A Machine-Learning Approach for Time-optimal Trajectory Generation for UAV's

Runyu Lai
Graduate Student
Rensselaer Polytechnic Institute
Troy, NY, USA

Di Zhao
PhD Candidate
Rensselaer Polytechnic Institute
Troy, NY, USA

Sandipan Mishra
Associate Professor
Rensselaer Polytechnic Institute
Troy, NY, USA

ABSTRACT

This paper presents a data-driven approach towards time-optimal trajectory generation for Unmanned Aerial Vehicles (UAV's) using a machine-learned trajectory generation mechanism for point-to-point time-optimal trajectories on-the-fly. To train this machine-learned black box trajectory generator off-line, a model-based optimization problem is first constructed for point-to-point time-optimal trajectory generation, with physical constraints on inputs, states, and rates. The formulated optimization problem is then solved off-line for a range of initial and terminal flight states to generate point-to-point data-sets that consist of the optimal state and input trajectories. This information is compressed by parameterizing the input and state trajectories using a set of basis functions. This data is then used to train the neural network-based trajectory planner. The output of the neural network is the basis function coefficient sets for the state and input trajectories (and the total flight time) which can then be used to reconstruct the flight trajectory. Once the neural network is trained, the data-driven on-board trajectory generator is ready to be deployed on the UAV for on-board planning. This approach is demonstrated for two scenarios: (1) the input to the neural network being the initial and terminal flight states and (2) the input to the neural network being initial and terminal flight states as well as physical constraints. To validate the performance of the machine-learned black-box trajectory generator, the root mean squared error between the neural network generated trajectories and the trajectories obtained from solving the optimization problem directly is statistically evaluated. These trajectories are also tested for violation of path constraints (which are not included explicitly in the training or input to the black box planner) by evaluating the mean constraint violation for each path-constrained variable.

INTRODUCTION

Trajectory planning is a key component of all guidance, navigation and control systems for autonomous unmanned aerial vehicles (UAV's). For mission planning, generally a set of waypoints are generated first, following which waypoint-to-waypoint trajectory planning for UAV's can be done by either parameterizing the kinematic variables (Ref. 1). Alternatively, the trajectories can be generated from an artificial potential field to guide the UAV path generation (Ref. 2). More recently, waypoint-to-waypoint trajectory generation has been posed as an optimization problem; for example as minimum-snap trajectory generation in (Ref. 3) and the time-optimal path generation problem presented in (Ref. 4).

Optimization-based trajectory generation methods are attractive since they can handle constraints and generate optimal paths based on simplified dynamics of the UAV. These methods use (1) a prescribed cost function to be minimized (either time of flight, or snap or jerk of the trajectory), (2) a model of the dynamics of the UAV, and (3) physical constraints on inputs and states, and (4) path constraints created by obstacles, etc. to generate feasible trajectories from the initial state to the terminal state. However, they can often be computationally expensive and are difficult to solve on the fly.

On the other hand, machine-learning based methods have seen significant interest over the past decade for path planning for robotics. (Ref. 5) presents a survey of machine learning approaches to robotic path-planning. These planning algorithms typically focus on determining sequences of maneuvers in complex environments while following a high-level task command. The mechanisms for obtaining the policies or path plans for the high-level task range from reinforcement learning to search over graphs. In (Ref. 6), reinforcement learning is applied to approximate the state value function of a UAV with a suspended load, which aims to determine a swing-free trajectory. In (Ref. 7), machine learning techniques are used to identify the obstacles for UAV flights, which further enables a two-layer obstacle avoidance algorithm to avoid obstacles with minimal effort. In (Ref. 8), a neural network is constructed to predict the trajectory of a small UAV under various wind conditions. In (Ref. 9), a Learning Automata (LA) machine learning technique is used to tune the weights of the Model Predictive Control (MPC) algorithm, which is used to achieve obstacle avoidance trajectory planning for the quadcopter. In (Ref. 10), a supervised feedforward neural network

Presented at the VFS International 76th Annual Forum & Technology Display, October 2020. Copyright © 2020 by the Vertical Flight Society. All rights reserved.

is trained offline to provide time allocation of minimum snap trajectories for quadcopter, which greatly reduced the computational time of onboard implementation. These machine-learning algorithms either do not address the low level dynamics of UAV and the multiple constraints that must be satisfied for planning UAV flight paths, or can not be applied to solve the free end time optimization problem (i.e. time-optimal trajectory). Hence, they may not be well-suited for waypoint-to-waypoint UAV trajectory planning.

In this paper, we present a machine learning approach to trajectory generation for UAV's. The key idea here is to *train a black box machine-learned model using data obtained from solving the trajectory optimization problem offline for a large number of scenarios*. The machine-learned trajectory planner can then be used to generate trajectories on-the-fly at a much lower computational cost than solving the optimization problem online. Towards this goal, we first construct the typical optimization problem for time-optimal trajectory generation and solve it off-line for a range of initial and terminal flight states to create the trajectory data-sets necessary to train the machine-learned model. Next, we compress the information in the planned trajectories by parameterizing them using a set of basis functions. These parameters are then used to train a neural network based model. Once trained, the neural network model is validated against out-of-sample data to study (1) the quality of the fit of trajectories generated by the machine learned trajectory planner and (2) whether constraints imposed by the original optimization problem are satisfied by the trajectories generated by machine-learned trajectory planner.

PROBLEM STATEMENT

As is shown in Fig.(1), the typical mechanism for trajectory generation for quadcopter is to formulate an optimization problem. The dynamics, initial/terminal boundary conditions and the physical capacity of the UAV are framed as constraints that the solution trajectory (i.e. $\boldsymbol{x}^*(\cdot)$ and $\boldsymbol{u}^*(\cdot)$) must satisfy. This approach is based on a white-box model and guarantees the rigorousness of the solution, however the nonlinear constrained optimization problem can be computationally expensive to solve, and thus difficult to implement on-the-fly.

To address this, the trajectory generation method proposed in this paper is based on a *data-driven approach*. Through an offline machine learning process, the data collected from solving the optimization problem for a range of scenarios are used to train a black-box model (e.g., a neural network) by seeking for the optimal weight $\boldsymbol{w}$ and bias $\boldsymbol{b}$ for each neuron in the network, which essentially creates a mapping from any admissible initial state x_0, terminal state x_f and the constraints $\mathbb{X}/\mathbb{U}$ to the corresponding state and input trajectories $\boldsymbol{x}^*(\cdot)$ and $\boldsymbol{u}^*(\cdot)$, respectively. With the black-box model trained by sufficient number of different flight scenarios, the proposed black-box algorithm can generate time-optimal trajectories for different initial and terminal flight states with much lower-computational cost that solving the optimization problem on-the-fly, for real-time trajectory planning.

OPTIMIZATION-BASED TRAJECTORY GENERATION

In this section, we present the typical model-based approach for generating time-optimal trajectory, which will be used to generate training data for the data-driven trajectory generator. We first present a simplified quadcopter dynamic model for trajectory planning and then formulate the optimization problem based on this model.

