2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

Learning High-Level Policies for Model Predictive Control

Yunlong Song, Davide Scaramuzza

***Abstract*— The combination of policy search and deep neural networks holds the promise of automating a variety of decision-making tasks. Model Predictive Control (MPC) provides robust solutions to robot control tasks by making use of a dynamical model of the system and solving an optimization problem online over a short planning horizon. In this work, we leverage probabilistic decision-making approaches and the generalization capability of artificial neural networks to the powerful online optimization by learning a deep high-level policy for the MPC (High-MPC). Conditioning on robot's local observations, the trained neural network policy is capable of adaptively selecting high-level decision variables for the low-level MPC controller, which then generates optimal control commands for the robot. First, we formulate the search of high-level decision variables for MPC as a policy search problem, specifically, a probabilistic inference problem. The problem can be solved in a closed-form solution. Second, we propose a self-supervised learning algorithm for learning a neural network high-level policy, which is useful for online hyperparameter adaptations in highly dynamic environments. We demonstrate the importance of incorporating the online adaption into autonomous robots by using the proposed method to solve a challenging control problem, where the task is to control a simulated quadrotor to fly through a swinging gate. We show that our approach can handle situations that are difficult for standard MPC.**

Code: `https://github.com/uzh-rpg/high_mpc`

I. Introduction

Model Predictive Control (MPC) [1], [2] is a powerful approach for dealing with complex systems with the capability of handling multiple inputs and outputs. MPC has become increasingly popular for robot control due to its robustness to model errors and its capability of incorporating actions limits and solving optimizations online. However, many popular MPC algorithms [3], [1], [4] rely on tools from constrained optimization, which means that convexification, such as a quadratic formulation of the cost function, and approximations of the dynamics are required [5]. The requirement of solving constrained optimization online limits the usage of MPC for dealing with high-dimensional states and complex cost formulation.

Model-free Reinforcement Learning (RL) offers the promise of automatically learning hard-to-engineer policies for complex tasks [6], [7], [8]. In particular, in combination with deep neural networks, deep RL [9], [10], [11] optimizes policies that are capable of mapping high-dimensional sensory inputs directly to control commands. However, the learning of deep neural network policies is highly data-inefficient and suffers from poor generalization. In addition, these methods typically provide little safety or stability guarantees for the system, which is particularly problematic when working with physical robots.

The authors are with the Robotics and Perception Group, Dep. of Informatics, University of Zurich, and Dep. of Neuroinformatics, University of Zurich and ETH Zurich, Switzerland. `http://rpg.ifi.uzh.ch`. This research was supported by the National Centre of Competence in Research (NCCR) Robotics through the Swiss National Science Foundation, the SNSF-ERC Starting Grant, and the European Union's Horizon 2020 Research and Innovation program through the AERIAL-CORE project (H2020-2019-871479).

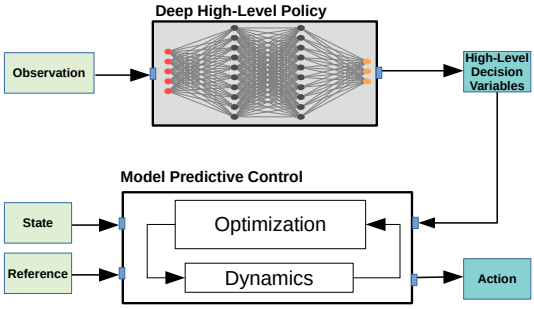

Fig. 1. An overview of our approach for online adaptations of model predictive control using a learned deep high-level policy. The neural network policy is trained using self-supervised learning (Algorithm 2).

Instead of learning end-to-end control policies that map observations directly to robot's control commands, we consider the problem of learning a high-level policy, where the policy chooses task-dependent decision variables for a low-level MPC controller. The MPC takes the decision variables as inputs and generates optimal control commands that are eventually executed on the robot. The policy parameters we are trying to learn can be hyperparameters that are hard-to-identify by human experts or a compact representation of high-dimensional states (see SectionIV).

Contributions: In this work, we leverage intelligent decision-making approaches to the powerful model predictive control. First, we formulate the search of high-level decision variables for MPC as a probabilistic policy search problem. We make use of a weighted maximum likelihood approach [7] for learning the policy parameters, since it allows a closed-form solution for the policy update. Second, we propose a novel self-supervised learning algorithm for learning a neural network high-level policy. Conditioning on the robot's observation in a rapidly changing environment, the trained policy is capable of adaptively selecting decision variables for MPC. We demonstrate the effectiveness of our approach, which incorporates a learned High-level policy into a MPC (High-MPC), by solving a challenging task of controlling a quadrotor to fly through a fast swinging gate.

II. Related Work

The study of combining machine learning or reinforcement learning with model predictive control has been conducted in learning-based control.

Sampling-based MPC are discussed in [5], [12], in which the MPC optimizations are capable of handling complex cost criteria and making use of learned neural networks for dynamics modelling. A crucial requirement for the sampling-based MPC is to generate a large number of samples in real time, where the sampling is generally performed in parallel using graphics processing units (GPUs). Hence, it is computationally expensive to run sampling-based MPC in real time. These methods generally focus on learning dynamics for tasks where a dynamical model of the robots or its environment is difficult to derive analytically, such as aggressive autonomous driving around a dirt track [5].

MPC-guided policy search [13], [14], [15] are methods that study the problems of learning a deep neural network control policy using an MPC as the teacher, and hence, they transform policy search into a supervised learning fashion. The trained end-to-end control policy can forgo the need for explicit state estimation and directly map sensor observations to actions. MPC-guided policy search has been demonstrated to be more data efficient than standard model-free reinforcement learning. However, it suffers from the problem of poor generalizations and stability.

Supervised learning for MPC [16], [17], [18], [19] has been studied in the literature. In [16], [19], the authors proposed to combine a CNN-based high-level policy with a low-level MPC controller to solve the problem of navigating a quadrotor to pass through multiple gates. The trained policy predicts three-dimensional poses of the gate's center from image observations, and then, the MPC outputs control commands for the quadrotor such that it navigates to the predicted waypoints. Similarly, the method in [17] tackles an aggressive high-speed autonomous driving problem by using a CNN-based policy to predict a cost map of the track, which is then directly used for online trajectory optimization. Here, the deep neural network policies are trained using supervised learning, which requires ground-truth labels.

