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

An Online Training Method for Augmenting MPC with Deep Reinforcement Learning

Guillaume Bellegarda and Katie Byl

Abstract— **Recent breakthroughs both in reinforcement learning and trajectory optimization have made significant advances towards real world robotic system deployment. Reinforcement learning (RL) can be applied to many problems without needing any modeling or intuition about the system, at the cost of high sample complexity and the inability to prove any metrics about the learned policies. Trajectory optimization (TO) on the other hand allows for stability and robustness analyses on generated motions and trajectories, but is only as good as the often over-simplified derived model, and may have prohibitively expensive computation times for real-time control, for example in contact rich environments. This paper seeks to combine the benefits from these two areas while mitigating their drawbacks by (1) decreasing RL sample complexity by using existing knowledge of the problem with real-time optimal control, and (2) allowing online policy deployment at any point in the training process by using the TO (MPC) as a baseline or worst-case scenario action, while continuously improving the combined learned-optimized policy with deep RL. This method is evaluated on tasks of successively navigating a car model to a series of goal destinations over slippery terrains as fast as possible, in which drifting will allow the system to more quickly change directions while maintaining high speeds.**

I. Introduction

Deep reinforcement learning (DRL) methods have shown recent success on continuous control tasks in robotics systems in simulation [1]–[3]. Such methods are applied using no prior knowledge of the systems, leading to problematic sample complexity and thus long training times. Unfortunately, little can be said about the stability or robustness of these resulting control policies, even if more traditional model-based optimal control solutions exist for these same systems.

Additionally, DRL has been almost exclusively applied in simulation, where a failed trial has no repercussions. In the real world a failure can have catastrophic consequences, including damaging the robot or causing injury to humans in the area. Some recent works have successfully learned a policy for a real robot [4] [5], or transferred policies learned in simulation to the real system [6] [7]. Of particular note in [7] is that the learned policies outperform the authors' previous model-based methods with respect to both energy-efficiency and speed. However, instead of training from scratch, it would seem intuitive to use the model-based methods as an initial starting point for DRL, with the reasoning that at any given moment, our learned policy should do *at least as well as* existing control solutions.

This work was funded in part by NSF NRI award 1526424.

Guillaume Bellegarda and Katie Byl are with the Robotics Laboratory, Department of Electrical and Computer Engineering, University of California at Santa Barbara (UCSB). gbellegarda@ucsb.edu, katiebyl@ucsb.edu

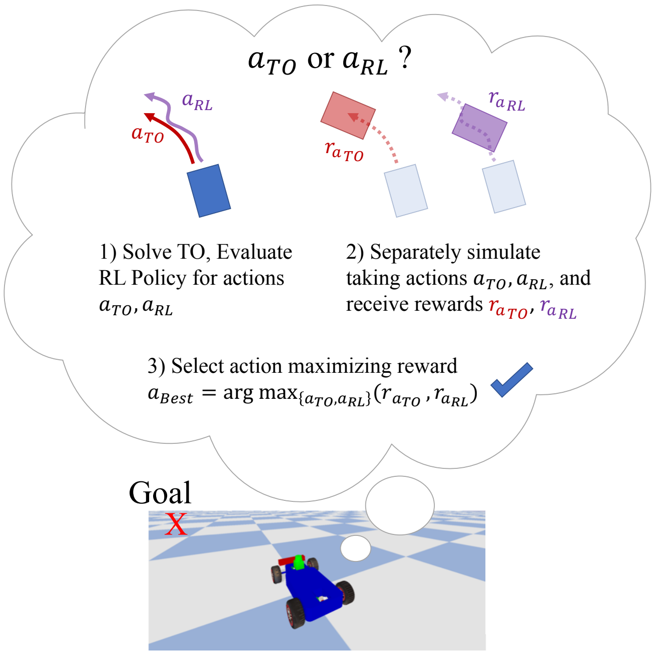
a_{TO} or a_{RL} ?
a_{RL}
a_{TO}
$r_{a_{TO}}$
$r_{a_{RL}}$
1) Solve TO, Evaluate RL Policy for actions a_{TO}, a_{RL}
2) Separately simulate taking actions a_{TO}, a_{RL}, and receive rewards $r_{a_{TO}}, r_{a_{RL}}$
3) Select action maximizing reward $a_{Best} = \arg\max_{\{a_{TO}, a_{RL}\}}(r_{a_{TO}}, r_{a_{RL}})$
Goal
X

Fig. 1: Agent picking between actions from trajectory optimization or from reinforcement learning, simulating taking each respectively, and then selecting the one leading to greater reward to execute in the real world.

