h_i)$
 12: end for
 13: return (θ, \mathcal{H}) as $(\theta^*, \mathcal{H}^*)$

During online adaptation (meta-testing), the most likely situational embedding h^{likely} is defined based on o recent observations $\mathcal{D}_j = \{(s_t, a_t, s_{t+1}) | t = 0, \dots, o\}$ as shown in Equation 9.

$$h^{\text{Likely}} = \arg \max_{h \in \mathcal{H}^*} \mathbb{E}_{\mathcal{D}_j} [\log p_{\theta^*}(s_{t+1} | s_t, a_t, h)] \quad (9)$$

Next, the dynamical model is updated online by simultaneously updating h^{likely} and θ^* taking g gradient steps:

$$\begin{aligned} \theta &\leftarrow \theta - \beta \nabla_{\theta} \mathcal{L}_{\mathcal{D}_j}(\theta, h^{\text{likely}}) \\ h_j &\leftarrow h^{\text{likely}} - \beta \nabla_h \mathcal{L}_{\mathcal{D}_j}(\theta, h^{\text{likely}}) \end{aligned} \quad (10)$$

IV. Agents forming adapted behavior through meta-learning

Figure 3 displays a robot incentivized to walk in one specific direction as fast as possible. The meta-learning objective is to walk successfully in different gravitational settings. First, as in algorithm 1, data is collected by randomly taking actions in different gravitational settings. Next, a meta-learning method (e.g.,

REPTILE) is used to train a meta-model. While the robot performs well during meta-testing with MPC, its adapted behavior in low gravitational settings is to jump and roll since this results in the highest reward (Fig. 1). In a real-world setting, similar adverse behaviors could have unknown consequences like damage to the robot or its environment. Designing a reward function that enables intended adaption is not a promising approach. First, developing the proper function for one specific task takes many trials, which is time-consuming. Moreover, finding a reward function that works across various tasks takes time and effort. For example, settings with low gravity require constraining the motion of the robot not to jump or roll, whereas high gravity settings demand rotation flexibility. More complex meta-learning tasks are even more challenging.

V. Apply preferred behavior to similar tasks

Rather than relying on a reward function to encourage desirable behaviors, we require a procedure that ensures the robot operates correctly with minimal supervision. One possible solution is to add constraints to the MPC optimization problem. These constraints force the predicted states to be similar to anchor states, denoted by S^{anch} , which satisfy a task-anchor condition. An example of such an anchor state is shown in Figure 3 (indicated by red arrows), where the task-anchor set is defined as:

$$S^{\text{anch}} = \{s_{\text{rot}} \in S \mid s_{\text{rot}}^{\text{min}} \leq s_{\text{rot}} \leq s_{\text{rot}}^{\text{max}}\}$$

Here, S represents the state space, and s_{rot} is a state variable that accounts for the robot's rotational motion and is constrained by a predefined, manually-selected interval, such as $[s_{\text{rot}}^{\text{min}}, s_{\text{rot}}^{\text{max}}]$. We formulated our constraint optimization problem as follows:

$$\begin{aligned} \max_{a_{[1]}} & \sum_{t=0}^{T-1} r(s_t, a_t) \\ \text{s.t.} & s_{t+1} = f_{\phi^{\text{test}}}(s_t, a_t, h^{\text{test}}) \quad \forall t \in \{0, \dots, T-1\} \\ & s_{t+1} \in S^{\text{anch}} \quad \forall t \in \{0, \dots, T-1\} \end{aligned} \quad (11)$$

We employed a random shooting method to solve this highly nonlinear optimization problem. We sampled many action trajectories, and the return of each was calculated using the model. During the state-action sequence calculation, the MPC algorithm checks for sequence states $s \notin S^{anch}$ that violate the constraints. This was done by assigning zero rewards to any sequence states that violated the constraints, which helped to steer the robot away from undesired behavior. By solving the optimization problem, the robot adapts to low-gravity conditions without jumping or rolling.

However, defining anchors for various tasks requires careful selection including many trials. Therefore, instead of empirically finding the desired movement across all tasks, we only choose a movement of one specific task that may work well in similar tasks. This movement can be extracted from learned RL policies or classical feedback controllers. We call this task a reference task and the used policy (controller) a reference policy. Our algorithm aims to find actions the robot has to take in a new test task to reach a similar outcome as the desired outcome in the reference task using the reference policy. To achieve that, our algorithm utilizes two variations of a meta-trained embedding neural network (ENN). The first variation entails the meta-trained network with meta parameters θ^* and learned embeddings \mathcal{H}^* . The second variation entails the meta-trained network with meta parameters θ^* conditioned on the embedding of a reference task h^{ref} .