III. Background

A. Model Predictive Control

We consider the problem of controlling an nonlinear deterministic dynamical system whose dynamics is defined by a differential equation $\dot{\mathbf{x}}_t = f(\mathbf{x}_t, \mathbf{u}_t)$, where $\mathbf{x}_t \in \mathbb{R}^n$ is the state vector, $\mathbf{u}_t \in \mathbb{R}^m$ is a vector of the control command, and $\dot{\mathbf{x}}_t \in \mathbb{R}^n$ is the derivative of current state. In model predictive control, we approximate the actual continuous time differential equation using a set of discrete time integration $\mathbf{x}_{h+1} = \mathbf{x}_h + d_t * \hat{f}(\mathbf{x}_h, \mathbf{u}_h)$, with d_t as the time interval between consecutive states and $\hat{f}$ as an approximated dynamical model.

At every time step t, the system is in state $\mathbf{x}_t$. MPC takes the current state $\mathbf{x}_t$ and a vector of additional references $\mathbf{p}$ as input. MPC produces a sequence of optimal system states and control commands $\boldsymbol{\tau} = \{(\mathbf{x}_1, \mathbf{u}_1), \cdots, (\mathbf{x}_{H-1}, \mathbf{u}_{H-1}), \mathbf{x}_H\}$ by solving an optimization online, using a mulitple-shooting scheme. The first control command is applied to the system, after which the optimization problem is solved again in the next state. MPC requires minimizing a quadratic cost over a fixed time horizon H at each control time step by solving a constrained optimization:

$$\begin{aligned} \min_{\mathbf{u}_{1:H}, \mathbf{x}_{1:H}} \quad & J = \sum_{h=1}^{H} c(\mathbf{x}_h, \mathbf{u}_h, \mathbf{p}, \mathbf{z}) \\ \text{subject to} \quad & \mathbf{g}(\mathbf{x}, \mathbf{u}) = 0, \quad \mathbf{h}(\mathbf{x}, \mathbf{u}) \leq 0 \\ & \mathbf{x}_{h+1} = \mathbf{x}_h + d_t * \hat{f}(\mathbf{x}_h, \mathbf{u}_h), \quad \mathbf{x}_1 = \mathbf{x}_{\text{init}} \end{aligned} \tag{1}$$

where $\mathbf{g}(\mathbf{x}, \mathbf{u})$ represents equality constraints and $\mathbf{h}(\mathbf{x}, \mathbf{u})$ represents inequality constraints. Here, $\mathbf{p}$ is a vector of reference states that are normally determined by a path planner and are directly related to the task goal. We represent a vector of high-level variables as $\mathbf{z}$, which has to be defined in advance by human experts, or learned using our policy search algorithm (Sec. IV).

B. Episode-based Policy Search

We summarize episode-based policy search by following the derivation from [8]. Unlike step-based policy search [11], [20], which explores in the action space by adding exploration noise directly to the executed actions, episode-based policy search perturbs the parameters of a low-level controller in parameter space [8]. This kind of exploration is normally added in the beginning of an episode and a reward function is used to evaluate the quality of trajectories that are generated by sampled parameters. A list of episode-based policy search algorithms have been discussed in literature [7], [21], [22], [8]. We focus on a probabilistic model in which the search of high-level parameters for the low-level controller is treated as a probabilistic inference problem. A visualization of the inference problem is given in Fig 2, the graphical model is inspired by [8].

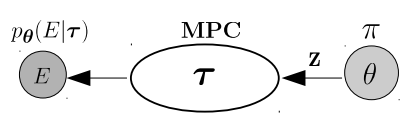

Fig. 2. Graphical model for learning a high-level policy π_θ for MPC.

We make use of an MPC as the low-level controller where the decision variables in MPC is represented as a vector of unknown variables $\mathbf{z}$. We define a reward function as $R(\boldsymbol{\tau})$, which is used to evaluate the goodness of the MPC solution $\boldsymbol{\tau}$ with respect to the given task. The goal of policy search is to find the optimal policy $\pi(\boldsymbol{\theta}^*)$ such that it automatically selects the high-level variables $\mathbf{z}$ for the MPC. Therefore, it is equivalent to maximize an expectation of the reward signal. Here, the reward function is different from the cost function optimized by the MPC, but directly related to the task goal.

To formulate the policy search as a latent variable inference problem, similar to [8], we introduce a binary "reward

event" as the observation, denoted as $E = 1$. Maximizing the reward signal implies maximizing the probability of this "reward event". This leads to the following maximum likelihood problem [8]:

$$\max_{\boldsymbol{\theta}} \quad \log p_{\boldsymbol{\theta}}(E=1) = \log \int_{\boldsymbol{\tau}} p(E|\boldsymbol{\tau}) p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) d\boldsymbol{\tau}, \tag{2}$$

which can be solved efficiently using Monte-Carlo Expectation-Maximization (MC-EM) [6], [23]. MC-EM algorithms find the maximum likelihood solution for the log marginal-likelihood (2) by introducing a variational distribution $q(\boldsymbol{\tau})$, and then, decompose the marginal log-likelihood into two terms:

$$\log p_{\boldsymbol{\theta}}(E=1) = \mathcal{L}_{\boldsymbol{\theta}}(q(\boldsymbol{\tau})) + \mathrm{KL}(q(\boldsymbol{\tau})||p_{\boldsymbol{\theta}}(\boldsymbol{\tau}|E)) \tag{3}$$

where $\mathcal{L}_{\boldsymbol{\theta}}(q(\boldsymbol{\tau}))$ is the lower bound of $\log p_{\boldsymbol{\theta}}(Z=1)$.

The MC-EM algorithm is an iterative method alternates between performing an Expectation (E) step and a Maximization (M) step. In the expectation step, we minimize the Kullback–Leibler (KL) divergence $\mathrm{KL}(q(\boldsymbol{\tau})||p_{\boldsymbol{\theta}}(\boldsymbol{\tau}|E))$, which is equivalent to setting $q(\boldsymbol{\tau}) = p_{\boldsymbol{\theta}}(\boldsymbol{\tau}|E) \propto p(E|\boldsymbol{\tau})p_{\boldsymbol{\theta}}(\boldsymbol{\tau})$. In the maximization, we use the sampled distributions for estimating the complete-data log-likelihood by maximizing the following weighted maximum likelihood objective:

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \left\{ \sum_i d^{[i]} \log \pi(\boldsymbol{z}^{[i]}; \boldsymbol{\theta}) \right\} \tag{4}$$

where $d^{[i]} = p(E|\boldsymbol{\tau}^{[i]})$ is an improper probability distribution for the trajectory $\boldsymbol{\tau}^{[i]}$. The trajectory $\boldsymbol{\tau}^{[i]}$ is collected by solving an MPC optimization problem using $\mathbf{z}^{[i]}$. The solution for updating the policy parameters $\boldsymbol{\theta}$ has a closed-form expression.