One might be tempted to perform imitation learning updates on trajectories taken from running a model-based optimal control policy, for example using DAgger [8]. However, due to often mismatching dynamics between the simplified system on which this model-based control policy is based off of, and the real physical system, this may lead to overfitting a suboptimal policy. There is also the additional concern of deviating too much from the expert trajectories into regions of the state space not previously visited, in which the policy learned only from expert data may perform poorly.

Instead, we propose interweaving optimal control samples during the policy rollouts of model-free DRL methods in the following manner: at each timestep we can evaluate our policy network to get action a_{RL}, as well as query our trajectory optimization to get action a_{TO} (assuming the TO can be solved in real-time as MPC). We then simulate the execution of each of these actions individually, and select the one which gave the larger reward as our true action to use in the real world. Such a scheme, shown in Figure 1, should ensure that at worst the agent will always do as well as the model-based optimal control policy, and can only do better. At the beginning of training, we expect this approach to almost exclusively pick a_{TO}; due to the network weights being randomly initialized, it is very unlikely to consistently outperform a model-based method. However as training progresses, and from added policy exploration/exploitation, the number of selected on-policy samples will increase.

A. Related Work

Related work on using trajectory optimization to help learn or guide a control policy include [9]–[11]. These works differ from the proposed method in this paper as they focus more on incorporating offline demonstrations or trajectories into training to guide the policy search, whereas in this work the trajectory optimization is run online at each timestep and compared with the current policy action, ensuring a worst case scenario.

Other related works take model-based reinforcement learning approaches. These include learning value functions for MPC [12], [13], learning the dynamics used by MPC [14], or shaping the cost function used for MPC with offline hindsight [15]. While these works focus on improving MPC from a model-based perspective, in this paper we consider using model-based approaches to aid model-free reinforcement learning, the latter of which has been shown to outperform model-based methods. We note that it would be possible to combine our proposed method with any of these works replacing the more traditional MPC we use here, but this would force training to be done offline while learning the dynamics, value functions, and/or cost functions, and also not ensure a worst-case scenario action.

A related work combining prior knowledge of the system with learning is [16], where the policy chooses between actions computed with a simple PID controller and from evaluating the current actor network. Although the authors observe this controller helps achieve faster and more stable learning performance, it is not optimal and much can still be improved in terms of sample complexity. Additionally, there are no guarantees on the policy at any given time, and no minimum time to goal estimates or worst case scenarios.

B. Problem Statement and Contribution

We consider a motion planning scenario in which the objective is to determine an optimal control policy. Specifically, we assume that we have access to a (possibly suboptimal) trajectory optimization framework that can be run in real-time as MPC, but which may result in a suboptimal control policy. The goal is to use such a suboptimal policy as a worst-case scenario action at any time step, to be improved with deep reinforcement learning. This has the simultaneous benefit of avoiding training from scratch and greater robustness, essentially combining strengths from both trajectory optimization and reinforcement learning.

The motion planning task considered in this work is navigating a car as quickly as possible to a series of goal locations over terrains with a priori unknown coefficients of friction (i.e. slippery terrains). In order to maximize progress and velocity, it will be beneficial to use aggressive turning maneuvers to quickly turn directions between successive goal locations. Trajectory optimization frameworks such as [17]–[20] for planning through contact motions potentially involving skidding are computationally expensive, not lending themselves well for real-time planning and control, additionally so because of the terrain uncertainty. In particular, the contact dynamics in these works are formulated as Linear Complementarity Problems (LCPs), creating a nonlinear program with expensive solve times. Even if we ignore the computational complexity, and if we assume that basic terrain properties are known, there may be a mismatch between the simplified model dynamics used for planning and the real physical system when performing such through contact motions as in [20].

We will thus leverage MPC on a constrained dynamic bicycle model, which will conservatively find solutions that do not involve wheel slip, around which DRL will explore and exploit the computationally expensive and difficult to accurately model contact dynamics. This suboptimal MPC will however ensure a performance baseline and "worst-case scenario action" (whenever it may outperform our current policy network learned with DRL). Better performing actions will then be used to update the policy network, either through behavioral cloning for MPC actions, or through standard DRL updates (in this work through Proximal Policy Optimization [2]).

The rest of this paper is organized as follows: Section II provides background details on reinforcement learning, imitation learning as behavioral cloning, and robot dynamics in the context of a car model. Section III describes the trajectory optimization framework used to calculate conservative optimal trajectories for the car avoiding slip, and our algorithm combining this trajectory optimization with deep reinforcement learning (in this case PPO) is presented in Section IV. Section V shows results on the benefits of using our algorithm, and a brief conclusion is given in Section VI.