Putting these pieces together, MAC (Fig. 2) optimizes a sequence of states $s_{[.]}$ and actions $a_{[.]}$ to maximize the predicted reward in the test-task while also eventually ensuring dynamics feasibility:

$$\begin{aligned} \max_{s_{[.]}, a_{[.]}} \sum_{t=0}^{T-1} r(s_t, a_t) \quad (12) \\ \text{s.t.: } s_{t+1} = f_{\phi^{test}}(s_t, a_t, h^{test}) \quad \forall t \in \{0, \dots, T-1\} \\ |s_{t+1} - s_{t+1}^{ref}| \leq \delta \quad \forall t \in \{0, \dots, T-1\} \end{aligned}$$

Where s_{t+1}^{ref} is the desired outcome of the reference task given the current state s_t and using the reference policy:

$$s_{t+1}^{ref} = f_{\theta^*}(s_t, \pi^{ref}(s_t), h^{ref}) \quad (13)$$

The constraint $|s_{t+1} - s_{t+1}^{ref}| \leq \delta$ is approximated using a similarity measurement between the predicted states and the reference state $Sim(s_{t+1}, s_{t+1}^{ref})$. As the similarity measure, we use the cosine similarity of the state vectors. After meta-training an ENN with algorithm 1, meta-testing (i.e., online adaption) is executed with algorithm 2. Here the most likely embedding is determined, and the ENN is adapted to approximate future environment states. Subsequently, MAC is used to find the action to take in a new task that reaches a similar outcome as in the reference task.

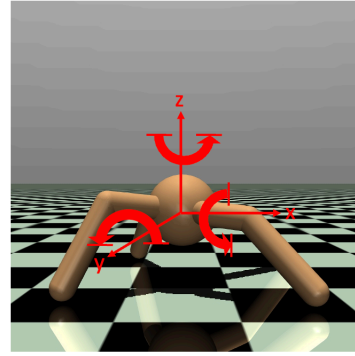


Fig. 3. An ant robot incentivized to walk as fast as possible. The red arrows depict an anchor that restricts the robot in its rotation

Algorithm 2 Meta-testing an Embedding Neural Network to adapt online using our control algorithm

Require: Meta-learned parameters θ^* and embeddings \mathcal{H}^*
 Require: Meta Adaption Controller MAC()
 Require: Empty set of o recent observations $\mathcal{D}_o = \{\}$

- 1: while task not solved do
- 2: Determine most likely embedding $h^{\text{likely}} \in \mathcal{H}^*$ given \mathcal{D}_o and θ^* {see Eq. 9}
- 3: Execute g steps of SGD using $\mathcal{D}_o, \theta^*, h^{\text{likely}}$ and receive ϕ^{test}, h^{test} {see Eq. 10}
- 4: Execute action $a_t = \text{MAC}(\phi^{test}, h^{test}, s_t)$ and receive state s_{t+1} {see Alg. 3}
- 5: Save observation $\mathcal{D}_o \leftarrow \mathcal{D}_o \cup \{(s_t, a_t, s_{t+1})\}$
- 6: if $size(\mathcal{D}_o) > o$ then remove oldest observation from \mathcal{D}_o
- 7: end while

The MAC procedure is shown in Algorithm 3, and can be summarized with the following steps:

- 1) Given the current state s_t , sample a set of reference actions A_t^{ref} using the reference policy $\pi^{ref}(s_t)$.
- 2) Using the embedding of the reference task h^{ref} , estimate a set of reference states S_{t+1}^{ref} that contains information about what states would follow if the reference actions in the reference task were executed.
- 3) Sample a set of actions A_t from an unconditional Gaussian distribution $\mathcal{N}(\mu, \Sigma)$. These actions will be used together with the test task embedding h^{test} to predict the following states S_{t+1} .
- 4) Use the reward function $r(S_t, A_t)$ to estimate a set of rewards R_t and measure the state similarity of the predicted states to the reference state $Sim(S_{t+1}, S_{t+1}^{ref})$.
- 5) From the sets S_{t+1} and A_{t+1} store only the most similar states and the actions used to reach these states.
- 6) Repeat the procedure according to the planning horizon.
- 7) Choose the first action a_t of the most similar action sequence with the highest reward.
- 8) Update the distribution $\mathcal{N}(\mu, \Sigma)$ towards action sequences with higher similarity and reward.

