Robotics: Science and Systems 2021
Held Virtually, July 12–16, 2021

HJB-RL: Initializing Reinforcement Learning with Optimal Control Policies Applied to Autonomous Drone Racing

Keiko Nagami
Department of Aeronautics and Astronautics
Stanford University
Stanford, CA
Email: knagami@stanford.edu

Mac Schwager
Department of Aeronautics and Astronautics
Stanford University
Stanford, CA
Email: schwager@stanford.edu

***Abstract*—In this work we present a planning and control method for a quadrotor in an autonomous drone race. Our method combines the advantages of both model-based optimal control and model-free deep reinforcement learning. We consider a single drone racing on a track marked by a series of gates, through which it must maneuver in minimum time. Firstly we solve the discretized Hamilton-Jacobi-Bellman (HJB) equation to produce a closed-loop policy for a simplified, reduced order model of the drone. Next, we train a deep network policy in a supervised fashion to mimic the HJB policy. Finally, we further train this network using policy gradient reinforcement learning on the full drone dynamics model with a low-level feedback controller in the loop. This gives a deep network policy for controlling the drone to pass through a single gate. In a race course, this policy is applied successively to each new oncoming gate to guide the drone through the course. The resulting policy completes a high-fidelity AirSim drone race with 12 gates in 34.89s (on average), outracing a model-based HJB policy by 33.20s, a supervised learning policy by 1.24s, and a trajectory planning policy by 12.99s, while a model-free RL policy was never able to complete the race.**

I. Introduction

In this paper, we build a deep Reinforcement Learning (RL) policy from a model-based optimal control policy for application to real-world robotics tasks. Our method seeks to combine the best attributes of both model-based optimal control and model-free RL. We demonstrate this method specifically in the context of autonomous drone racing, a problem involving complex nonlinear dynamics, and where computational speed is essential in order to produce a safe and fast racing policy. Specifically, we solve the Hamilton-Jacobi-Bellman (HJB) partial differential equation for a reduced order drone model to get a feedback policy for drone racing. We then use the HJB policy to train a supervised network to imitate this HJB policy. This network then serves as an initialization for a policy gradient RL method, which uses roll-outs from a full drone dynamics simulation (including drag and a low-level feedback controller) to improve the racing policy. We call the network resulting from this multi-stage training pipeline an HJB-RL policy. Fig. 1 shows HJB-RL in action in a drone race simulated in AirSim.

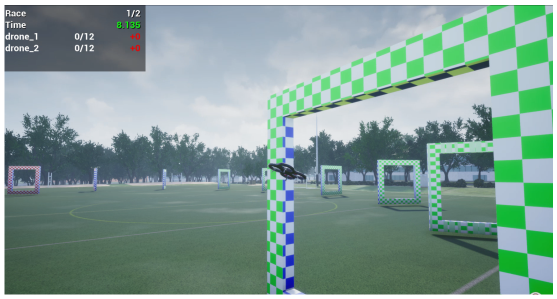

Fig. 1. A drone using our HJB-RL policy is shown traversing a gate in a simulated race in AirSim. Our HJB-RL policy initializes policy gradient RL with a model-based HJB policy, leading to a combined policy that significantly outperforms either HJB or RL on its own, and also outperforms an MPC policy based on trajectory optimization.

The motivation for this multi-stage training pipeline is to leverage the benefits of each component method. HJB methods require discretizing the state space to numerically solve a partial differential equation (PDE). These methods give discrete approximations to the optimal feedback policy, which can be queried online with minimal computation. However, these methods only give solutions on a restricted subset of the state space and are limited to low-dimensional state spaces. The coarse state space discretization and low state space dimensionality make HJB methods impractical for high-dimensional robotics problems. Conversely, a deep network can represent a policy over a large region of the state space and can apply to high-dimensional systems, but training that policy through reinforcement learning may take a large number of simulation rollouts, and may never result in an adequate policy without a good initialization. Indeed, in our drone racing example, we were not able to train an adequate policy from scratch with the policy gradient RL method regardless of the training time or number of rollouts.

Our proposed training pipeline bridges these two methods, providing superior performance to either one alone. We train the RL policy from roll-outs starting within a small region of the state space where the HJB policy performs well. We then expand this region over sequential training epochs to encompass an ever increasing region in the state space. We train the RL policy specifically for a drone to pass through a single gate, then complete a full race by applying the policy successively one gate at a time until the race is complete. Furthermore, we show how to adapt the policy trained with fixed gate dimensions to suit gates of arbitrary dimensions through a fast, online re-scaling procedure. We assume the relative pose between the drone and the gate is known, or can be measured accurately on-line through an existing perception stack, for example, the ones in [13, 14]. In future work, we will consider including unknown gate pose within RL training to produce an end-to-end deep racing policy.

In an ablation study, we show that our HJB-RL policy strongly outperforms other policies derived from any subset of the stages in our pipeline. We compare performance in a high-fidelity, 12-gate drone race in AirSim, competing against a model-based HJB policy, a supervised network policy trained from the HJB policy, and a standard policy gradient RL policy produced without our HJB initialization. We also compare against a baseline Model Predictive Control (MPC) trajectory planning policy. We find that our HJB-RL policy completes the race in 34.89s (on average), which is 33.20s faster that the HJB policy alone, 1.24s faster than the supervised learning policy, and 12.99s faster than the baseline MPC policy. By contrast, we were not able to train a standard policy gradient RL to complete a single race.

We discuss related work in Sec. II and formalize the drone racing problem in Sec. III. We introduce our HJB-RL pipeline in Sec. IV, discuss the low-level controller in Sec. V, and describe our simulation results in Sec. VI. Finally, we give conclusions in Sec. VII

II. Related Work

Our work brings together methods from both model-based optimal control and deep RL. Both of these approaches, in various forms, have been applied to autonomous drone racing, as well as to many other problem in robotics. Within optimal control, one can consider methods based on trajectory optimization, versus methods based on solving the Hamilton-Jacobi-Bellman equation. We review the literature in each of these areas below.