II. Preliminaries

A. Reinforcement Learning

The reinforcement learning framework, which is described thoroughly in [21] and elsewhere, typically consists of an agent interacting with an environment modeled as a Markov Decision Process (MDP). An MDP is given by a 4-tuple (S, A, T, R), where S is the set of states, A is the set of actions available to the agent, $T : S \times A \times S \to \mathbb{R}$ is the transition function, where $T(s, a, s')$ gives the probability of being in state s, taking action a, and ending up in state s', and $R : S \times A \times S \to \mathbb{R}$ is the reward function, where $R(s, a, s')$ gives the expected reward for being in state s, taking action a, and ending up in state s'. The goal of an agent is thus to interact with the environment by selecting actions that will maximize future rewards.

In this paper, the states consist of a subset of a robot's positions and velocities, the actions are motor torques or positions, the transition function is modeled by a physics engine [22], and the reward is a potential-based function to minimize distance to a target goal.

B. Proximal Policy Optimization

Although we expect to see benefits from combining trajectory optimization with any deep reinforcement learning algorithm, for this paper we use the current state-of-the-art, Proximal Policy Optimization (PPO) [2]. In particular, PPO has achieved breakthrough results for continuous control robotics tasks by optimizing the following surrogate objective with clipped probability ratio:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t] \quad (1)$$

where $\hat{A}_t$ is an estimator of the advantage function at time step t as in [23], and $r_t(\theta)$ denotes the probability ratio

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \quad (2)$$

where π_θ is a stochastic policy, and θ_{old} is the vector of policy parameters before the update. This objective seeks to penalize too large of a policy update, which means penalizing deviations of $r_t(\theta)$ from 1.

As PPO is an actor-critic algorithm, the full objective function combines terms for this policy surrogate with a value function error term, as well as an entropy bonus to ensure sufficient exploration [2], [24], which we more succinctly define as $L^{PPO}(\theta)$ as

$$\begin{aligned} L^{PPO}(\theta) &:= L_t^{CLIP+VF+S}(\theta) \\ &= \hat{\mathbb{E}}_t\left[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)\right] \end{aligned} \quad (3)$$

where c_1, c_2 are hyperparameters, S denotes an entropy bonus, and $L_t^{VF}(\theta)$ is a squared-error loss:

$$L_t^{VF}(\theta) = \left(V_\theta(s_t) - V_t^{targ}\right)^2 \quad (4)$$

C. Learning from Demonstration

In this work we use the classical behavioral cloning (BC) approach to imitation learning where we seek to minimize the error between an expert action and the maximum likelihood estimate action from the current policy:

$$L^{BC}(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left(a_i^* - \arg\max_{a_i} \pi_\theta(a_i|s_i)\right)^2 \quad (5)$$

for expert demonstration state-action pairs $\{s_i, a_i^*\}_{i=1}^N$, where $\arg\max_{a_i} \pi_\theta(a_i|s_i)$ is the maximum likelihood estimate action a_i for state s_i using policy π_θ.

D. Robot Dynamics

The equations of motion for a robotic system can be written as:

$$D(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) + A(q)^T\lambda = B(q)u + F \quad (6)$$

where q are the generalized coordinates, $D(q)$ is the inertial matrix, $C(q,\dot{q})\dot{q}$ denotes centrifugal and Coriolis forces, $G(q)$ captures potentials (gravity), $A(q)^T\lambda$ are constraint forces (where λ are unknown multipliers a priori), $B(q)$ maps control inputs u into generalized forces, and F contains non-conservative forces such as friction.

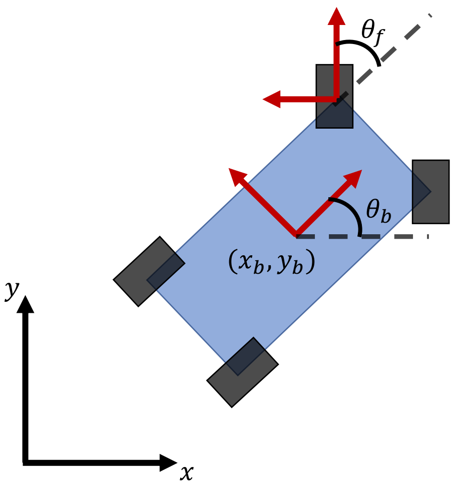

Fig. 2: Car model used for trajectory optimization.