Algorithm 3 Meta Adaption Controller (MAC)

Require: Reward function $r()$; similarity measure $Sim()$
 Require: Action elites ϵ ; planning horizon η
 Require: Reference policy $\pi^{ref}()$
 Require: Task-specific configuration ϕ^{test}, h^{test}
 Require: Reference task configuration θ^{ref}, h^{ref}
 Require: Set of initial states S_t
 Require: Empty set of next states $\mathcal{S} = \{\}$
 Require: Empty set of actions to next states $\mathcal{A} = \{\}$
 Require: Empty set of calculated rewards $\mathcal{R} = \{\}$

- 1: Use random distribution $\mathcal{N}(\mu, \Sigma)$ to sample actions
- 2: for $t \leq \eta$ do
- 3: Sample set of ref. actions A_t^{ref} with $\pi^{ref}(S_t)$
- 4: Calculate ref. states S_{t+1}^{ref} with $f_{\theta^*}(S_t, A_t^{ref}, h^{ref})$
- 5: Sample set of actions \mathcal{A}_t from $\mathcal{N}(\mu, \Sigma)$
- 6: Get set of predictions of next states S_{t+1} with $f_{\theta^{test}}(S_t, A_t, h^{test})$
- 7: Calculate rewards R_t with $r(S_t, A_t)$
- 8: Calculate state similarity S_t^{sim} with $Sim(S_{t+1}, S_{t+1}^{ref})$
- 9: Update \mathcal{A}_t based on S_t^{sim} and R_t to consists of the ϵ most similar states with the highest rewards
- 10: $\mathcal{S} \leftarrow \mathcal{S} \cup \{S_{t+1}\}$
- 11: $\mathcal{R} \leftarrow \mathcal{R} \cup \{R_t\}$
- 12: $\mathcal{A} \leftarrow \mathcal{A} \cup \{A_t\}$
- 13: end for
- 14: Extract the first action a_t of the most similar action sequence with the highest reward.
- 15: Update $\mathcal{N}(\mu, \Sigma)$ based on \mathcal{A}
- 16: return a_t

VI. Experiments

We compare our meta adaption controller (MAC) algorithm with three baseline algorithms. The algorithms are compared during meta-testing within four different environments where the agent must quickly adapt to new tasks and collect as much reward as possible (Figure 4). The environments are based on the MuJoCo physics engine developed by Todorov et al. [20]. They have been used in previous meta-RL literature [9]: Halfcheetah-disabled, Halfcheetah-pier, Ant-disabled, and Ant-gravity.

All three baseline algorithms use a multilayer perceptron (MLP) with three hidden layers, each consisting of 512 neurons. The first baseline, RMPC, trains the MLP using the first-order implementation of Model-Agnostic Meta-Learning (MAML) known as REPTILE by Nichol et al. [18]. RMPC uses model predictive control (MPC) for meta-testing. The second baseline, FMPC, follows the approach of FAMLE by Kaushik et al. [11] and extends the MLP with an additional embedding input. FMPC then meta-trains the resulting embedding neural network (ENN) using REPTILE and also uses MPC for meta-testing. The third baseline, ANC, supplements RMPC with an anchor state constraint (see equation 11). ANC is only evaluated in the ant environments. MAC, on the other hand, uses a policy from an actor-critic module trained on one reference task per environment with the soft actor-critic algorithm by Haarnoja et al. [21] to extract reference actions. The reference task in each environment corresponds to the goal of its base environment

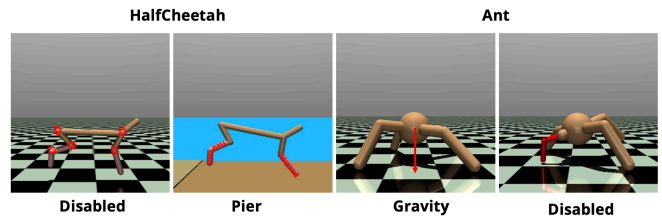


Fig. 4. Environments used to test our algorithm. In each environment, the corresponding robot needs to run as fast as possible in one direction. Halfcheetah-disabled blocks different joints of a cheetah robot. Halfcheetah-pier changes the cheetah’s limb flexibility while running on moving ground. Ant-disabled resizes different legs of an ant robot. In Ant-Gravity, the environment’s gravitational setting is adjusted.

TABLE I
Meta-testing in different environments

Environment	j tasks	steps	RMPC	FMPC	ANC	MAC
Ant-gravity	8	20000	23289	23957	22870	48465
Ant-disabled	1	2500	2234	1856	2454	4863
Hc-disabled	2	5000	4034	9439	-	35927
Hc-pier	2	5000	2956	5260	-	20410

without any meta-task (i.e., Ant without disability and in standard gravitational setting; HalfCheetah without disability). Additionally, MAC reuses the trained ENN as explained in algorithm 2.

