Data-Driven Hierarchical Predictive Learning in Unknown Environments

Charlott Vallon and Francesco Borrelli

***Abstract*— We propose a hierarchical learning architecture for predictive control in unknown environments. We consider a constrained nonlinear dynamical system and assume the availability of state-input trajectories solving control tasks in different environments. A parameterized environment model generates state constraints specific to each task, which are satisfied by the stored trajectories. Our goal is to find a feasible trajectory for a new task in an unknown environment. From stored data, we learn strategies in the form of target sets in a reduced-order state space. These strategies are applied to the new task in real-time using a local forecast of the new environment, and the resulting output is used as a terminal region by a low-level receding horizon controller. We show how to *i)* design the target sets from past data and then *ii)* incorporate them into a model predictive control scheme with shifting horizon that ensures safety of the closed-loop system when performing the new task. We prove the feasibility of the resulting control policy, and verify the proposed method in a robotic path planning application.**

I. Introduction

Classical Iterative Learning Controllers (ILCs) aim to improve a system's closed-loop reference tracking performance at each iteration of a repeated task [1]–[3]. Recent work has also explored reference-free ILC for applications whose goals are better defined through a performance metric, such as racing autonomously around a track or harvesting wind energy [4]–[6].

In general, the learned ILC policy is not cost-effective, or even feasible, if the task environment changes [7], [8]. In the Artificial Intelligence and Reinforcement Learning communities, the ability to generate a control policy which performs well under different environment conditions is a common challenge, often referred to as generalization or transfer learning [9], [10]. These data-driven approaches typically focus on minimizing the performance loss between solving tasks in the original and new environment, rather than guaranteeing feasibility of the policy in a new environment.

Methods that do guarantee feasibility are generally model-based [11]–[15]. Approaches have been proposed for autonomous vehicles [16] and robotic manipulation applications [8], [17], [18]. These strategies often require maintaining a trajectory library, and adapting the stored trajectories online to the new constraints of the changed tasks, which can be both time-consuming and computationally expensive.

*This research was sustained in part by fellowship support from the National Physical Science Consortium, the National Institute of Standards and Technology, and by the grant NSF-1931853.

[1]Charlott Vallon, and Francesco Borrelli are with the Department of Mechanical Engineering, University of California, Berkeley, Berkeley, CA 94701, USA {charlottvallon, fborrelli} @ berkeley.edu

This article proposes a data-driven method for tackling a simple abstraction of the "changing environment" control problem. We consider availability of state-input trajectories which solve a set of control tasks $\{\mathcal{T}^1,\ldots,\mathcal{T}^n\}$. A successful execution of the i^{th} control task $\mathcal{T}^i$ is defined as a trajectory of states and inputs evolving according to a nonlinear difference equation, satisfying system state and input constraints, and satisfying constraints imposed by the task environment. In each of the n control tasks the system model, system constraints and objective function are identical. However, a parameterized environment descriptor function $\boldsymbol{\theta}$ generates a set of state constraints specific to each task $\mathcal{T}^i$, where $\boldsymbol{\theta} = \boldsymbol{\theta}^i$.

Our goal is to use stored executions of n previous control tasks in order to complete a successful execution of a new task $\mathcal{T}^{n+1}$ in a new environment described by $\boldsymbol{\theta}^{n+1}$. This paper presents a hierarchical predictive learning architecture to solve such a problem. Specifically, we use stored data to design strategy sets in reduced-dimension state space. These strategy sets represent high-level strategies learned from previous control tasks, and are used as waypoint regions in a receding horizon control. In this paper, we:

1) propose interpreting "strategies" as sets in reduced-dimension state space, referred to as "strategy sets",
2) show how to design strategy sets from past data,
3) demonstrate how to incorporate the strategy sets into a receding horizon control,
4) prove the proposed method will lead to a successful execution of the new control task, and
5) apply the controller to a robotic path planning example.

II. Problem Formulation

We consider a discrete-time system with dynamical model

$$x_{k+1} = f(x_k, u_k), \tag{1}$$

subject to system state and input constraints

$$x_k \in \mathcal{X},\ u_k \in \mathcal{U}. \tag{2}$$

The vectors $x_k \in \mathbb{R}^{n_x}$ and $u_k \in \mathbb{R}^{n_u}$ collect the states and inputs at time k.

A. Task Environments

The system (1) solves a series of n control tasks $\{\mathcal{T}^1, \ldots, \mathcal{T}^n\}$ in different task environments parameterized by $\{\boldsymbol{\theta}^1, \ldots, \boldsymbol{\theta}^n\}$. For each task $\mathcal{T}^i$, the environment descriptor function $\boldsymbol{\theta}^i$ maps the state x_k at time k to a description of the local task environment $\boldsymbol{\theta}^i(x_k, k)$. Examples of $\boldsymbol{\theta}^i(x_k, k)$ include camera images, the coefficients of a polynomial

describing a race track lane boundaries, or simple waypoints for tracking.

In each control task the system model (1) and constraints (2) are identical. However, the environment descriptor function generates additional local environment-specific state constraints, denoted $E(\boldsymbol{\theta}^i(x_k, k))$. In task $\mathcal{T}^i$, we write the combined system and environment constraints as

$$x_k \in \mathcal{X}(\boldsymbol{\theta}^i, x_k, k), \tag{3}$$

where

$$\mathcal{X}(\boldsymbol{\theta}^i, x_k, k) = E(\boldsymbol{\theta}^i(x_k, k)) \cap \mathcal{X}.$$

For notational simplicity, we refer to the combined system and environment constraints (3) as $\mathcal{X}(\boldsymbol{\theta}^i)$.

