MODEL PREDICTIVE PATH INTEGRAL CONTROL FOR LEARNED DYNAMICS OF QUADCOPTERS

ANDREW GARRETT [GEANDE@SEAS], RAMYA MUTHUKRISHNAN [RAMYAMU@SEAS], PHILIP SIEG [PHILSIEG@SEAS],

ABSTRACT. In our final project, we implement a model predictive path integral (MPPI) control algorithm that uses a neural network dynamics model for a CrazyFlie quadcopter to operate in a physics simulator. We train the neural network to predict the quadcopter accelerations at each time sample in the simulated dataset of drone dynamics, before deploying the neural network as a dynamics model in the trajectory rollout step of the MPPI control algorithm. The controller is able to successfully complete navigation tasks and closely track desired trajectories. We compare our controller to a similar MPPI controller that uses an analytical dynamics model, which models the exact physics dynamics of the system. We demonstrate that our MPPI controller's performance approaches that of this controller. A direct future extension of this work is to demonstrate that for an environment with physical phenomena like downwash and drag, a neural network can encapsulate these complex, non-linear dynamics using a purely data-driven approach. This model would lead to increased controller performance over a naive analytical dynamics model.

1. INTRODUCTION

In robotics, control algorithms manipulate robot actuators to get robots to successfully perform desired tasks. Many of these control problems can be viewed as reinforcement learning (RL) problems. In the RL problem setup, the robot aims to maximize a utility (reward) function over the data gathered by the system; this function quantifies the successful completion of the task. Despite many recent advances in RL techniques, poor generalization of RL algorithms in practice continues to hinder progress and deployment of RL-based control algorithms. To be successful, a robot operating in a complex, evolving environment must be able to quickly make informed decisions. Model predictive control (MPC) helps the generalization of RL methods via online optimization of a cost function. However, a primary issue with traditional MPC is that it depends on approximating the cost function as a convex function so that traditional constrained optimization tools can be applied. To address this issue, model predictive path integral (MPPI) control is a sampling-based MPC approach that can optimize for a general cost, even if it is non-convex. MPPI can be used on systems without control affine dynamics, allowing us to pair it with arbitrary system dynamics models, including those that are learned, data-driven neural networks. Since accuracy of the dynamics model directly impacts the performance of the controller, neural networks' ability to represent complex, non-linear system dynamics is very powerful. It allows us to represent relationships that would be difficult (or impossible) to model otherwise with traditional, heuristic methods. Quadcopters are highly maneuverable but relatively simple, making them great candidates for unmanned aerial tasks through tight, complicated environments. Additionally, they are underactuated systems subject to highly non-linear dynamics. Their dynamics are affected by complex forces due to phenomena like ground effect and downwash. Combined, this provides a compelling example for the outlined framework.

1.1. Contributions.

In our final project, we develop learned dynamics models and pair them with an MPPI framework to carry out navigation tasks for a CrazyFlie drone in simulation. The work presented is integrated into gym-pybullet-drones [1], a simulation environment for quadcopters built from openAI's GYM. It also utilizes Pybullet, a physics simulator for robots in Python. Building on prior work, we further examine the different performance between learned models and analytical models for dynamics. We show that the controller using the learned model approaches the performance of the controller using an analytical model that knows the exact physical system dynamics, demonstrating the power of using a neural network to model dynamics for MPPI control. Additionally, we provide insight into cost function development and hyper parameter tuning for drones using MPPI.

2. BACKGROUND

The drone used in this project is a CrazyFlie 2 and we adopt the "X" coordinate frame convention. This means that the rotor arms are offset by 45° from the body frame's x and y axes (creating an "X" shape when the top view is aligned with axes). This is an increasingly popular convention as it allows for a camera along one of the body-frame axes with a less obstructed view. Additionally, we use the convention where the z-axis of the drone's body frame is pointed

upwards, in the direction of the motors' thrust. The physical drone and our body frame coordinate representation can be seen in figure 1. Our control inputs will be the four motor speeds in RPM, which can easily be translated to forces and moments via the following equations: $T_i = K_F\omega_i^2,\ M_i = K_M\omega_i^2$ where K_F and K_M are motor specific constants.