IV. Methodology

A. Problem Formulation

We make use of a Gaussian distribution $\pi_{\boldsymbol{\theta}} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ to model the high-level policy, where $\boldsymbol{\mu}$ is the mean vector, $\boldsymbol{\Sigma}$ is a covariance matrix, and hence, $\boldsymbol{\theta} = \{\boldsymbol{\mu}, \boldsymbol{\Sigma}\}$ represents all policy parameters. We design a model predictive control with a vector of unknown decision variables $\boldsymbol{z}$ to be specified. The variables are directly related to the goal of a task and have to be specified in advance before MPC solves the optimization problem. MPC produces a trajectory $\boldsymbol{\tau}$ that consists of a sequence of optimal system states and control commands $(\mathbf{s}, \mathbf{u})$. The cost function is defined by the variables and additional references states, such as a target position or a planned trajectory.

We define a reward function $R(\boldsymbol{\tau})$ which evaluates the goodness of the predicted trajectory $\boldsymbol{\tau}$ with respect to the task goal. The design of this reward function is more flexible than the cost function optimized by MPC, which allows us to work with complex reward criteria, such as exponential reward, discrete reward, and even sparse reward. For example, we can compute the reward by counting the total number of non-collision states in the predicted trajectory. Maximizing this reward can hence find the optimal collision free trajectory.

B. Probabilistic Policy Search for MPC

We first focus on solving the problem of learning a high-level policy $\pi_{\boldsymbol{\theta}}$ that does not depend on robot's observations, where our goal is to find an optimal policy which maximizes the expected reward of predicted trajectories denoted as $\boldsymbol{\tau}$. We used a weighted maximum likelihood algorithm to solve the maximum likelihood estimation problem, where maximizing the reward is equivalent to maximizing the probability of the binary "event", denoted as $p_{\boldsymbol{\theta}}(E|\boldsymbol{\tau})$ (Section III).

The maximization problem corresponds to weighted maximum likelihood estimation of $\pi_{\boldsymbol{\theta}}$ where each sample $\boldsymbol{\tau}^{[i]}$ is weighted by $d^{[i]} = p(E|\boldsymbol{\tau})$. To transform the reward signal $R(\boldsymbol{\tau}^{[i]})$ of a sampled trajectory $\boldsymbol{\tau}^{[i]}$ into a probability distribution $d^{[i]}$, we use the exponential transformation [8]:

$$d^{[i]} = \exp\left\{\beta R(\boldsymbol{\tau}^{[i]})\right\} \tag{5}$$

where the parameter $\beta \in \mathbb{R}_+$ denotes the inverse temperature of the soft-max distribution, higher value of β implies more greedy policy update. A comparison of using different β for the policy update is shown in Fig. 3. A complete episode-based policy search for learning a high-level policy in MPC is given in Algorithm 1.

Algorithm 1: Probabilistic Policy Search for MPC

Input: $\pi_{\boldsymbol{\theta}}(\boldsymbol{\mu}, \boldsymbol{\Sigma}), N, \mathrm{MPC}, \mathbf{x}_0, \mathbf{p}$
While not converged
  Sample variables: $\boldsymbol{z}^{[i]} \sim \pi_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{\mu}, \boldsymbol{\Sigma})_{i=1\ldots N}$
  Sample trajectories: $\boldsymbol{\tau}^{[i]} = \mathrm{MPC.solve}(\mathbf{x}_0, \boldsymbol{z}^{[i]}, \mathbf{p})$
  Expectation:
    $d^{[i]} = \exp\left\{\beta R(\boldsymbol{\tau}^{[i]})\right\}$
  Maximization:
    $\boldsymbol{\mu} = \left(\sum_{i=1}^N d^{[i]}\mathbf{z}^{[i]}\right) / \left(\sum_{i=1}^N d^{[i]}\right)$
    $\boldsymbol{\Sigma} = \left(\sum_{i=1}^N d^{[i]}(\mathbf{z}^{[i]} - \boldsymbol{\mu})(\mathbf{z}^{[i]} - \boldsymbol{\mu})^T\right) / Y$
  $Y = \left((\sum_{i=1}^N d^{[i]})^2 - \sum_{i=1}^N (d^{[i]})^2\right) / \left(\sum_{i=1}^N d^{[i]}\right)$
  $\rightarrow \boldsymbol{\theta}_{\text{new}} = [\boldsymbol{\mu}, \boldsymbol{\Sigma}]$
Output: Learned high-level policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{\mu}^*, \boldsymbol{\Sigma}^*)$

We represent our policy $\pi_{\boldsymbol{\theta}}$ using a normal distribution with randomly initialized policy parameters $\boldsymbol{\theta}$. We consider the robot at a fixed state $\mathbf{x}_0$, which does not change during the learning. At the beginning of each training iteration, we randomly sample a list of parameters of length N from the current policy distribution $\pi_{\boldsymbol{\theta}}$ and evaluate the parameters via a predefined reward function $R(\boldsymbol{\tau})$, where $\boldsymbol{\tau}^{[i]}$ are the trajectories predicted by solving the MPC with sampled variables $\boldsymbol{z}^{[i]}$.

In the Expectation step, we transform the computed reward signal $R(\boldsymbol{\tau})$ into a non-negative weight $d^{[i]}$ (improper probability distribution) via the exponential transformation (5). In the Maximization step, we update the policy parameters by optimizing the weighted maximum likelihood objective (4), where the policy parameter, both the mean and the covariance, are updated using a closed-form expression.

We repeat this process until the expectation of sampled reward converges. Here, $\mathbf{p}$ is a vector of auxiliary variables. After training (during policy evaluation), we simply take the mean vector of the Gaussian policy as the optimal decision variables for the MPC. Therefore, $\mathbf{z} = \boldsymbol{\mu}^*$ is the optimal MPC decision variables found by our approach.

C. Learning A Deep High-Level Policy

We extend Algorithm 1 of learning a high-level policy to learning a deep neural network high-level policy, where the trained neural network policy is capable of selecting adaptive decision variables for the MPC given different observations of the robot. Such properties are potentially useful for the robot to adapt its behavior online in a highly dynamic environment. For example, it is important to use an adaptive control scheme for mobile robots since the robot's dynamics and its surrounding environment changes frequently.

First, we characterize an observation vector of the robot as $\mathbf{o}$, where the observation can be either high-dimensional sensory inputs, such as images, or low-dimensional states, such as the robot's pose. Second, we define a general-purpose neural network denoted as $f_{\boldsymbol{\Phi}}$, with $\boldsymbol{\Phi}$ being the network weights to be optimized. We train the deep neural network policy by combining the episode-based policy search (Algorithm 1) with a self-supervised learning approach. Our algorithm of learning a deep high-level policy is summarized in Algorithm 2.