Trajectory optimization methods, which plan paths online repeatedly in a Model Predictive Control (MPC) loop, represent the standard method in motion planning for drones [25, 26, 15, 2, 20], and are common in many other problems in robot motion planning. These are the most common methods in autonomous drone racing [13, 14, 32, 35], however they suffer from requiring significant online computation, which leads to slower, less reactive policies. In this work, we show that our HJB-RL based policy outperforms a more traditional trajectory optimization based MPC approach.

Conversely, the Hamilton-Jacobi-Bellman (HJB) approach to optimal control involves numerically solving a partial differential equation offline to find a feedback policy [3]. This policy can then be applied online quickly, as minimal online computation is required. The problem with this approach is the curse of dimensionality: numerical solution of the HJB equation is intractable for problems with more than 5-6 state-dimensions, and solutions can only be found over a restricted portion of the state space. Quadrotor dynamics are 12 dimensional, which is far beyond the reach of a numerical HJB solution. However, HJB methods have been incorporated into control and planning methods for both drones and cars, usually through a reduced-order approximation of the system dynamics. For example, in [5] the authors solve the HJB equation for a simplified car model to pass through stochastic gates (similar to a drone race, but in 2D). Similarly, [19] incorporates the solution of an HJ equation for a reduced-order model of relative motion between two cars, to enforce safety within a more traditional MPC planning loop for autonomous driving. The paper [8] presents a method to use an HJB controller derived from a low-order model approximation to safely control the original higher-order system. Methods to approximate the value function of the HJB solution have also been explored, e.g., in [11]. To our knowledge, HJB solutions have not yet been applied to autonomous drone racing.

As opposed to model-based optimal control, Reinforcement Learning (RL) has the advantage that no explicit model is required. Instead RL statistically tunes a control policy (usually represented as a deep neural network) by repeatedly rolling out trajectories in a simulation of the dynamics of the system. The disadvantage is that RL is known to suffer from poor sample complexity, typically requiring a very large number of simulation roll-outs. Furthermore, there are no general guarantees about whether an adequate policy can be learned for a given problem, or whether a learned policy will perform robustly or stably when transferred to a real robot. Indeed, these are currently topics of vigorous research. RL can be applied in the form of Q-learning [27, 21], or policy gradient approaches, where the mapping between the state input and the optimal action is learned directly. Several forms of policy gradient methods have been studied [28], including natural policy gradient [12], and off-policy methods [31, 7]. In this work, we focus on a basic policy gradient approach that tunes parameters of a policy network directly based on the policy gradient theorem [33, 4] from roll-outs produced under the current policy. Other applications of policy gradient techniques such as [30] have further developed these approaches, where changes to the natural policy gradient result in monotonic improvement. Several works apply learning to quadrotor platforms [17, 10, 22, 34], where [10, 22, 34] develop algorithms using policy gradient techniques.

In this work, we focus on combining a model-based control method with a learning-based approach. Other works have also considered combining model-based control methods with RL. For example, [36, 1] combine model-based control with RL by learning residual corrections to the dynamics and controls of the system. Other methods use a model-based controller to train a network, as done in [37], where MPC is used in guided policy search. In our method, we use the Hamilton-

Jacobi-Bellman equation to pre-train a network using model-based information, and use reinforcement learning as a means of improving the resulting policy. Somewhat similar to our method, [23] computes a time-to-reach function from an HJ PDE to improve data-efficiency in reinforcement learning. In [6], the authors bridge safety analysis techniques of Hamilton-Jacobi methods to reinforcement learning. It is not clear how either of these methods overcome the challenges of small state space dimension and restricted state space regions that are inherent in HJB methods. In contrast to these works, our method computes the solution to the HJB equation for a simplified low-order model restricted to a small part of the state space, transfers the simplified policy to a deep network through supervised training, then expands the state dimension and state space region in phases through RL.

III. Problem Setup

In this problem, we consider a single autonomous drone racing through a track marked by a series of gates. This problem setup will assume that the global ground truth pose of each oncoming gate is known. Because the quadrotor will be flying at high speeds, we consider body drag forces in the dynamics model, which is given by

$$\dot{\mathbf{p}} = \mathbf{v} \tag{1}$$

$$\dot{\mathbf{v}} = \frac{1}{m}\mathbf{R}\mathbf{T} - \frac{1}{m}\mathbf{R}\mathbf{D}\mathbf{R}^T\|\mathbf{v}\|\mathbf{v} - g\mathbf{z}_w \tag{2}$$

$$\dot{\mathbf{R}} = \mathbf{R}\hat{\boldsymbol{\omega}} \tag{3}$$

$$\dot{\boldsymbol{\omega}} = \mathbf{I}^{-1}(\boldsymbol{\tau} - \boldsymbol{\omega} \times \mathbf{I}\boldsymbol{\omega}), \tag{4}$$

where $\mathbf{p} = [x\ y\ z]^T$ is the position, $\mathbf{v} = [v_x\ v_y\ v_z]^T$ is the velocity, T is the thrust control input $\mathbf{T} = [0\ 0\ T]^T$, which is always applied along the vertical body fixed axis, $\mathbf{D}$ is a diagonal matrix of the drag coefficients, m is the mass of the quadrotor, $\mathbf{R}$ is the orientation between the body frame and the gate centered frame, which is defined as,

$$\begin{aligned}\mathbf{R} &= \mathbf{R}_z\mathbf{R}_y\mathbf{R}_x \\ &= \begin{bmatrix} C\psi & \text{-}S\psi & 0 \\ S\psi & C\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} C\phi & 0 & S\phi \\ 0 & 1 & 0 \\ \text{-}S\phi & 0 & C\phi \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & C\theta & \text{-}S\theta \\ 0 & S\theta & C\theta \end{bmatrix},\end{aligned} \tag{5}$$