The kinematics of our problem is trivial given the velocities, and the challenge lies in calculating the accelerations (dynamics). The state $x = (q,\ \dot{q})$ is composed of linear position, angular orientation, linear velocity, and angular velocity. We wish to find a function $f(x_t,\ u_t)$ so that we can write the state transition as follows

$$x_{t+1} = \begin{bmatrix} q_t + \dot{q}_t\Delta t \\ \dot{q}_t + f(x_t,\ u_t)\Delta t \end{bmatrix} \tag{1}$$

With regard to representing the state, we use the following conventions to interface with the simulator. The linear position and velocity are expressed in the world frame. The orientation is described via Euler Angles using an "XYZ" convention (roll, pitch, yaw). The angular velocities are also expressed in the world frame. It is often useful to convert vectors between these two frames, so we can utilize the rotation matrix, ${}^W R_b$, which when post-multiplied by a vector in the body frame coordinates transforms the coordinates to be in the world frame. To generate this matrix, we can either manually or with a package like *scipy.spatial.transform* take the orientation Euler angles and convert them to rotation matrix representation.

To readers less fluent in drone aerodynamics, we would like to briefly introduce some of the more complicated forces imparted on a drone during flight. Generally speaking, these are difficult to model, which is why a data-driven approach is appealing. Drag is the force that opposes the direction of motion as an object traverses through a fluid. Even with an analytical model, in practice drag coefficients are determined via flight or wind tunnel data. Ground effect occurs when the drone is flying closely above a large surface (i.e. the ground) and the air pushed down by the rotors has nowhere to go. As a result, pressure builds up below the drone and increases the lift force [2]. Downwash can cause unsteady flows as the rotors pass air below the drone. This can lead to problems when vertically descending since the drone is entering the disturbed air, leading to a reduction in lift generation capabilities and stability [3].

In order to use our control algorithms we must have a task we would like to drone to carry out. We define a desired task (such as a figure 8 loop) as a sequence of waypoints through which the drone must travel, each represented as xyz coordinates in the 3D world coordinate system. We can then use a trajectory generator (constant velocity, minimum snap, etc.) to create a desired trajectory, which contains a corresponding state at each time step. We define the navigation task as manipulating drone controls such that the drone tracks this desired trajectory, or sequence of states, as closely as possible.

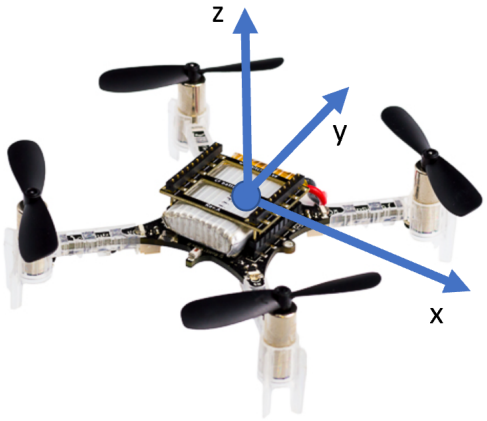


FIGURE 1. Local Coordinate Frame For Crazyflie

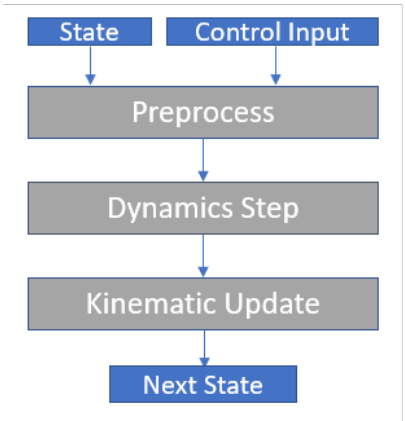


FIGURE 2. Block Diagram for Simulating Dynamics

3. RELATED WORK

MPPI was created by leveraging existing path integral control techniques and taking an MPC based, on-the-fly approach. In [4] the authors derive this sampling-based approach which relies on relative entropy minimization. They

show that for a physical, small-form auto rally vehicle, the controller outperforms its competition. Unlike a hierarchical controller, it is able to push the vehicle to its limits without much knowledge of the environment, and unlike traditional optimal control algorithms, it can perform computations in real-time. The authors further developed their algorithm in [5] by deriving the update law without making a control affine assumption, enabling a purely data-driven dynamics model, such as a neural network, to be integrated into the MPPI pipeline. Further work in [6] applies MPPI to a drone with onboard sensing to perform obstacle avoidance in cluttered environments.