B. Task Executions

A feasible execution of the task $\mathcal{T}^i$ in environment $E(\boldsymbol{\theta}^i)$ is defined as a pair of state and input trajectories

$$\begin{aligned} \mathrm{Ex}(\mathcal{T}^i, \boldsymbol{\theta}^i) &= [\mathbf{x}^i, \mathbf{u}^i] \qquad (4)\\ \mathbf{x}^i &= [x_0^i, x_1^i, ..., x_{D^i}^i],\ x_k^i \in \mathcal{X}(\boldsymbol{\theta}^i) \quad \forall k \in [0, D^i],\\ & \qquad x_{D^i} \in \mathcal{P}(\boldsymbol{\theta}^i),\\ \mathbf{u}^i &= [u_0^i, u_1^i, ..., u_{D^i}^i],\ u_k \in \mathcal{U} \qquad \forall k \in [0, D^i], \end{aligned}$$

where $\mathbf{u}^i$ collects the inputs applied to the system (1) and $\mathbf{x}^i$ is the resulting state evolution. D^i is the duration of the execution of task $\mathcal{T}^i$. The final state of a feasible task execution, x_{D^i}, is in the task target set $\mathcal{P}(\boldsymbol{\theta_i}) \subset \mathcal{X}(\boldsymbol{\theta}^i)$.

Problem Definition: Given a dynamical model (1) with state and input constraints (2), (3), and a collection of feasible executions (4) that solve a series of n control tasks, $\{\mathrm{Ex}(\mathcal{T}^1, \boldsymbol{\theta}^1), ..., \mathrm{Ex}(\mathcal{T}^n, \boldsymbol{\theta}^n)\}$, our aim is to find an execution for a new task in an unseen environment: $\mathrm{Ex}(\mathcal{T}^{n+1}, \boldsymbol{\theta}^{n+1})$.

Remark 1: For notational simplicity, we write that the collected data set contains one execution from each task. However, in practice one may collect multiple executions of each task $\mathcal{T}^i$. In this case, the executions can simply be stacked, and the procedure proposed in this paper can proceed as described.

C. Environment Forecasts

The control architecture proposed in this paper relies on forecasts of the task environment. At time k, we use forecasts of the system state across a horizon N, $\hat{x}^i_{k:k+N}$, to predict the corresponding environments $[\boldsymbol{\theta}^i(\hat{x}_k, k), \ldots, \boldsymbol{\theta}^i(\hat{x}_{k+N}, k + N)]$. For notational simplicity, we write this environment forecast as $\theta^i_{k:k+N}$.

III. Hierarchical Predictive Learning Control

We propose a data-driven controller that uses stored executions from previous tasks, $\{\mathrm{Ex}(\mathcal{T}^1, \boldsymbol{\theta}^1), \ldots, \mathrm{Ex}(\mathcal{T}^n, \boldsymbol{\theta}^n)\}$, to find an execution for a new task in a new environment, $\mathrm{Ex}(\mathcal{T}^{n+1}, \boldsymbol{\theta}^{n+1})$. Instead of simply adapting the stored executions from previous tasks to the constraints of the new task, our aim is to learn a generalizable and interpretable *strategy* from past tasks, and apply it to the new task.

A. Approach

Our approach is inspired by how navigation tasks are typically explained to humans, who can easily generalize learning to new environments. For example, if a human has learned to race a vehicle by driving around a single track, they can easily adapt their learned strategy when racing a new track. Based on real-life intuition, we propose three principles of strategy:

1) Strategies are a function of a local environment forecast.
2) Strategies work in a reduced-order state space.
3) Strategies provide target regions in the (reduced-order) state space for which to aim.

The control architecture proposed in this paper formalizes the above principles of strategy, and shows how to incorporate this strategy framework into a hierarchical learning control. In particular, we focus on two aspects. First, we show how to learn generalizable strategies from stored executions of previous control tasks. Second, we show how to integrate the learned strategies in an MPC framework so as to guarantee feasibility when solving a new control task in real-time.

Vehicle Racing Example: Consider, for example, learning how to race a vehicle around a track. A snippet of common racing strategies[1] taught to new racers is depicted in Fig. 1. The environmental constraints imposed on the vehicle are

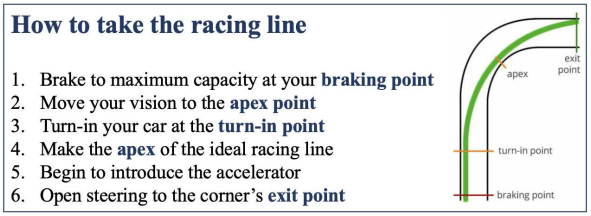

Fig. 1: Sample of strategies taught at various online racing schools.

parameterized by the race track curvature. This environmental descriptor gives rise to physical areas in state space with respect to which racing rules are then explained, e.g. "cut the curve at the apex and aim for the outside of the straightaway." The strategies consist of sections of the track towards which to aim the vehicle as well as acceleration profiles to apply along the way. The strategies are explained using only a subset of the state space: the distance from the centerline. Given guidelines on this subset, the driver is free to adjust other states and inputs such as vehicle velocity and steering as necessary.

B. Implementation

Hierarchical Predictive Learning (HPL) is a data-driven control scheme based on high-level strategies, learned from previous executions of different tasks according to our principles of strategy. The HPL controller modifies its behavior whenever new strategies become applicable, and operates

[1]As taught at online racing schools such as Driver 61: `https://driver61.com/uni/racing-line/`

in coordination with a safety controller to ensure constraint satisfaction at all future time steps.

At each time k, an N-step local environment forecast $\theta^i_{k:k+N}$ is used to determine if a new high-level control strategy is available. A strategy consists of state and input sets in reduced dimensions, $\tilde{\mathcal{X}}_{k+T}$ and $\tilde{\mathcal{U}}_{k:k+T}$, that provide a set towards which to steer the system in the next T timesteps, as well as input guidelines for getting there. The strategy sets are used to construct a target set in the full state space, $\mathcal{X}_{k+T}$. Lastly, a receding horizon controller calculates a low-level input to get the system to the target set. In the following sections, we explain each of the three hierarchy levels in detail and prove the feasibility of the resulting control law.

We note the difference between the environment forecast horizon N and the strategy prediction horizon T. The two need not be the same, and it is reasonable to select $N > T$.

IV. Learning Strategies From Data

This section addresses the first aim of our paper: learning generalizable strategies from previously solved tasks. There are several ways of learning strategies, including model-based methods that use an explicit model for how variations in task environments affect the optimal control input [11]. In this work we instead opt for a data-driven approach, using stored executions (4) that solved related tasks.