Algorithm 2: Learning A Deep High-Level Policy

Input: $f_{\boldsymbol{\Phi}}, \mathcal{D} = \{\}$, **Algorithm** 1
Data collection (repeat)
  Randomly reset the system: $\mathbf{x}_t, \mathbf{o}_t, \mathbf{p}_t, t = 0$
  While not done:
    $(\mathbf{z}_t = \boldsymbol{\mu}^*) \leftarrow$ **Algorithm 1** $(\mathbf{x}_0 = \mathbf{x}_t, \mathbf{p}_t)$
    Data collection: $\mathcal{D} \leftarrow \mathcal{D} \cup \{\mathbf{o}_t, \mathbf{z}_t\}$
    MPC optimization: $\mathbf{u}_t^* = \text{MPC.solve}(\mathbf{x}_t, \mathbf{z}_t, \mathbf{p}_t)$
    System transition: $\mathbf{x}_t \leftarrow f(\mathbf{x}_t, \mathbf{u}_t^*)$
Policy learning
  $\boldsymbol{\Phi}_{\text{new}} = \arg\min_{\boldsymbol{\Phi}} \|f_{\boldsymbol{\Phi}}(\mathbf{o}_t) - \mathbf{z}_t\|^2$
Output: Learned deep high-level policy $f_{\boldsymbol{\Phi}^*}$

We divide the learning process into two stages: 1) data collection, 2) policy learning. In the data collection stage, we randomly initialize the robot in a state $\mathbf{x}_t$ and find the optimal decision variables $\mathbf{z}_t^*$ via Algorithm 1. We aggregate our dataset by $\mathcal{D} \leftarrow \mathcal{D} \cup (\mathbf{o}_t, \mathbf{z}_t^*)$, where $\mathbf{o}_t$ is the current observation of the robot. An sequence of optimal control actions $\mathbf{u}_t^*$ are computed by solving the MPC optimization, given the current state $\mathbf{x}_t$ of the robot and the learned variable $\mathbf{z}_t^*$. The first control command is applied to the system, subsequently, the robot transitions to the next state. Incrementally, we collect a set of data that consists of a variety of observation-optimal-variables pairs $(\mathbf{o}_t, \mathbf{z}_t^*)$. In the policy learning stage, we optimize the neural network by minimizing the mean-squared-error between the labels $\mathbf{z}_t^*$ and the prediction of the network $f_{\boldsymbol{\Phi}}(\mathbf{o}_t)$, using stochastic gradient descent.

V. Experiments

A. Problem Formulation

1) Passing Through a Fast Moving Gate: To demonstrate the effectiveness of our approach, we aim at solving a challenging control problem. Our task is to maneuver a quadrotor to pass through the center of a swinging gate that hangs from the ceiling via a cable. We assume that the gate oscillates in a same two-dimensional plane (Fig. 5). Thus, we model the motion of the gate as a simple pendulum. Such a quadrotor control problem can be solved via a traditional modular planning-tracking pipeline, where an explicit trajectory generator, such as a minimum snap trajectory [24] or motion primitives [25] is combined with a low-level controller. To forgo the need for an explicit trajectory generator, we intend to solve this problem using our proposed High-MPC, where we make use of a high-level policy to adaptively select a decision variable for a low-level MPC controller. Our approach automatically find an optimal trajectory for flying through the gate by solving an adaptive MPC optimization online,

Quadrotor Dynamics: We model the quadrotor as a rigid body controlled by four motors. We use the quadrotor dynamics proposed in [25]:

$$\begin{aligned}
\dot{\mathbf{p}}_{WB} &= \mathbf{v}_{WB} \\
\dot{\mathbf{v}}_{WB} &= \mathbf{q}_{WB} \odot \mathbf{c} - \mathbf{g} \\
\dot{\mathbf{q}}_{WB} &= \frac{1}{2}\boldsymbol{\Lambda}(\boldsymbol{\omega}_B) \cdot \mathbf{q}_{WB}
\end{aligned}$$

where $\mathbf{p}_{WB} = [x_q, y_q, z_q]^T$ and $\mathbf{v}_{WB} = [v_{q,x}, v_{q,y}, v_{q,z}]^T$ are the position and velocity of the quadrotor in the world frame W. We use a unit quaternion $\mathbf{q}_{WB} = [q_{q,w}, q_{q,x}, q_{q,y}, q_{q,z}]^T$ to represent the orientation of the quadrotor and use $\boldsymbol{\omega}_B = [\omega_x, \omega_y, \omega_z]^T$ to denote the body rates (roll, pitch, and yaw respectively) in the body frame B. Here, $\mathbf{g} = [0, 0, -g_z]^T$ with $g_z = 9.81m/s^2$ is the gravity vector, and $\boldsymbol{\Lambda}(\boldsymbol{\omega}_B)$ is a skew-symmetric matrix. Finally, $\mathbf{c} = [0, 0, c]^T$ is the mass-normalized thrust vector. We use a state vector $\mathbf{x}_q = [\mathbf{p}_{WB}, \mathbf{v}_{WB}, \mathbf{q}_{WB}]$ and an action vector $\mathbf{u}_q = [c, \omega_x, \omega_y, \omega_z]$ to denote the quadrotor's states and control commands separately.

Pendulum Dynamics: We use a simple pendulum which is modeled as a bob of mass m_p attached to the end of a massless cord L. The cord is hinged at a fixed pivot point denoted as $\boldsymbol{P}_{WP} = [x_f, y_f, z_f]$. The pendulum is subject to three forces: the gravity, the tension force exerted by the cord upon the bob, and a damping force due to friction and air drag. The damping force is proportional to the angular velocity $\dot{\theta}$ and denoted as $f_d = -b * \dot{\theta}$, where $b \in \mathbb{R}_+$ is a damping factor. Hence, we use the following dynamical model

$$\begin{aligned}
\dot{\theta} &= \theta_v \\
\dot{\theta}_v &= -\frac{g_z}{L}\sin(\theta) - \frac{b}{m_p}\dot{\theta}
\end{aligned}$$

to simulate the motion of our gate, where θ is the angle displacement with respect to the vertical direction. We constrain the pendulum's motion in the $y-z$ plane, where $x = x_f$ and $v_x = 0$. A Cartesian coordinate representation of the pendulum in the world frame W can be obtained from the pendulum's angle displacement θ with respect to $\boldsymbol{P}_{WP}$ and L. We can represent the state of the gate's center using the state vector $\mathbf{x}_p = [x_p, y_p, z_p, v_{p,x}, v_{p,y}, v_{p,z}, q_{p,w}, q_{p,x}, q_{p,y}, q_{p,z}]$.

