2019 International Conference on Robotics and Automation (ICRA)
Palais des congres de Montreal, Montreal, Canada, May 20-24, 2019

Real-Time Minimum Snap Trajectory Generation for Quadcopters: Algorithm Speed-up Through Machine Learning

Marcelino M. de Almeida, Rahul Moghe and Maruthi Akella[1]

***Abstract*— This paper addresses the problem of generating quadcopter minimum snap trajectories for real time applications. Previous efforts addressed this problem by either employing a gradient descent method, or by greatly sacrificing optimality for faster solutions that are amenable for onboard implementation. In this work, outputs of the gradient descent method are used offline to train a supervised neural network. We show that the use of neural networks results typically in two orders of magnitude reduction in computational time. Our proposed approach can be used for warm-starting onboard implementable iterative methods with an "educated" initial guess. This work is motivated by the application for human-machine interface in which a human provides desired trajectory through a smart-tablet interface, which has to be translated into a dynamically feasible trajectory for a quadcopter. The proposed solution is tested in thousands of different examples, demonstrating its effectiveness as a booster for minimum snap trajectory generation for quadcopters.**

I. Introduction

Real time path planning for quadcopters have been subject of extensive research in the last decade. This is typically an arduous task due to the fact that quadcopters are nonlinear underactuated systems. Some prior literature that explores trajectory generation for quadcopters include the use of learning from demonstration [1], sequential composition with parameter adaptation [2], reinforcement learning and iterative adaptation [3], reachable sets [4], among others. However, the solution that has been most extensively used in literature for trajectory generation stems from the work of [5], which uses a set of truncated basis functions to plan a trajectory that satisfies waypoint specifications by solving a quadratic programming problem to minimize translational snap (fourth derivative of position).

The solution of [5] stands out among other solutions in the literature due to its flexibility, optimality, and simplicity. The work in [5] showed that quadcopters are *differentially flat* systems [6] for outputs $\boldsymbol{y} = \begin{bmatrix}\boldsymbol{r}^T & \psi\end{bmatrix}^T$, where $\boldsymbol{r}$ is the vehicle's position, and ψ is the vehicle's heading. It has been proven that the states and inputs of a quadcopter can all be uniquely determined from given values of $\boldsymbol{r}$, $\dot{\boldsymbol{r}}$, $\ddot{\boldsymbol{r}}$, $\boldsymbol{r}^{(3)}$, $\boldsymbol{r}^{(4)}$, ψ, $\dot{\psi}$ and $\ddot{\psi}$. Then, one can plan trajectories in the output space and then map them back into the original space [6]. The work of [5] proposed a path planner that minimizes $\boldsymbol{r}^{(4)}$ (translational snap) and $\ddot{\psi}$ (heading acceleration). This method is commonly known as the minimum snap planner for quadcopters.

The minimum snap problem statement [5] can be summarized as follows: given initial and final states for a quadcopter, and given a set of ordered intermediate waypoints for it to traverse, find the solution that goes through all the waypoints minimizing the integral of the two-norm of snap, arriving at its final state at a prescribed time T.

A key component for solving the minimum-snap algorithm is to determine the optimal time t_i to go through every intermediate waypoint $\boldsymbol{P}_i$ (time allocation problem). Solving for the time allocation makes the problem non-convex, requiring the use of nonlinear programming methods.

When solutions are not needed in real time[1], or if the number of waypoints is low (say, less than 5), the time allocation problem can be quickly solved through gradient descent methods until convergence to a local minimum [5][7][8][9]. These methods typically start from a naive initial guess, such as based on a trapezoidal velocity profile [9].