In this work, we specifically consider a simple car model, shown in Figure 2, with $q = [x_b, y_b, \theta_b, \theta_f]^T$, where (x_b, y_b) are the center of mass coordinates in the world frame, θ_b is the yaw of the body with respect to the global x-axis, and θ_f is the steering angle of the front wheels.

For general wheeled mobile robots, $A(q)^T\lambda$ typically contains constraints ensuring no slip (free rolling in the direction the wheel is pointing) and no skid (no velocity along the wheel's rotation axis perpendicular to the free rolling direction), which come from writing these constraints in Pfaffian form $A(q)\dot{q} = 0$. λ can be explicitly solved for by differentiating $A(q)\dot{q} = 0$ and substituting in $\ddot{q}$ from Equation 6.

1) Remark: Because this constraint is in place, the trajectory optimization will not find solutions where skidding is a viable option allowing for greater reward (such as skidding into a parking space instead of parallel parking, or an aggressive turning maneuver to more quickly change directions). The alternative to this constraint would be to add friction approximations such as in [25], for which the optimization must then solve a Linear Complementarity Problem at each contact point. As this can be a very expensive computation, we avoid this consideration and instead entrust the DRL algorithm to use the trajectory optimization as a guide towards learning a better policy, in which slip may or may not be optimal.

III. Trajectory Optimization and MPC

This section provides details for formulating the locomotion problem for a robotic system as a trajectory optimization. This optimization can be solved in real-time as MPC due to the simplified dynamics model with nonholonomic constraints, producing conservative optimal trajectories that do not involve wheel slip. At a high level, the full nonlinear system is discretized, and direct collocation along with backward Euler integration is used to generate motion as in [18]–[20]. More precisely, the problem is formulated as:

$$\begin{aligned}
\text{find} \quad & q, \dot{q}, u \text{ at discrete timesteps } k = 1...N && (7)\\
\text{s.t.} \quad & \text{minimize cost } J \\
& \text{-State Constraints} \\
& \phi(q,\dot{q},u) = 0, \;\; \psi(q,\dot{q},u) \geq 0 && (8)\\
& \text{-Dynamics Constraints} \\
& D(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) + A(q)^T\lambda = B(q)u + F && (9)
\end{aligned}$$

where each of the above constraints are detailed below, along with cost function considerations.

A. Objectives

The cost function J is defined as the weighted squared error between the goal coordinates (x_g, y_g, θ_g) and the body coordinates (x_N, y_N, θ_N), where N is the number of sample points for the trajectory:

$$J = \alpha(x_g - x_N)^2 + \beta(y_g - y_N)^2 + \gamma(\theta_g - \theta_N)^2 \qquad (10)$$

where weights α, β, γ can vary based on the desired task, i.e. if final body orientation is important.

B. State Constraints

The initial states q_0 and $\dot{q}_0$ are constrained exactly based on the robot's current state. For the rest of the N time points, q and $\dot{q}$ are bounded by joint position and velocity limits. The input torques u are also bounded explicitly by the physical constraints of the robot, as well as implicitly by $\dot{q}$ ranges.

C. Dynamics Constraints

At each time step k, with $h = \Delta t$ the time step interval, the dynamics are constrained:

$$q_{k+1} = q_k + h\dot{q}_{k+1} \qquad (11)$$

$$\dot{q}_{k+1} = \dot{q}_k + h\ddot{q}_{k+1} \qquad (12)$$

with

$$\begin{aligned}
\ddot{q}_{k+1} = D_{k+1}^{-1}(& B_{k+1}u_{k+1} + F_{k+1} - C_{k+1}\dot{q}_{k+1} \\
& - G_{k+1} - A_{k+1}^T\lambda_{k+1}) \qquad (13)
\end{aligned}$$

where we write $D(q_{k+1})$ as D_{k+1}, and similar for other terms.

IV. Cooperative Trajectory Optimization and Deep Reinforcement Learning

In this section we detail our algorithm, Cooperative Trajectory Optimization and PPO (CoTO-PPO), shown in Algorithm 1. The main idea is that for each new observation s_t at time step t, the current PPO actor network is queried for action a_{RL}, and a trajectory optimization is solved for action a_{TO}. Each of these actions is simulated individually to get rewards $r_{a_{RL}k+1}$ and $r_{a_{TO}k+1}$. The action that produces the larger simulated reward is the one that is selected as the true best action and used in the real world (or to step the actual simulation). Necessary transition information is then appended to either the PPO dataset D_{PPO} or Supervised Learning (SL) dataset D_{SL}, depending on which action was selected. After T time steps corresponding to the current policy/trajectory optimization roll out, the actor-critic PPO networks are updated by optimizing $L^{PPO}(\theta)$ on dataset D_{PPO}, and the actor network is additionally updated with supervised learning by optimizing $L^{BC}(\theta)$ on dataset D_{SL}.

