A Simple Neural Network for Efficient Real-time Generation of Dynamically-Feasible Quadrotor Trajectories

Mohammad Jawad Lakis
Electrical & Computer Engineering Department
American University of Beirut
Beirut, Lebanon
mbl00@mail.aub.edu

Naseem Daher
Electrical & Computer Engineering Department
American University of Beirut
Beirut, Lebanon
nd38@aub.edu.lb

2021 IEEE 3rd International Multidisciplinary Conference on Engineering Technology (IMCET) | 978-1-6654-0587-4/21/$31.00 ©2021 IEEE | DOI: 10.1109/IMCET53404.2021.9665549

***Abstract*—In this work, we study the problem of efficiently generating dynamically-feasible trajectories for quadrotors in real-time. A supervised learning approach is used to train a simple neural network with two hidden layers. The training data is generated from a well-established trajectory generation method for quadrotors that minimizes jerk given a fixed time interval. More than a million dynamically-feasible trajectories between two random points in the three-dimensional (3D) space are generated and used as training data. The input of the neural network is a vector composed of initial and desired states, along with the final trajectory time. The output of the neural network generates the motion primitives of the trajectories, as well as the duration or final time of a segment. Simulation results show extremely fast generation of dynamically-feasible trajectories by the proposed learning algorithm, which makes it suitable for real-time implementation.**

***Index Terms*—Keywords – Neural Networks, Supervised Learning, Trajectory Generation, Quadrotor UAVs.**

I. INTRODUCTION

Generating dynamically-feasible trajectories in real-time is essential for maneuvering quadrotors between an initial and a desired state. Depending on the nature of the generated trajectories, this task might be too computationally expensive, thus infeasible, to be executed in real-time. For example, time-optimal trajectories required around one hour to be computed in [1]. While a quadrotor might be able to follow the generated trajectories, there is no guarantee that it can recover from unexpected perturbations. A slight wind gust might hinder the quadrotor from reaching a desired state, and the controller might saturate the actuators while trying to stabilize the quadrotor about the original trajectory. Therefore, it is common practice to rely on real-time trajectory generation methods. To resolve this issue, the condition for time optimality is relaxed. Instead of trying to minimize the total duration between two desired states, an alternative can be generating smooth trajectories that satisfy minimum-jerk [2] or minimum-snap objectives [3], [4].

Alternatively, data-driven approaches can be used to learn certain trajectories and thus be able to generate them with less computational effort. In [5], an expert operator repeats a set of maneuvers for a remote controlled (RC) helicopter, and an apprenticeship learning algorithm is used to leverage these expert trials and learn a suitable control policy. However, this approach does not generalize to maneuvers other than those executed during the data collection procedure, which is expensive and time consuming. In [6], quadrotors are controlled using neural networks with two hidden layers that are trained using reinforcement learning, with the unique feature that the neural network directly predicts the control signals. This method is computationally attractive as it only requires 7 μs to compute a control signal, and the results show that the quadrotor can execute impressive dynamic maneuvers. However, end-to-end deep learning approaches do not provide stability guarantees, and the only way to validate such an approach is via statistical methods [7]–[9] and extensive testing in real-life scenarios, which can be prohibitively expensive and dangerous. In [10], a supervised learning method is used to generate paths between obstacles in two-dimensional (2D) maps. While the feasibility of the results can be checked for collisions, there is no guarantee of dynamic feasibility of the trajectories that are generated along this path.

A trajectory optimization approach was introduced in [11] by calculating optimal trajectories offline to train a neural network, which predicts trajectories for new initial state and cost function, and sparse quadratic programming is used to further optimize the prediction. A computationally efficient path-planning scheme that generates energy-optimal tours for quadcopters to visit all trees in a dense forest environment was proposed in [12], [13]; by leveraging optimal control theory, graph theory, and integer linear programming, the routes are chosen to tour the entire forest and return to a base station in an energy optimal fashion. Along similar lines, a time-optimal control problem was solved in [14] to ensure completely visiting and exploring all of the sites in surveillance missions. The authors employ convex optimization to derive a geometrical representation of the local minima for path headings, and

they solve for the optimal Dubin's maneuver corresponding to the global shortest path. A deep neural network (DNN)-based algorithm was proposed in [15] as an add-on module that improves the tracking performance of a classical feedback controller, which provides a tailored reference input to the controller based on their gained experience.