The authors in [7] give a broad overview of data-driven approaches for system identification of quad-copters. Their work shows that multiple multi-layer perceptron (MLP) architectures have been successful in characterizing drone dynamics. They note that one should be cautious of creating a network too large since it can more easily lead to overfitting. They also discuss how RBF networks offer promising alternatives to standard MLP setups. Work in [8] shows that even small, shallow neural networks can successfully learn quadrotor dynamics and generalize to many cases, not just for specific types of trajectories. They also confirm that these neural networks can perform well when leveraged in control algorithms like MPPI. In this project, we similarly train a neural network to learn drone dynamics, and we extend this prior work by novelly incorporating this data-driven dynamics model into a sampling-based MPPI controller. We detail our approach in the following section.

4. Approach

4.1. **Data Collection and Bootstrapping.** To collect the data used to develop our learned dynamics model, we generate a bootstrap dataset with a drone simulator provided by the gym-pybullet-drones Python package [1]. We create 100 episodes, each with varying duration, of 27 drones following a desired flat trajectory using a PID controller provided by gym-pybullet-drones to select drone controls. Drone states and controls are sampled at a frequency of 48 Hz, and for each sample, we calculate the drone acceleration (to be described in the following section) and record it as a state-control-acceleration tuple in the bootstrapped dataset. Furthermore, in order to train the neural network to learn the system dynamics in an accurate, generalizable manner, the sampled trajectories must have significant variation. Thus, we vary the trajectory task (straight lines and figure-8 loops), drone speed, height of the flat trajectory, and the radius of curvature (for figure-8 trajectories) across different episodes to introduce variation into the bootstrapped dataset. **A video illustrating some examples from the data collection process can be found here.**

We generate two different datasets that use two different physics simulators to model drone dynamics: (1) a simple, explicitly defined dynamics model (found in the BaseAviary class in the gym-pybullet-drones package) and (2) a PyBullet physics simulator that can model more complex systems (using bullet physics too apply forces and torques to rigid bodies). We develop (1) as a proof-of-concept to compare MPPI controller performance between various dynamics models, and (2) as an example of a more complex dynamical system that can be learned by a neural network. For the remainder of the paper, we will refer to (1) as the explicit physics dataset and (2) as the PyBullet physics dataset.

4.2. **Dynamics Model.** On a high level, all of the models follow the structure shown in Figure 2. Taking the current state and control commands, they then pre-process the data, calculate the accelerations, and then apply them in a kinematic update step (as shown in equation 1) to generate the state at the next time-step. As a baseline, we implemented an analytical, first principles model similar to what is outlined in [9]. This model assumes that the only forces acting on the body are the four thrusts from the motors and gravity. It also assumes that only moments come from either forces or torques applied to the motor. Using the conservation of linear and angular momentum, we can derive equations that map the current state and control input into accelerations.

For our learned model, we trim the input state by removing the linear position, since it has no effect on accelerations. We also represent each Euler angle in the input state by its sine and cosine. To normalize the control inputs, we divide by the maximum allowable RPM value, specified by the drone manufacturer. We further normalize all state inputs and acceleration outputs to the range [-1, 1] using min-max normalization on the training dataset. For our model architecture, we use a fully connected network with two hidden layers, each with hyperbolic tangent (tanh) activation functions and batch normalization. We do not use dropout in our model architecture. We experimented with multiple layer sizes, but the final model has 64 nodes in each hidden layer. The objective function used to train the network is the mean squared error between the predicted and ground-truth acceleration values. As is done in [5], the ground-truth acceleration labels a_t are calculated with the following formula:

$$a_t = (\dot{q_{t+1}} - \dot{q}_t)/\Delta t \tag{2}$$

The network is trained with an RMSProp optimizer, a learning rate of 0.001, a weight decay of 0.0001, and a batch size of 64. Each batch consists of randomly selected, independent time steps across different generated episodes/trajectories in the training dataset. We train the neural network on the explicit physics dataset.

4.3. **MPPI.** We use the same MPPI algorithm to choose control inputs as the one described in [5]. For our drone problem, the state-dependent runtime cost $q(x)$ can be written as shown below, where C is an indicator that is enabled if the drone crashes and x_d represents the desired state at the time of evaluation.

$$q(x) = (x - x_d)^T Q (x - x_d) + 10000C \tag{3}$$

The terminal cost function is the same as the run-time state-dependent cost function.

We evaluate our control algorithm on desired trajectories generated from the same tasks used to create the bootstrapped dataset, namely following a straight linear trajectory to a stationary goal point and following a curved, figure-8 trajectory. On the simulator using the explicit physics, we evaluate: (1) the MPPI controller using an analytical dynamics model (the exact dynamics used in the simulation), and (2) the MPPI controller using the neural network dynamics model trained on the explicit physics dataset. On the simulator using the PyBullet physics, we only evaluate (1). We do this because we had trouble getting MPPI to work on the neural network trained on the PyBullet physics dataset (we consider potential ideas for improvement in the discussion).