Algorithm 1: Cooperative Trajectory Optimization and PPO (CoTO-PPO)

```
Initialize function approximation parameters $\theta$
Initialize PPO and Supervised Learning datasets
  $D_{PPO}, D_{SL}$
for training epoch=1,2,... do
    for timestep=1,2,...T do
        Solve trajectory optimization for action $a_{TO}$
        Evaluate PPO policy network for
          $a_{RL} \sim \pi_\theta(a_t, s_t)$
        Simulate taking each action separately and
          select action maximizing next step reward:
            $a_{Best} = \underset{(a_{TO}, a_{RL})}{\arg\max}(r_{a_{RL}k+1}, r_{a_{TO}k+1})$
        Step environment with $a_{Best}$:
          $s_{t+1} \sim p(s_{t+1}|s_t, a_{Best})$
        if $a_{Best} == a_{RL}$ then
            $D_k \sim$ partial trajectory, transition
              information
            $D_{PPO} = D_{PPO} \cup D_k$
        else if $a_{Best} == a_{TO}$ then
            $D_{SL} = D_{SL} \cup \{(s_t, a_{TO})\}$
        end
    end
    for K epochs on $D_{PPO}$ do
        normal PPO updates by optimizing $L^{PPO}(\theta)$
    end
    for K epochs on $D_{SL}$ do
        supervised learning LfD updates by optimizing
          $L^{BC}(\theta)$
    end
end
```

V. Results

A. Implementation Details

We use a combination of OpenAi Gym [26] to represent the MDP and PyBullet [22] as the physics engine for training and simulation purposes.

We additionally use the OpenAI Baselines [27] implementation of PPO (which optimizes $L^{PPO}(\theta)$ as discussed in Sec. II-B) with the default hyperparameters, but with the Beta distribution to select continuous actions as suggested in [28] to avoid the bias introduced with limited control ranges when using the standard Gaussian distribution. The Gym environment we use is similar to the standard HumanoidFlagRun environments, but the humanoid is replaced with the Bullet MIT racecar. Example snapshots from the environment are shown in Figure 3. The goal destination/flag is updated only when the car center of mass lies within 0.2 [m] of the current goal location, and then placed randomly in a 10 by 10 [m]

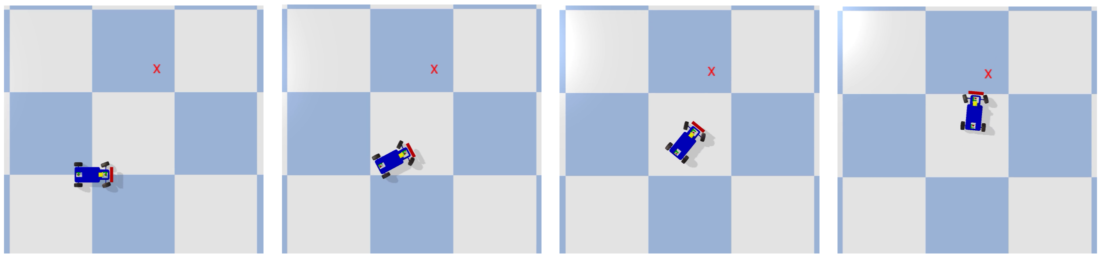

Fig. 3: Environment snapshots of Bullet racecar with desired goal denoted by the red ‘X’.

grid. The agent has 10 seconds to maximize rewards each trial, which will typically consist of reaching several goal locations consecutively.

Our neural network architecture is the default Multi-Layer Perceptron, consisting of 2 fully connected hidden layers of 64 neurons each, with tanh activation. The policy and value networks each have this same structure.

The observation space is:

$$[\ ||(x_g, y_g) - (x_b, y_b)||,\ \theta_g - \theta_b,\ \theta_f,\ \dot{x}_b,\ \dot{y}_b,\ \dot{\theta}_b,\ \dot{\theta}_f\ ] \quad (14)$$

which is the distance from the center of mass to the goal location, the angle from the current body heading to the goal, the steering angle of the front wheels, the body velocity in the global x and y directions, the body yaw rate, and the yaw rate of the steering wheels.

The action space is $[\ v,\ \theta_f\ ]$, which is a desired body velocity to be set with velocity control mapped to a differential drive, and desired steering angle to be set with position control.