Simplified Quadcopter Dynamic Model

For planning purposes, we assume the dominant forces acting on the quadcopter to be the thrust T and the gravity mg. The differential RPM input u_ψ generates the yawing moment. We also assume that roll angle ϕ and pitch angle θ dynamics are much faster than translational motion in north X, east Y, down Z directions and the yaw angle ψ dynamics. Thus, we assume ϕ and θ to be instantaneously achievable and thus function as the input variables. With these assumptions, the simplified quadcopter dynamics can be written as:

$$\begin{aligned} \ddot{X} &= -\tfrac{T}{m}(\cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi); \\ \ddot{Y} &= -\tfrac{T}{m}(\cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi); \\ \ddot{Z} &= g - \tfrac{T}{m}\cos\phi\cos\theta; \\ \ddot{\psi} &= u_\psi \end{aligned} \tag{1}$$

Eq.(1) can be denoted as the general form $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x},\boldsymbol{u})$ (shown in Fig.(1)), where the state $\boldsymbol{x} = [X,Y,Z,\psi,\dot{X},\dot{Y},\dot{Z},\dot{\psi}]$ and the input $\boldsymbol{u} = [T,\phi,\theta,u_\psi]$. Note that these dynamics are differential flat as the input variables can be represented by:

$$\begin{aligned} T &= m\sqrt{\ddot{X}^2+\ddot{Y}^2+(g-\ddot{Z})^2}; \\ \phi &= \tan^{-1}\Big(\tfrac{-\ddot{X}\sin\psi+\ddot{Y}\cos\psi}{\sqrt{(g-\ddot{Z})^2+(\ddot{X}\cos\psi+\ddot{Y}\sin\psi)^2}}\Big); \\ \theta &= -\tan^{-1}\Big(\tfrac{\ddot{X}\cos\psi+\ddot{Y}\cos\psi}{g-\ddot{Z}}\Big); \\ u_\psi &= \ddot{\psi} \end{aligned} \tag{2}$$

Eq.(2) indicates that the original nonlinear dynamics in Eq.(1) can be transformed into equivalent linear dynamics: $\dot{\boldsymbol{q}} = \boldsymbol{F}\boldsymbol{q} + \boldsymbol{G}\boldsymbol{w}$, where $\boldsymbol{q} = [X,Y,Z,\psi,\dot{X},\dot{Y},\dot{Z},\dot{\psi}]^T$ and $\boldsymbol{w} = [\ddot{X},\ddot{Y},\ddot{Z},\ddot{\psi}]^T$. Note that the original state $\boldsymbol{x}$ and $\boldsymbol{u}$ can be uniquely determined by $\boldsymbol{q}$ and $\boldsymbol{w}$ by the smooth mappings $\boldsymbol{x} = \mathcal{N}_x(\boldsymbol{q})$ and $\boldsymbol{u} = \mathcal{N}_u(\boldsymbol{q},\boldsymbol{w})$. Specifically for the dynamics in Eq.(1), $\mathcal{N}_x$ is the identical function and $\mathcal{N}_u$ is determined by Eq.(2) (Ref. 11).

Optimization-based Trajectory Design

The time-optimal trajectory optimization problem can be formulated as:

$$\begin{aligned} \underset{t_f^*,\boldsymbol{q}^*,\boldsymbol{w}^*}{\arg\min} \quad & J = \int_{t_0}^{t_f} 1dt, && \textit{flight time} \\ s.t \quad & \dot{\boldsymbol{q}} = \boldsymbol{F}\boldsymbol{q} + \boldsymbol{G}\boldsymbol{w}, && \textit{dynamics} \\ & \boldsymbol{q}(t_0) = \boldsymbol{q}_0, \boldsymbol{q}(t_f) = \boldsymbol{q}_f && \textit{boundary constraints} \\ & \mathcal{N}_x(\boldsymbol{q}) \in \mathbb{X}, \mathcal{N}_u(\boldsymbol{q},\boldsymbol{w}) \in \mathbb{U} && \textit{path constraints} \end{aligned} \tag{3}$$

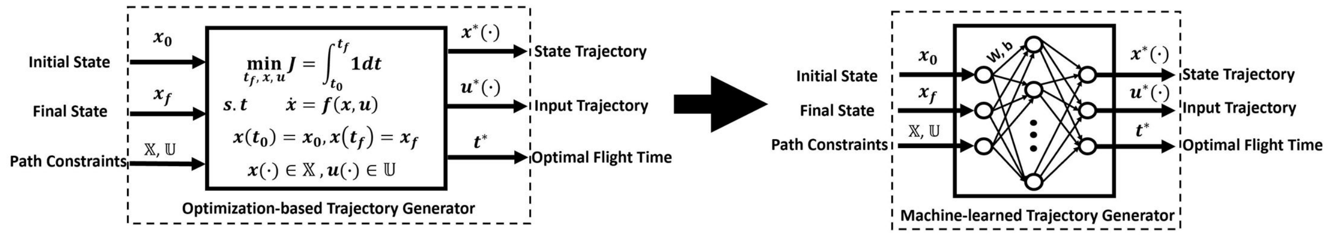

Initial State
Final State
Path Constraints
x_0
x_f
$\mathbb{X}, \mathbb{U}$
$\min_{t_f, x, u} J = \int_{t_0}^{t_f} 1dt$
s.t $\dot{x} = f(x,u)$
$x(t_0) = x_0, x(t_f) = x_f$
$x(\cdot) \in \mathbb{X}, u(\cdot) \in \mathbb{U}$
$x^*(\cdot)$
$u^*(\cdot)$
t^*
State Trajectory
Input Trajectory
Optimal Flight Time
Optimization-based Trajectory Generator
W, b
Machine-learned Trajectory Generator

Figure 1: Optimization-based and Machine-learned Trajectory Generation.

In Eq.(3), $\boldsymbol{q}_0$ and $\boldsymbol{q}_f$ represent the initial and terminal constraints of the trajectory.

For path constraints, the horizontal velocity $V_h = \sqrt{\dot{X}^2 + \dot{Y}^2} \leq V_{h,\max}$, vertical velocity $|\dot{Z}| \leq V_{z,\max}$, vertical acceleration $|\ddot{Z}| \leq a_{z,\max}$ are enforced. Moreover, box constraints on the input variables $\boldsymbol{U}_{\min} \leq |\mathscr{N}_u(\boldsymbol{q}, \boldsymbol{w})| \leq \boldsymbol{U}_{\max}$ are also included.

Note that the problem in Eq.(3) is *an exact reformulation of the original problem* shown in Fig.(1), via differential flatness. By solving the optimization problem in (3), $\boldsymbol{q}^*(\cdot)$ and $\boldsymbol{w}^*(\cdot)$ are determined, and the corresponding state and input trajectories can be obtained from $\boldsymbol{x}^*(\cdot) = \mathscr{N}_x(\boldsymbol{q}^*(\cdot))$ and $\boldsymbol{u}^* = \mathscr{N}_u(\boldsymbol{q}^*(\cdot), \boldsymbol{w}^*(\cdot))$.

The optimization problem above is discretized into a nonlinear programming problem and solved by in the numerical solver CasADi in MATLAB, which eventually generates the reference time-optimal trajectory.

TRAJECTORY GENERATION USING A NEURAL NETWORK

In this section, we develop a methodology to create data-driven trajectory generators as an alternative. For illustrative purposes, two cases are discussed in this paper. In the first case, the inputs to the trajectory generator are the initial state and the terminal state of UAV. In the second case, we also add bounds on path constraints as input to the trajectory generator. We first generate training and testing data sets from the optimization-based trajectory generator and then compress the data sets so that we can retrieve complete information of the time-optimal trajectories from smaller data sets. A neural network model is then built, trained and tested based on these data sets. This neural network is the machine-learned trajectory generator as shown in Fig.(1).