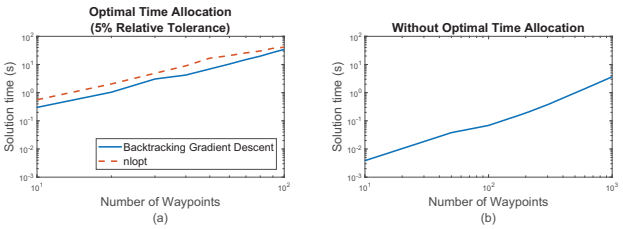

Fig. 1. (a) Minimum snap solution time for Backtracking Gradient Descent and for NLOPT. (b) Minimum snap solution time when time allocation is fixed. Solutions obtained using an Intel Core i5-4690K - 3.5GHz.

Figure 1 (a) shows the computational time that it takes to solve minimum snap with time allocation using either the Backtracking Gradient Descent method from [5] or the NLopt solver [10]. Clearly, the solution time scales poorly with the number of waypoints, preventing either method from being a real-time generalized solution for large number of waypoints. This contrasts with the solution time for minimum snap using fixed time allocation (see Figure 1 (b)[2]), which shows that the solution without optimal time allocation is typically obtained two orders of magnitude faster than the gradient descent method. Hence, if the optimal time allocations are known a priori (Figure 1 (b)) then the optimization problem is solved 100 times faster than when there is need to solve for those (Figure 1 (a)).

[1]The authors are with Department of Aerospace Engineering and Engineering Mechanics, University of Texas at Austin, Austin, TX 78712, USA marcelino.malmeidan@utexas.edu, rahul.moghe@utexas.edu, makella@mail.utexas.edu

[1]For purposes of this paper, we take real time planners to be those that can be calculated at 10Hz or faster.

[2]These solutions were obtained with a modified version of the software in https://github.com/ethz-asl/mav_trajectory_generation, which was optimized by exploiting sparsity.

978-1-5386-6027-0/19/$31.00 ©2019 IEEE

When minimum snap is required to be calculated in real time, common approaches tend to loosen the problem constraints or sacrifice optimality to promote faster solutions. In [11], the authors use a kinodynamic planner to find $\boldsymbol{P}_i$ together with t_i, and uses this time allocation without performing further optimizations. The work in [12] starts from a trapezoid velocity profile and increases time allocations until the computed solutions satisfies bounds on maximum velocity, acceleration and jerk. The solution in [13] does not optimize the time segments, but allows the intermediate waypoints $\boldsymbol{P}_i$ to change within a bounded safe region, allowing for an implicit time optimization (although limited by the safe region).

In this paper, we show that feedforward Neural Networks (NN) can be trained to learn and determine the optimal time segments. Apart from [11] (which precludes the use of planners other than the kinodynamic one), none of the aforementioned methods provide a definitive approach to determine good initial guesses for the time allocation. Hence, a well-trained NN-based solution has the potential to significantly complement currently available techniques by quickly providing a mathematically interpolated initial guess for the time allocations. In this paper, we show how to exploit certain invariance properties of the minimum snap problem to train Neural Networks that can work for any set of input waypoints. Although our work focuses on minimum snap problems, the methodology herein can be easily adapted to the minimization of different derivatives of position (velocity, acceleration, jerk, etc).

A. Motivation

Although most of the prior literature that uses minimum snap focuses on increasing autonomy of a quadcopter, this work is strongly motivated by the need for allowing real-time human interface for quadcopter flight missions. If a human operator commands a quadcopter by sketching a 2D trajectory on a tablet, this trajectory needs to be transformed into a dynamically feasible trajectory, which can be accomplished using minimum snap interpolation. However, the number of points in a hand-drawn trajectory (spatial discretization) is usually considerable, and the solution time for such a problem can usually take minutes, which is unacceptable for most real-time applications (see Figure 1 (a)). Also, waypoints too close to each other (spatially) over-constrain the problem, making it hard to generate dynamically feasible trajectories.

In light of the aforementioned major technical challenges, we compress the hand-drawn trajectory until it is reduced to a set of waypoints that robustly maintains the structure of the original trajectory. To accomplish such a reasonable compression, we adopt the algorithm of [14], which was originally developed for extracting keyframes from motion capture system data.