where ϕ, θ, and ψ are the roll, pitch, and yaw angles and $C\cdot$ and $S\cdot$ are cosine and sine operations. g is acceleration due to gravity, $\mathbf{z}_w$ is the world frame z-axis, $\boldsymbol{\omega}$ is the vector of angular rates in the body frame $\boldsymbol{\omega} = [p\ q\ r]^T$, $\hat{\boldsymbol{\omega}}$ is a skew symmetric matrix of $\boldsymbol{\omega}$, $\boldsymbol{\tau}$ is the input torque on the quadrotor in the body frame applied by the rotors, and $\mathbf{I}$ is the inertia matrix of the quadrotor. The control input vector for the 12D system is $\mathbf{u}_{12} = [T\ \boldsymbol{\tau}]^T$, the torque and total thrust magnitude applied by the rotors. We assume limits on both control inputs, where the maximum magnitude of the thrust vector is T_{max} and the maximum angular acceleration in the body frame is $\dot{\boldsymbol{\omega}}_{max} = [\dot{p}_{max}\ \dot{q}_{max}\ \dot{r}_{max}]^T$. The position, velocity, thrust, and orientation are defined in a gate centered reference frame as defined in Fig. 2.

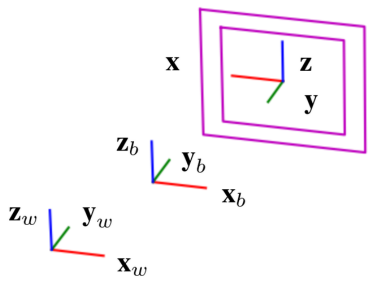

Fig. 2. Definition of world frame ($\mathbf{x}_w$, $\mathbf{y}_w$, $\mathbf{z}_w$) , quadrotor body frame ($\mathbf{x}_b$, $\mathbf{y}_b$, $\mathbf{z}_b$), and gate frame ($\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$).

In this problem, our objectives are to go through each oncoming gate and to do so quickly. Since we will be using a reduced order model to define our state input to the policy, we formulate our cost function as

$$J(\mathbf{u}(t)) = \mathrm{E}[g(\mathbf{s}(t_f)) + \int_0^{t_f} 1dt] \tag{6}$$

$$\mathbf{s}(t_f) \in \mathcal{T} \cup \mathcal{A} \tag{7}$$

$$\mathbf{s} = [x \quad y \quad z \quad v_x \quad v_y \quad v_z]^T \tag{8}$$

$$\mathbf{u} = [T_x \quad T_y \quad T_z]^T, \tag{9}$$

where $g(\mathbf{s}(t_f))$ is the terminal cost, our running cost is only dependent on the time, and our state input $\mathbf{s}$ and control input $\mathbf{u}$ are defined for a reduced order model described in the next section. The terminal cost is defined at the terminal states, which are defined to be the states with positions inside the gate frame as the target set $\mathcal{T}$ and positions at the gate frame as the avoid set $\mathcal{A}$.

IV. HJB-RL

Our approach consists of three main segments. First, we generate a control policy, $\mathbf{u} = f_{HJB}(\mathbf{s})$, in the form of a look-up table by formulating the HJB equation, and discretize the problem to solve by value iteration. We then use a k-nearest neighbor interpolation to generate a larger look-up table that is used to generate state-input data pairs to train a stochastic neural network policy $(\boldsymbol{\mu}, \boldsymbol{\sigma}^2) = f_{SL}(\mathbf{s})$ using supervised learning. Here we use a stochastic policy to allow for exploration during training in the reinforcement learning stage of our framework. The 6D state serves as the input to the model, with a mean and fixed variance for the 3D thrust vector as the output. Finally, this pretrained network is used to further learn a better policy $(\boldsymbol{\mu}, \boldsymbol{\sigma}^2) = f_{HJB-RL}(\mathbf{s})$ on the full 12D state model through policy gradient reinforcement learning, using simulated roll-outs of trajectories going through a gate. When applying the control actions using this final policy, we will execute the mean $\boldsymbol{\mu}$ as the control input, as having a stochastic policy is unfavorable after training. Fig. 3 visualizes the multiple stages in this training process, and Fig. 4 shows the block diagram of the final planning and control method that we implement after all training stages. The block diagram in Fig. 4 shows how the HJB-RL policy is applied to the quadrotor system with the low level controllers for orientation.

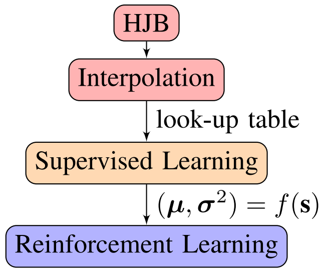

Fig. 3. HJB-RL multi-stage training procedure. First the state space in front of a single gate is discretized for a lower dimensional model of a quadrotor to obtain an optimal control policy using an HJB equation. This look-up table is then used as training data for a neural network for a supervised learning stage. Lastly, reinforcement learning is applied using the full 12D quadrotor dynamics to further improve the policy.

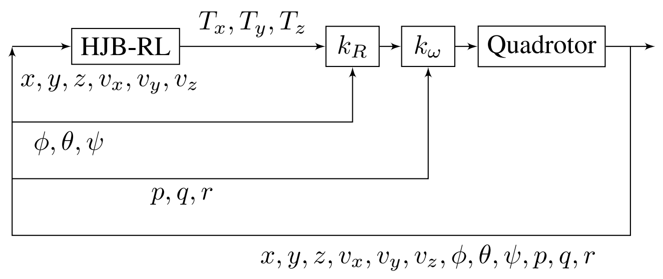

Fig. 4. Block diagram of our policy acting on a quadrotor. The six-dimensional state is used as input to our HJB-RL block, which outputs a three-dimensional thrust vector in the gate centered coordinate frame. Two low-level controllers are used to align the quadrotor with the desired thrust vector.

