Paper Implimentation

Policy Search for Model Predictive Control with Application to Agile Drone Flight

Shahab Nikkhoo Xiaoao Song

Abstract— **We have implemented the framework proposed in the paper [6], which combines policy search and model predictive control (MPC) for robot control. The authors leverage policy search to select high-level decision variables for MPC, resulting in a self-supervised optimization of policies. The framework's validation focuses on agile drone flight, particularly navigating a quadrotor through fast-moving gates. Our experimental results demonstrate robust and real-time control performance, both in simulation and real-world scenarios. This integration of learning and control offers a new perspective and holds potential for future advancements in the field.**

I. INTRODUCTION

Mobile robots, particularly agile quadrotors, operate in dynamic and complex environments, enabling them to access areas that are otherwise inaccessible to humans. However, sudden environmental changes, such as dynamic obstacles, pose challenges for vehicle control, requiring fast reactions and rapid trajectory replanning. In agile drone flight, adapting the vehicle trajectory swiftly in response to environmental changes is crucial. Model predictive control (MPC) has emerged as a robust model-based approach for quadrotor control in both static and dynamic environments [1], [3]. It simultaneously handles complex nonlinear dynamic systems while satisfying various constraints. However, MPC faces challenges in terms of accurate modeling, online trajectory optimization, and sub-optimal solutions due to design choices.

On the other hand, reinforcement learning (RL) methods, like policy search, offer a way to solve continuous control problems with minimal prior knowledge [4], [2]. RL maximizes task performance through trial and error, but the lack of interpretability is a concern in the control community. Combining the strengths of model-based controllers like MPC and RL's ability to learn complex policies using experience data would be ideal. This integration could handle large-scale inputs, reduce human-in-the-loop design, and achieve adaptive and optimal control. However, designing such a system is a significant challenge.

Research in the learning community focuses on developing data-efficient policy search methods using model priors or learning-based MPC that leverage real-world data. However, black-box control policies and computational expense remain issues. The paper we are reimplementing aims to address these challenges by proposing a framework that combines intelligent decision-making approaches with model predictive control. It formulates the search for high-level decision variables as a probabilistic policy search problem, allowing the learning of a neural network high-level policy. This policy adaptively selects decision variables for MPC, resulting in a High-MPC framework. The approach's effectiveness is demonstrated through the challenging task of controlling a quadrotor to navigate a fast-swinging gate.

In summary, the paper aims to integrate intelligent decision-making with model predictive control, providing a solution for agile drone flight in dynamic environments. The combination of policy search and MPC holds promise for robust and adaptive control performance, offering advancements in autonomous robot control.

II. METHODOLOGY

In this section, we describe the methodology presented in the paper to address and solve the aforementioned problem. We provide a simplified explanation of the proposed method and its implementation. Additionally, we explore further ideas and conduct experiments to assess the integrity of the approach and examine corner cases. These additional experiments serve as valuable insights for future research and potential improvements to the proposed method.

A. Paper Methodology

1) Quadrotor and Gate Dynamics: The dynamics of a quadrotor, which is controlled by four motors, can be described by the following set of equations:

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

$$\dot{\mathbf{q}}_{WB} = \frac{1}{2}\Lambda\left(\omega_B\right) \cdot \mathbf{q}_{WB} \tag{2}$$

$$\dot{\mathbf{v}}_{WB} = \mathbf{q}_{WB} \odot \mathbf{c} - \mathbf{g} \tag{3}$$

$$\dot{\omega}_B = \mathbf{J}^{-1}\left(\eta - \omega_B \times \mathbf{J}\omega_B\right) \tag{4}$$

The position and velocity vectors of the quadrotor in the world frame are denoted by $\mathbf{p}^q_{WB}$ and $\mathbf{v}^q_{WB}$ respectively. The orientation of the quadrotor is represented by a unit quaternion $\mathbf{q}_{WB}$. The body rates, which correspond to the roll, pitch, and yaw, are given by ω_B. The quadrotor's motion is governed by equations that involve the mass-normalized thrust vector $\mathbf{c}$, the gravity vector $\mathbf{g}$, the inertia matrix $\mathbf{J}$, and the three-dimensional torque η. These equations describe the dynamics of the quadrotor and provide a comprehensive representation of its state $\mathbf{x}^q$, which includes position, orientation, and velocity.