B. Structure of the paper

The paper is organized as follows: Section II presents the dynamical model for a quadcopter, highlighting its properties of differential flatness. Section III presents the minimum snap problem formulation, its solution, and some properties therein. Section IV presents our technical approach to render the problem within the framework of neural networks. Section V presents results and performance analysis of our method. Section VI provides some final considerations, and Section VII presents conclusions to this work.

II. Equations of Motion and Differential Flatness

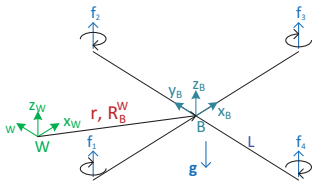

Fig. 2. Quadcopter and its frames.

This work considers a dynamic model based on the coordinate frames from Fig. 2, involving a fixed world frame W and a moving body frame B, with $\boldsymbol{z}_W$ parallel to the gravity vector $\boldsymbol{g}$. The transformation from body frame B to world frame W is parametrized by the translation $\boldsymbol{r}$, and the rotation matrix $\boldsymbol{R}_B^W$.

The input vector for this system is composed by the quadcopter's total thrust T and body moments $\boldsymbol{\tau} = \begin{bmatrix}\tau_1 & \tau_2 & \tau_3\end{bmatrix}^T$. Thus, the control input for the system is defined as $\boldsymbol{u} = \begin{bmatrix}T & \boldsymbol{\tau}^T\end{bmatrix}^T$. The combination of these quantities can be linearly mapped into thrust applied by each propeller $f_i, i \in \{1,2,3,4\}$.

Assuming that the vehicle's center of mass is located at the origin of B, the translational motion can be described as:

$$m\ddot{\boldsymbol{r}} = T\boldsymbol{z}_B - mg\boldsymbol{z}_W \tag{1}$$

where g is the norm of the gravity vector $\boldsymbol{g}$, and m is the vehicles's mass.

Defining the angular velocity of the quadcopter's body with respect to the world frame as $\boldsymbol{\omega}_{BW} = p\boldsymbol{x}_B + q\boldsymbol{y}_B + r\boldsymbol{z}_B$, the rotational dynamics is described as follows:

$$\dot{\boldsymbol{R}}_B^W = \boldsymbol{R}_B^W \left[\boldsymbol{\omega}_{BW\times}\right], \tag{2}$$

$$\dot{\boldsymbol{\omega}}_{BW} = \boldsymbol{J}^{-1}\left(-\boldsymbol{\omega}_{BW} \times \boldsymbol{J}\boldsymbol{\omega}_{BW} + \boldsymbol{\tau}\right), \tag{3}$$

where $\boldsymbol{J}$ is the inertia tensor matrix referenced to the vehicle's center of mass along the axes of the body frame B.

As shown in [5] and [15], quadcopters are *differentially flat* systems [6] for outputs $\boldsymbol{y} = \begin{bmatrix}\boldsymbol{r}^T & \psi\end{bmatrix}^T$, where ψ is the vehicles yaw angle, or heading. The work in [5] shows that the states $\boldsymbol{r}$, $\dot{\boldsymbol{r}}$, $\boldsymbol{R}_B^W$, $\boldsymbol{\omega}_{BW}$, and inputs $\boldsymbol{u}$ can all be uniquely determined from given values of $\boldsymbol{r}$, $\dot{\boldsymbol{r}}$, $\ddot{\boldsymbol{r}}$, $\boldsymbol{r}^{(3)}$, $\boldsymbol{r}^{(4)}$, ψ, $\dot{\psi}$ and $\ddot{\psi}$. Then, one can plan trajectories in the output space and then map them back into the original space [6] with the state variables of Eqs. 1 to 3. The plan is made upon the following linear model [15]:

$$\begin{bmatrix}\boldsymbol{r}^{(4)} \\ \ddot{\psi}\end{bmatrix} = \begin{bmatrix}\boldsymbol{w}_{\boldsymbol{r}} \\ w_\psi\end{bmatrix}, \tag{4}$$