We consider potential-based shaping reward functions of the form:

$$F(s,a,s') = \gamma\Phi(s') - \Phi(s) \quad (15)$$

to guarantee consistency with the optimal policy, as proved by Ng et. al in [29]. The real-valued function $\Phi : S \rightarrow \mathbb{R}$ seeks to minimize the distance to a target goal (x_g, y_g):

$$\Phi(s) = -\sqrt{(x_b - x_g)^2 + (y_b - y_g)^2} \quad (16)$$

This reward scheme gives dense rewards at each time step on the full simulated system, towards ensuring the optimal policy is learned, rewarding incremental motion in the direction of the current goal. Having dense rewards is important in this framework as we are choosing between actions based on the simulated instantaneous reward, which would likely be 0 at most time steps under sparse reward scenarios.

The trajectory optimization is implemented in Python with CasADi [30], using IPOPT [31] to solve the NLP. Due to the imposed torque and velocity limits as well as the non-holonomic constraints added to the dynamics, the trajectory optimization will find solutions that are suboptimal to the true best policy. We will see in the following subsection that even this suboptimal trajectory optimization has a large benefit when combined with policies either learned from scratch or with our method, both during training and testing.

B. Experiments

We seek to compare and evaluate the following methods:

1) *pure PPO*
2) *pure trajectory optimization*
3) *CoTO-PPO* - (our method)
4) *CoTO-PPO, policy only* - how well does the policy learned from CoTO-PPO perform on its own, without the fail-safe action of the trajectory optimization?
5) *CoTO-(pure PPO)* - how well does combining trajectory optimization with an entirely separately trained agent with PPO perform?

Figure 4 shows the episode reward mean vs. number of training time steps for running pure PPO as well as CoTO-PPO on the CarFlagRun environment. The episode reward mean indicates how well the agent was able to continue to progress in the direction of the goal location(s) during each trial. Due to doing at worst as well as the trajectory optimization, CoTO-PPO begins with very high reward mean, and only improves from there as the networks are updated with both DRL and supervised learning updates. Pure PPO on the other hand is forced to learn from scratch, and even after training for 1 million time steps, is only able to do as well as the trajectory optimization combined with essentially uniformly distributed random noise of the uninitialized policy from CoTO-PPO.

Figure 5 shows the percentage of samples picked by the policy network of PPO in CoTO-PPO that outperform the actions from the trajectory optimization over training. As expected, when the policies are randomly initialized, few samples from PPO will outperform even a suboptimal trajectory optimization. Eventually as training progresses, the percentage of maximal reward samples picked with PPO converges to around 75% of the time. This shows there is still a benefit to using the trajectory optimization as a worst case scenario, as it is still being picked 25% of the time after 1 million training time steps.

Table I details the reward mean and percentage of PPO actions picked (if relevant) with various algorithms and scenarios across 100 trials, after training for 1 million time steps. We do trials of evaluating each policy by sampling from the output Beta distributions stochastically, as well as deterministically evaluating the distributions with the maximum likelihood estimate. We see that combining the trajectory optimization with PPO significantly increases the mean reward, with our method CoTO-PPO having the best

	Reward Mean		Percent PPO Actions	
Algorithm	Stochastic	Deterministic	Stochastic	Deterministic
pure PPO	12.9	13.8	-	-
Trajectory Optimization	-	12.1	-	-
CoTO-PPO	15.1	**15.7**	75	**81**
CoTO-PPO, policy only	14.1	14.7	-	-
CoTO-(pure PPO)	14.5	14.5	44	57

TABLE I: Episode reward mean and percent of samples chosen with PPO for different algorithms across 100 trials, using either stochastic or deterministic (maximum likelihood) actions from the output distributions of the policy network. The percent of PPO actions are only listed for algorithms which choose between both DRL and TO actions, and the trajectory optimization action is always evaluated deterministically.

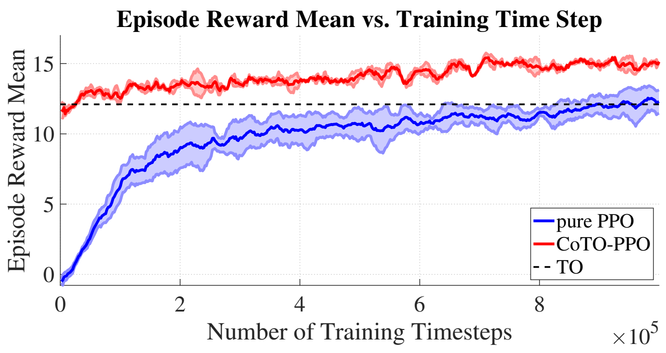

Fig. 4: Episode reward mean for pure PPO, and cooperative trajectory optimization and PPO (CoTO-PPO). The episode reward mean from using only the trajectory optimization is plotted as a dashed line.