The authors of the study also present a model for the dynamic gate, approximating it as a pendulum consisting of a point mass suspended by a weightless and inextensible string from a fixed support. The motion of the pendulum is influenced by various factors including gravity, tension in the string, and damping caused by friction and air resistance. The authors simplify the representation of damping by considering it as a linear function of the pendulum's angular velocity. The rotational dynamics of the pendulum are described by a set of differential equations that take into account the roll angle and moment of inertia. The translational motion equations incorporate the gate's position and the distance between the gate's center and the fixed point. The gate's orientation is represented using a quaternion. To capture the complete state of the gate, including its position, orientation, and velocity, a state vector is employed.

2) Policy Search: In the paper, the author utilizes episodic policy search to learn a high-level policy, symbolized as π_θ, which is independent of the robot's observations. The aim is to optimize this policy in order to maximize the expected reward of predicted trajectories, denoted by τ. This is achieved through the application of a weighted maximum likelihood algorithm. In this context, maximizing the reward is equivalent to amplifying the probability of a binary "event", denoted as $p_\theta(E|\tau)$. The problem of maximization is resolved by employing a weighted maximum likelihood estimation of π_θ. Here, each trajectory sample, $\tau[i]$, is assigned a weight determined by the probability $p(E|\tau)$. An exponential transformation is applied, thereby converting the reward signal $R(\tau[i])$ of each sampled trajectory $\tau[i]$ into a probability distribution. This distribution is defined as $d[i] = \exp(\beta R(\tau[i]))$, where $\beta \in R^+$ signifies the inverse temperature of the softmax distribution. It should be noted that a higher β value corresponds to a more greedy policy update.

The paper presents three methods of policy serach but only examine two of them, here we explain those two:
Gaussian Policy with Constant Mean: The study begins by addressing the simpler problem of learning a Gaussian policy, $\pi_\theta(z|\mu,\Sigma)$, with an unknown vector as the mean. During each training iteration, They sample a list of parameters from the current policy distribution, π_θ, and evaluate them using a reward function, $R(\tau)$, with $\tau[i]$ being trajectories predicted by the Model Predictive Control using the sampled variables $z[i]$.

The reward signal, $R(\tau)$, with help of an exponential transformation is transformed into a non-negative weight, $d[i]$, in the Expectation step. In the Maximization step, policy parameters are updated by optimizing a weighted maximum likelihood objective:

$$\theta^* = \arg\max_\theta \left\{ \sum_i d^{[i]} \log \pi_\theta \left(z^{[i]} \right) \right\} \tag{5}$$

Then the the mean and the covariance, can be updated using a closed-form expressions:

$$\mu = \left(\sum_{i=1}^{N} d^{[i]} \mathbf{z}^{[i]} \right) / \left(\sum_{i=1}^{N} d^{[i]} \right) \tag{6}$$

$$\Sigma = \left(\sum_{i=1}^{N} d^{[i]} \left(\mathbf{z}^{[i]} - \mu \right) \left(\mathbf{z}^{[i]} - \mu \right)^T \right) / Y \tag{7}$$

$$Y = \left(\left(\sum_{i=1}^{N} d^{[i]} \right)^2 - \sum_{i=1}^{N} \left(d^{[i]} \right)^2 \right) / \left(\sum_{i=1}^{N} d^{[i]} \right) \tag{8}$$

This process continues until the expectation of the sampled reward stabilizes. Following training (in the policy evaluation phase), the mean vector of the Gaussian policy is chosen as the optimal decision variables for the Model Predictive Control. As such, the optimal MPC decision variables, as identified by the policy search, is $z = \mu^*$. The algorithm is showed in algorithm 1.