where $\boldsymbol{w} \triangleq \begin{bmatrix}\boldsymbol{w}_{\boldsymbol{r}}^T & w_\psi\end{bmatrix}^T \in \mathbb{R}^4$ is the input vector in the flat space. The remainder of this paper assumes constant yaw angle ($\dot{\psi} = \ddot{\psi} = w_\psi = 0$), without loss of generality. Thus, it plans using only the snap input $\boldsymbol{w_r}$. This paper pursues generating minimum snap trajectories, which seeks to minimize the cost function:

$$J = \int_0^T \left\|\boldsymbol{r}^{(4)}(t)\right\|^2 dt = \int_0^T \left\|\boldsymbol{w_r}(t)\right\|^2 dt. \tag{5}$$

III. The Minimum Snap Algorithm

This section briefly reviews the solution of [5] for generating minimum snap trajectories. This algorithm solves the problem of starting at an initial point $\boldsymbol{P}_0$ at time $t_0 = 0$ and passing through *m waypoints* $\boldsymbol{P}_k, k \in \{1, 2, \cdots, m\}$, reaching the final state $\boldsymbol{P}_m$ at time $t_m = T$.

The minimum snap algorithm solves separately for each cartesian degree of freedom x, y and z. The trajectory for each of these states, denoted as $\sigma_i(t), i \in \{x, y, z\}$, is written as piecewise polynomials of order n over m time intervals:

$$\sigma_i(t) = \begin{cases} \sum_{j=0}^n a_{j,1} t^j, & t_0 \le t < t_1 \\ \sum_{j=0}^n a_{j,2} t^j, & t_1 \le t < t_2 \\ \vdots & \\ \sum_{j=0}^n a_{j,m} t^j, & t_{m-1} \le t \le t_m \end{cases}, \tag{6}$$

This problem can be written in the following quadratic programming form [5]:

$$\begin{cases} \min\limits_{\boldsymbol{a}, \boldsymbol{\delta t}} & \mathbb{J}(\boldsymbol{\delta t}) = \boldsymbol{a}^T \boldsymbol{Q} \boldsymbol{a} \\ s.t. & \boldsymbol{A}_{eq} \boldsymbol{a} = \boldsymbol{b}_{eq} \\ & \boldsymbol{A}_{ineq} \boldsymbol{a} \le \boldsymbol{b}_{ineq} \end{cases}, \tag{7}$$

where $\boldsymbol{a} \triangleq \begin{bmatrix}\boldsymbol{a}_1^T & \cdots & \boldsymbol{a}_m^T\end{bmatrix}^T$ is the vector of solution variables, and $\boldsymbol{a}_i \triangleq \begin{bmatrix}a_{0,k} & \cdots & a_{n,k}\end{bmatrix}^T$ is the vector of polynomial coefficients for the interval $t_{k-1} \le t < t_k$. The matrices $\boldsymbol{Q}$, $\boldsymbol{A}_{eq}$ and $\boldsymbol{A}_{ineq}$ are function of the time allocation $\boldsymbol{\delta t} \triangleq \begin{bmatrix}\delta t_1 & \cdots & \delta t_m\end{bmatrix}^T$, where $\delta t_k \triangleq t_k - t_{k-1}$. Equality constraints in (7) are used to constrain position, velocity, orientation, or angular velocity through waypoints, as well as enforcing continuity of $\sigma_i(t)$ and its derivatives. Inequality constraints are added to enforce safety constraints, such as collision avoidance, maximum velocity, or maximum acceleration.

For a fixed time allocation $\boldsymbol{\delta t}$, the problem in Eqs. 7 reduces to a linear quadratic programming problem, which can be solved in polynomial time. On the other hand, if $\boldsymbol{dt}$ is free, the optimal time allocation problem can be stated as:

$$\begin{cases} \min\limits_{\boldsymbol{\delta t}} & \mathbb{J}(\boldsymbol{\delta t}) \\ s.t. & \sum \delta t_k = t_m \\ & \delta t_k \ge 0 \end{cases} \tag{8}$$

Assuming that (i) the problem has only position equality constraints in the waypoints, (ii) it has no inequality constraints, and (iii) that the vehicle starts from rest and finishes at rest.