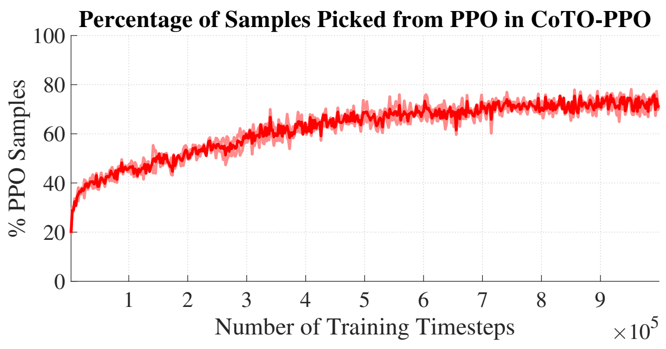

Fig. 5: At the beginning of training, about 20% of the random actions selected by PPO produce larger rewards than the optimal actions found with the trajectory optimization. Over time the policy is guided towards better regions and roughly 75% of the samples selected from PPO result in maximal reward. As a fail safe, the TO action is taken to keep our bounds on time to goal.

performance. If we evaluate only the policy trained from CoTO-PPO, it is still a significant improvement over the policy trained from pure PPO alone. We also evaluate the effect of combining the policy trained with pure PPO with the trajectory optimization, labeled CoTO-(pure PPO), which makes it clear that pure PPO has learned a suboptimal policy, as the combination with the TO leads to a higher reward mean.

In this latter case, we also track what percent of the time CoTO-(pure PPO) picks the TO action vs. the action selected from the policy network of pure PPO. Despite training for 1 million time steps, our algorithm finds that the trajectory optimization performs better than the policy trained from scratch roughly half of the time. In comparison, the policy trained from our method CoTO-PPO is picked over 75% of the time, despite far fewer on-policy samples (due to using the TO samples for supervised updates).

C. *Maximum Instantaneous Reward Discussion*

A first look at the algorithm may seem to imply that it is greedy, rather than optimal, as the agent selects the action leading to the maximum instantaneous reward, rather than a function of expected returns. We experimented with simulating taking multiple actions from the TO and from RL over varying horizons, but found significantly worse performance with this method. One plausible explanation for this result is that when first initialized, the PPO actor network is essentially taking random actions. As the horizon h increases on which we simulate taking h actions from TO and RL separately, the expected returns of taking a series of random actions regresses toward 0 under our reward scheme, and thus the probability that RL will outperform the TO tends to 0. Said another way, it becomes increasingly unlikely to have multiple "lucky actions" from RL in a row during exploration as the horizon h increases. Since the agent will correspondingly almost always choose the actions from the suboptimal TO, the policy network will almost always be updated with supervised learning updates and will correspondingly converge, approximately, to this same suboptimal policy.

On the other hand, by using the maximum instantaneous transition reward from a horizon of 1, there is a much higher probability of sampling a "good" action from an uninitialized random policy. This allows for more efficient exploration of the environment than by overwhelmingly following the (suboptimal) trajectory optimization actions, leading to a better overall policy (such as more aggressive throttle values, steering angles during turns, etc.), while still ensuring a reasonable worst case scenario action from the more dynamically conservative TO solution, for cases in which we sample a worse-performing action with RL.

The short horizon also avoids overfitting to the suboptimal trajectory optimization expert.

VI. Conclusion

In this work we have shown the benefits of combining trajectory optimization and deep reinforcement learning methods into one training process. Using these two methods cooperatively allows for online use of our algorithm at any point in the training process, knowing that the worst case scenario will be as good as a model-based trajectory optimization. This additionally leads to much greater sample efficiency, and avoids unnecessary exploration of randomly initialized policies, towards avoiding local optima. There is also the advantage of leaving the difficult to model (and computationally expensive to optimize over) contact dynamics to be learned, rather than bias a trajectory optimization with potentially inaccurate dynamics, while at the same time using prior knowledge to structure the motion planning task.

Even if the trajectory optimization is suboptimal due to mismatching dynamics or overly conservative constraints, there is a clear advantage to incorporating prior knowledge of the system to speed up and guide learning. We also observe that trained policies, whether exclusively learned with deep reinforcement learning or from our combined method, are likely to converge to local optima and cannot exhaustively span all observation states, showing the benefit of model-based methods as a proven fail-safe option. The need to be able to put bounds on learned policies and guarantee some sort of behavior is clear, and this work presents preliminary steps in this direction.