A. *Hamilton-Jacobi-Bellman Equation*

To formulate the HJB equation, we first choose a lower dimensional model, where we reduce equation (2) and approximate the quadrotor as a point-mass subject to drag forces,

$$\mathrm{d}\mathbf{p} = \mathbf{v}\mathrm{dt} \tag{10}$$

$$\begin{aligned} \mathrm{d}\mathbf{v} &= (\frac{1}{m}\mathbf{u} - \frac{d}{m}\|\mathbf{v}\|\mathbf{v} - g\mathbf{z}_w)\mathrm{dt} + \sigma\mathrm{d}\mathbf{W} \\ &= \mathbf{a}\mathrm{dt} + \sigma\mathrm{d}\mathbf{W}, \end{aligned} \tag{11}$$

where d is a scalar constant drag coefficient, $\mathbf{a} = [a_x \; a_y \; a_z]^T$ is the acceleration, and $\mathbf{u}$ here is a thrust vector control input that is defined in the gate centered coordinate frame. In this reduced order model everything is defined either in a gate centered frame or the world frame. Additionally, we consider some uncertainty on the velocity of the quadrotor in the gate frame since we are using a simplified model of the full dynamics. In this reduced order dynamics model, we use a 6D state vector expressed in equation (8).

To formulate the HJB problem we use the cost function as defined in equation (6), and define the target set as the discrete states inside the gate frame and the avoid set as the discrete states at the gate frame. We account for the arm lengths of the drone and also include discretized states inside the gate frame in the avoid set as shown in Fig. 5. The cost at these terminal states are low in the target set and high in the avoid set,

$$g(\mathbf{s}(t_f)) = \begin{cases} 0, & \mathbf{s}(t_f) \in \mathcal{T} \\ 10, & \mathbf{s}(t_f) \in \mathcal{A}. \end{cases} \tag{12}$$

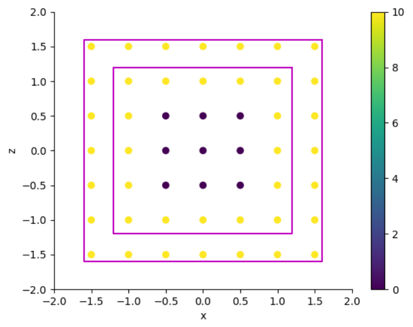

Fig. 5. Front view of gate definition and target and avoid sets. Color bar indicates values associated with discretized points of target set and avoid set. Pink lines mark the inner and outer dimensions of the gate.

For all states not in the target and avoid sets, the optimal control is found through the HJB equation,

$$\inf_{\mathbf{u}}[\mathcal{L}^{\mathbf{u}}V(\mathbf{s}) + 1] = 0 \tag{13}$$

where $V(\mathbf{s})$ is the value function, and $\mathcal{L}^{\mathbf{u}}$ is a differential operator as described in [5] to account for the uncertainty in the relative velocity between the gate frame and the quadrotor,

$$\begin{aligned} \mathcal{L}^{\mathbf{u}}V(\mathbf{s}) &= v_x\frac{\delta V}{\delta x} + v_y\frac{\delta V}{\delta y} + v_z\frac{\delta V}{\delta z} \\ &+ a_x\frac{\delta V}{\delta v_x} + a_y\frac{\delta V}{\delta v_y} + a_z\frac{\delta V}{\delta v_z} \\ &+ \frac{\sigma^2}{2}\frac{\delta^2 V}{\delta v_x^2} + \frac{\sigma^2}{2}\frac{\delta^2 V}{\delta v_y^2} + \frac{\sigma^2}{2}\frac{\delta^2 V}{\delta v_z^2}. \end{aligned} \tag{14}$$

From here we discretize the problem to find the discrete value function as described in [5] and use value iteration to find a policy which minimizes the value at each point in the state space.

B. *Supervised Learning*

Once the solution to the HJB equation is obtained in the form of a look-up table, additional data is generated using a k-nearest neighbor interpolation. The network is then pretrained by using the input output pairs of the 6D state vector and 3D thrust vector at each of the discretized points of the HJB policy so that the network represents the function $(\boldsymbol{\mu}, \boldsymbol{\sigma}^2) = f_{SL}(\mathbf{s})$, where $\boldsymbol{\mu}$ is a mean vector and $\boldsymbol{\sigma}^2$ are the diagonal elements of a covariance matrix, $\boldsymbol{\Sigma} = \mathrm{diag}(\boldsymbol{\sigma}^2)$, that are used to parameterize a Gaussian distribution of the control input. The network is comprised of four ReLU activation layers each with 50 nodes for the mean, and one sigmoid activation layer with 50 nodes for the variance. While the mean is trained by using data from the HJB policy, the variance is trained to be 0.5. We choose this initial variance value of 0.5 so that during the reinforcement learning training, the policy will draw actions that are close to the mean, while still allowing for exploration during training. We use a Huber loss function [9] to compute the loss,

$$L_\delta(\mathbf{w},\hat{\mathbf{w}}) = \frac{1}{M}\sum_{j=1}^{2M} \begin{cases} \frac{1}{2}(w_j - \hat{w}_j)^2, \text{ for } \mid w_j - \hat{w}_j \mid \leq \delta \\ \delta \mid w_j - \hat{w}_j \mid - \frac{1}{2}\delta^2, \text{ otherwise }, \end{cases} \tag{15}$$

where $\mathbf{w}$ is the vector of target values, $\hat{\mathbf{w}}$ is a vector of the outputs from $f_{SL}(\mathbf{s})$, so that $\hat{\mathbf{w}} = [\boldsymbol{\mu}; \boldsymbol{\sigma}^2]$, M is the dimension of the thrust control input vector, j is the index of vectors $\hat{\mathbf{w}}$ and $\mathbf{w}$, and the parameter δ is set to a value of 1. This loss is minimized using the Adam optimizer [16] with a learning rate of 10^{-1}.