We propose using Gaussian Processes (GPs) to learn a mapping from a current system state and environment forecast, $(x_k, \theta^i_{k:k+N})$, to hyperrectangular state and input constraint sets in reduced dimension, $(\tilde{\mathcal{X}}_{k+T}, \tilde{\mathcal{U}}_{k:k+T})$, known as strategy sets. The GPs are trained *offline*, and the strategy sets are built *online* during the execution of a new control task using the estimated mean and standard deviation provided by the GPs. A review of GPs in control is provided in [19], and implementation details in the Appendix.

A. Training the GP (Offline)

1) GPs in MPC: GPs have been used in recent predictive control literature as data-driven estimates of unknown nonlinear dynamics [12], [20], [21]. Specifically, GPs are used to approximate vector-valued functions with real (scalar) outputs. Given training data (input vectors and output values), GPs learn a similarity measure known as "kernel" between inputs to estimate the true underlying function. The kernel represents the learned covariance between two function evaluations.

2) Why use GPs?: Once a kernel has been learned, the GP can be queried at a new input vector. Given an input, the GP returns a one-dimensional Gaussian distribution over output estimates; thus GPs provide a best guess for the output value corresponding to an input *and* a measure of uncertainty about the estimate based on the distribution's standard deviation. This allows us to gauge how confident the GP is in its prediction at a particular input.

Note: The robust MPC community often prefers stochastic models with bounded support [22] to GPs, in order to have strict safety guarantees. Our approach can be extended to these types of models as well.

3) GPs for Hierarchical Predictive Learning: We consider strategies to be maps from a state and environment forecast to reduced-dimension strategy sets. The choice of appropriate strategy states and inputs depends on the tasks. We define the strategy state at each time k as

$$\tilde{x}_k = g(x_k) \in \mathbb{R}^{n_{\tilde{x}}}, \tag{5}$$

where $g(\cdot)$ maps the full-dimensional state x_k into the corresponding lower-dimensional strategy state. Similarly, the strategy inputs are

$$\tilde{u}_k = r(u_k) \in \mathbb{R}^{n_{\tilde{u}}}, \tag{6}$$

where $r(\cdot)$ maps the full-dimensional input at time k into lower-dimensional strategy inputs. We denote the strategy state and strategy input spaces as

$$\tilde{\mathcal{X}} = \{\tilde{x} \mid \tilde{x} = g(x),\ x \in \mathcal{X}\}, \tag{7}$$

$$\tilde{\mathcal{U}} = \{\tilde{u} \mid \tilde{u} = r(u),\ u \in \mathcal{U}\}. \tag{8}$$

In HPL, we train GPs to predict the values of the strategy states (5) and inputs (6) at T timesteps into the future, based on the current state and environment forecast. Each GP learns to approximate the mapping between a state and environment forecast and a particular strategy state or input:

$$(\mu, \sigma^2) = \mathrm{GP}(\tilde{x}_k, \theta^i_{k:k+N}), \tag{9}$$

where μ and σ^2 represent statistics of the Gaussian distribution over strategy state estimates which are used to build strategy sets in Sec.IV-B. GPs best approximate functions with scalar outputs, so we train one GP for each strategy state and input (a total of $n_{\tilde{x}} + n_{\tilde{u}}$ number of GPs).

4) Building Training Data: We use stored executions (4) from previous control tasks to create GP training data. Each GP uses the same training input data. The training output data for each GP contains the strategy state or input the GP is learning to predict. After solving n control tasks at least once (see Remark 1), the training data consists of:

$$\begin{aligned}\mathcal{D} = \{&\mathbf{z} = [z^1_0, z^1_1, \ldots, z^1_{D^1-T}, z^2_0, \ldots, ..., z^n_{D^n-T}] \\ &\mathbf{y} = [y^1_0, y^1_1, \ldots, y^n_{D^n-T}]\}.\end{aligned} \tag{10}$$

Each input vector z^i_j corresponds to the output y^i_j, where

$$\begin{aligned}z^i_j = [x^i_j, \theta^i_j, \theta^i_{j+1}, ..., \theta^i_{j+N}], \\ i \in [1, n],\ j \in [0, D^i - T],\end{aligned} \tag{11}$$

and θ^i_j denotes the local environment at time k of the ith control task. For example, this can correspond to the instantaneous track curvature or the camera image recorded at that time step. The output entry y^i_j contains the value of the strategy state or input of interest at T time steps in the future:

$$\begin{aligned}y^i_j = \tilde{x}^i_{j+T} \text{ or } \tilde{u}^i_{j+T}, \\ i \in [1, n],\ j \in [0, D^i - T].\end{aligned}$$

Again, we emphasize the difference between the environment forecast horizon N and the strategy prediction horizon T.

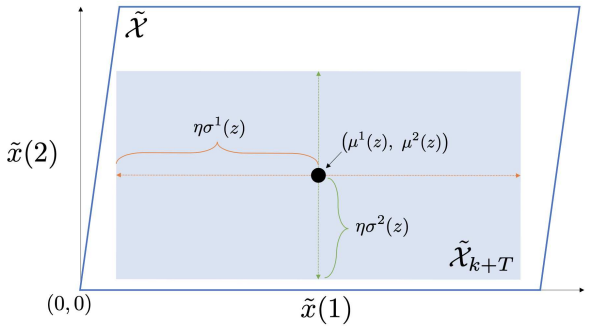

Fig. 2: Each dimension $\tilde{x}(i)$ of the strategy set $\tilde{\mathcal{X}}_{k+T}$ is bounded using a GP (12).

5) Training: We use the Matlab Statistics and Machine Learning toolbox[2] to learn kernel hyperparameters that best match the training data (10). Specifically, we train a squared-exponential kernel with separate length scales for each predictor (see Appendix). New hyperparameters can be learned whenever more executions become available.

B. Constructing the Strategy Sets (Online)

Once the GPs are trained, they can be used to solve a new task. At time k of a new task $\mathcal{T}^{n+1}$, we evaluate the GPs at the new query vector z_k^{n+1}, formed as in (11), to construct hyperrectangular strategy sets in reduced-dimension space.