Model Predictive Control: We solve the problem of passing through the swinging gate using non-linear model predictive control. We make use of discrete time models, where a list of quadrotor states $\mathbf{x}_{q,h}, \forall h \in [0, H]$ and control commands $\mathbf{u}_{q,h}, \forall h \in [0, H-1]$ are sampled with a discrete time step d_t. We define the objective $\mathcal{L}$ as a sum over three different cost components: a goal cost $\mathcal{L}_{\text{g}}$, a tracking cost $\mathcal{L}_{\text{tr}}$, and an action regularization cost $\mathcal{L}_{\text{u}}$. Thus, we solve the following constrained optimization problem:

$$\begin{aligned}
\min_{\mathbf{u}_q, \mathbf{x}_q} \quad & \mathcal{L}_{\text{g}}(\mathbf{x}_{q,H}, \mathbf{r}_g) + \sum_{h=0}^{H-1} \mathcal{L}_{\text{tr}}(\mathbf{x}_{q,h}, \mathbf{r}_h, t_{\text{tra}}) + \mathcal{L}_{\text{u}}(\mathbf{u}_h) \\
& = \boldsymbol{\delta}_{\text{g},H}^T \mathbf{Q}_{\text{g}} \boldsymbol{\delta}_{\text{g},H} + \sum_{h=0}^{H-1} \boldsymbol{\delta}_{\text{tr},h}^T \mathbf{Q}_{\text{tr}}(t_{\text{tra}}, h) \boldsymbol{\delta}_{\text{tr},h} + \boldsymbol{\delta}_{\text{u},h}^T \mathbf{Q}_{\text{u}} \boldsymbol{\delta}_{\text{u},h} \\
\text{s.t.}: \quad & c_{\text{min}} \leq c \leq c_{\text{max}} \\
& -\boldsymbol{\omega}_{\text{max}} \leq \boldsymbol{\omega}_B \leq \boldsymbol{\omega}_{\text{max}}
\end{aligned}$$

where $\boldsymbol{\delta}_{\text{tr},h} = (\mathbf{x}_{\text{q},h} - \mathbf{r}_h)$ are differences between the vehicle's states $\mathbf{x}_{\text{q},h}$ and reference states $\mathbf{r}_h$ at the stage h, and $\boldsymbol{\delta}_{\text{g},H} = (\mathbf{x}_{\text{q},H} - \mathbf{r}_{\text{g}})$ defines the difference between the vehicle's terminal state $\mathbf{x}_{\text{q},H}$ and a hovering state $\mathbf{r}_{\text{g}}$. Here, $\boldsymbol{\delta}_{u,h} = (\mathbf{u}_h - \mathbf{u}_{\text{r}})$ is a regularization for predicted control commands $\mathbf{u}_h$, where the reference command $\mathbf{u}_{\text{r}} = [g_z, 0, 0, 0]$ is the command required for hovering the quadrotor. The control commands are constrained by $c_{\text{min}}, c_{\text{max}}, \boldsymbol{\omega}_{\text{max}} \in \mathbb{R}_+$.

Cost Functions: In MPC, we minimize a sum of quadratic cost functions over the receding horizon T using a sequential quadratic program (SQP). We design quadratic cost functions using positive definite diagonal matrices $\mathbf{Q}_{\text{g}}$, $\mathbf{Q}_{\text{tr}}(t_{\text{tra}}, h)$, and $\mathbf{Q}_{\text{u}}$. In particular, both $\mathbf{Q}_{\text{g}}$ and $\mathbf{Q}_{\text{u}}$ are time-invariant matrices. Here, $\mathbf{Q}_{\text{g}}$ defines the importance of reaching to a hovering state $\mathbf{r}_g$ at the end of the horizon and $\mathbf{Q}_{\text{u}}$ corresponds to the importance of taking the control commands that are not diverging too much from the reference command $\mathbf{u}_r$.

Since the gate is swinging in the $y-z$ plane, in order to pass through the gate, the quadrotor has to fly forward in the x direction and simultaneously minimize its distance to the center of the gate in both y and z axes. Hence, the quadrotor has to track the pendulum's motion in both axes when it approaches to the gate. To do so, we use a time-varying cost matrix $\mathbf{Q}_{\text{tr}}(\phi, h)$, which is defined as:

$$\mathbf{Q}_{\text{tr}}(t_{\text{tra}}, h) = \mathbf{Q}_{\text{tr, max}} * \exp\left(-\alpha * (h * d_t - t_{\text{tra}})^2\right) \tag{6}$$

where the exponential function defines the temporal importance for each states $\mathbf{x}_q$, and $\alpha \in \mathbb{R}_+$ defines the temporal spread of states in terms of tracking the pendulum's motion. Here, $t_{\text{tra}} \in [0, 2T]$ is a time variable that defines the best traversal time for the quadrotor, having $T < t_{\text{tra}} < 2T$ helps the quadrotor go to the hovering point after passing through the gate. Hence, for states $\mathbf{x}_q$ that are close to the t_{tra}, we have $\mathbf{Q}_{\text{tr}}(t_{\text{tra}}, h) \approx (\mathbf{Q}_{\text{tr, max}} * 1)$, which means that these states should strictly follow the pendulum in y and z. However, for states that are faraway from t_{tra}, we have $\mathbf{Q}_{\text{tr}}(t_{\text{tra}}, h) \approx (\mathbf{Q}_{\text{tr, max}} * 0)$, which indicates that it is not necessary for these states to follow the pendulum's motion. Here, $\mathbf{Q}_{\text{tr, max}}$ defines the maximum weight that should be assigned for tracking the pendulum. Without considering the importance of each state at different time stages, e.g., weighting the tracking loss $\boldsymbol{\delta}_{\text{tr},h}$ in all time stages using the same constant cost matrix, the quadrotor flies trajectories that would oscillate around the forward axis (see Fig. 6).

Therefore, a key requirement for our MPC to solve the problem is to obtain the optimal traversal time t_{tra} in advance. A similar problem was discussed in [26], where a time variable at which a desired static waypoint should be reached by a quadrotor was determined by human experts. In our case, the time variables are more difficult to obtain, especially when we consider adapting the variable online.

B. Learning Traversal Time

We first consider the scenario where the quadrotor always starts from the same initial hovering state with $[x_q, y_q, z_q] = [-1, 0, 2]$ and the pendulum is hinged at a fixed pivot point $[x_f, y_f, z_f] = [2, 0, 3]$ with cord length $L = 2$ meter (m). The pendulum's initial angle and angular velocity are $[\theta, \dot{\theta}] = [\pi/2, 0]$ (in radians). We define a hovering state $[x_g, y_g, z_g] = [4, 0, 2]$ as a goal state for the quadrotor to hover after passing through the gate. Given the dynamics of the vehicle and the pendulum, we want to plan a trajectory in the future time horizon $T = 2$ seconds, such that the produced quadrotor trajectory $\boldsymbol{\tau}$ intersects the center of the gate at the traversal time t_{tra}.