Data Generation

In order to obtain adequate learning data for training the neural network, M different reference trajectories characterized by different initial and terminal states are solved by randomly selecting the corresponding $\boldsymbol{q}_0$ and $\boldsymbol{q}_f$ as below:

$$\begin{aligned} \boldsymbol{q}_0 &= [0,0,0,0,0,0,0,0,0]^T; \\ \boldsymbol{q}_f^i &= [X_f^i, Y_f^i, Z_f^i, 0,0,0,0,0,0]^T \quad i \in [1,M] \end{aligned} \tag{4}$$

where

$$X_f^i \in \mathscr{U}(\underline{X}_f, \bar{X}_f),\ Y_f^i \in \mathscr{U}(\underline{Y}_f, \bar{Y}_f),\ Z_f^i \in \mathscr{U}(\underline{Z}_f, \bar{Z}_f) \tag{5}$$

and $\mathscr{U}(\cdot,\cdot)$ represents a uniformly random distribution; $\underline{X}_f, \underline{Y}_f, \underline{Z}_f$ and $\bar{X}_f, \bar{Y}_f, \bar{Z}_f$ represent the lower and upper bounds of the sample for the terminal state X, Y and Z respectively.

Note that the boundary conditions $\boldsymbol{q}_0$ and $\boldsymbol{q}_f$ in Eq.(4) represent point to point maneuvers that start with hover state at the origin, and end with another hover state at (randomized) case-specific coordinates. Also, despite the initial position chosen to be the origin in this paper, this can be translated to arbitrary coordinates without loss of generality. Furthermore, the quadcopter is chosen to head north at the beginning and end of the trajectory (i.e. $\psi_0 = \psi_f = 0$).

For the second case, these M trajectories are not only characterized by q_0 and q_f as shown in Eq.(4) but are also characterized by varying lower bounds of path constraints $\boldsymbol{U}_{min}$ including constraints on θ_{min},ϕ_{min} and thrust T_{min} as well as upper bounds of path constraints $\boldsymbol{U}_{max}$ including constraints on θ_{max},ϕ_{max} and thrust T_{max}. Constraint on horizontal velocity $\boldsymbol{V}_{max}$ are included as well. These trajectories are sampled as below:

$$\begin{aligned} \boldsymbol{U}_{min}^i &= [T_{min}^i, \theta_{min}^i, \phi_{min}^i]^T; \\ \boldsymbol{U}_{max}^i &= [T_{max}^i, \theta_{max}^i, \phi_{max}^i]^T; \\ \boldsymbol{V}_{max}^i &= V_{max}^i; \quad i \in [1,M] \end{aligned} \tag{6}$$

where

$$\begin{aligned} &T_{min}^i \in \mathscr{U}(\underline{T}_{min}, \bar{T}_{min}),\ T_{max}^i \in \mathscr{U}(\underline{T}_{max}, \bar{T}_{max}), \\ &\theta_{min}^i \in \mathscr{U}(\underline{\theta}_{min}, \bar{\theta}_{min}),\ \theta_{max}^i \in \mathscr{U}(\underline{\theta}_{max}, \bar{\theta}_{max}), \\ &\phi_{min}^i \in \mathscr{U}(\underline{\phi}_{min}, \bar{\phi}_{min}),\ \phi_{max}^i \in \mathscr{U}(\underline{\phi}_{max}, \bar{\phi}_{max}), \\ &V_{hmax}^i \in \mathscr{U}(\underline{V}_{hmax}, \bar{V}_{hmax}) \end{aligned} \tag{7}$$

and $\mathscr{U}(\cdot,\cdot)$ represents a uniformly random distribution; $\underline{T}_{min}$,$\underline{T}_{max}$,$\underline{\theta}_{min}$, $\underline{\theta}_{max}$, $\underline{\phi}_{min}$, $\underline{\phi}_{max}$, $\underline{V}_{hmax}$ and $\bar{T}_{min}$, $\bar{T}_{max}$, $\bar{\theta}_{min}$, $\bar{\theta}_{max}$, $\bar{\phi}_{min}$, $\bar{\phi}_{max}$, $\bar{V}_{hmax}$ represents lower and upper bounds of the sample for the path constraints U_{min}, U_{max} and V_{max}.

For both cases each pair of $\boldsymbol{q}_0$ and $\boldsymbol{q}_f^i, i \in [1,M]$, the corresponding time-optimal trajectory t_f^{i*}, $\boldsymbol{q}^{i*}(t_n)$ and $\boldsymbol{w}^{i*}(t_n)$ is solved and later used as the learning data for the neural network. In the second case discussed above, each constraint $\boldsymbol{U}_{min}^i$, $\boldsymbol{U}_{max}^i$ and $\boldsymbol{V}_{max}^i$ applied is saved as part of learning data as well. Note that $t_n = t_0 + \frac{n}{N-1}(t_f - t_0), n \in [0, N-1]$ denotes the time stamp of each discretization node.

Data Compression

Note that each of the time-optimal trajectories generated above has a size of $\mathbb{R}^{12N+1}$, where N is the number of (discretized time) nodes in the trajectory. Although complete information of the trajectories can be directly retrieved and then used to train the neural network, this large and complex learning data will lead to a very large neural network model and potentially cause over-fitting issues as well as increase computational burden. On the other hand, the time-optimal trajectories solved from the trajectory optimization problem can be indeed be well-approximated by parameterizing the trajectories through a set of basis functions, which eventually compresses the size of the data that is necessary for training the neural network.

Parameterization of Time-Optimal Trajectory: The general idea behind parameterizing the optimal state $\boldsymbol{q}^*(\cdot)$ and input $\boldsymbol{w}^*(\cdot)$ is to approximate them by the summation of a set of basis functions that are scaled by the corresponding coefficients, in the form $\hat{(\star)}^*(t) = \sum_{k=0}^{R} a^*_{(\star)k}\varphi_{(\star)k}(t)$, where $\hat{}$ represents the approximated trajectory as opposed to the actual optimal solution; $(\star)$ denotes each variable $(x, y, z, \psi$ etc.) in the state $\boldsymbol{q}$ and input $\boldsymbol{w}$; $\varphi_{(\star)k}(t)$ represents the k^{th} basis function, with $a^*_{(\star)k}$ being the corresponding parametric coefficients. Typically, all the variables in the state and input vectors should be parameterized respectively. However, for the specific problem in Eq.(3), because of the differential flatness property, we only require parametrized functions of the optimal flat output $\boldsymbol{o}^*(\cdot) = [X^*(\cdot), Y^*(\cdot), Z^*(\cdot), \psi^*(\cdot)]^T$, while higher order time derivatives in $\boldsymbol{q}^*$ and $\boldsymbol{w}^*$ are automatically determined.

Selection of Basis Functions: In general, there are many choices for determining the basis functions, such as Legendre polynomials, radial basis functions, splines and principal components from data. In this paper, the variables $\boldsymbol{o}$ are parameterized by R^{th} order polynomials. The The UAV position $X^*(\cdot)$, $Y^*(\cdot)$, $Z^*(\cdot)$ and yaw angle $\psi^*(\cdot)$ of UAV are parameterized as:

$$\hat{\psi}^*(t) = \sum_{k=0}^{R_\psi} a^*_{(\psi)k}(t-t_0)^k, t \in [t_0, t_f^*] \tag{8}$$

In order to determine the highest order $R_{(\star)}$ for each flat output variable, the approximation performance of the polynomial with different order is evaluated by the mean squared error of the fit: (i.e. $MSE = \frac{1}{N}\sum_{n=0}^{N-1}((\star)^*(t_n) - \hat{(\star)}^*(t_n))^2$)

We observed that the approximation performance is generally good with polynomial orders larger than 4 for all 4 flat output variables. Consequently, in order to balance trade-off between the complexity of the neural network and the degree of freedom for shaping the trajectory, $R_{(\star)} = 4$ is chosen for fitting.