Each GP returns a one-dimensional Gaussian distribution over output scalars, parameterized by a mean μ and variance σ^2. Once means and variances have been determined for each strategy state and input from the learned kernel hyperparameters (see (26) in Appendix), we form one-dimensional bounds on each ith strategy state and jth strategy input as

$$\tilde{\mathcal{X}}_{k+T}(i) = [\mu^i(z_k^{n+1}) \pm \eta\sigma^i(z_k^{n+1})],\ \forall i \in [1, n_{\tilde{x}}] \quad (12)$$
$$\tilde{\mathcal{U}}_{k:k+T}(j) = [\mu^j(z_k^{n+1}) \pm \eta\sigma^j(z_k^{n+1})], \quad \forall j \in [n_{\tilde{x}}+1, n_{\tilde{x}}+n_{\tilde{u}}].$$

In (12), $\mu^i(\cdot)$ and $\sigma^i(\cdot)$ are the means and standard deviations computed by the ith GP. The parameter $\eta > 0$ determines the size of the range. When these one-dimensional bounds are combined for all strategy states and strategy inputs, respectively, we form hyperrectangular strategy sets in strategy space, with each dimension constrained according to (12). An example is shown in Fig. 2. The hyperrectangular strategy sets are denoted $\tilde{\mathcal{X}}_{k+T}$ and $\tilde{\mathcal{U}}_{k:k+T}$, and indicate where (in strategy space) the system should be in T timesteps and what inputs to apply to get there.

V. Safely Applying Learned Strategies

We now address the second aim of our paper: using the strategy sets in a low-level controller while maintaining safety guarantees. Our approach consists of *i)* lifting the reduced-dimension strategy sets back into full-dimension, and *ii)* integrating the full-dimension set with a safety controller. The result is a target set that can be used in a low-level MPC controller.

Assumption 1: There exists a safety control policy that can prevent the system (1) from violating both system- and task-specific environment constraints (3). In particular, there exists a safe set

$$\mathcal{X}_E \subseteq \mathcal{X}(\boldsymbol{\theta}), \quad (13)$$

and a corresponding safety control law

$$u = \pi_{\mathrm{e}}(x, \boldsymbol{\theta}), \quad (14)$$

such that $\forall x \in \mathcal{X}_E,\ f(x, \pi_{\mathrm{e}}(x, \boldsymbol{\theta})) \in \mathcal{X}_E$.

Remark 2: Given a safety control law (14), the safe set (13) may be found using a variety of data-driven methods, including backwards reachability from a known subset of $\mathcal{X}_E$ (such as physical standstill), or sample-based forward reachability from a gridded state space $\mathcal{X}$.

A. Converting to full-dimensional hyperrectangles

At each time k of solving a task $\mathcal{T}^{n+1}$, new reduced-dimension strategy sets (12) are constructed. These strategy sets are converted to a target set in full-dimensional state space, to be used as a terminal constraint in a low-level MPC controller. Critically, the target set must belong to the safe set (13). This ensures that once the system has reached the target set, there will always exist at least one feasible input (the safety control (14)) that allows the system to satisfy all state constraints. Given strategy sets (12) found at time k, we define a corresponding lifted strategy set as:

$$\mathcal{X}_{k+T} = \{x \in \mathcal{X}_E \mid g(x) \in \tilde{\mathcal{X}}_{k+T}\}, \quad (15)$$

where $g(x)$ (5) is the projection of the full-dimensional state x onto the set of chosen strategy states. $\mathcal{X}_{k+T}$ is a full-dimensional set in which the strategy states (5) are constrained to lie in the strategy sets (12), and the remaining states are constrained such that for any state $x_k \in \mathcal{X}_{k+T}$, the safety control (14) can be applied if necessary to ensure constraint satisfaction in future time steps.

B. Incorporating the Uncertainty Measure

A benefit of using GPs is that the standard deviation around an estimate may be used to evaluate how confident the GP is in its prediction at a particular input. At time k, consider the uncertainty measure $\mathcal{C}_k$ which is a vector containing the standard deviations of the evaluated GPs:

$$\mathcal{C}_k = [\sigma^1(z_k^{n+1}), \ldots, \sigma^{n_{sx}+n_{su}}(z_k^{n+1})]. \quad (16)$$

If the GPs return a strategy set with standard deviations (16) larger than a chosen threshold d_{thresh}, we may opt not to use this strategy. We expect $\mathcal{C}_k > d_{\mathrm{thresh}}$ if:

1) the system did not encounter a similar environment forecast in a previous control task, or
2) in previous control tasks this environment forecast did not lead to a single coherent strategy, resulting in a wide distribution of potential future strategy states.

With high uncertainty measures, the strategy sets are not likely to contain valuable control information for the system.

[2] https://www.mathworks.com/help/stats/index.html

In this case, HPL sets the target set to be *empty*: $\mathcal{X}_{k+T} = [\,]$. As explained in Sec. VI, this results in a horizon shift for the low-level MPC, and the system (1) re-uses the target set from the previous time step.

VI. Low-level Controller Design

The low-level MPC controller is responsible for calculating the input to be applied to the system at each time k. This input is calculated based on the sequence of target sets (15) found during the last T timesteps.

A. Target Set List

At each time k of solving task $\mathcal{T}^{n+1}$, a new target set (15) is constructed by lifting the strategy sets (12). However, if the standard deviations (16) are too high, or there is no feasible input sequence to reach the lifted strategy set $\mathcal{X}_{k+T}$, the target set for time k will be empty: $\mathcal{X}_{k+T} = [\,]$.

The "target set list" keeps track of which target sets (empty or not) were constructed during the most recent T timesteps:

$$\text{SetList}_k = [\mathcal{X}_{k+1}, \mathcal{X}_{k+2}, \ldots, \mathcal{X}_{k+T}]. \tag{17}$$

At each new time step, the first set gets removed and the target set found at the current time step gets appended to the end. In this way, the target set list (17) always maintains exactly T sets, though some (including the last one) may be empty. This list is used to guide the objective function and constraints of the MPC controller.