To evaluate each control algorithm on each task, we run the simulation with the controller for 100 episodes, following both straight line and figure-8 desired trajectories. To compare tracked and desired trajectories, which are sampled at the same timestamps, we use the following metrics: (1) positional error and (2) rotational error. The positional error is the Euclidean distance between the desired and current at each timestamp, and the rotational error is the magnitude of the rotation representing the tracked drone orientation (stored as Euler angles) composed with the inverse of the rotation representing the desired drone orientation. Both metrics are averaged across all timestamps and all simulated trajectories.

5. Experimental Results

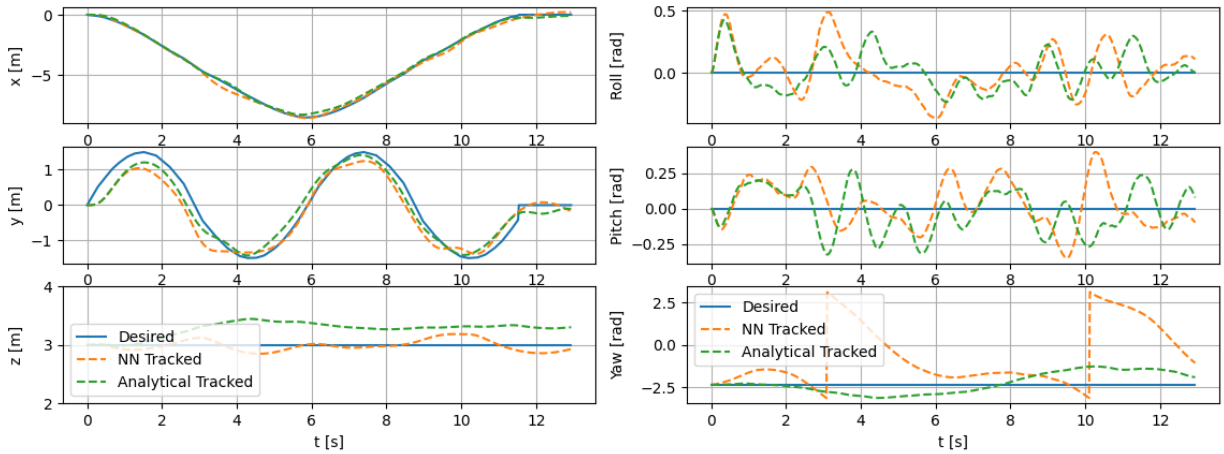
Using the cost function, error metrics, and test cases outlined above, we tested our models in simulation to gauge their performance. Table 1 summarizes the results with a gain of $Q = \text{Diag}(125, 125, 1000, 50, 50, 50, 5, 5, 5, 0, 0, 0)$ for the three primary evaluation setups discussed in the previous section. These results indicate that the MPPI controller using the neural network dynamics model approaches the performance of the controller using the analytical dynamics model when tested on the explicit physics simulator, since their error bars overlap. This demonstrates the power of using deep learning to learn a dynamics model given simulation data. The results also show that the MPPI controller using the analytical model is able to generalize to the PyBullet physics simulator. **Here is a link to a folder containing videos of the drone in simulation during testing.**

Physics Environment	Explicit		PyBullet
Model Type	Analytical	Neural Network	Analytical
Positional error	0.3249 ± 0.1260	0.6054 ± 0.2880	0.3846 ± 0.1154
Orientation error	0.8816 ± 0.4037	1.4891 ± 0.3841	0.9436 ± 0.4279

TABLE 1. MPPI Evaluation Results Across Models

To supplement the error metric, Figure 3 shows the difference in how the MPPI controller is able to track the same desired figure-eight trajectory using different dynamics models. The qualitative performance is quite similar, and in both cases, the desired x-y positions as well as roll and pitch are tracked closely. One key difference is that the wrapping of Euler angles causes problems for the learned model. We will revisit this issue in the discussion.

It is important to note that the tuning of the gains directly impacts the results reported. Depending on the task at hand, it may make sense to tune the controller gains differently to achieve the desired performance. When using the gains reported above, the drone flies in a very controlled manner, staying close to the desired velocities and orientation throughout the entire flight. This comes at the cost of not tracking the positional components of the desired trajectory as well. This is useful in a situation where stability and orientation are a priority. One could think of a scenario where a camera is mounted underneath the drone to capture data, and slight deviations in the course are not crucial because the FOV of the camera can still capture the scene of interest.