In this paper, we address the problem of generating real-time trajectories in 3D space without obstacles. Applications for this problem vary from aerial photography, to benchmarking controllers in indoor environments. Our proposed solution builds upon the success of relatively simple machine learning approaches. We utilize a neural network to approximate complex trajectory generation methods, which are typically nonlinear in terms of the input and constraints. In other words, we aim to encode the mapping between the inputs and outputs of deterministic trajectory generation algorithms. The inputs to the neural network are the initial and final desired states, and the outputs of the network are the motion primitives and the final time of the trajectory. Utilizing learning has a significant advantage in terms of computational efficiency, which becomes even more pronounced in the presence of a graphics processing unit (GPU) that can handle machine learning models in parallel computation, which enables the simultaneous evaluation of numerous trajectories in a single time step.

For our model of choice, simulation results show that it is possible to compute around 1000 trajectory motion primitives in less than $1\mu s$ on a standard personal computer (PC). Another key advantage of the proposed learning method is that its generated trajectories are verifiable, which is not possible with other learning approaches such as reinforcement learning. Since the neural network outputs motion primitives, the feasibility of the trajectories can be checked by the method described in [2].

The remainder of this paper is organized as follows. Section *II* introduces the quadrotor's coordinate systems, dynamic equations of motion, and existing trajectory generation methods. Section *III* describes the data generation and collection method. Section *IV* describes the procedure to train the proposed learning algorithm. Section *V* furnishes simulation results obtained on a validation dataset. Final thoughts and an outlook are provided in Section *VI*.

II. BACKGROUND

In this section, we introduce the quadrotor model and the trajectory generation method. The general approach can be summarized by the block diagram in Fig. 1. First, a trajectory generator is used in real-time, whose output consists of desired position, velocity, and acceleration. A controller then computes the desired forces and torques to track the desired trajectory. In general, a quadrotor control system is split into position and attitude controllers. Typical frequencies for attitude control are around $1KHz$, while position control can be carried out at $100Hz$, which is due to the fact that the attitude (rotational) dynamics of quadrotors are much faster than the translational dynamics.

A. COORDINATE FRAMES

The quadrotor has six degrees-of-freedom: $\{x, y, z, \phi, \theta, \psi\}$. x, y, and z are the translation coordinates, whereas the rotation angles ϕ, θ, and ψ correspond to roll, pitch, and yaw, respectively. Therefore, we define the state vector to be:

$$X^T = \begin{bmatrix} x & y & z & \phi & \theta & \psi & \dot{x} & \dot{y} & \dot{z} & \dot{\phi} & \dot{\theta} & \dot{\psi} \end{bmatrix}. \quad (1)$$

Similar to [3] and [16], we adopt the $Z-X-Y$ Euler angles for the rotation matrices. The rotation from body-fixed to the world coordinates systems is handled by a transformation matrix, R, given by:

$$R = \begin{bmatrix} c_\psi c_\theta - s_\phi s_\psi s_\theta & -c_\phi s_\psi & c_\psi s_\theta + c_\theta s_\phi s_\psi \\ c_\theta s_\psi + c_\psi s_\phi s_\theta & c_\phi c_\psi & s_\psi s_\theta - c_\psi c_\theta s_\phi \\ -c_\phi s_\theta & s_\phi & c_\phi c_\theta \end{bmatrix} \quad (2)$$

where letters c and s are used in short of cosine and sine, respectively.

Generating trajectories in this high dimensional space presents computational difficulties. However, the differential flatness property of quadrotors [17] allows us to encode the dynamics in the space of the vector of deferentially flat outputs:

$$\sigma(t) = \begin{bmatrix} x(t) & y(t) & z(t) & \psi(t) \end{bmatrix}^T, \quad (3)$$

which makes the trajectory generation problem simpler.

B. DYNAMICS

Quadrotor dynamics have been studied rigorously for different sizes and shapes as in [2], [3], and [18]. We use a simple nonlinear model that incorporates the thrusts, moments, and gyroscopic effects on a point-mass representation of the quadrotor frame. Denote by F_i and M_i the thrust and moment of each motor such that:

$$F_i = c_T \omega_{m-i}^2, \quad (4)$$

and

$$M_i = c_Q \omega_{m-i}^2, \quad (5)$$

where ω_{m-i} represents the speed of the i^{th} motor, and c_T and c_Q are the thrust and torque constants that approximately represent the aerodynamics effects on the blades of the quadrotor. It is worth noting that the quadratic relationship that maps the motor speeds to thrust forces and torques is the main source of nonlinearity in the model of multi-rotor aerial vehicles [16], [18], [19].