C. Reinforcement Learning

The reinforcement learning portion of this algorithm uses the network that results from supervised learning as an initial policy, and improves the policy through reinforcement learning by training the network using a 12D quadrotor dynamics model. Here we use the policy gradient theorem to approximate the gradient from roll-outs of the dynamics,

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{t_f} \nabla_\theta \log \pi_\theta(\mathbf{u}_{i,t} \mid \mathbf{s}_{i,t}) \times \sum_{t'=t}^{t_f} r(\mathbf{s}_{i,t'}, \mathbf{u}_{i,t'}), \tag{16}$$

where N is the number of trajectory roll-outs, and θ represents the parameters of the network. With this approximation of the gradient, we perform gradient ascent to maximize the objective

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta), \tag{17}$$

where α is the learning rate. In this problem, our objective is to pass through the gate and to do so quickly. Thus, we use the reward function,

$$r(\mathbf{s}(t)) = \begin{cases} +5, \text{ if } \mathbf{s}(t) \in \mathcal{T} \\ -5, \text{ if } \mathbf{s}(t) \in \mathcal{A} \\ -\Delta t, \text{ otherwise }, \end{cases} \tag{18}$$

so that a reward of $+5.0$ is obtained for passing through the gate and a reward of -5.0 is charged for hitting the gate at the terminal states, while a reward of $-\Delta t$ will be applied at every state in the trajectory so that the objective will minimize the time length of the trajectories.

The policy resulting from the supervised learning stage is able to bring the quadrotor through the gate, however, our goal is to see if the number of gates that the quadrotor could enter would increase after training with reinforcement learning, and if the time length of the trajectories could simultaneously be decreased. In our training process, we execute ten roll-outs in simulation for each batch. The control inputs used are drawn from a Gaussian distribution parameterized by the mean and variance output of the model. Each of these trajectories are generated by using a random initial state within a sub-region of the full state space. This specified region of the state space then grows when all ten trajectories in a batch reach the target set (or once a maximum number of iterations is reached), and continues to grow until the state space with bounds defined by the HJB problem is explored. A visualization of this procedure is shown in Fig. 6.

V. Low Level Control

The HJB-RL policy gives a desired 3D thrust control input for each discrete state in the state space. To align the body-z axis of the quadrotor with the desired thrust vector, a lower level controller must be used to obtain a desired angular rate of the quadrotor. In this work, proportional control will be used on the rotation error between the desired orientation and the current orientation. We express the attitude tracking error $\mathbf{e}_R$ as used in [18],

$$\mathbf{e}_R = \frac{1}{2}(\mathbf{R}_d^T\mathbf{R} - \mathbf{R}^T\mathbf{R}_d)^\vee, \tag{19}$$

where $(\cdot)^\vee$ is the vee operator, $\mathbf{R}_d$ is the desired thrust vector orientation and $\mathbf{R}$ is the current orientation. The desired body angular rates control input is then $\boldsymbol{\omega}_d = k_R\mathbf{e}_R$ where k_R is a proportional constant. In the AirSim simulation environment, $\boldsymbol{\omega}_d$ along with a normalized thrust magnitude are used as inputs to an AirSim controller API. In our 12D quadrotor simulation, we implement another lower level proportional controller to achieve the desired angular rates using the current body rates of the quadrotor.

VI. Simulation Experiments

In this section, we compare the performance of our policy to a model-based planning method that we refer to as Move-On-Spline (MOS), an HJB policy (HJB), a policy learned through reinforcement learning without any pretraining or priors (RL), and the network resulting from the supervised learning step described in this work (SL). The model-based planning method MOS is based on [29] and is made available in the AirSim environment as an API [24]. The HJB policy used in these tests will use a k-nearest neighbor interpolation between each discretized state point to find the appropriate thrust vector control input. The policy learned through reinforcement learning is trained using a the standard policy gradient method used in our work.

A. 12D Quadrotor Simulation

To test our method, we execute each policy from a set of initial states in the region in front of a single gate. 1000 of these trajectories of each policy were run, from a set of 1000 initial states. The thrust control inputs that were used in all simulation tests were the mean output of the model. Additionally, we consider how our approach varies

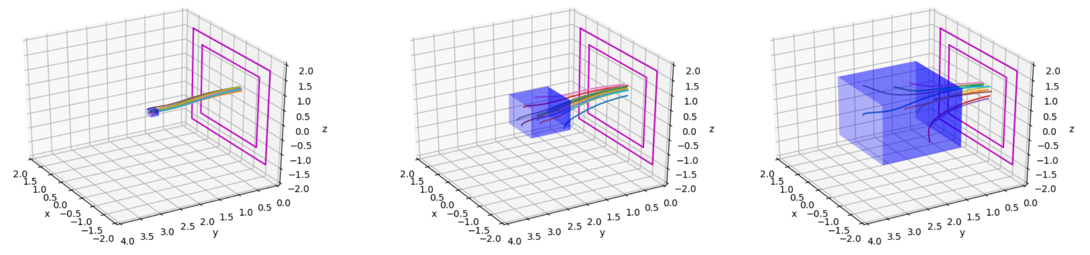
Fig. 6. Visualization of reinforcement learning training procedure. Region from which trajectories are initialized during training grows are trajectories become more successful. Blue cube marks the region within the state space from which trajectories are randomly initialized. Lines mark roll-outs of the quadrotor dynamics.

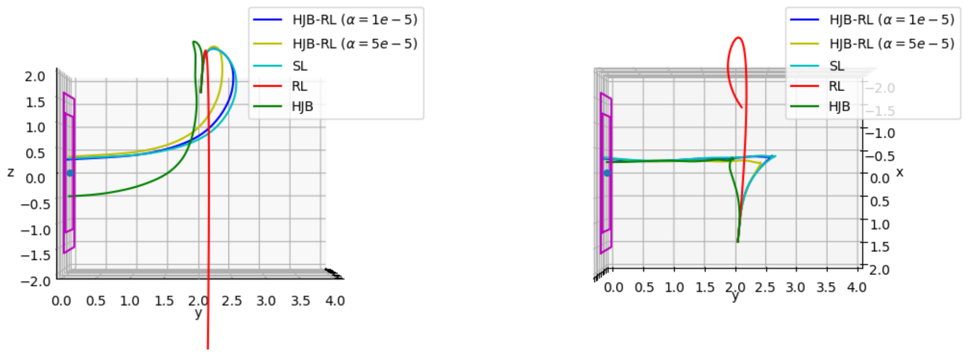

Fig. 7. Trajectories of different policies from same initial state. Comparing HJB-RL, Reinforcement Learning Only (RL), Supervised Learning (SL), HJB Only. The HJB-RL policy is able to get through the gate more quickly than the other policies. The HJB policy takes a safer path, while the RL policy is unable to make it through the gate. (Left) Side view of trajectories of different policies. (Right) Top view of different policies.