B. Shifting Horizon MPC Formulation

We formulate an MPC controller to calculate our input at each time step:

$$\begin{aligned}\mathbf{u}^\star(x_k) = \underset{u_{0|k},\ldots,u_{N_k-1|k}}{\arg\min} & \sum_{j\in\mathcal{S}_k}^{N_k-1} \text{dist}\big(x_{j|k}, \mathcal{X}_{k+j}\big) \\ \text{s.t.}\quad & x_{j+1|k} = f(x_{j|k}, u_{j|k}) \\ & x_{j|k} \in \mathcal{X}(\boldsymbol{\theta}^{n+1}) \quad \forall j \in \{0, N_M - 1\} \\ & u_{j|k} \in \mathcal{U} \qquad\quad \forall j \in \{0, N_M - 1\} \\ & x_{N_k|k} \in \mathcal{X}_{k+N_k} \\ & x_{0|k} = x_k,\end{aligned} \tag{18}$$

where $\mathcal{S}_k$ is the set of indices with non-empty target sets,

$$\mathcal{S}_k = \{s \mid \text{notEmpty}(\mathcal{X}_{k+s-1})\}.$$

The MPC objective function (18) penalizes the Euclidean distance from each predicted state to the target set corresponding to that prediction time. If a smoother cost is desired, the objective could be augmented to take the input effort into account.

The MPC uses a time-varying shifting horizon $0 < N_k \leq T$ that corresponds to the largest time step into the future for which a non-empty target set results in feasibility of (18):

$$N_k = \max s \;:\; \{s \in \mathcal{S}_k,\ (18) \text{ is feasible with } N_k = s\}. \tag{19}$$

This ensures that the MPC controller (18) has a non-empty terminal constraint and the optimization problem is feasible. To avoid unnecessary repeated computations, all target sets in the target set list (17) which lead to infeasibility of (18) when used as the terminal constraint are set as *empty* in (17). At time step k, we apply the first optimal input to the system:

$$u_k = u^\star_{0|k}. \tag{20}$$

Note: Here, the target sets (15) are found at the same frequency as the controller (18)-(20) is updated, but our framework can easily be adapted to use asynchronous loops.

C. Safety Control

If no target sets in (17) can be used as a terminal constraint in (18) such that (18) is feasible, all sets in the target set list (17) will be empty, and the MPC horizon is $N_k = 0$. When this occurs, the system enters into *Safety Control mode*. The safety controller (14) turns on, and controls the system in a safe manner until a time when a satisfactory target set is found (at which point the MPC horizon resets to $N_k = T$). The HPL approach ensures that the system (1) will always be in the safe set (13) when $N_k = 0$ and the safety controller (14) needs to turn on. We prove this in Sec. VIII.

VII. The HPL Algorithm

Alg. 1 summarizes the HPL control policy. Importantly, HPL only requires a local forecast of the new task environment, rather than the entire environment description. Gaussian Processes, trained offline on trajectories from past control tasks, are used to construct reduced-dimension strategy sets. Target sets are formed in conjunction with a safety controller, and are used as terminal sets in a shifting-horizon MPC controller.

VIII. Feasibility Proof

We prove that Alg. 1 outputs a feasible execution for a new control task $\mathcal{T}^{n+1}$.

Theorem 1: Let Assumption 1 hold. Consider the availability of feasible executions (4) by a constrained system (1)-(2) of a series of control tasks $\{\mathcal{T}^1, \ldots, \mathcal{T}^n\}$ in different environments $\{E(\boldsymbol{\theta}^1), \ldots, E(\boldsymbol{\theta}^n)\}$. Consider a new control task $\mathcal{T}^{n+1}$ in a new environment $E(\boldsymbol{\theta}^{n+1})$. If $x_0^{n+1} \in \mathcal{X}_E$, then the output of Alg. 1 is a feasible execution of $\mathcal{T}^{n+1}$: $\text{Ex}(\mathcal{T}^{n+1}, \boldsymbol{\theta}^{n+1})$.
For ease of reading, we drop the task index $(\cdot)^{n+1}$.

Proof: We use induction to prove that for all $k \geq 0$, the iteration loop (Lines 10-23) in Alg. 1 finds an input u_k such that the resulting closed-loop trajectory satisfies system and environment constraints.

At time $k = 0$ of the new task $\mathcal{T}^{n+1}$, the target set list can contain at most one non-empty set, $\mathcal{X}_T$ (15). If $\mathcal{X}_T$ is non-empty, and the resulting (18) is feasible, then there exists an input sequence $[u_{0|0}, \ldots, u_{T-1|0}]$ calculated by (18) satisfying all state and input constraints (3), with $x_{T|0} \in \mathcal{X}_T$. However, if $\mathcal{X}_T$ is empty or (18) is infeasible, we instead apply the safety control law $u_0 = \pi_e(x_0)$. By assumption, $x_0 \in \mathcal{X}_E$, so this input is feasible. Thus we have shown that the iteration loop in Alg. 1 is feasible for $k = 0$.

Algorithm 1 HPL Control Policy

1: **parameters:** $d_{\text{thresh}}, T, N, \mathcal{X}_E, \pi_E$
2: **input:** f, $\mathcal{X}$, $\mathcal{U}$, $\{\text{Ex}(\mathcal{T}^1, \boldsymbol{\theta}^1), ..., \text{Ex}(\mathcal{T}^n, \boldsymbol{\theta}^n)\}$, $\boldsymbol{\theta}^{n+1}$
3: **output:** $\text{Ex}(\mathcal{T}^{n+1}, \boldsymbol{\theta}^{n+1})$
4:
5: **offline:**
6: **train** GPs using stored executions as in Sec.IV-A
7:
8: **online:**
9: **initialize** $k = 0$, $N_k = T$, SetList = [ ]
10: **for** each time step k **do**
11: **collect** $(x_k, \theta^{n+1}_{k:k+N})$
12: **find** $[\tilde{\mathcal{X}}_{k+T}, \tilde{\mathcal{U}}_{k:k+T}, \mathcal{C}_k]$ using (9) - (12)
13: **construct** $\mathcal{X}_{k+T}$ using (15)
14: **if** $\mathcal{C}_k < d_{\text{thresh}}$ **then**
15: $\mathcal{X}_{k+T} = [\,]$
16: **append** $\mathcal{X}_{k+T}$ to SetList (17) and shift sets
17: **if** all sets in SetList (17) are **empty then**
18: $u_k = \pi_e(x_k)$
19: **else**
20: **calculate** N_k using (19)
21: **solve** MPC (18) with horizon N_k
22: $u_k = u^\star_{0|k}$ (20)
23: **end**