Determination of coefficients for basis functions: Note that approximating the optimal trajectory of the flat output $\boldsymbol{o}^*(\cdot)$ may be lead to the undesirable error in the initial and terminal position in the parameterized trajectories, if all the parametric coefficients $a^*_{(\star),k}, k \in [0,4]$ are unconstrained. In order to satisfy the hard constraints set by the initial and terminal condition of the quadcopter position and yaw angle, the corresponding coefficients are fixed by:

$$\begin{aligned} a^*_{(\star),0} &= (\star)^*(t_0), \\ a^*_{(\star),4} &= \frac{1}{t_f^{*4}}\left((\star)^*(t_f^*) - a^*_{(\star),0} - \sum_{k=1}^{4} a^*_{(\star)k}(t_f^* - t_0)^k\right) \end{aligned} \tag{9}$$

The rest of the coefficients $a^*_{(\star),1,2,3}$ for parameterizing the optimal trajectory of the flat output variables are then obtained by following a standard constrained least squares problem. Note that the above problem can be solved for any time-optimal trajectory t_f^{i*}, $\boldsymbol{q}^{i*}$ and $\boldsymbol{w}^{i*}$, which provides the corresponding optimal parametric coefficient vector $\boldsymbol{a}^{i*} = [(\boldsymbol{a}^{i*}_X)^T, (\boldsymbol{a}^{i*}_Y)^T, (\boldsymbol{a}^{i*}_Z)^T, (\boldsymbol{a}^{i*}_\psi)^T]^T$ and $\boldsymbol{a}^{i*}_{(\star)} = [a^{i*}_{(\star),0}, a^{i*}_{(\star),1}, \cdots, a^{i*}_{(\star),4}]^T$. This data is used as the training data for the neural network, which is discussed in the next subsection.

Structure and Training of Neural Network

In this subsection, we present the methodology for developing a data-driven path planner for generating time-optimal trajectories for (1) given initial and terminal states, and (2) given input constraints.

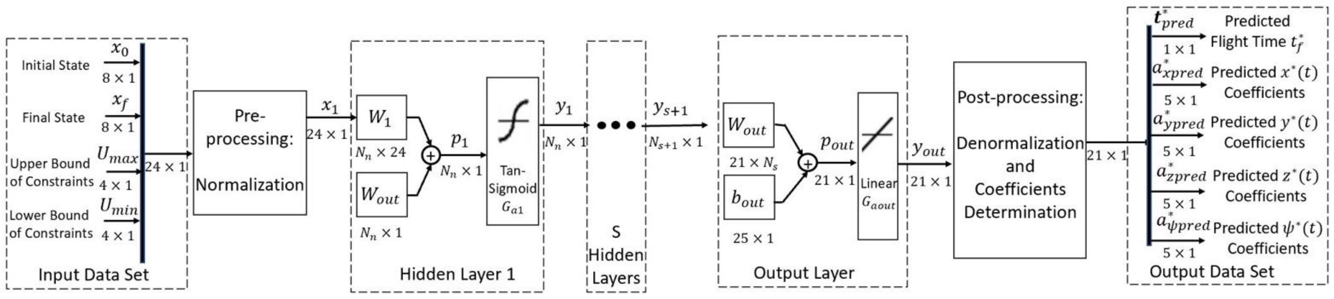


Figure 2: Structure of Neural Network for Time-optimal Trajectory Prediction

Create Network Object: We create a feedforward network which is able to predict t^* and $(\star)^*$, where $(\star)^*$ denotes each individual element in the optimal state $\mathbf{q}^*$ and input $\mathbf{w}^*$. The input vector to this neural network model is the initial state $\mathbf{x}_0$, final state $\mathbf{x}_f$ of the quadcopter and the input constraints $\mathbf{U}_{max}$ and $\mathbf{U}_{min}$ enforced during each flight. We combine all these characteristics of the desired flight and generate the input vector to neural network as:

$$input^i = [(\mathbf{x}_0^i)^T, (\mathbf{x}_f^i)^T, (\mathbf{U}_{min}^i)^T, (\mathbf{U}_{max}^i)^T]^T \tag{10}$$

Note that both $\mathbf{x}_0^i$ and $\mathbf{x}_f^i$ are of size $\mathbb{R}^{8\times1}$; both $\mathbf{U}_{min}^i$ and $\mathbf{U}_{max}^i$ are of size $\mathbb{R}^{4\times1}$, thus the output vector as indicated in Eq.(10) is of size $\mathbb{R}^{24\times1}$. However, the true input vector we applied may have some of the elements in Eq.(10) fixed, which is determined by the physical constraints as well as the goal of the quadcopter, we denote the actual size of the input vector we used for our proposed neural network as $\mathbf{m}$.

The output of the neural network predicts the optimal flight time (t^{i*}) combined with optimal parametric coefficient vector a^{i*}, which can be written as:

$$output^i = [t^{i*}, (a_X^{i*})^T, (a_Y^{i*})^T, (a_Z^{i*})^T, (a_\psi^{i*})^T]^T; \tag{11}$$

Note that the optimal time $t^{i*} \in \mathbb{R}$ and $\boldsymbol{a}_{(\star)}^{i*} \in \mathbb{R}^{5\times1}$, thus the output vector as indicated in Eq.(11) is of size $\mathbb{R}^{21\times1}$. According to Eq.(9), there are always two elements in each $a_{(\star)}^i$ calculated from the other elements, we can denote the actual size of neural network output vector as $\mathbf{n} = 21$.

Determine Neural Network Structure: In order to determine the structure of the neural network, we first obtain the number of hidden layers and neurons per hidden layer. It is generally accepted that increasing the number of hidden layers can reduce error and improve accuracy of network prediction (Ref. 12). However, a larger number of hidden layers also leads to a higher neural network model complexity, which gives rise to increasing in training time as well as the tendency to overfit.

For simplicity, we use a network with one hidden layer, and then lower prediction error by adjusting the number of neurons in the network. The choice of number of neurons in the hidden layer is based on the number of input elements $\mathbf{m}$, and number of output elements $\mathbf{n}$. To avoid overfitting and reduce training time, the neural network model should be as compact as possible. We choose the default number of neurons N_n by checking the geometric mean $\sqrt{\mathbf{m}\times\mathbf{n}}$ of the input and output vectors. Then we increase the number of N_n and conduct K-fold cross validation (Ref. 13). The optimal N_n is chosen as the number of neurons that results in the smallest testing error. All neurons N_k in the hidden layer are connected with each element of the input layer and each connection is separately weighted by a weight w_{ik}. The input x_{hid} and output y_{hid} of the hidden layer of the proposed prediction neural network can be written as shown in Eq.(12):

$$\begin{aligned} x_{hid} &= [\mathbf{x}_0^T, \mathbf{x}_f^T, \mathbf{U}_{min}^T, \mathbf{U}_{max}^T]^T; \\ y_1 &= G_{a1}(W_1 x_1 + b_1)); \end{aligned} \tag{12}$$

where b_1 is a vector consists of all bias for each neurons and G_{a1} is the activation function of hidden layer. All weights W_{ik} are initialized with a small random numbers $W_{ik0} \in (0, 0.001)$. We choose G_{a1} as the hyperbolic tangent sigmoid function, $G_{a1}(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$. The output of output layer can be written as shown in Eq.(13) and an identical function is used for the activation function $G_{aout}(x) = x$.

$$y_{out} = G_{aout}(W_{out} y_{s+1} + b_{out}));) \tag{13}$$

where s is hidden layer number. Since the sigmoid activation function in the hidden layer bounds the output between $[-1, 1]$, the output of proposed neural network is in fact the normalized output. We then calculate the predicted output from these normalized values. The structure of this neural network model is shown in Fig.(2).