Algorithm 1 Learning Gaussian Policies for MPC [6]

```
Require: $\pi_\theta(\mu, \Sigma), N, MPC, \mathbf{x}_0, \mathbf{p}$
1: while not converged do
2:     Sample variables: $z^{[i]} \sim \pi_\theta(\mu, \Sigma)_{i=1\ldots N}$
3:     Sample trajectories: $\tau^{[i]} = MPCsolve\left(\mathbf{x}_0, z^{[i]}, \mathbf{p}\right)$
4:     Expectation:
5:     $d^{[i]} = \exp\left\{\beta R\left(\tau^{[i]}\right)\right\}$
6:     Maximization:
7:     $\theta^* = \arg\max_\theta \left\{\sum_i d^{[i]} \log \pi_\theta\left(z^{[i]}\right)\right\}$
8: end while
9: return  Trained Policy $\pi_{\theta^*}\left(\mu^*, \Sigma^*\right)$
```

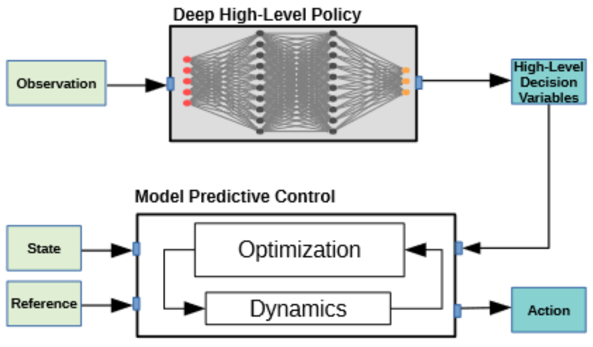

Fig. 1. Overview of the method for real-time adaptations of model predictive control via a trained deep high-level policy.

Learning Neural Network Policies:
The policy presented in Algorithm 1 is adept at learning a Gaussian policy optimized for specific scenarios or experiments. Additionally, the author develops a sophisticated neural network policy capable of adjusting decision variables and processing high-dimensional observations. This policy allows the robot to adapt its behavior effectively in a dynamic environment.

This advanced neural network policy is trained using a combination of Algorithm 1 and supervised learning, resulting in Algorithm 2. The learning process is divided into two stages: **data collection** and **policy learning**.

In the **data collection** stage, the robot starts at a state x_t and determines the optimal decision variables z_t using Algorithm 1. The dataset is then expanded by incorporating pairs of the current robot observation o_t and z_t. An optimal control action sequence u_t is computed via the MPC optimization, given x_t and z_t. The first control command is executed by the robot, which then moves to the next state.

Each simulation step, Algorithm 1 is utilized to solve multiple MPC optimizations in order to determine the optimal decision variable for the current state. This step essentially represents online learning within the simulator, which presents challenges for real-time execution.

During the **policy learning** stage, the neural network is trained to minimize the mean squared error between labels z^*t and the network output $f\Phi(o_t)$, using standard stochastic gradient descent. Following the training, the neural network policy is deployed in conjunction with the MPC for vehicle control.

Algorithm 2 Learning Neural Network Policies for MPC

```
Require: $f_\theta, \mathcal{D} = \{\}$
 1: Data collection (repeat)
 2: Randomly reset the system: $\mathbf{x}_t, \mathbf{o}_t, \mathbf{p}_t, t = 0$
 3: while not done do
 4:     $(\mathbf{z}_t = \mu^*) \leftarrow$ Algorithm 1 $(\mathbf{x}_0 = \mathbf{x}_t, \mathbf{p}_t)$
 5:     Data collection: $\mathcal{D} \leftarrow \mathcal{D} \cup \{\mathbf{o}_t, \mathbf{z}_t\}$
 6:     MPC optimization: $\mathbf{u}_t^* = MPC.solve^* (\mathbf{x}_t, \mathbf{z}_t, \mathbf{p}_t)$
 7:     System transition: $\mathbf{x}_{t+1} \leftarrow f(\mathbf{x}_t, \mathbf{u}_t^*)$
 8: end while
 9: Policy learning
10: $\theta_{new} = \arg\min_\theta \| f_\theta(\mathbf{o}_t) - \mathbf{z}_t \|^2$
11: return Learned deep high-level policy $f_{\theta^*}$
```

3) Model Predictive Control: The study aims to address the challenge of programming a drone to navigate through dynamic gates. The goal is to pinpoint a trajectory that enables the drone to pass directly through the center of the rapidly moving gates. This task of trajectory optimization consists of two intertwined aspects: 1) deciding the sequence of moments at which the drone should traverse the dynamic gates, and 2) given these traversal moments, finding the trajectory that facilitates passage through these gates.