TABLE I
COMPARISON OF WINS BETWEEN HJB-RL AND OTHER METHODS IN 12D QUADROTOR SIMULATION

HJB-RL ($\alpha = 5e-5$) vs.	HJB-RL Faster/Success	Other Method Faster/Success	Avg. Time Difference to Gate (s)	Neither Finishes
Supervised Learning	62.48%	10.11%	0.0438	38.7%
HJB	56.03%	29.73%	0.2808	36.1%
Reinforcement Learning Only	80.91%	7.67%	0.0443	38.7%

TABLE II
PERCENTAGE OF INITIAL STATES FROM WHICH EACH POLICY PASSES THROUGH A SINGLE GATE IN 12D QUADROTOR SIMULATION

Method	Percentage of Gates Passed
HJB-RL ($\alpha = 1e-5$)	61.0%
HJB-RL ($\alpha = 5e-5$)	61.2%
Supervised Learning	61.2%
HJB	63.1%
Reinforcement Learning Only	12.4%

with different learning rates α. Fig. 7 shows a sample set of trajectories of each policy from one initial state. While the trajectories are not constrained to go through the center point of the gate, the resulting HJB-RL policy does produce trajectories that favor entering the gate closer to the center, likely as a result of training with a stochastic policy. Table I outlines the performance of the HJB-RL method in racing against other methods when the trajectories started from the same initial state. The percentages listed in the second column mark the frequency with which our method with a learning rate of $\alpha = 5e-5$ either entered the gate before the other method, or when our method entered the gate while the other did not, normalized by the number of races for which at least one policy did enter the target set. The third column indicates the same metric for the method that is listed in the first column. These columns do not sum to 100% because there were cases in which the methods tied. The fourth column lists the average time difference between the methods when the HJB-RL policy was faster, and the final column lists the number of trials from which neither of the methods are able to pass through the gate.

From the same set of 1000 initial states, we also compared the percentage of trials for which each method was able to pass through a single gate, shown in Table II. From this data, we see that there is a slight reduction in the number of gates that the HJB-RL policy is able to pass through with a learning rate of $\alpha = 1e-5$ compared to the supervised learning model and the HJB policy. The initial states from which each of these trajectories were executed were chosen randomly from velocity ranges $-5\text{m/s} \leq v_x \leq 5\text{m/s}$, $-15\text{m/s} \leq v_y \leq 5\text{m/s}$,$-5\text{m/s} \leq v_z \leq 5\text{m/s}$. These high velocity initial states

TABLE III
PERFORMANCE OF PLANNING METHODS IN AIRSIM SOCCER FIELD ENVIRONMENT

Method	Races Finished	Avg. and Standard Deviation Gates per Race	Avg. and Standard Deviation Race time (sec)	Avg. and Standard Deviation Collision Counts
HJB-RL ($\alpha = 5e-5$)	8	11.38 $\pm$0.52	30.36 $\pm$2.67	1.10 $\pm$1.79
HJB-RL ($\alpha = 1e-5$)	9	11.89 $\pm$0.33	34.89 $\pm$8.76	2.56 $\pm$3.50
Supervised Learning	10	11.70 $\pm$0.48	36.14 $\pm$13.32	2.90 $\pm$6.10
Move-On-Spline API	10	12.0	47.88 $\pm$7.39	3.60 $\pm$1.43
HJB	8	12.0	68.10 $\pm$26.74	9.38 $\pm$7.21
Reinforcement Learning Only	0	-	-	-

are likely the source of the lower percentages shown in Table II. However, as shown in Table I, the frequency with which the HJB-RL policy is able to pass through the gate faster than the other policies is higher.

B. Full AirSim Drone Race

In order to test the performance of this policy on a full race track, the same policies were tested on tracks with multiple gates along with a planning API called Move-On-Spline that is available in the AirSim environment [24], which is described to use [29] as the trajectory planning backend. Ten races of each method were conducted and the average performance of each method compared in Table III. The first performance metric we compare in this table is the number of races finished by each method out of ten. In some cases, the quadrotor would be unable to recover from collisions. We also compare the average number of gates the quadrotor is able to pass through out of the 12 on the race track. The collision counts only consider the number of collisions made with the gates, and do not consider collisions with other objects in the environment.

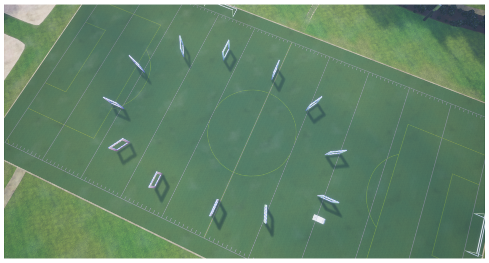
Fig. 8. Top view of Soccer Field Track in AirSim environment, where the HJB-RL policy is tested in simulation. At each oncoming gate, the HJB-RL policy is applied using the relative state of the quadrotor in a gate centered frame as input.

In the HJB-RL, HJB-Only, RL-Only, and the supervised learning methods, the policy is only defined in a specified domain ahead of each oncoming gate. In the regions between each gate, the controller used here switches to a position controller that holds the same speed of the quadrotor directed toward the next gate. Additionally, each of these methods are given the position of each next gate, only after it passes through the previous gate. For this reason, we test the Move-On-Spline API in the same way, so that the quadrotor only has access to the pose of the next oncoming gate. The results are shown in Table III.