The equations of motion for translation and rotation are derived from Newton's second law of motion as:

$$m\ddot{r} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + R \begin{bmatrix} 0 \\ 0 \\ F_1 + F_2 + F_3 + F_4 \end{bmatrix}. \quad (6)$$

The rotational motion equations are:

$$I\frac{d\omega}{dt} + \omega \times I\omega = \tau = \begin{bmatrix} \tau_x \\ \tau_y \\ \tau_z \end{bmatrix}, \quad (7)$$

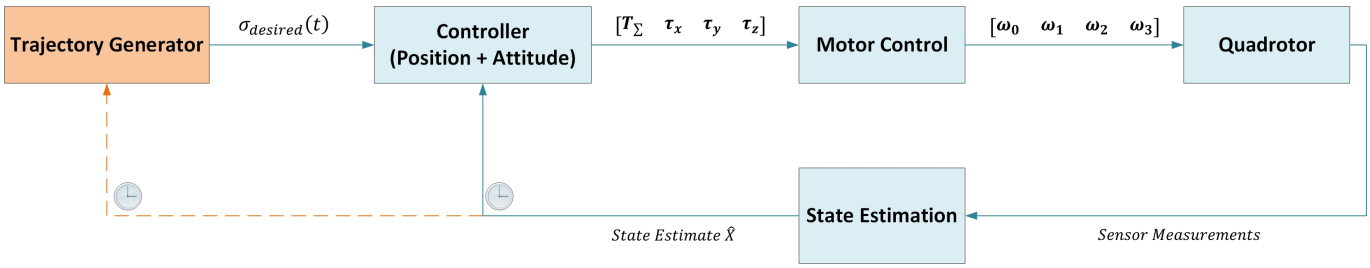

Fig. 1. Overall block diagram for a quadrotor motion control system. A trajectory generator computes the desired position. The desired position along with the current state estimates of the quadrotor are used by the position controller to compute the total thrust and desired orientation. The attitude controller computes the desired torques. The clocks represent different sampling times for the trajectory generator and controller.

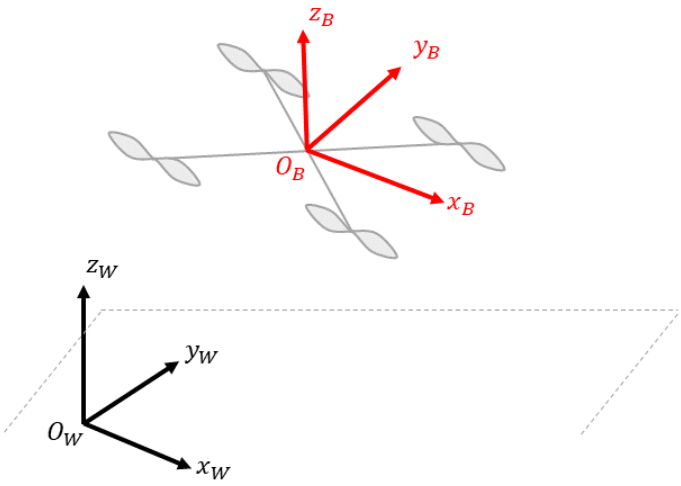

Fig. 2. Global and body frames of references of quadrotor in 3D space.

where I is the inertia tensor, which is a diagonal matrix due to the assumed symmetry of the quadrotor about each axis:

$$I = \begin{bmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z \end{bmatrix}. \tag{8}$$

τ is the torque vector generated by the motors about the axis of the body frame:

$$\tau = \begin{bmatrix} L(F_2 - F_4) \\ L(F_3 - F_1) \\ M_1 - M_2 + M_3 - M_4 \end{bmatrix}. \tag{9}$$

Since the motors saturate at a maximum speed, the torques and moments are limited by the following constraints:

$$\begin{aligned} 0 \leq f_{min} < F < f_{max}, \\ |\omega| \leq \omega_{max}. \end{aligned} \tag{10}$$