Train Neural Network: Once the structure of the neural network is established and the weights and bias are initialized, the neural network is ready for training. We use the well-known Levenberg-Marquardt algorithm (Ref. 14) for training. We form input and output vectors by combining input and output vectors of every single sample as shown in Eq.(14):

$$\begin{aligned} Input &= [input^1, input^2, \cdots, input^i]; \\ Output &= [output^1, output^2, \cdots, output^i]; \end{aligned} \tag{14}$$

Note that the output training samples should be normalized before training. During training, the weights and bias of the network is adjusted to minimize the mean square error between the network output and desired outputs. Thus, the weight matrices W_1, W_{out} and bias vectors b_1, b_{out} are obtained.

VALIDATION OF NEURAL NETWORK TRAJECTORY GENERATION

In this section, we outline the methodology for validating the proposed data-driven trajectory generation algorithm. The performance of the trajectory generator is evaluated based on approximation quality and violation of constraints.

Evaluating Quality of Approximation

To evaluate the neural network prediction of optimal flight time t_{pred} against true optimal time t^*, we evaluate the (normalized) error as shown in Eq.(15):

$$e_t = \frac{|t_{pred} - t^*|}{t^*} \tag{15}$$

To determine the quality of the predicted trajectories of the flat outputs by the neural network, we calculate the root mean squared error (RMSE) between the predicted flat output $(\star)_{pred}$ and flat output obtained from the optimizer $(\star)^*$ in the validation data set. This RMSE is calculated as shown in Eq.(16):

$$e_{(\star)pred} = \sqrt{\frac{1}{N}\sum_{n=0}^{N-1}((\star)_{pred}(t_n) - (\star)^*(t_n))^2} \tag{16}$$

To test the quality of trajectories generated by the neural networks, we analyze the distribution of prediction mean square errors $e_{(\star)pred}$ for different validation samples and provide a statistical analysis of the approximation quality of the proposed neural network.

Evaluating Violation of Constraints

Since the data-driven model has no inherent mechanism to encode the dynamics and path constraints, it is critical to confirm that the neural network predicted trajectories meet the dynamics and path constraints that the original trajectories were designed to satisfy.

From the flat output trajectories generated by the neural network, we can compute the entire state trajectory. We can then evaluate violation of constraint of the proposed machine-learned model by comparing the neural-network-predicted states with the constraints enforced in the original optimization problem.

We now evaluate the violation/satisfaction of roll, pitch, thrust and horizontal velocity limits by the trajectories generated by the neural network. The profiles of roll angle ϕ and pitch angle θ for each trajectory in the validation sample set are obtained by using Eq.(2). We then utilize the same criteria to verify the constraint on thrust T and horizontal velocity V_h. Mean violation of path constraints are calculated by Eq.(17),where $\triangle$ denotes ϕ, θ, T and V_h.

$$e_{(\star)pred} = \frac{1}{N}\sum_{n=0}^{N} |(\triangle)_{violated}(t_n) - (\triangle)_{Constraint}(t_n)| \tag{17}$$

RESULTS, VALIDATION, AND DISCUSSION

To apply and test the methodology for the data-driven trajectory generator, we build two neural networks: one with only the desired initial and terminal states as input (Case 1) and another with both desired states and constraints enforced in optimization problem as input to the neural network (Case 2). We evaluate the quality of these two trajectory generators by testing them with different validation samples and provide a statistical evaluation.

Case 1: Path Planner with Fixed Constraints

Following the proposed methodology, we build a neural network with one hidden layer and 13 neurons to predict the optimal flight time and all four elements $(\star)^*$ of the flat outputs. In this case, input to the neural network are the initial and final states of quadcopter as shown in Fig.(4). We randomly generate a sample set of 2000 trajectories as the learning data set and then another data set of size 500 trajectories for validation purposes, in both of which, $\underline{X}_f = \underline{Y}_f = 10$ m, $\underline{Z}_f = -20$ m and $\bar{X}_f = \bar{Y}_f = 20$ m, $\bar{Z}_f = -10$ m are selected.

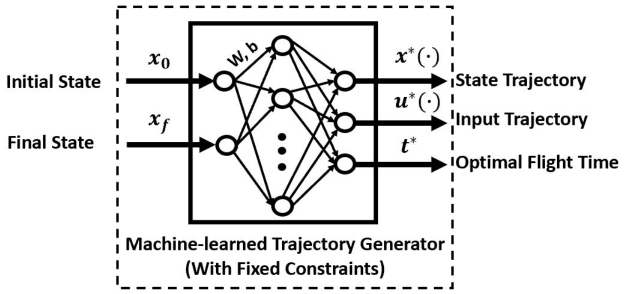
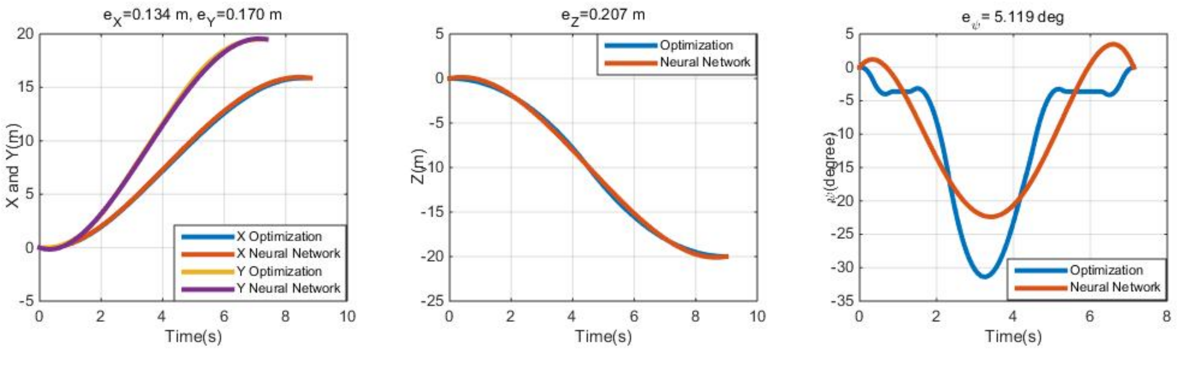


Figure 4: Machine-learned Trajectory Generator with Fixed Constraints (Case 1)

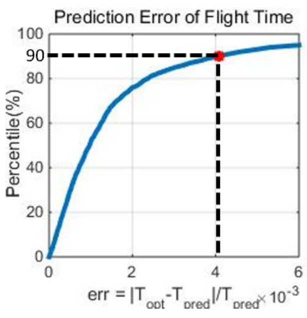
We first present a comparison of representative trajectories obtained from the neural network model against those determined by direct optimization. For illustrative purposes, we show one particular scenario for trajectories predicted from the machine learned trajectory generator compared against those from the optimization-based approach, in Fig.(3). We observe that the trajectories planned by the machine-learned model approximate those obtained from the optimization-based planner well. The RMSE of X trajectory prediction is 0.134 m, the RMSE of Y trajectory prediction is 0.170 m, the RMSE of Z trajectory prediction is 0.207 m and RMSE of ψ trajectory prediction is 5.119 deg.