From the results of the full track races and the single gate trajectories, we can see that the HJB-RL method seems to prioritize increasing speed over consistently entering gates. While the average race times for the HJB-RL policy that was trained with a learning rate of $5e-5$ was lower, the policy does so by missing a gate more frequently than the other methods when racing on a full track. This is likely attributed to applying reinforcement learning on a single gate as opposed to a sequence of gates. The drone will speed through each gate as fast as possible without considering how this speed will affect its ability to enter the following gate. In future work we will address this issue by training on a sequence of gates.

In Fig. 9 we show the trajectories for the best races of each method, and also give the race statistics of each. In all trajectories shown, all 12 gates are passed through. In some trajectories, the drone makes contact with the ground and recovers, however these collisions are not added to the collision counts as the policies do not account for obstacles other than the gate frames. The HJB-RL policy plotted in Fig. 9 was trained using a learning rate of $\alpha = 1e-5$.

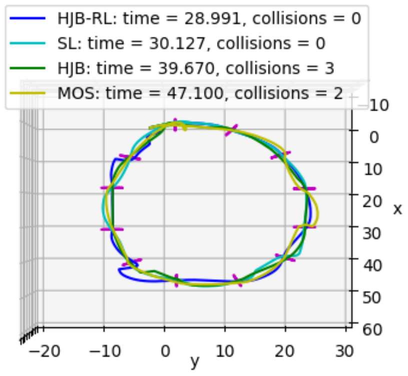

Fig. 9. Top view of trajectories with lowest collision count and time for each method. HJB-RL has the fastest best race with no gate collisions. The model-based methods HJB and MOS have longer race times with gate collisions.

C. Generalization to Different Gate Sizes

In this section, we also consider the generalizability of this method to race track cases where the gates along the track vary in size. Assuming that the gates are rectangular, and only their height and width change, we can scale the x, z, v_x, and v_z dimensions of the state space according to the gate size and apply the corresponding control input returned from the model to the system. In Fig. 10 we show a trajectory resulting from using this scaling method with the HJB-RL policy that was trained using a learning rate of $\alpha = 1e-5$.

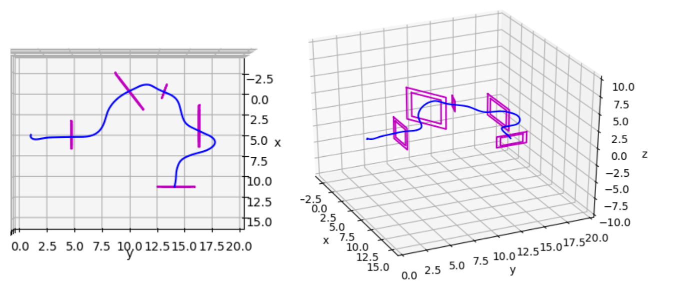

Fig. 10. Trajectory passing through gates of different sizes. (Left) Top view of trajectory for multiple gates of different sizes. (Right) Trajectory for multiple gates of different sizes.

VII. Conclusion

In this paper, we proposed a method of obtaining a policy that could leverage information from a model-based controller and the adaptability of a reinforcement learning policy. We show through simulation that this method is able to outrace a model-based HJB policy, a supervised learning policy, a model-free RL policy and a trajectory planning policy in a race track environment with 12 gates. We also note a reduction in the number of states from which the gate is able to successfully enter the gate, suggesting a priotization of speed over entering a gate. In future work we consider applying reinforcement learning to a sequence of gates so that the policy can anticipate how passing through gates affects the ability to pass through the following gates. Additionally, we plan to apply other variations of reinforcement learning methods to this framework, and extend the method to be applied to local coordinate frames of dynamic obstacles or to opponents in a race. Finally, this framework could be extended to cases where a perception module must be used to find the next gate along the track.

Acknowledgments

Toyota Research Institute ("TRI") provided funds to assist the authors with their research but this article solely reflects the opinions and conclusions of its authors and not TRI or any other Toyota entity. This work was also supported in part by the Stanford Ford Alliance, by DARPA grant HR001120C0107, and by ONR grant N00014-18-1-2830.

References

[1] Anurag Ajay, Jiajun Wu, Nima Fazeli, Maria Bauza, Leslie P. Kaelbling, Joshua B. Tenenbaum, and Alberto Rodriguez. Augmenting physical simulators with stochastic neural networks: Case study of planar pushing and bouncing. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3066–3073, 2018. doi: 10.1109/IROS.2018.8593995.

[2] Anil Aswani, Humberto Gonzalez, S Shankar Sastry, and Claire Tomlin. Provably safe and robust learning-based model predictive control. *Automatica*, 49(5):1216–1226, 2013.

[3] Somil Bansal, Mo Chen, Sylvia Herbert, and Claire J. Tomlin. Hamilton-jacobi reachability: A brief overview and recent advances. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 2242–2253, 2017. doi: 10.1109/CDC.2017.8263977.

[4] Jonathan Baxter and Peter L Bartlett. Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15:319–350, 2001.

[5] Marco A. Carmona, Alexey A. Munishkin, Megan Boivin, and Dejan Milutinović. Stochastic optimal approach to the steering of an autonomous vehicle through a sequence of roadways. In *2019 American Control Conference (ACC)*, pages 3279–3284, 2019. doi: 10.23919/ACC.2019.8814762.

[6] Jaime F. Fisac, Neil F. Lugovoy, Vicenç Rubies-Royo, Shromona Ghosh, and Claire J. Tomlin. Bridging hamilton-jacobi safety analysis and reinforcement learning. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8550–8556, 2019. doi: 10.1109/ICRA.2019.8794107.

[7] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870, 2018.

[8] Sylvia L. Herbert, Mo Chen, SooJean Han, Somil Bansal, Jaime F. Fisac, and Claire J. Tomlin. Fastrack: A modular framework for fast and guaranteed safe motion planning. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 1517–1522, 2017. doi: 10.1109/CDC.2017.8263867.

[9] Peter J Huber. Robust estimation of a location parameter. In *Breakthroughs in statistics*, pages 492–518. Springer, 1992.