In this scenario, it is assumed that there exists a vector of desired traversal times, denoted as $t = [t1, ..., ti]$ for each gate i, where $0 < t_i < t_j$ if $i < j$ and i, j are part of the set $[1, ..., n]$. In this context, n corresponds to the total number of moving gates. With the current states and dynamic model of the moving gates known, it's possible to predict the future trajectory $\tau_i = [x_{g_0}, ..., x_{g_{ti}}]$ for each gate i, where x_g is a state vector composed of position p_g, linear velocity v_g, and orientation q_g. Therefore, a gate-pass cost $L_{gate-pass}$ is defined as a specific quadratic cost.

$$\mathcal{L}_{gate-pass} = \sum_{h=1}^{H-1} \left(\mathrm{x}_h^q - \mathbf{x}_{t_i}^g\right)^T \mathbf{Q}_p \left(\mathbf{x}_h^q - \mathrm{x}_{t_i}^g\right) \cdot p_h, \quad (9)$$

where $\mathbf{Q}_p$ is a diagonal cost matrix and p_h a Boolean variable which is 1 if and only if $h = \lfloor t_i/d_t \rfloor$. Then they define a gate follow cost and a terminal cost $\mathcal{L}_{terminal}$ and an action regularization cost $\mathcal{L}_\mathrm{u}$

$$\mathcal{L}_{gate-follow} = \sum_{h=1}^{H-1} \left(\mathbf{x}_h^q - \mathbf{x}_{l_i}^g\right)^T \mathbf{Q}_f \left(\mathbf{x}_h^q - \mathbf{x}_{t_i}^g\right) \cdot w_h \cdot (1 - p_h) \quad (10)$$

where $w_h = \exp\left(-\alpha \cdot (h \cdot d_t - t_i)^2\right) \cdot \gamma_i$.

$$\mathcal{L}_{terminal} = \left(\mathrm{x}_H^q - \mathrm{x}^{goal}\right)^T \mathbf{Q}_{goal} \left(\mathbf{x}_H^q - \mathrm{x}^{goal}\right) \quad (11)$$

$$\mathcal{L}_\mathrm{u} = \sum_{h=1}^{H-1} \left(\mathbf{u}_h^q - \mathbf{u}_r\right)^T \mathbf{Q}_\mathrm{u} \left(\mathbf{u}_h^q - \mathbf{u}_r\right) \quad (12)$$

In their $\mathcal{L}terminal$ cost function, they consider x^{goal} as a target state for hovering. Additionally, in their $\mathcal{L}$u cost function, $\mathbf{u}_r = [g_z, 0, 0, 0]$ is used. They employ body-rate control $\mathbf{u}_h = [c, \omega_x, \omega_y, \omega_z]$ as the inputs. This terminal cost is designed to motivate the quadrotor to navigate towards a desired goal state, denoted by x^{goal}.

and the following is the optimization problem:

$$\min_\tau \mathcal{L}(\mathbf{x}_1, \mathbf{z}, \mathbf{r}) = \mathcal{L}_{terminal} + \mathcal{L}_{gate-pass} + \mathcal{L}_{gate-follow} + \mathcal{L}_\mathrm{u} \quad (13)$$

$$s.t.: \quad \mathbf{u}_{\min} \leq \mathbf{u} \leq \mathbf{u}_{\max}$$

$$\mathbf{x}_{h+1} = \mathbf{x}_h + d_t \cdot \hat{f}(\mathbf{x}_h, \mathbf{u}_h), \quad \mathbf{x}_1 = \mathbf{x}_{init}$$

In this model, τ represents the generated trajectory comprises the state vector $x_q = [p_q, q_q, v_q]$ and the control inputs u. They also define a vector of high-level decision variables $z = [t, \gamma]$ that includes the desired traversal time t and the desired weights γ for each moving gate. Given these decision variables z, the quadrotor's current state x_0, and the predicted future trajectories r of all moving gates, they are solving the trajectory optimization problem and determine the optimal trajectory for the quadrotor to navigate through all moving gates.

B. Our Modifications

1) Code Implementation: Although the code for this project was provided by the author, it appeared not to be the most recent version as it was non-executable. We modified it to make it runnable. Our contributions are as follows:

- **Creating the Shared Object (.so) file:** The authors used CasADi as an optimizer to solve the objective function. CasADi generates a .C file from the high-level mathematical models. In our case, these functions had to be compiled in a 64-bit architecture.