C. TRAJECTORY GENERATION

In this paper, we leverage the work of [20], which describes a method to generate minimum-jerk trajectories between initial and final states for a fixed time interval, T. Formally, we want to generate trajectories such that the L2-norm of the jerk is minimized:

$$min \quad \frac{1}{T}\int_0^T \|j(t)\|^2 dt \tag{11}$$

where the jerk j is the derivative of the acceleration a. This minimization is subject to the constraints in (10). Moreover, we would like to find the minimum final trajectory time, T, such that we are able to find a solution to the optimization problem in (11) without violating the specified constraints. The proposed solution finds decoupled quadratic polynomial trajectories for each axis as:

$$j_i(t) = \frac{1}{2}a_i t^2 + b_i t + c_i, \tag{12}$$

where a_i, b_i, and c_i are the motion primitives of each axis. After computing these primitives, we find the acceleration, velocity, and position by integration to obtain:

$$\begin{aligned} a_i(t) &= \frac{1}{6}a_i t^3 + b_i t^2 + c_i t + a_0, \\ v_i(t) &= \frac{1}{24}a_i t^4 + \frac{1}{6}b_i t^3 + c_i t^2 + a_0 t + v_0, \\ p_i(t) &= \frac{1}{120}a_i t^5 + \frac{1}{24}b_i t^4 + \frac{1}{6}c_i t^3 + a_0 t^2 + v_0 t + p_0. \end{aligned} \tag{13}$$

We will next show how these motion primitives along with the final time, T, can be efficiently computed using a supervised learning approach.

III. DATA COLLECTION

In order to train a neural network to generate trajectories, training data is first sampled from the proposed method of computing motion primitives in [2]. The input of the training data is a vector composed of the initial and final state:

$$X_i = \begin{bmatrix} r_{t_0} & \dot{r}_{t_0} & \ddot{r}_{t_0} & r_T & \dot{r}_T & \ddot{r}_T \end{bmatrix}. \tag{14}$$

The output is a vector of motion primitives along with the optimal final time, T:

$$y_i = \begin{bmatrix} a_x & b_x & c_x & a_y & b_y & c_y & a_z & b_z & c_z & T \end{bmatrix}. \tag{15}$$

Note that it is challenging to sample feasible states from this high dimensional input (18 variables). To avoid selecting infeasible data points, we randomize positions of the initial and final states, along with segment final times. For these samples, the velocity and acceleration are set to $[0,0,0]^T$, i.e. the quadrotor is at rest. After generating a trajectory between

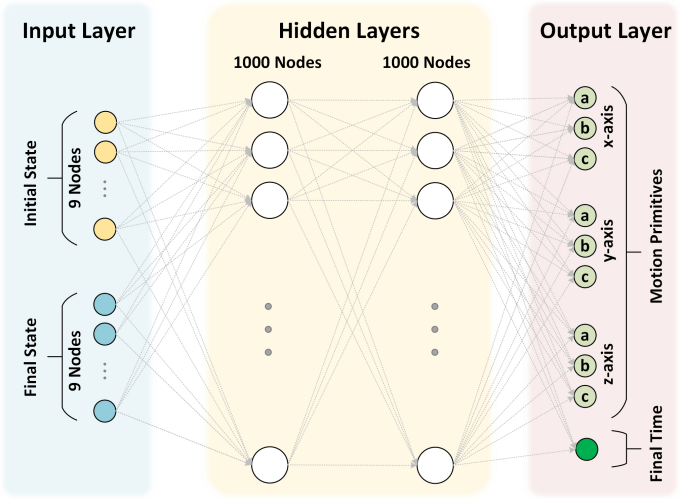

Fig. 3. Fully connected neural network architecture used for computing motion primitives and final trajectory time. The input layer is formed of the initial and final states described by position, velocity, and acceleration. The two hidden layers have 1000 nodes each. The network has around one million trainable parameters.

the initial and desired positions, we randomly sample two time instants, t_A and t_B, along this trajectory such that:

$$0 \leq t_A < t_B < T. \tag{16}$$

A new set of motion primitives is computed for the new datapoint. To avoid biasing the neural network, we keep the number of states at rest equal to the number of states in motion. Hence, we randomly choose, with probability $p = 0.5$ to have a balanced distribution between states at rest and states in motion, to re-sample the initial and final states.