Next, we present a statistical evaluation of the data-driven trajectory generation mechanism. We plot the distribution of prediction errors for optimal time t^* and all flat outputs $(\star)$ for all 500 validation trajectories, shown in Fig.(5). The dashed lines

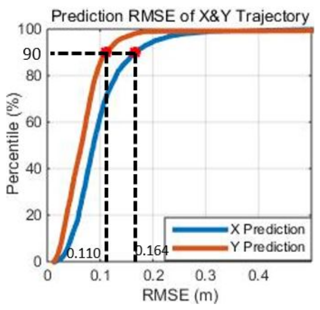


(a) X Trajectories (b) Z Trajectories (c) Yaw ψ Trajectory

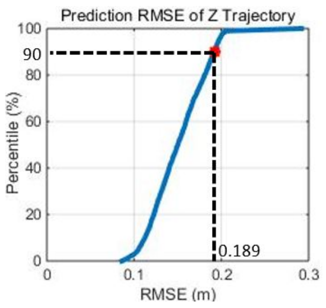
Figure 3: Sample Trajectories from the Machine-learned Planner and the Optimization-based Planner.



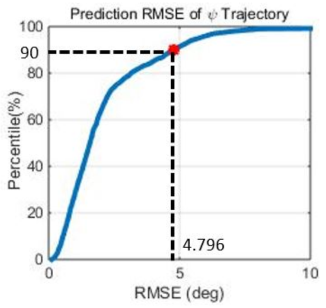
(a) Optimal time t^* Prediction



(b) X and Y Prediction RMSE

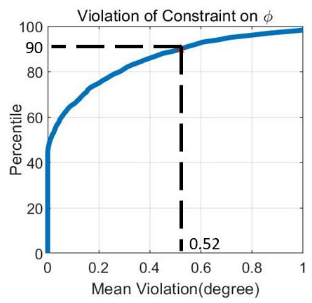


(c) Z Prediction RMSE

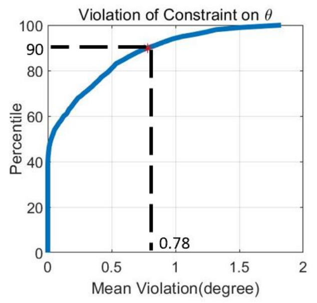


(d) ψ Prediction RMSE

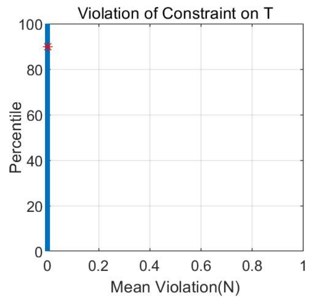
Figure 5: Percentile vs Prediction RMSE between Neural Network and Optimization-based Trajectories, from 500 Validation Cases.(The dashed lines indicates RMS of 90th-percentile Prediction)



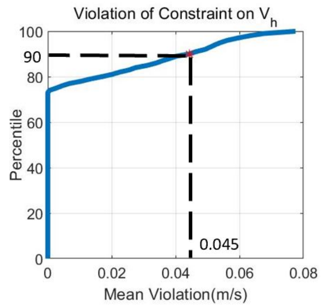
(a) Evaluation of Constraint on ϕ



(b) Evaluation of Constraint on θ



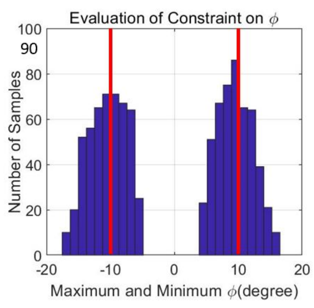
(c) Evaluation of Constraint on T



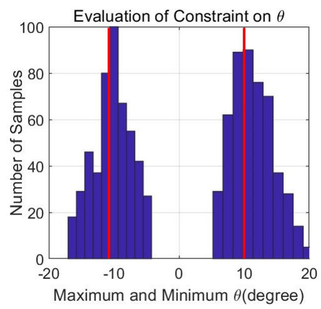
(d) Evaluation of Constraint on V_h

Figure 6: Evaluation of Path Constraints, for 500 validation cases (Case 1)

show the RMS prediction error for the 90th-percentile of all test samples to be within 0.110 m for X, within 0.164 m for Y, within 0.189 m for Z and within 4.796 deg for yaw, which



(a) Evaluation of bounds of ϕ



(b) Evaluation of Bounds of θ

Figure 7: Evaluation of Bounds of Constraints, for 500 validation cases (Case 1)

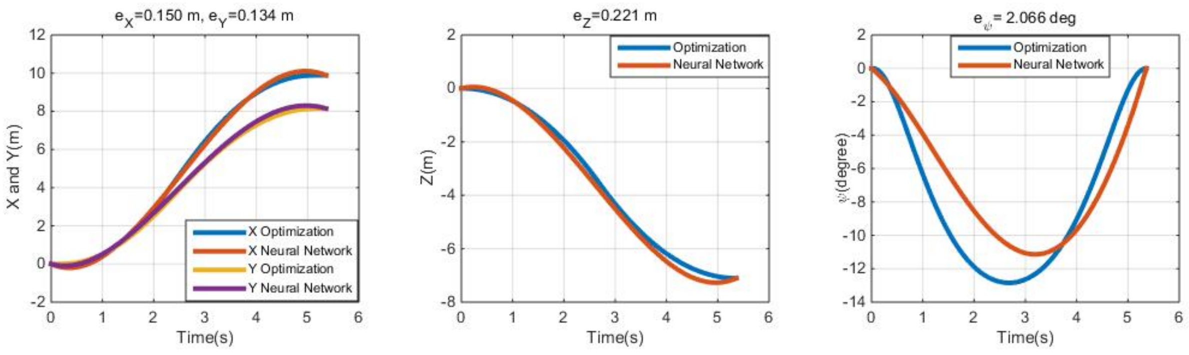
means the neural network model generates trajectories with better RMSE than those in Fig.(3), 90% of the time.

Finally, we test each trajectory for the degree of constraint violations. We check the violations on max/min $\theta = \pm 10$ degree , $\phi = \pm 10$ degree, thrust $T = [1.96\,\text{N}, 19.6\,\text{N}]$, and $V_h = [0, 5\text{ m}]$ during each flight in the validation sample, and provide a statistical analysis as shown in Fig.(6). We can see that for 90% of the trajectories in the validation data set, the mean violation of constraints on ϕ is no more than 0.52 degree and the mean violation of constraints on θ is no more than 0.78 degree. We can observe from Fig.(6) that only 50 % of trajectories in the validation set always satisfy the constraint $\theta, \phi \in [-10\text{ deg}, 10\text{ deg}]$. The distribution of the maximum and minimum roll, pitch angle is illustrated in Fig.(7). Note that the distribution of upper and lower bounds of each trajectories predicted is centered around the enforced upper and lower bounds of constraints. The mean violation of constraint on horizontal velocity V_h is less than 0.045 m/s for 90% of the sample trajectories in validation data set from Fig.(6). Interestingly, the thrust bound is never violated in the cases we studied.