[10] Jemin Hwangbo, Inkyu Sa, Roland Siegwart, and Marco Hutter. Control of a quadrotor with reinforcement learning. *IEEE Robotics and Automation Letters*, 2(4):2096–2103, 2017. doi: 10.1109/LRA.2017.2720851.

[11] Frank Jiang, Glen Chou, Mo Chen, and Claire J Tomlin. Using neural networks to compute approximate and guaranteed feasible hamilton-jacobi-bellman pde solutions. *arXiv preprint arXiv:1611.03158*, 2016.

[12] Sham M Kakade. A natural policy gradient. In *Advances in neural information processing systems*, pages 1531–1538, 2002.

[13] Elia Kaufmann, Antonio Loquercio, Rene Ranftl, Alexey Dosovitskiy, Vladlen Koltun, and Davide Scaramuzza. Deep drone racing: Learning agile flight in dynamic environments. In *Conference on Robot Learning*, pages 133–145. PMLR, 2018.

[14] Elia Kaufmann, Mathias Gehrig, Philipp Foehn, René Ranftl, Alexey Dosovitskiy, Vladlen Koltun, and Davide

Scaramuzza. Beauty and the beast: Optimal methods meet learning for drone racing. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 690–696, 2019. doi: 10.1109/ICRA.2019.8793631.

[15] H Jin Kim, David H Shim, and Shankar Sastry. Nonlinear model predictive tracking control for rotorcraft-based unmanned aerial vehicles. In *Proceedings of the 2002 American control conference (IEEE Cat. No. CH37301)*, volume 5, pages 3576–3581. IEEE, 2002.

[16] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[17] Nathan O. Lambert, Daniel S. Drew, Joseph Yaconelli, Sergey Levine, Roberto Calandra, and Kristofer S. J. Pister. Low-level control of a quadrotor with deep model-based reinforcement learning. *IEEE Robotics and Automation Letters*, 4(4):4224–4230, 2019. doi: 10.1109/LRA.2019.2930489.

[18] Taeyoung Lee, Melvin Leok, and N. Harris McClamroch. Geometric tracking control of a quadrotor uav on se(3). In *49th IEEE Conference on Decision and Control (CDC)*, pages 5420–5425, 2010. doi: 10.1109/CDC.2010.5717652.

[19] Karen Leung, Edward Schmerling, Mengxuan Zhang, Mo Chen, John Talbot, J Christian Gerdes, and Marco Pavone. On infusing reachability-based safety assurance within planning frameworks for human–robot vehicle interactions. *The International Journal of Robotics Research*, 39(10-11):1326–1345, 2020.

[20] Weiwei Li and Emanuel Todorov. Iterative linear quadratic regulator design for nonlinear biological movement systems. In *ICINCO (1)*, pages 222–229, 2004.

[21] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2019.

[22] Sergei Lupashin, Angela Schöllig, Michael Sherback, and Raffaello D'Andrea. A simple learning strategy for high-speed quadrocopter multi-flips. In *2010 IEEE International Conference on Robotics and Automation*, pages 1642–1648, 2010. doi: 10.1109/ROBOT.2010.5509452.

[23] Xubo Lyu and Mo Chen. Ttr-based reward for reinforcement learning with implicit model priors. *arXiv preprint arXiv:1903.09762*, 2019.

[24] Ratnesh Madaan, Nicholas Gyde, Sai Vemprala, Matthew Brown, Keiko Nagami, Tim Taubner, Eric Cristofalo, Davide Scaramuzza, Mac Schwager, and Ashish Kapoor. Airsim drone racing lab. In *NeurIPS 2019 Competition and Demonstration Track*, pages 177–191. PMLR, 2020.

[25] Daniel Mellinger and Vijay Kumar. Minimum snap trajectory generation and control for quadrotors. In *2011 IEEE International Conference on Robotics and Automation*, pages 2520–2525, 2011. doi: 10.1109/ICRA.2011.5980409.

[26] Daniel Mellinger, Nathan Michael, and Vijay Kumar. Trajectory generation and control for precise aggressive maneuvers with quadrotors. *The International Journal of Robotics Research*, 31(5):664–674, 2012. doi: 10.1177/0278364911434236.

[27] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[28] Jan Peters and Stefan Schaal. Reinforcement learning of motor skills with policy gradients. *Neural networks*, 21 (4):682–697, 2008.

[29] Charles Richter, Adam Bry, and Nicholas Roy. Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments. In *Robotics Research*, pages 649–666. Springer, 2016.

[30] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015.

[31] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International Conference on Machine Learning*, pages 387–395. PMLR, 2014.

[32] Riccardo Spica, Eric Cristofalo, Zijian Wang, Eduardo Montijano, and Mac Schwager. A real-time game theoretic planner for autonomous two-player drone racing. *IEEE Transactions on Robotics*, 36(5):1389–1403, 2020. doi: 10.1109/TRO.2020.2994881.

[33] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NIPs*, volume 99, pages 1057–1063. Citeseer, 1999.

[34] Yuanda Wang, Jia Sun, Haibo He, and Changyin Sun. Deterministic policy gradient with integral compensator for robust quadrotor control. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 50(10):3713–3725, 2020. doi: 10.1109/TSMC.2018.2884725.

[35] Zijian Wang, Tim Taubner, and Mac Schwager. Multi-agent sensitivity enhanced iterative best response: A real-time game theoretic planner for drone racing in 3d environments. *Robotics and Autonomous Systems*, 125: 103410, 2020.

[36] Andy Zeng, Shuran Song, Johnny Lee, Alberto Rodriguez, and Thomas Funkhouser. Tossingbot: Learning to throw arbitrary objects with residual physics. *IEEE Transactions on Robotics*, 36(4):1307–1319, 2020. doi: 10.1109/TRO.2020.2988642.

[37] Tianhao Zhang, Gregory Kahn, Sergey Levine, and Pieter Abbeel. Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 528–535, 2016. doi: 10.1109/ICRA.2016.7487175.