For each training example, we compute the minimum final time, T, using binary search. We choose an upper-bound $T_{max} = 15s$ and a lower-bound $T_{min} = 0.1s$. The initial guess, T_{guess}, for T is set as the midpoint of T_{max} and T_{min}. We check if the trajectory generated between the selected states is feasible for the current guess of the optimal trajectory time. If the current time is feasible, we set $T_{max} = T_{guess}$. On the other hand, if T_{guess} is not feasible, we set $T_{min} = T_{guess}$. By repeating this process several times, the true optimal time for the trajectory is bounded between T_{max} and T_{min}. The loop is terminated when:

$$T_{max} - T_{min} = \alpha, \tag{17}$$

where α is a tolerance value. We sample one million trajectories for training and validation. We also sample another dataset consisting of 100,000 trajectories for testing after training is finished. It is noted here that the gradient descent method described in [4] can be used as an alternative for selecting the final time.

IV. ALGORITHM TRAINING

Before training the machine learning algorithm, we normalize and standardize the training data, which is a particularly important step for the fidelity of the output data given that the scale of the outputs varies by two orders of magnitude. Normalization and standardization help with avoiding imbalanced gradients during training, and they also remove hockey stick learning curves by eliminating the need to learn any biases in the output vector. We train a fully connected neural network with two hidden layers each consisting of $N_{hidden} = 1000$ hidden nodes, as shown in Fig. 3. While tuning the hyperparameters, it is noticed that increasing the number of hidden layers and the number of nodes per hidden layer results in lower losses and higher accuracies. However, the final values for these parameters are chosen to strike a balance between the accuracy of predictions and the computational power required. This is important especially in the case of a quadrotor, where the computations are executed on-board. The first two layers have rectified linear units (”ReLu”s) [21] as activation functions for the two hidden layers, and the final layer has a linear activation function. A similar neural network was able to learn complex end-to-end control policies in [22] and [6]. We choose to minimize the mean absolute error (MAE) between the trajectories as the loss function to be minimized:

$$MAE = \frac{1}{N}\sum_{i=1}^{N} |y_i - \hat{y}_i| \tag{18}$$

We use Adam optimizer [23] with learning rate, $lr = 1 \times 10^{-4}$, with a decay of 2×10^{-6}. Training is carried for 250 epochs, and we perform 5-fold cross validation by randomly partitioning the data into five sets. For five training iterations, we keep one partition for validation and train on the rest of the data. Training is stopped when the training and validation losses consistently plateau around 0.0028. After cross validation, the neural network is trained again for 250 epochs over the entire training dataset. All hyperparameters are summarized in the Table I.

Training Hyperparameters	
Number of hidden layers	2
Number of nodes in hidden layers	1000
1^{st} and 2^{nd} layer activations	Relu
Final layer activation	Linear
Batch size	256
Learning rate	10^{-4}
Rate decay	10^{-6}
Number of epochs	250
Optimizer	Adam
Cost function	Mean absolute error

TABLE I
THE FINAL SELECTED TRAINING HYPERPARAMETERS.

V. RESULTS

Training the neural network converges to a mean absolute error loss of $MAE = 0.0028$. This loss shows that the error is very small between the true labels and the predicted values. To get a better understanding of the output, we visualize the results by plotting the jerk, acceleration, velocity, and position along each axis for the true and predicted values. A sample of the results is shown in Fig. 4. The plotted polynomials show extremely close match to the true labels (almost identical)