We then tested the neural network trajectory planner on another sample set consisting of randomly generated trajectories in which $\underline{X}_f = \underline{Y}_f = 5$ m, $\underline{Z}_f = -10$ m and $\bar{X}_f = \bar{Y}_f = 10$ m, $\bar{Z}_f = -5$ m are selected as shown in Fig.(8). This flight is 'shorter' than the training flights (used to train the network) and upper bound of constraints on roll, pitch angle are not active often. The trajectory generator is adequate for all X, Y, Z and ψ trajectories.

Case 2: Path Planner with Constraints as Input

In addition to initial and terminal states, we now train the neural network to generate trajectories with different bounds on the path constraints. The inputs to the neural network are the initial state, terminal state, and bounds on row, pitch angle as well as thrust and maximum horizontal velocity. To do this, we train the network on learning data with varying constraints U including constraints on row, pitch angle as well as thrust and maximum horizontal velocity V_h, as shown in Fig.(9). Upper and lower bounds $V_{max}, \boldsymbol{U}_{max}$ and $\boldsymbol{U}_{min}$ for these path constraints are randomly generated within reasonable physical limits. In this case, the constraints are sampled within the



(a) X and Y Trajectories (b) Z Trajectories (c) Yaw ψ Trajectory

Figure 8: Sample Trajectories from the Path Planner and the Optimization-based Planner (Different Flight Range).

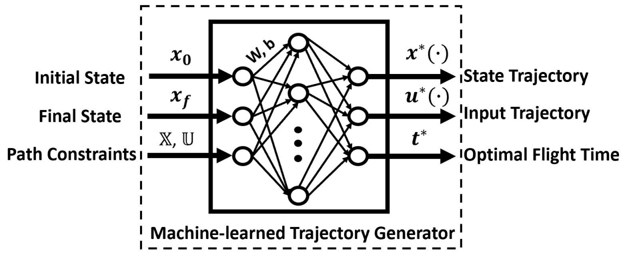


Figure 9: Machine-learned Trajectory Generator with Constraints as Input (Case 2).

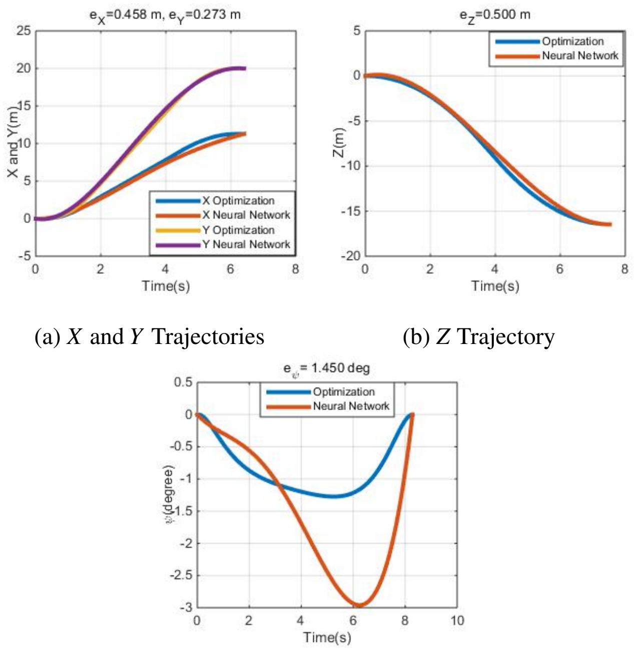
range stated in Eq.(18):

$$\begin{aligned} &V_{max} \in [4\text{m/s}, 10\text{m/s}];\\ &\boldsymbol{U}_{min}^{lower} = [1.96\text{N}, -20\text{deg}, -20\text{deg}];\\ &\boldsymbol{U}_{min}^{upper} = [6.87\text{N}, -10\text{deg}, -10\text{deg}];\\ &\boldsymbol{U}_{max}^{lower} = [14.72\text{N}, 10\text{deg}, 10\text{deg}];\\ &\boldsymbol{U}_{max}^{upper} = [19.60\text{N}, 20\text{deg}, 20\text{deg}] \end{aligned} \tag{18}$$

A new training data set consisting of 2000 trajectories is generated following these requirements and a validation sample set of size 500 is generated in the same manner as well, in both of which, $\underline{X}_f = \underline{Y}_f = 10$ m, $\underline{Z}_f = -20$ m and $\bar{X}_f = \bar{Y}_f = 20$ m, $\bar{Z}_f = -10$ m are selected. Based on these two data sets, a new neural network model is built, trained and performance of this neural network is evaluated following the methodologies used in the previous case.

We first show one particular scenario for illustrative purposes in Fig.(10) and then demonstrate the accuracy of this path planner by presenting a statistical evaluation as shown in Fig.(11). The trajectories planned by this new machine-learned model approximate those obtained from the optimization-based planner well. The RMSE of X trajectory prediction is 0.707 m, the RMSE of Y trajectory prediction is 0.836 m, the RMSE of Z trajectory prediction is 0.740 m and RMSE of ψ trajectory prediction is 7.681 deg. Compared with the neural network developed in case 1, the prediction accuracy of this new path planner with some constraints as input is lower.

Path constraints are evaluated as shown in Fig.(12) by checking the violations on max/min θ, ϕ, thrust T, and V_h during

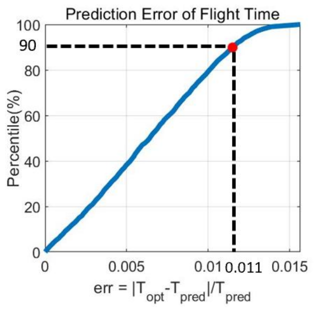


(a) X and Y Trajectories (b) Z Trajectory

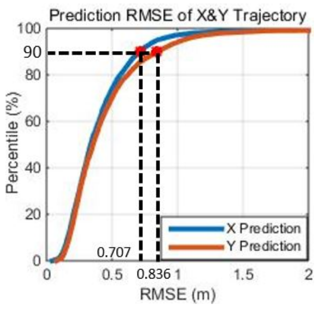
(c) Yaw ψ Trajectory

Figure 10: Sample Trajectories from the Machine-learned Planner and the Optimization-based Planner.

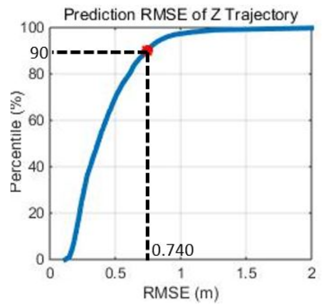
each flight in the validation sample. Note that each trajectory has different upper and lower bounds of constraints within the range stated in Eq.(18). These bounds are included in the inputs to the neural network. We can see that for 90% of the trajectories in the validation data set, the mean violation of constraints on ϕ and θ is no more than 0.10 degree which is smaller than the mean violation angle in case 1. Comparing with Fig.(6) with only 50% of trajectories always satisfying the enforced constraints on ϕ and θ, we can see that up to 80% of the predicted trajectories do not violate enforced constraints in Fig.(12). We can also observe that more than 90% of the prediction satisfy constraint on horizontal velocity V_h. In other words, risk of constraint violation is decreased by inputting constraints into our neural network model.