We learn the decision variable t_{tra} using Algorithm 1 (Section IV), where t_{tra} is modeled as a high-level policy and is represented using a Gaussian distribution $t_{\text{tra}} \sim \pi(\mu, \sigma)$. We first sample a list of $t_{\text{opt}}^{[i]}, i \in [0 \cdots N-1]$ of size N, and then, collect a vector of predicted trajectories $\boldsymbol{\tau}^{[i]}$ by solving N MPC optimizations. We evaluate the sampled trajectories $\boldsymbol{\tau}$ using the following reward function:

$$R(\boldsymbol{\tau}) = -\sum_j (\|\mathbf{x}_{q,j} - \mathbf{x}_{p,j}\| + \|\mathbf{y}_{q,j} - \mathbf{y}_{p,j}\| + \|\mathbf{z}_{q,j} - \mathbf{z}_{p,j}\|)$$

where $j \in [(j^* - 5) \cdots (j^* + 4)]$ correspond to 10 time stages that are close to the time stage determined by the samples $t_{\text{tra}}^{[i]}$ via $j^* = \text{int}(t_{\text{tra}}/d_t)$. Maximizing this reward signal indicates that the high-level policy $\pi(\mu, \sigma)$ tends to sample t_{tra} that allows the MPC to plan a trajectory that has a minimum distance between the quadrotor's state $\mathbf{x}_q$ and the center of the gate $\mathbf{x}_p$ during the traversal. This reward is maximized by solving the weighted maximum likelihood objective (4) using Algorithm 1.

1) High-Level Policy Training: Fig. 3 shows the learning progress of the high-level policy. The learning of such a high-level policy is extremely data-efficient and stable, where the

policy converges in only a few trials. For example, the policy is converged after around 6 training iterations when using $\beta = 3.0$, where in total 6×30 trajectories (equivalent to 180 MPC optimizations) were sampled. We use CasADi [27], which is an open-source tool for nonlinear optimization and algorithmic differentiation, for our MPC implementation. We use a discretization time step of $dt = 0.04s$ and a prediction horizon of $T = 2s$. On average, each MPC optimization takes around $0.03s$ on a standard laptop.

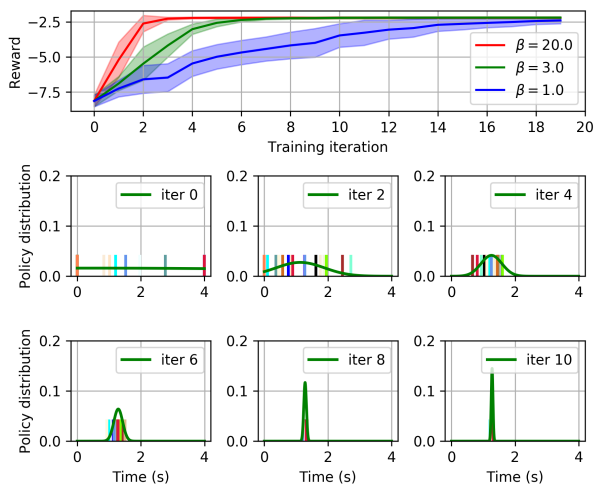

Fig. 3. This figure shows the learning progress of the high-level policy. **Top:** Averaged rewards using different β over 7 runs of each, where policies are randomly initialized with different random seeds. **Bottom:** A visualization of policy distributions and sampled t_{tra} during training. The policy converges to an optimal solution after around 6 iterations.

2) Traverse Trajectory Planning: Fig. 4 shows a comparison between the planned trajectory using our High-MPC (along with an optimized decision variable $t_{\text{tra}} = 1.25$ seconds) and the solution from a standard MPC. The standard MPC minimizes the same cost function with a constant cost matrix $\mathbf{Q}_{\text{tr}} = \mathbf{Q}_{\text{tr, max}}$ for all states and does not use the exponential weighting scheme. As a result, both methods are capable of planning trajectories that pass through the swinging gate, where absolute position errors at the traversal point in the $y - z$ plane are 0.24 meters for High-MPC and 0.30 meters for the standard MPC, respectively. Nevertheless, the control actions (the total thrust and body rates) produced by High-MPC are better for real-world deployment since the inputs reach their limit for lower amount of time, leaving more control authority to counteract disturbance. Our approach only tries to follow the pendulum's motion in y and z directions at the time stages closed to the learned traversal time t_{tra}.

C. Learning Adaptive Traversal Time

Learning a single high-level policy without taking the robot's observation into account is only useful for selecting time-invariant variables or for planning a one-shot trajectory, where the dynamics are perfectly modeled. This, however, is generally not the case. For example, our task requires the

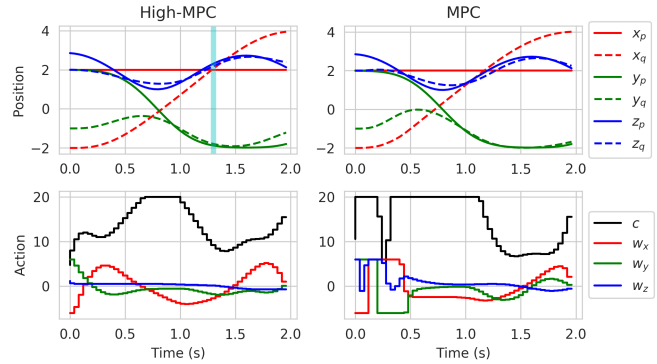

Fig. 4. A comparison of planned trajectories between our High-MPC (with trained $t^*_{\text{tra}} = 1.25$ (s)) and a standard MPC. The vertical line indicates the passing moment. Our High-MPC is better for real-world deployment since the produced actions are much smoother than the standard MPC and reach the limit for lower amount of time.

MPC to constantly update its prediction based on the the vehicle's state with respect to that of the dynamic gate. Hence, we also want to find a high-level policy which is capable of adaptively selecting the time variable t^*_{tra} depending on the robot's observation.