Property 1 - If we solve (8) with $t_m = T$ and find a trajectory $\sigma_i(t)$ with derivatives $\dot{\sigma}_i(t)$, $\ddot{\sigma}_i(t)$, ..., $\sigma_i^{(N)}(t)$, then the optimal trajectory for $t_m = T/\alpha$, for $\alpha > 0$, is $\sigma_i(\alpha \cdot t)$ with derivatives $\alpha\dot{\sigma}_i(\alpha \cdot t)$, $\alpha^2\ddot{\sigma}_i(\alpha \cdot t)$, ..., $\alpha^N \sigma_i^{(N)}(\alpha \cdot t)$, for $0 \le t \le T/\alpha$ (see [5]).

Property 2 - If we solve (8) with $t_m = T$ and find a solution $\boldsymbol{\delta t} = \boldsymbol{\delta t}^*$ and optimal cost $\mathbb{J} = \mathbb{J}^*$, then the solution for the same problem with $t_m = \alpha T$ is $\boldsymbol{\delta t} = \alpha \cdot \boldsymbol{\delta t}^*$ and has optimal cost $\mathbb{J} = \alpha^{-7}\mathbb{J}^*$. In other words, the optimal time allocation scales linearly with the final time t_m, but the optimal cost scales with the inverse of the 7-th power of the final time.

Property 3 - If we solve (8) for waypoints $\boldsymbol{P}_k, k \in \{0, 1, \cdots, m\}$ and find a solution $\boldsymbol{\delta t} = \boldsymbol{\delta t}^*$, then the solution for the problem with waypoints $s\boldsymbol{H}\boldsymbol{P}_k, k \in \{0, 1, \cdots, m\}$, where $s > 0$ is a scaling factor and $\boldsymbol{H}$ is a homogeneous transformation, will also have optimal time allocation $\boldsymbol{\delta t} = \boldsymbol{\delta t}^*$. In other words, the optimal time allocation is invariant to scaling and homogeneous transformations on the waypoints.

IV. Neural Network Methodology

This section presents our methodology to render the minimum snap time allocation problem within the framework of a supervised feedforward Neural Network. The goal is to train a NN offline that takes as inputs a set of waypoints, and produces a set of time allocations as outputs. Since the main objective of this work is to transform user-provided trajectories into quadcopter feasible trajectories, the approach herein presented is developed to work with 2D waypoints. At the same time, we note that our approach can be readily extended to 3D cases as well, which would be more applicable to Minimum Snap fitting combined with sampling-based methods[8][9][12], instead of the human-machine interface explored in this work. In 3D, the training set can be obtained from the solution of multiple path planning problems using, for instance, sampling-based methods [16][17][18][19].

This section is divided as follows: Subsection IV-A discusses about the datasets that were used to train the NN. Subsection IV-B presents the algorithm that is used to decimate hand-drawn trajectories into a set of keyframes. Subsection IV-C introduces a methodology for pre-processing the data such that the solution exploits the minimum snap *properties* of Section III. Subsection IV-D presents the Neural Network architecture that is used to learn optimal time allocations, and Subsection IV-E summarizes the algorithm pipeline.

A. Data Collection

Properly training a NN often requires a *hefty* amount of data. The quantity of data needed is typically a function of the problem itself, but our experience shows that it is enough to use some ten-thousands of data points. In addition, a crucial practice when training a NN is to have different datasets to train, to validate, and to test. The present work uses the SIGN dataset [20] for training and validation, while the Intuidoc-Loustic Gestures DataBase (ILGDB) [21] is used for testing.

Both datasets comprise of single-stroke pen gestures. The SIGN dataset has 33154 tablet hand-drawn gestures, while

ILGDB has 6627. The reason to use different datasets is to demonstrate that one can train a NN with data obtained from a particular dataset and have it still working with independently collected data. In fact, the SIGN dataset comprises of mostly simple and smooth gestures, while the ILGDB has a wider variety of trajectories (see [20] and [21] for description of their gestures).