Next, we show that the iteration loop of Alg. 1 is recursively feasible. Assume that at time $k > 0$, the low-level policy (18-20) is feasible with horizon N_k, and let $\mathbf{x}^\star_{k:k+N_k|k}$ and $\mathbf{u}^\star_{k:k+N_k-1|k}$ be the optimal state trajectory and input sequence according to (18), such that

$$u_k = u^\star_{k|k} \tag{21}$$

$$x^\star_{k+N_k|k} \in \mathcal{X}_{k+N_k}. \tag{22}$$

If at time $k+1$ a non-empty target set $\mathcal{X}_{k+T+1}$ is constructed according to (15) such that (18) is feasible, then there exists a feasible input sequence $[u_{k+1|k+1}, \ldots, u_{k+T|k+1}]$ satisfying all state and input constraints such that $x_{k+T+1} \in \mathcal{X}_{k+T+1}$.

If at time $k+1$ the target set is *empty*, or (18) is infeasible, we must consider two cases separately:

Case 1: The MPC horizon at time step k is $N_k > 1$. In the absence of model uncertainty, when the closed-loop input u_k (21) is applied, the system (1) evolves such that

$$x_{k+1} = x^\star_{k+1|k}. \tag{23}$$

According to Alg. 1, when the empty target set $\mathcal{X}_{k+1+T}$ is added to the target set list (17), the MPC horizon is shortened and the most recent non-empty target set is used again. Since $N_k > 1$, we are guaranteed at least one non-empty target set in (17) that may used as a feasible terminal constraint in the low-level controller (18). At time step $k+1$, the shifted input sequence $\mathbf{u}^\star_{k+1:k+N_M-1|k}$ will be optimal for this shifted horizon optimal control problem (with a corresponding state trajectory $\mathbf{x}^\star_{k+1:k+N_M|k}$). At time step $k+1$, Alg. 1 applies the second input calculated at the previous time step: $u_{k+1} = u^\star_{k+1|k}$.

Case 2: $N_k = 1$. In this scenario, the target set list (17) at time step $k+1$ is empty, resulting in $N_{k+1} = 0$ (19). However, combining the fact that $N_k = 1$ with (22) and (23), we note that

$$x_{k+1} \in \mathcal{X}_{k+1} \subseteq \mathcal{X}_E,$$

by construction of the target set (15). This implies that at time step $k+1$, system (1) is necessarily in the safe set (13), and so application of the safety controller will result in a feasible input, $u_{k+1} = \pi_e(x_{k+1})$.

We have shown that *i)* the online iteration loop (Lines 10-23) in Alg. 1 finds a feasible input at time step $k = 0$, and *ii)* if the loop finds a feasible input at time step k, it must also find a feasible input at time $k+1$. We conclude by induction that the iteration loop in Alg. 1 finds a feasible input $u_k \forall k \in \mathbb{Z}_{0+}$ in the new task $\mathcal{T}^{n+1}$. This results in a feasible execution of $\mathcal{T}^{n+1}$. ■

IX. Robot Path Planning Example

We demonstrate the effectiveness of our proposed control architecture in a robotic path planning example.

A. System and Task description

Consider a UR5e[3] robotic arm tasked with maneuvering its end-effector through a tube with varying slope. The UR5e has high end-effector reference tracking accuracy, allowing us to use a simplified end-effector model in place of a discretized second-order model [23]. At each time step k, the state of the system is z_k,

$$z_k = [x_k,\ \dot{x}_k,\ y_k,\ \dot{y}_k],$$

where x_k and y_k are the coordinates of the end-effector, and $\dot{x}_k$ and $\dot{y}_k$ their respective velocities. The inputs to the system at time step k are

$$u_k = [\ddot{x}_k,\ \ddot{y}_k], \tag{24}$$

the accelerations of the end-effector in the x and y direction, respectively. We model the base-and-end-effector system as a quadruple integrator:

$$z_{k+1} = Az_k + Bu_k \tag{25}$$
$$A = \begin{bmatrix} 1 & dt & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & dt \\ 0 & 0 & 0 & 1 \end{bmatrix}, B = \begin{bmatrix} 0 & 0 \\ dt & 0 \\ 0 & 0 \\ 0 & dt \end{bmatrix},$$

where $dt = 0.01$ seconds is the sampling time. This model holds as long as we operate within the region of high end-effector reference tracking accuracy, characterized experimentally as the following state and input constraints:

$$\mathcal{X} = \begin{bmatrix} -3 \\ -3 \end{bmatrix} \leq \begin{bmatrix} \dot{x}_k \\ \dot{y}_k \end{bmatrix} \leq \begin{bmatrix} 3 \\ 3 \end{bmatrix}$$
$$\mathcal{U} = \sqrt{\ddot{x}_k + \ddot{y}_k} \leq 1,$$

[3] https://www.universal-robots.com/products/ur5-robot/

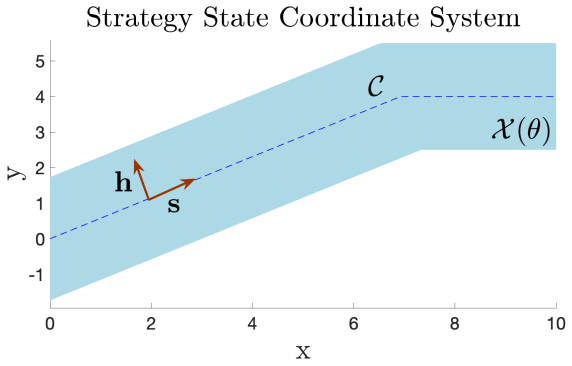

Fig. 3: The end-effector is constrained to stay in the light blue tube $\mathcal{X}(\boldsymbol{\theta})$, whose centerline is plotted in dashed blue. The strategy states s and h measure the cumulative distance along the centerline and the distance from the centerline.

where the states x_k and y_k are not constrained by the system, but by the particular task environment.