1) Deep High-Level Policy Training: To do so, we make use of a multilayer perceptron (MLP) to generalize the t^*_{tra} to different contexts $\mathbf{o}_t$. We represent $\mathbf{o}_t$ as an observation of the vehicle using $\mathbf{o}_t = \mathbf{x}_{q,t} - \mathbf{x}_{p,t}$, which represents the difference between the vehicle's state $\mathbf{x}_{q,t}$ and the pendulum's state $\mathbf{x}_{p,t}$ at time step t. We use Algorithm 2 (Section IV), where we combine the learning of an optimal high-level policy online with a supervised learning approach to train the MLP. We first randomly initialize the system, which means we use random initial states for the quadrotor, and drop the pendulum from random angles; then, we find the optimal traversal time t^*_{tra} at this state. We solve the MPC optimization using t^*_{tra} and apply the optimal control action to a simulated quadrotor. We repeat this process again at each simulation time step until the quadrotor flies through the gate or it reaches the maximum simulation steps. In total, we collect 40,000 samples that consist of observation-traversal-time pairs $(\mathbf{o}_t, t_{\text{tra}})$. It takes a single core CPU several hours to collect the data, however, the total sampling time can be significantly reduced using parallel processing or multithreading. We use Tensorflow [28] to implement the a fully-connected MLP with two hidden layers of 32 units, and *ReLU* nonlinearities. The training of network weights takes less than 5 minutes on a notebook with a *Nvidia Quadro P1000* graphics card.

2) Passing Through a Fast Moving Gate via High-MPC: We evaluate the effectiveness of our High-MPC by controlling a simulated quadrotor to pass through a fast swing gate, where the quadrotor and the pendulum are randomly initialized in different states. Based on the state of the quadrotor, the motion of the pendulum (including 2s of predicted pendulum motion in the future), and the predicted traversal time, our High-MPC simultaneously plans a trajectory and

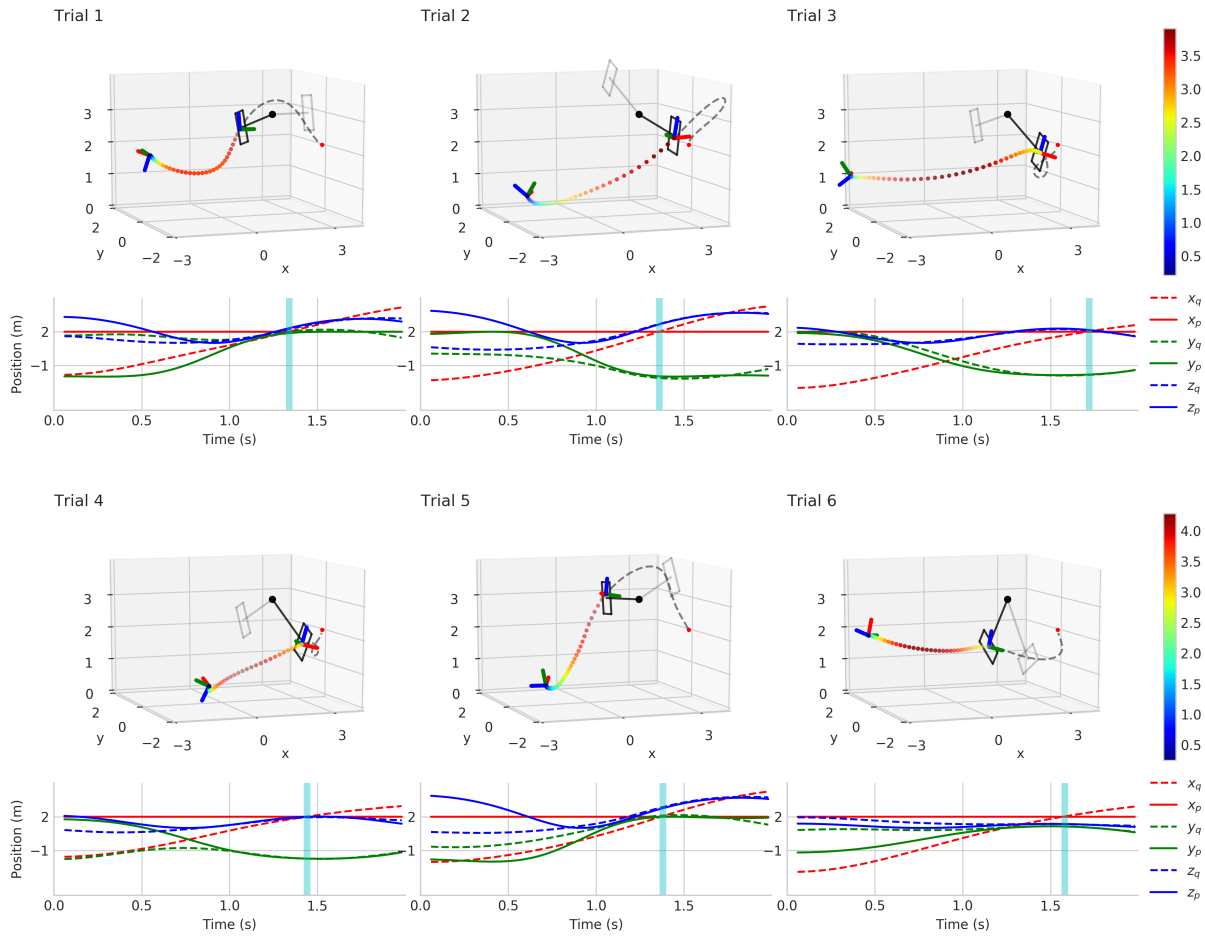

Fig. 5. Demonstrations of our High-MPC for flying through a swinging gate. The initial states of the quadrotor and the pendulum are randomly initialized. In the 3D plots, the initial states of the pendulum are indicated by the grey color, and the black gates show the moment when the quadrotor is intersecting in the gate. The color bars on the right side specify the quadrotor speed in the x direction. The grey dash lines are planned trajectory by our MPC and colored dots are traveled trajectories. The quadrotor's body frame is indicated by $[x_q, y_q, z_q]$. The 2D plots show travelled trajectories of the quadrotor and the pendulum.

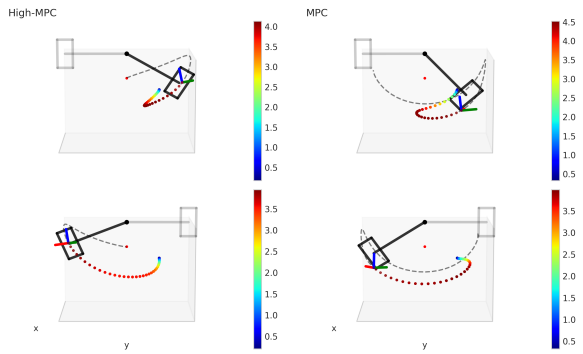

Fig. 6. Comparisons between our High-MPC (left) and a standard MPC (right), where initial states of the system are the same for both methods. **Top:** the swinging gate is released from $\theta = 1.57$ (rad). **Bottom:** the swinging gate is released from $\theta = -1.57$ (rad).

controls the vehicle to pass through the gate. Fig. 5 shows six random examples of the quadrotor successfully flying through the swinging gate.