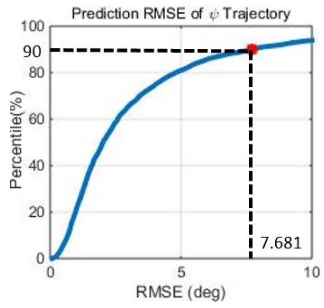
(a) t^* Prediction Error



(b) X and Y Prediction RMSE

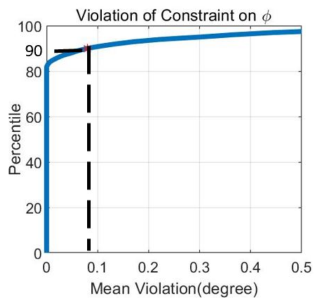


(c) Z Prediction RMSE

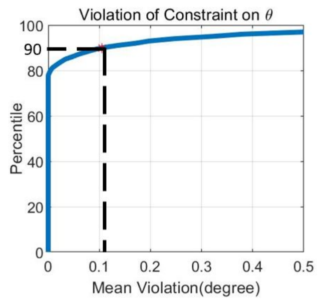


(d) ψ Prediction RMSE

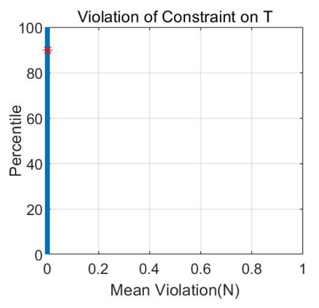
Figure 11: Percentile vs Prediction RMSE between Neural Net and Optimization-based Trajectories, from 500 Validation Cases.(The dashed lines indicates RMSE of 90^{th}-percentile Prediction)



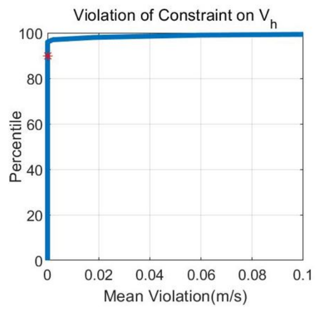
(a) Evaluation of Constraint on ϕ



(b) Evaluation of Constraint on θ



(c) Evaluation of Constraint on T



(d) Evaluation of Constraint on V_h

Figure 12: Evaluation of Path Constraints, for 500 validation cases (Case 2)

CONCLUSIONS

The goal of this research was to determine a data-driven approach for real-time trajectory planning for UAV's. To this end, a method to create a 'black-box' model that approximates a time-optimal trajectory given the initial and terminal flights states was proposed. Data from a model-based trajectory optimization algorithm was used to train the neural network model. This trajectory data was compressed by using projection onto a set of (piecewise) polynomial functions. The compressed data was used to train a (relatively simple) neural network. Subsequent testing and validation showed that the neural network can indeed produce trajectories very similar to those generated from the optimization algorithm, and violates the constraints for a very small fraction of the flight time. With this data-driven black-box trajectory planner, we can reduce the computational time since the optimization problem is no longer solved on-the-fly.

There are several interesting directions to follow based on results in this paper. One of them is to design a trajectory planner that can avoid obstacles with fixed states. We can take the size and location of fixed state obstacles into account by adding one or more features that capture the obstacle's effect on the trajectory we generate. Another interesting direction is to create on-board path planner with flexible dynamic constraints. While the planner developed here produces trajectories with specified (fixed in time) state and input constraints, adding dynamic constraints to the neural network model as new features will enable adaptation to changing flight constraints.

ACKNOWLEDGMENTS

This work was sponsored by in part by the Office of Naval Research (ONR), under contract number N00014-16-1-2705, and in part under the Army/Navy/NASA Vertical Lift Research Center of Excellence (VLRCOE) Program, grant number W911W61120012, with Dr. Mahendra Bhagwat and Dr. William Lewis as Technical Monitors.

REFERENCES

1. Jang, J. T., Moon, S. T., Han, S., Gong, H. C., Choi, G., Hwang, I. H., and Lyou, J., "Trajectory generation with piecewise constant acceleration and tracking control of a quadcopter," 2015 IEEE International Conference on Industrial Technology (ICIT), Seville, Spain, March 2015.

2. Vílez, P., Certad, N., and Ruiz, E., "Trajectory generation and tracking using the AR. Drone 2.0 quadcopter UAV," 2015 12th Latin American Robotics Symposium and 2015 3rd Brazilian Symposium on Robotics (LARS-SBR), Uberlandia, Brazil, October 2015.

3. Mellinger, D., and Kumar, V., "Minimum snap trajectory generation and control for quadrotors," 2011 IEEE International Conference on Robotics and Automation, Shanghai, China, May 2011.

4. Hu, B., and Mishra, S., "Time-Optimal Trajectory Generation for Landing a Quadrotor Onto a Moving Platform," *IEEE/ASME Transactions on Mechatronics*, Vol. 24, (2), April 2019, pp. 585–596.

5. Otte, M., “A Survey of Machine Learning Approaches to Robotic Path-Planning,” Technical Report PhD Preliminary Exam, University of Colorado at Boulder, 2008.

6. Faust, A., Palunko, I., Cruz, P., Fierro, R., and Tapia, L., “Learning swing-free trajectories for UAVs with a suspended load,” 2013 IEEE International Conference on Robotics and Automation, Karlsruhe, Germany, May 2013.

7. Choi, Y., Jimenez, H., and Mavris, D. N., “Two-layer obstacle collision avoidance with machine learning for more energy-efficient unmanned aircraft trajectories,” *Robotics and Autonomous Systems*, Vol. 98, 2017, pp. 158 – 173.

8. Xue, M., “UAV Trajectory Modeling Using Neural Networks,” 17th AIAA Aviation Technology, Integration, and Operations Conference, Denver, CO, June 2017.

9. Jardine, P. T., Givigi, S. N., and Yousefi, S., “Experimental results for autonomous model-predictive trajectory planning tuned with machine learning,” 2017 Annual IEEE International Systems Conference (SysCon), Montreal, QC, Canada, April 2017.

10. Almeida, M., Moghe, R., and Akella, M., “Real-Time Minimum Snap Trajectory Generation for Quadcopters: Algorithm Speed-up Through Machine Learning,” 2019 International Conference on Robotics and Automation (ICRA), Montreal, QC, Canada, May 2019.

11. Zhao, D., Krishnamurthi, J., Mishra, S., and Gandhi, F., “A trajectory generation method for time-optimal helicopter shipboard landing,” American Helicopter Society 74th Annual Forum, Phoenix, AR, May 2018.

12. Gaurang, P., Ganatra, A., Kosta, Y., and Panchal, D., “Behaviour Analysis of Multilayer Perceptronswith Multiple Hidden Neurons and Hidden Layers,” *International Journal of Computer Theory and Engineering*, Vol. 3, 01 2011, pp. 332–337.

13. Kohavi, R., “A Study of Cross-validation and Bootstrap for Accuracy Estimation and Model Selection,” IJCAI’95, Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2, Montreal, Quebec, Canada, 1995.

14. Yu, H., and Wilamowski, B., *Levenberg Marquardt Training Industrial Electronics Handbook, vol. 5 Intelligent Systems, 2nd Edition*, CRC Press, 01 2011, Chapter 12, pp. 12–1 to 12–15.