Each control task $\mathcal{T}^i$ requires the end-effector to be controlled through a different tube, described using the environment descriptor function $\boldsymbol{\theta}^i$, as quickly as possible. Here, the function $\boldsymbol{\theta}^i$ maps a state in a tube segment to the slope of the constant-width tube. Different control tasks $\{\mathcal{T}^1, \ldots, \mathcal{T}^n\}$ correspond to maneuvering through tubes of constant width but different piecewise-constant slopes.

We choose two strategy states for these tasks:

$$\tilde{x}_k = [s_k,\ h_k],$$

where s_k is the cumulative distance along the centerline of the tube from the current point (x_k, y_k) to the projection onto the centerline, and h_k is the distance from (x_k, y_k) to the centerline. The strategy states therefore measure the total distance traveled along the tube up to time step k, and the current signed distance from the center of the tube. The system inputs (24) are used as strategy inputs.

Our safety controller (14) is an MPC controller which tracks the centerline of the tube at a slow, constant velocity of 0.5 meters per second. The system (25) in closed-loop with this centerline-tracking controller is able to solve each of the considered tasks without breaking the state constraints imposed by the environment (i.e. without hitting the tube boundary). The safe set $\mathcal{X}_E$ (13) is determined offline using sampling-based forward reachability.

B. Hierarchical Predictive Learning Results

We test our proposed control architecture in simulation. We begin by collecting executions that solve a series of 20 control tasks $\{\mathcal{T}^1, ..., \mathcal{T}^{20}\}$, with each control task corresponding to a different tube shape. The executions are used to create training data for our GPs, using an environment forecast horizon of $N = 10$ and a control horizon of $T = 5$. GPs are then found to approximate the strategies learned from solving the 20 different control tasks.

Figure 4 shows the closed-loop trajectory and target set list at various time steps of solving a new task, $\mathcal{T}^{21}$, using

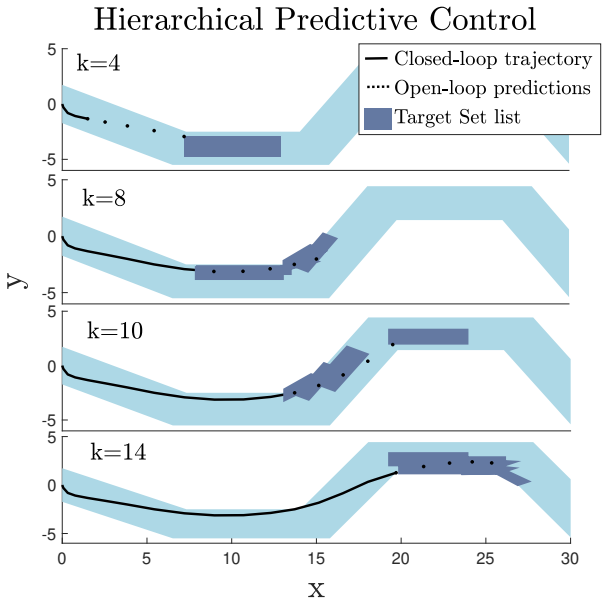

Fig. 4: At each time step, the target set list (17) provides different regions in the task space for the system to track.

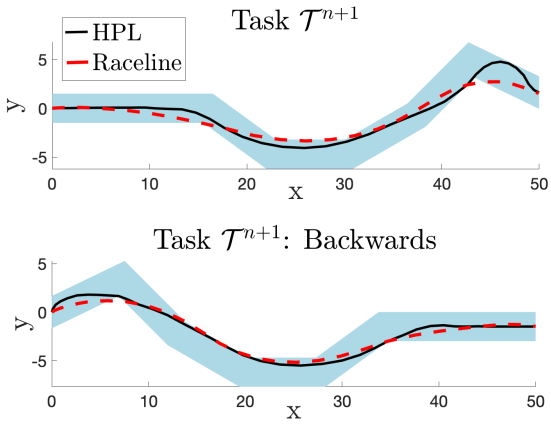

Fig. 5: The HPL execution is compared to the raceline (the fastest possible execution), as determined by an LMPC [7]. Respective execution times in [s] are 6.5 (LMPC), 8.8 (HPL), and 12.8 (Centerline-tracking π_e).

the HPL framework. At each time step, the final predicted state lies within the last set in the target set list; the other predicted states track any non-empty target sets as closely as possible. The formulation allows us to visualize what strategies have been learned, by plotting at each time step where the system thinks it should go. Indeed, we see that the system has learned to maneuver along the insides of curves, and even takes the direct route between two curves going in opposite directions.

Figure 5 shows the resulting executions for a new task solved forwards and backwards. We emphasize that HPL generalizes the strategies learned from training data to unseen tube segments. Specifically, for the tasks shown here the GPs were trained on executions solving tasks in the

'forward' direction, i.e. constructed left to right using tube segments as shown in the top images. When the tasks are solved backwards, the environment descriptors are piecewise mirror images of previously seen environment descriptors. For example, tube segments of certain slopes had only been traversed upwards in previous control tasks, never down. The HPL control architecture was able to handle this change very well. As in Fig. 4, the forward and backwards trajectories demonstrate good maneuvering strategies, including moving along the insides of curves and cutting consecutive corners.

X. Conclusion

A data-driven hierarchical predictive learning architecture for control in unknown environments is presented. The HPL algorithm uses stored executions that solve a variety of previous control tasks in order to learn a generalizable control strategy for new, unseen tasks. Based on a local description of the task environment, the learned control strategy proposes regions in the state space towards which to aim the system at each time step, and provides input constraints to guide the system evolution according to previous task solutions. We prove that the resulting policy is guaranteed to be feasible for the new tasks, and evaluate the effectiveness of the proposed architecture in a simulation of a robotic path planning task. Our results confirm that HPL architecture is able to learn applicable strategies for efficient and safe execution of unseen tasks.