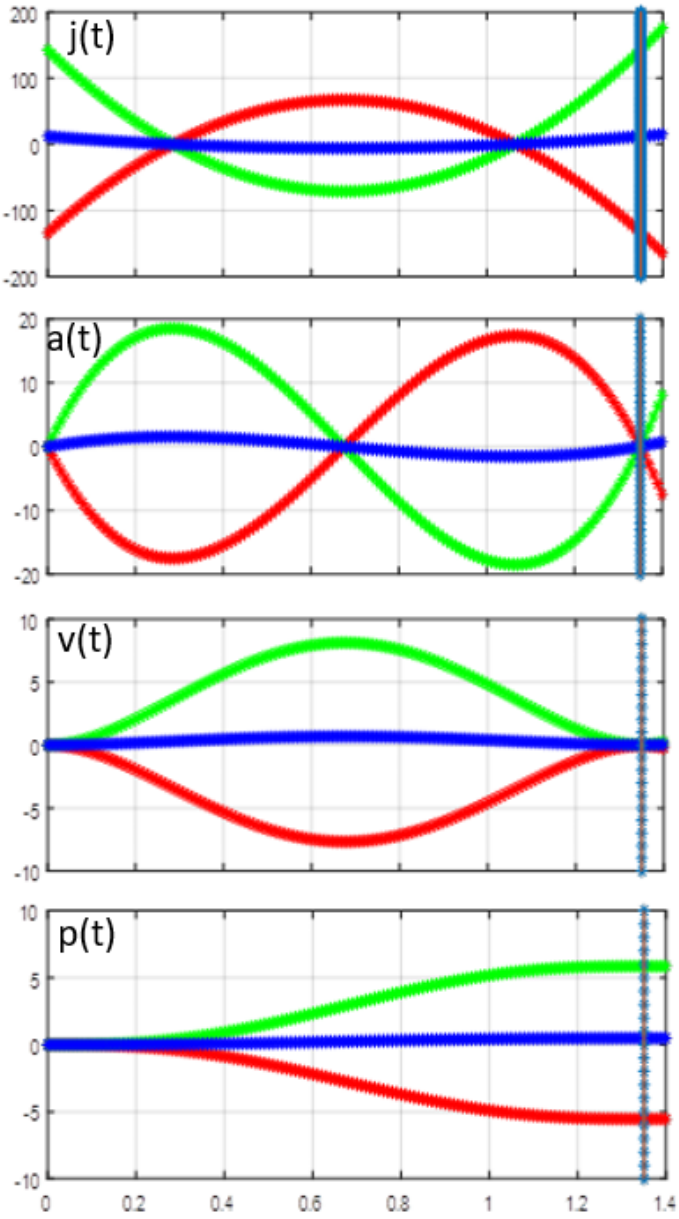

Fig. 4. From top to bottom: jerk(m/s^3), acceleration (m/s^2), velocity (m/s), and position (m) versus time (s) for the true and predicted motion primitives for a single test datapoint. Continuous lines represent true labels, while crosses represent predicted values. Red, green, and blue plots represent values for x, y, and z, respectively. All variables are in SI units. The vertical line on 1.37 seconds represents the predicted final time of the trajectories.

along the entire duration of the trajectory. In fact, the predicted final trajectory time, T, is only about 0.5 ms away from the true label. Also, note that the trajectory generated converges to the desired state at the estimated final time. For the particular example shown in Fig. 4, the final accelerations and velocities are zeros, and the final positions are $< 1mm$ away from the desired positions on each axis.

To estimate how long each set of motion primitives and final time requires to be computed, a set of 100,000 trajectories are sequentially tested. Using a standard PC, the average time to compute a single trajectory is around $0.4\mu s$. Moreover, each 1024 datapoints can be computed in one batch with an average time of $0.64\mu s$.

A final remark is that even though the predictions of the neural network are very accurate, some of the generated trajectories are slightly infeasible. To address this issue, we utilize the predicted final trajectory time, T, as the initial guess for the sequential method described in the data collection section (similar approach in [10], and we notice that by adding $1ms$ to the final time, almost all generated trajectories ($> 99\%$) become dynamically feasible. This is summarized in Fig. 5.

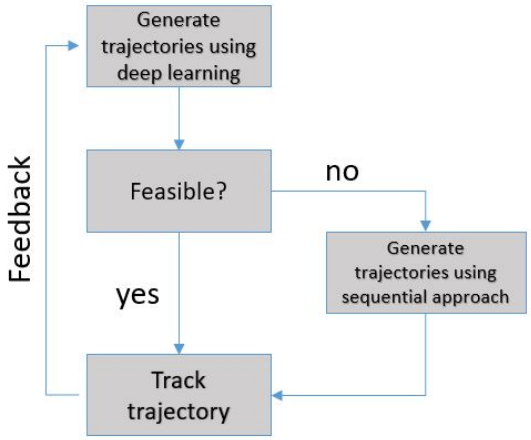

Fig. 5. Flow chart for real-time feasibility checking of the trajectories generated using the proposed learning algorithm.

VI. CONCLUSION

We present a method for generating dynamically-feasible polynomial trajectories using supervised learning. After generating millions of trajectories from a well-established state-of-the-art trajectory generation method, the data is used to train a neural network to predict motion primitives along with minimum final trajectory time per segment. Results show very accurate predictions of motion primitives as well as final time. One of the main advantages for using our proposed method is that it enables checking the feasibility of the trajectories being generated, since it directly computes motion primitives. This is a property that is missing from several learning approaches, such as deep reinforcement learning. Since the neural network is able to learn a complex trajectory generation method, we can train similar networks for predicting more complex trajectories. A good alternative is time-optimal trajectories, which take long time to compute and therefore cannot be used for real-time applications. On the other hand, the current method suffers from its inability to incorporate obstacles and corridor constraints. Properly representing cluttered environments in the training data can allow this method to be extended to

new applications such as trajectory generation for fleets of quadrotors.