In addition, we compared the performance of our High-MPC to a standard MPC (Fig. 6), where the standard MPC optimizes a cost function without considering the temporal importance of difference states in tracking the pendulum motion. The standard MPC failed to pass through the gate and results in trajectories that are oscillating about the forward direction (x axis).

VI. Discussion and Conclusion

In this work, we introduced the idea of formulating the design of hard-to-engineer decision variables in MPC as a probabilistic inference problem, which can be solved efficiently using an EM-based policy search algorithm. We combined self-supervised learning with the policy search method to train a high-level neural network policy. After training, the policy is capable of adaptively making online decisions for the MPC. We demonstrated the success of our approach by combining a trained MLP policy with a MPC to solve a challenging control problem, where the task is to maneuver a quadrotor to fly through the center

of a fast-moving gate. We compared our approach (High-MPC) to a standard MPC and showed that ours achieve more robust results, and hence, it is more promising to deploy our method on real robots, thanks to the online decision variable adaptation scheme realized by the deep high-level policy. Besides, our approach has the advantage of tightly coupling planning and optimal control together, and hence, forgo the need for an explicit trajectory planner.

Nevertheless, our approach has limitations such as it requires multiple MPC optimizations in-the-training-loop in order to find optimal variables. It is possible to learn a vector of high-dimensional decision variables and more complex neural network policies, however, the sample complexity will increase by a large margin. To fully exploit the potential of automatically learning high-level policies for optimal control, we hope that our work sparks more researchers' interests in this domain to derive new algorithms and opens up opportunities for solving more complex robotic problems, such as real-world robot navigation in a complex dynamic environment. To test the scalability and generalization of our High-MPC, in the near future we intend to deploy the algorithm on a real robot system.

REFERENCES

[1] J. Rawlings and D. Mayne, *Model Predictive Control: Theory and Design*. Nob Hill Pub., 2009. [Online]. Available: https://books.google.ch/books?id=3_rfQQAACAAJ

[2] D. Q. Mayne, "Model predictive control: Recent developments and future promise," *Automatica*, vol. 50, no. 12, pp. 2967–2986, 2014.

[3] D. Falanga, P. Foehn, P. Lu, and D. Scaramuzza, "PAMPC: Perception-Aware Model Predictive Control for Quadrotors," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct 2018, pp. 1–8.

[4] M. Kamel, M. Burri, and R. Siegwart, "Linear vs nonlinear MPC for trajectory tracking applied to rotary wing micro aerial vehicles," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 3463–3469, 2017.

[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," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 1714–1721.

[6] J. Kober and J. R. Peters, "Policy search for motor primitives in robotics," in *Advances in neural information processing systems*, 2009, pp. 849–856.

[7] J. Peters and S. Schaal, "Reinforcement learning by reward-weighted regression for operational space control," in *Proceedings of the 24th international conference on Machine learning*, 2007, pp. 745–750.

[8] M. P. Deisenroth, G. Neumann, J. Peters *et al.*, "A survey on policy search for robotics," *Foundations and Trends® in Robotics*, vol. 2, no. 1–2, pp. 1–142, 2013.

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

[10] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, "Control of a Quadrotor With Reinforcement Learning," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2096–2103, Oct 2017.

[11] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. Stockholmsmässan, Stockholm Sweden: PMLR, 10–15 Jul 2018, pp. 1861–1870. [Online]. Available: http://proceedings.mlr.press/v80/haarnoja18b.html

[12] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Aggressive driving with model predictive path integral control," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, May 2016, pp. 1433–1440.

[13] S. Levine and V. Koltun, "Guided Policy Search," in *Proceedings of the 30th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, S. Dasgupta and D. McAllester, Eds., vol. 28, no. 3. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 1–9. [Online]. Available: http://proceedings.mlr.press/v28/levine13.html

[14] T. Zhang, G. Kahn, S. Levine, and P. Abbeel, "Learning deep control policies for autonomous aerial vehicles with MPC-guided policy search," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, May 2016, pp. 528–535.

[15] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, 2016.

[16] E. Kaufmann, M. Gehrig, P. Foehn, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza, "Beauty and the Beast: Optimal Methods Meet Learning for Drone Racing," in *2019 International Conference on Robotics and Automation (ICRA)*, May 2019, pp. 690–696.

[17] P. Drews, G. Williams, B. Goldfain, E. A. Theodorou, and J. M. Rehg, "Aggressive Deep Driving: Combining Convolutional Neural Networks and Model Predictive Control," in *Proceedings of the 1st Annual Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, S. Levine, V. Vanhoucke, and K. Goldberg, Eds., vol. 78. PMLR, 13–15 Nov 2017, pp. 133–142. [Online]. Available: http://proceedings.mlr.press/v78/drews17a.html

[18] G. Kahn, P. Abbeel, and S. Levine, "BADGR: An Autonomous Self-Supervised Learning-Based Navigation System," *arXiv preprint arXiv:2002.05700*, 2020.

[19] A. Loquercio, E. Kaufmann, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza, "Deep Drone Racing: From Simulation to Reality With Domain Randomization," *IEEE Transactions on Robotics*, vol. 36, no. 1, pp. 1–14, Feb 2020.

[20] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html

[21] F. Stulp and O. Sigaud, "Path integral policy improvement with covariance matrix adaptation," *arXiv preprint arXiv:1206.4621*, 2012.

[22] C. Daniel, G. Neumann, O. Kroemer, and J. Peters, "Hierarchical Relative Entropy Policy Search," *Journal of Machine Learning Research*, vol. 17, no. 93, pp. 1–50, 2016. [Online]. Available: http://jmlr.org/papers/v17/15-188.html

[23] N. Vlassis and M. Toussaint, "Model-free reinforcement learning as mixture learning," in *Proceedings of the 26th Annual International Conference on Machine Learning*, 2009, pp. 1081–1088.

[24] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *2011 IEEE International Conference on Robotics and Automation*, May 2011, pp. 2520–2525.

[25] M. W. Mueller, M. Hehn, and R. D'Andrea, "A Computationally Efficient Motion Primitive for Quadrocopter Trajectory Generation," *IEEE Transactions on Robotics*, vol. 31, no. 6, pp. 1294–1310, Dec 2015.

[26] M. Neunert, C. De Crousaz, F. Furrer, M. Kamel, F. Farshidian, R. Siegwart, and J. Buchli, "Fast nonlinear model predictive control for unified trajectory optimization and tracking," in *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2016, pp. 1398–1404.

[27] J. Andersson, J. Gillis, G. Horn, J. Rawlings, and M. Diehl, "CasADi: a software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, 07 2018.

[28] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/