XI. Appendix

We use Gaussian Processes trained on data (4) from previous tasks to construct strategy sets for solving a new task. Specifically, GPs use a similarity measure known as "kernel" between input vectors to learn a nonlinear approximation of the underlying input-output mapping. The kernel function represents the learned covariance between two function evaluations. In this paper we use the squared-exponential kernel for our GPs. Given two entries of $\mathbf{z}$ in (10), this kernel evaluates as

$$k(z_i^o, z_j^w) = \sigma_f^2 \exp -\frac{1}{2} \sum_{m=1}^{n_x+N+1} \frac{(z_i^o(m) - z_j^w(m))^2}{\sigma_m^2},$$

where $z_i^o(m)$ is the mth entry of the vector z_i^o. The hyperparameters σ_f, σ_1, ..., σ_{n_x+N+1} are learned from the training data (10) using maximum log-likelihood regression. We use the Matlab GP toolbox for this process.

Given a new query vector z_k^{n+1}, hyperrectangular strategy sets are formed using the mean μ and variance σ^2 of the resulting Gaussian distribution:

$$\begin{aligned} \mu(z_k^{n+1}) &= \mathbf{k}(z_k^{n+1})\bar{K}^{-1}\mathbf{y} \\ \sigma(z_k^{n+1})^2 &= k(z_k^{n+1}, z_k^{n+1}) - \mathbf{k}(z_k^{n+1})\bar{K}^{-1}\mathbf{k}^\top(z_k^{n+1}), \end{aligned} \tag{26}$$

where

$$\mathbf{k}(z_k^{n+1}) = [k(z_k^{n+1}, z_0^1), \ldots, k(z_k^{n+1}, z_{b_n}^n)],$$

and the matrix $\bar{K}$ is formed out of the covariances between training data samples such that

$$\bar{K}_{i,j} = k(\mathbf{z}_i, \mathbf{z}_j).$$

The strategy sets are then constructed according to (12).

References

[1] D. A. Bristow, M. Tharayil, and A. G. Alleyne, "A survey of iterative learning control," *IEEE Control Systems Magazine*, vol. 26, no. 3, pp. 96–114, 2006.

[2] J. H. Lee and K. S. Lee, "Iterative learning control applied to batch processes: An overview," *Control Engineering Practice*, vol. 15, no. 10, pp. 1306–1318, 2007.

[3] J. Lu, Z. Cao, C. Zhao, and F. Gao, "110th anniversary: An overview on learning-based model predictive control for batch processes," *Industrial & Engineering Chemistry Research*, vol. 58, no. 37, pp. 17 164–17 173, 2019.

[4] U. Rosolia, A. Carvalho, and F. Borrelli, "Autonomous racing using learning model predictive control," in *2017 American Control Conference (ACC)*. IEEE, 2017, pp. 5115–5120.

[5] J. Kabzan, L. Hewing, A. Liniger, and M. N. Zeilinger, "Learning-based model predictive control for autonomous racing," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3363–3370, 2019.

[6] M. K. Cobb, K. Barton, H. Fathy, and C. Vermillion, "Iterative learning-based path optimization for repetitive path planning, with application to 3-d crosswind flight of airborne wind energy systems," *IEEE Transactions on Control Systems Technology*, pp. 1–13, 2019.

[7] U. Rosolia and F. Borrelli, "Learning model predictive control for iterative tasks. a data-driven control framework," *IEEE Transactions on Automatic Control*, vol. 63, no. 7, pp. 1883–1896, 2017.

[8] C. Vallon and F. Borrelli, "Task decomposition for iterative learning model predictive control," in *2020 American Control Conference (ACC)*. IEEE, 2020.

[9] L. Torrey and J. Shavlik, "Transfer learning," *Handbook of Research on Machine Learning Applications*, 01 2009.

[10] K. Weiss, T. M. Khoshgoftaar, and D. Wang, "A survey of transfer learning," *Journal of Big data*, vol. 3, no. 1, p. 9, 2016.

[11] D. Bertsimas and B. Stellato, "The voice of optimization," 2018.

[12] L. Hewing, J. Kabzan, and M. N. Zeilinger, "Cautious model predictive control using gaussian process regression," *IEEE Transactions on Control Systems Technology*, 2019.

[13] M. Stolle and C. Atkeson, "Finding and transferring policies using stored behaviors," *Autonomous Robots*, vol. 29, no. 2, pp. 169–200, 2010.

[14] D. Berenson, P. Abbeel, and K. Goldberg, "A robot path planning framework that learns from experience," in *2012 IEEE International Conference on Robotics and Automation*. IEEE, 2012, pp. 3671–3678.

[15] K. Pereida, M. K. Helwa, and A. P. Schoellig, "Data-efficient multirobot, multitask transfer learning for trajectory tracking," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 1260–1267, 2018.

[16] W. Zhi, T. Lai, L. Ott, G. Francis, and F. Ramos, "Octnet: Trajectory generation in new environments from past experiences," 2019.

[17] A. Paraschos, G. Neumann, and J. Peters, "A probabilistic approach to robot trajectory generation," in *2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*. IEEE, 2013, pp. 477–483.

[18] T. Fitzgerald, E. Short, A. Goel, and A. Thomaz, "Human-guided trajectory adaptation for tool transfer," in *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, ser. AAMAS '19, 2019, pp. 1350–1358.

[19] J. Kocijan, *Modelling and control of dynamic systems using Gaussian process models*. Springer.

[20] A. Jain, T. X. Nghiem, M. Morari, and R. Mangharam, "Learning and control using gaussian processes: Towards bridging machine learning and controls for physical systems," in *Proceedings of the 9th ACM/IEEE International Conference on Cyber-Physical Systems*, ser. ICCPS 18, 2018, p. 140149.

[21] E. D. Klenske, M. N. Zeilinger, B. Schölkopf, and P. Hennig, "Gaussian process-based predictive control for periodic error correction," *IEEE Transactions on Control Systems Technology*, vol. 24, no. 1, pp. 110–121, 2015.

[22] M. Bujarbaruah, X. Zhang, and F. Borrelli, "Adaptive MPC with chance constraints for FIR systems," in *2018 Annual American Control Conference (ACC)*, June 2018, pp. 2312–2317.

[23] M. Spong and M. Vidyasagar, *Robot Dynamics And Control*, 01 1989.