ACKNOWLEDGMENT

This work is supported by the University Research Board (URB) at the American University of Beirut (AUB).

REFERENCES

[1] M. Hehn, R. Ritz, and R. DAndrea, "Performance benchmarking of quadrotor systems using time-optimal control," *Autonomous Robots*, vol. 33, no. 1-2, pp. 69–88, 2012.

[2] M. Hehn and R. D'Andrea, "Quadrocopter trajectory generation and control," *IFAC proceedings Volumes*, vol. 44, no. 1, pp. 1485–1491, 2011.

[3] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 2520–2525.

[4] C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," in *Robotics Research*. Springer, 2016, pp. 649–666.

[5] P. Abbeel, A. Coates, and A. Y. Ng, "Autonomous helicopter aerobatics through apprenticeship learning," *The International Journal of Robotics Research*, vol. 29, no. 13, pp. 1608–1639, 2010.

[6] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, "Control of a quadrotor with reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2096–2103, 2017.

[7] S. Webb, T. Rainforth, Y. W. Teh, and M. P. Kumar, "A statistical approach to assessing neural network robustness," 2018.

[8] M. Althoff and J. M. Dolan, "Online verification of automated road vehicles using reachability analysis," *IEEE Transactions on Robotics*, vol. 30, no. 4, pp. 903–918, 2014.

[9] M. O'Kelly, A. Sinha, H. Namkoong, R. Tedrake, and J. C. Duchi, "Scalable end-to-end autonomous vehicle testing via rare-event simulation," in *Advances in Neural Information Processing Systems*, 2018, pp. 9827–9838.

[10] T. Watanabe and E. N. Johnson, *Trajectory Generation using Deep Neural Network*. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2018-1893

[11] G. Tang, W. Sun, and K. Hauser, "Learning trajectories for real-time optimal control of quadrotors," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2018, pp. 3620–3625.

[12] C. Aoun, N. Daher, and E. Shammas, "An energy optimal path-planning scheme for quadcopters in forests," in *2019 IEEE 58th Conference on Decision and Control (CDC)*, 2019, pp. 8323–8328.

[13] C. Aoun, E. Shammas, and N. Daher, "Energy-optimal tours for quadrotors to scan moth-infested trees in densely-packed forests," in *2020 American Control Conference (ACC)*, July 2020, pp. 58–63.

[14] M. Tuqan, N. Daher, and E. Shammas, "A simplified path planning algorithm for surveillance missions of unmanned aerial vehicles," in *2019 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, July 2019, pp. 1341–1346.

[15] Q. Li, J. Qian, Z. Zhu, X. Bao, M. K. Helwa, and A. P. Schoellig, "Deep neural networks for improved, impromptu trajectory tracking of quadrotors," in *2017 IEEE International Conference on Robotics and Automation*. IEEE, 2017, pp. 5183–5189.

[16] R. Mahony, C. Kumar, and P. Vijay, "Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor," *IEEE Robotics and Automation Magazine*, vol. 19, no. 3, p. 2032, 2012.

[17] R. M. Murray, M. Rathinam, and W. Sluis, "Differential flatness of mechanical control systems: A catalog of prototype systems," in *ASME international mechanical engineering congress and exposition*. Citeseer, 1995.

[18] G. Hoffmann, H. Huang, S. Waslander, and C. Tomlin, "Quadrotor helicopter flight dynamics and control: Theory and experiment," in *AIAA guidance, navigation and control conference and exhibit*, 2007, p. 6461.

[19] P. Pounds, R. Mahony, and P. Corke, "Modelling and control of a large quadrotor robot," *Control Engineering Practice*, vol. 18, no. 7, pp. 691–699, 2010.

[20] M. W. Mueller, M. Hehn, and R. D'Andrea, "A computationally efficient motion primitive for quadrocopter trajectory generation," *IEEE Transactions on Robotics*, vol. 31, no. 6, pp. 1294–1310, 2015.

[21] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.

[22] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[23] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.