(A) Position Tracking

(B) Orientation Tracking

FIGURE 3. Comparison of Analytical and Learned Model using Explicit Physics for an Ellipse

If orientation and steady velocities are less important to the user, one could use other gains (for example $Q = \text{Diag}(25, 25, 25, 0, 0, 0, 0, 0, 0, 0, 0, 0)$). This would cause the controller to follow the positional portion of the trajectory almost exactly, in a more aggressive manner. A natural application of this is quickly needing to maneuver through tight spaces or checkpoints. **Here is a link to videos showing the drone behavior with the updated gains.** One thing we observe is that there is much more overshoot, and the drone often finds itself ahead of the trajectory and must wait. There are also larger, more rapid changes in orientation, however overall, it as able to stay closer to some point in the desired XYZ desired path throughout the flight.

6. DISCUSSION

In this project, we have shown that a neural network can accurately learn the dynamics of a simple drone system. Therefore, it can thus be successfully used in an MPPI controller; the performance lost by replacing the actual, explicit dynamics model with the neural network in the MPPI control algorithm is small. In doing so have also fully integrated our own dynamics models and MPPI controller into the Py-bullet framework and built tools for data visualization. A link to our repository can be found here.

In the short term, we would like to improve performance of the MPPI controller trained on the PyBullet physics dataset. We are confident this is possible given a better network architecture and training hyperparameters, and a more robust normalization scheme for neural network inputs and outputs. The first step would be to shift our entire framework to use quaternions to represent orientation. This would help us avoid the Euler angle wrapping problem during the post-processing step while simultaneously being well-normalized for the network. We would also like to improve all MPPI controllers by experimenting with more cost functions and tuning the gains corresponding to these cost functions. We have set up a *Weights and Biases* pipeline to do so now that we are confident our algorithms are working.

In the future, we would like to train a dynamics model on data from the PyBullet simulation environment when complex physical phenomena like drag, downwash, and ground effect are enabled. Then, we could compare the performance of the simple analytical model to the learned model which should do a better job at capturing the complicated drone dynamics. This improved dynamics model would correlate to an improved MPPI controller since the rollouts would be more accurate and better represent the true cost of taking certain control actions. Another next step after getting the controller to work on simulated data with complex physics would be to train a neural network on data from an actual drone or use a network to bridge the real-to-sim gap. This is an even more challenging problem than using the advanced simulator dynamics.

REFERENCES

[1] Jacopo Panerati, Hehui Zheng, SiQi Zhou, James Xu, Amanda Prorok, and Angela P. Schoellig. Learning to fly—a gym environment with pybullet physics for reinforcement learning of multi-agent quadcopter control. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021.

[2] B Y Sebbane. *A First Course in Aerial Robots and Drones (Chapman & Hall/CRC Artificial Intelligence and Robotics Series)*. Chapman and Hall/CRC, 2022.

[3] Marcel Veismann, Christopher Dougherty, and Morteza Gharib. Experimental studies of the rotor flow downwash on the Stability of multi-rotor crafts in descent. In *APS Division of Fluid Dynamics Meeting Abstracts*, APS Meeting Abstracts, page M18.002, November 2017.

[4] Grady Williams, Paul Drews, Brian Goldfain, James M. Rehg, and Evangelos A. Theodorou. Aggressive driving with model predictive path integral control. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1433–1440, 2016.

[5] Grady Williams, Nolan Wagener, Brian Goldfain, Paul Drews, James M. Rehg, Byron Boots, and Evangelos A. Theodorou. Information theoretic mpc for model-based reinforcement learning. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1714–1721, 2017.

[6] Ihab S. Mohamed, Guillaume Allibert, and Philippe Martinet. Model predictive path integral control framework for partially observable navigation: A quadrotor case study. pages 196–203, 12 2020.

[7] Mohammad Pairan, Syariful Shamsudin, and Mohd Fadhli. Neural network-based system identification for quadcopter dynamic modeling: A review. *Journal of Advanced Mechanical Engineering Applications*, 1, 11 2020.

[8] Somil Bansal, Anayo K. Akametalu, Frank J. Jiang, Forrest Laine, and Claire J. Tomlin. Learning quadrotor dynamics using neural network for flight control, 2016.

[9] 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.