- **Refining the MPC Loss Equation:** The provided code is based on the first paper [5]. This paper introduces additional terms for the loss function to maintain the drone as close as possible to the center of the gate, which was not implemented.
- **Rectifying Dimension Inconsistency:** We resolved issues in matrix calculations due to inconsistent dimensions. We improved the simulation calculations by omitting the incorrect matrices.
- **Creating a Smaller Dataset:** We generated a smaller dataset from 5000 random samples and used this to train the Multi-Layer Perceptron (MLP).
- **Adding an Animation Saver:** The original MPC did not include animations, so we added this feature.

C. Discussion

In this section, we discuss aspects of the paper [6] that can be improved or questioned.

1) More Dynamic Environment: The paper's key focus is addressing the issue of pathing through a dynamic environment. However, their assumption of a single type of movement could be seen as limiting. We added gates with a higher degree of freedom, specifically making the gate act as a pendulum in the x-z plane. Moving the gate in the x-axis adds more variance to the traversal time, increasing the variance in Equation 7. The model has not been trained to deal with such cases, dramatically decreasing the success rate of the mission. Owing to testing limitations, we cannot quantify the decrease. Possibly, adding more layers to the MLP could help estimate more complex dynamics. However, this might result in decreased smoothness in velocity, which is the primary objective of this paper.

2) Other than Smoothness: In the context of drones, other factors beyond smoothness are relevant, such as battery life, robustness, safety, and reliability. The focus of this paper is on smoothness, but we observed that in some cases, the drone behaves unrealistically to follow the pendulum's (gate's) pattern. Such aggressive behavior can lead to higher battery consumption and safety issues. To tackle this, we could add another loss function to the MPC to minimize these situations.

3) Replacing Parameter w_h: The parameter w_h in the paper ensures that the drone mimics the movement of the gate when it's close, thereby achieving smooth velocity. The problem here is that there are other hyperparameters in function w_h that require manual tuning. Instead, we could train another ML model to solve this issue and determine the best policy for estimating the optimal gate traversal time.

4) Real-world Noise: In real-world scenarios, dealing with noise is a significant challenge apart from solving the problem. As we need an exact model to solve the issue, building the model based on real-world noise can cause serious problems. We added Gaussian noise with a mean of 0 and a variance of 0.1 to the state of the gate. Our experiment showed that the approach in this paper is not robust in the face of noise, resulting in a decrease in the success rate.

III. Conclusion

In our work, we attempted to replicate the study presented in paper [6]which the authors proposed an innovative concept of transforming the process of creating sophisticated decision variables in Model Predictive Control (MPC) into a probabilistic inference issue solvable via policy search. They successfully amalgamated self-supervised learning with this policy search method to cultivate an advanced neural network policy. Upon finalizing the training, this policy was capable of adaptively making online decisions for the MPC. The effectiveness of their method was showcased through the integration of a trained Multilayer Perceptron (MLP) policy with an MPC to tackle a challenging control problem - guiding a quadrotor to navigate through the center. Further, we aimed to explore this approach in detail by examining specific corner cases that pose challenges to the method outlined in the paper.

References

[1] Falanga, D., Foehn, P., Lu, P., and Scaramuzza, D. Pampc: Perception-aware model predictive control for quadrotors, 2018.

[2] Kober, J., and Peters, J. Policy search for motor primitives in robotics. In *Advances in Neural Information Processing Systems* (2008), D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, Eds., vol. 21, Curran Associates, Inc.

[3] Neunert, M., de Crousaz, C., Furrer, F., Kamel, M., Farshidian, F., Siegwart, R., and Buchli, J. Fast nonlinear model predictive control for unified trajectory optimization and tracking. In *2016 IEEE International Conference on Robotics and Automation (ICRA)* (2016), pp. 1398–1404.

[4] Peters, J., and Schaal, S. Reinforcement learning by reward-weighted regression for operational space control. In *Proceedings of the 24th International Conference on Machine Learning* (New York, NY, USA, 2007), ICML '07, Association for Computing Machinery, p. 745–750.

[5] Song, Y., and Scaramuzza, D. Learning high-level policies for model predictive control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2020).

[6] Song, Y., and Scaramuzza, D. Policy search for model predictive control with application to agile drone flight. *IEEE Transactions on Robotics* (2022), 1–17.