The method detailed in this paper can be readily applied to any robotic system, and should be an effective way to reduce sampling complexity, accelerate training, guide the policy search, deploy policies online at any point in the training process, and give an upper bound estimate on time-to-goal through the trajectory optimization.

References

[1] N. Heess, D. TB, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. M. A. Eslami, M. A. Riedmiller, and D. Silver, "Emergence of locomotion behaviours in rich environments," *CoRR*, vol. abs/1707.02286, 2017.

[2] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017.

[3] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *CoRR*, vol. abs/1509.02971, 2015.

[4] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 3389–3396.

[5] T. Haarnoja, A. Zhou, S. Ha, J. Tan, G. Tucker, and S. Levine, "Learning to walk via deep reinforcement learning," *CoRR*, vol. abs/1812.11103, 2018.

[6] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," *CoRR*, vol. abs/1804.10332, 2018.

[7] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning agile and dynamic motor skills for legged robots," *Science Robotics*, vol. 4, no. 26, 2019.

[8] S. Ross, G. J. Gordon, and J. A. Bagnell, "No-regret reductions for imitation learning and structured prediction," *CoRR*, vol. abs/1011.0686, 2010.

[9] I. Mordatch and E. Todorov, "Combining the benefits of function approximation and trajectory optimization," in *In Robotics: Science and Systems (RSS)*, 2014.

[10] S. Levine and V. Koltun, "Variational policy search via trajectory optimization," in *Advances in Neural Information Processing Systems*, 2013, pp. 207–215.

[11] ——, "Guided policy search," in *International Conference on Machine Learning*, 2013, pp. 1–9.

[12] K. Lowrey, A. Rajeswaran, S. Kakade, E. Todorov, and I. Mordatch, "Plan online, learn offline: Efficient learning and exploration via model-based control," in *International Conference on Learning Representations*, 2019.

[13] M. Zhong, M. Johnson, Y. Tassa, T. Erez, and E. Todorov, "Value function approximation and model predictive control," in *2013 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, 2013, pp. 100–107.

[14] 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)*, 2017, pp. 1714–1721.

[15] A. Tamar, G. Thomas, T. Zhang, S. Levine, and P. Abbeel, "Learning from the hindsight plan – episodic mpc improvement," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 336–343.

[16] L. Xie, S. Wang, S. Rosa, A. Markham, and N. Trigoni, "Learning with training wheels: Speeding up training with a simple controller for deep reinforcement learning," *CoRR*, vol. abs/1812.05027, 2018.

[17] S. Feng, E. Whitman, X. Xinjilefu, and C. G. Atkeson, "Optimization-based full body control for the DARPA Robotics Challenge," *Journal of Field Robotics*, vol. 32, no. 2, pp. 293–312, 2015.

[18] M. Posa and R. Tedrake, "Direct trajectory optimization of rigid body dynamical systems through contact," in *Algorithmic foundations of robotics X*. Springer, 2013, pp. 527–542.

[19] G. Bellegarda and K. Byl, "Trajectory optimization for a wheel-legged system for dynamic maneuvers that allow for wheel slip," in *2019 IEEE 58th Conference on Decision and Control (CDC)*, 2019, pp. 7776–7781.

[20] G. Bellegarda and K. Byl, "Versatile trajectory optimization for dynamic vehicle maneuvers," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 7905–7911.

[21] R. S. Sutton and A. G. Barto, *Reinforcement learning - an introduction*, ser. Adaptive computation and machine learning. MIT Press, 1998.

[22] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," http://pybullet.org, 2016–2019.

[23] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *CoRR*, vol. abs/1506.02438, 2015.

[24] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," *CoRR*, vol. abs/1602.01783, 2016.

[25] D. E. Stewart and J. C. Trinkle, "An implicit time-stepping scheme for rigid body dynamics with Coulomb friction," in *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2000, pp. 162–169.

[26] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[27] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Openai baselines," https://github.com/openai/baselines, 2017.

[28] P.-W. Chou, D. Maturana, and S. Scherer, "Improving stochastic policy gradients in continuous control with deep reinforcement learning using the beta distribution," in *International Conference on Machine Learning*, 2017, pp. 834–843.

[29] A. Y. Ng, D. Harada, and S. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *ICML*, vol. 99, 1999, pp. 278–287.

[30] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – A software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.

[31] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical programming*, vol. 106, no. 1, pp. 25–57, 2006.