To compare the performance of each algorithm in each environment, we conducted a hyperparameter search for each combination of algorithm and environment for meta-training and meta-testing. We sampled a new task after every 500 steps and tested each task five times with different environment seeds. Our experiments show that MAC outperforms all baselines in all environments. The results of our experiments are summarized in Table I.

VII. Conclusion

In this paper, we presented MAC, a robot control algorithm that employs meta-reinforcement learning to apply a preferred robot behavior from one task to many similar tasks. At its core is the combination of a meta-trained embedding neural network and a RL policy of a reference task. While adapting the neural network online, it predicts which actions need to be taken in unseen tasks to mimic the behavior of the reference task obtained by the policy. Our experiments demonstrated that this mechanism works across various tasks in different environments and outperforms meta-testing with model predictive control in different model-based meta-reinforcement learning setups.

References

- [1] P. Sterling and S. Laughlin, *Principles of neural design*. MIT press, 2015.
- [2] J. Schmidhuber, “Evolutionary principles in self-referential learning. on learning now to learn: The meta-meta-meta...-hook,” Diploma Thesis, Technische Universitat Munchen, Germany, 14 May 1987.
- [3] C. B. Finn, “Meta Learning Dissertation,” pp. 1–3, 6–8, 2018.
- [4] C. G. Atkeson and J. C. Santamaria, “Comparison of direct and model-based reinforcement learning,” *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 4, pp. 3557–3564, 1997.
- [5] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou, “Information Theoretic MPC for Model-Based Reinforcement Learning.”
- [6] M. P. Deisenroth and C. E. Rasmussen, “PILCO: A model-based and data-efficient approach to policy search,” *Proceedings of the 28th International Conference on Machine Learning, ICML 2011*, pp. 465–472, 2011.
- [7] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, “Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning,” *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 7579–7586, aug 2017.
- [8] K. Chatzilygeroudis, V. Vassiliades, F. Stulp, S. Calinon, and J.-B. Mouret, “A survey on policy search algorithms for learning robot controllers in a handful of trials,” *IEEE Transactions on Robotics*, vol. 36, no. 2, pp. 328–347, jul 2018.
- [9] A. Nagabandi, I. Clavera, S. Liu, R. S. Fearing, P. Abbeel, S. Levine, and C. Finn, “Learning to adapt in dynamic, real-world environments through meta-reinforcement learning,” *arXiv*, pp. 1–17, 2018.
- [10] S. Sæmundsson, K. Hofmann, and M. P. Deisenroth, “Meta Reinforcement Learning with Latent Variable Gaussian Processes,” *34th Conference on Uncertainty in Artificial Intelligence 2018, UAI 2018*, vol. 2, pp. 642–652, mar 2018.
- [11] R. Kaushik, T. Anne, and J.-B. Mouret, “Fast Online Adaptation in Robotics through Meta-Learning Embeddings of Simulated Priors,” *IEEE International Conference on Intelligent Robots and Systems*, pp. 5269–5276, mar 2020.
- [12] M. Zhang, S. Vikram, L. Smith, P. Abbeel, M. J. Johnson, and S. Levine, “Solar: Deep structured representations for model-based reinforcement learning,” 2018.
- [13] A. Nagabandi, K. Konolige, S. Levine, and V. Kumar, “Deep dynamics models for learning dexterous manipulation,” pp. 1101–1112, 2020.
- [14] Y. Yang, K. Caluwaerts, A. Iscen, T. Zhang, J. Tan, and V. Sindhwani, “Data efficient reinforcement learning for legged robots,” pp. 1–10, 2020.
- [15] Y. Lee and S. Choi, “Gradient-based meta-learning with learned layerwise metric and subspace,” *35th International Conference on Machine Learning, ICML 2018*, vol. 7, pp. 4574–4586, 2018.
- [16] C. Finn, P. Abbeel, and S. Levine, “Model-agnostic meta-learning for fast adaptation of deep networks,” *34th International Conference on Machine Learning, ICML 2017*, vol. 3, pp. 1856–1868, 2017.
- [17] A. Nichol, J. Achiam, and J. Schulman, “On First-Order Meta-Learning Algorithms,” *Tech. Rep.*, 2018.
- [18] A. Rajeswaran, C. Finn, S. Kakade, and S. Levine, “Meta-learning with implicit gradients,” *arXiv*, 2019.
- [19] E. Todorov, T. Erez, and Y. Tassa, “Mujoco: A physics engine for model-based control,” in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 5026–5033.
- [20] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor,” *35th International Conference on Machine Learning, ICML 2018*, vol. 5, pp. 2976–2989, jan 2018.