B. Curve Decimation

As mentioned in Section I-A, hand drawn trajectories are decimated into a set of keyframes. This speeds up the minimum snap calculation (by reducing the number of waypoints), and also avoids over-constraining the trajectory. The approach employed by this work uses a decimation procedure [14] that ranks importance of all points in a space, then iteratively removes the least important one.

Assuming that the initial curve has N points $\{\boldsymbol{r}_1, \boldsymbol{r}_2 \cdots \boldsymbol{r}_N\}$, the algorithm in [14] always preserve the points $\boldsymbol{r}_1$ and $\boldsymbol{r}_N$. Given that the desired final keyframes are expected to deviate from the initial curve by at most D_{max} and the final set has at least N_{min} points, the method is described below:

Step 1. For $i = 2$ to $i = N-1$, for each i, define the line that connects $\boldsymbol{r}_{i-1}$ to $\boldsymbol{r}_{i+1}$ as $\boldsymbol{l}(i)$.

Step 2. Calculate the importance $I(i)$ of the point $\boldsymbol{r}_i$ as being the distance between $\boldsymbol{r}_i$ and the line $\boldsymbol{l}(i)$:

$$I(i) = D\left(\boldsymbol{r}_i, \boldsymbol{l}(i)\right). \tag{9}$$

Step 3. Define k as the index for the least important point. If $\boldsymbol{l}(k) \geq D_{max}$, then terminate the algorithm. Otherwise, decimate the least important point $\boldsymbol{r}_k$. Set $N = N-1$.

Step 4. If $N > N_{min}$, return to Step 1. Otherwise, terminate the algorithm.

Figure 3 shows some examples of original curves from the databases along with their decimated counterparts. The termination criterion for those examples was to have $N_{min} =$ 30 points in the final curve.

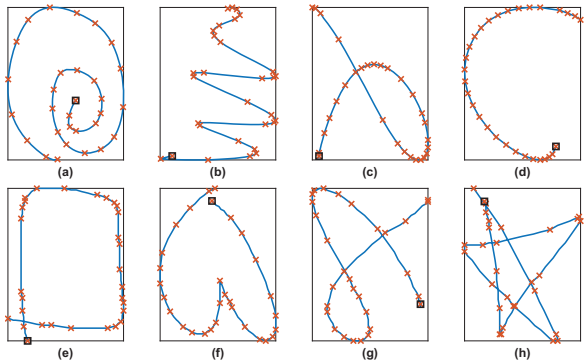

Fig. 3. Sample trajectories from the datasets (blue solid curves starting at black square) and their corresponding decimated counterparts (red crosses). (a) to (d) are examples from SIGN, while (e) to (h) come from ILGDB.

C. Data Pre-Processing

Training a Supervised Neural Network requires an input set of data $\mathbb{X} = \{\mathbb{X}_1, \cdots, \mathbb{X}_n\}$ and an output set of data $\mathbb{O} = \{\mathbb{O}_1, \cdots, \mathbb{O}_n\}$. Each $\mathbb{X}_j$ represents a set of $m+1$ waypoints, and each $\mathbb{O}_j$ represents a set of m time allocations. When trained, a Neural Network is expected to work only within the range of values for which it was trained. In order to train a NN for finding the optimal $\boldsymbol{\delta t}$'s, it is possible to invoke the properties from Section III to keep inputs and outputs of the training data within compact sets, allowing to use the trained NN for any set of inputs.

The output sets $\mathbb{O}_j$ can be obtained offline by solving minimum snap through gradient descent for every set of input waypoints. The optimal outputs were obtained through the Backtracking Gradient Descent (BGD) method from [5] with termination criterion of 1% relative error. The BGD method is sensitive to the initial guess, and it can be driven to local minima.

In order to keep the possible values of the time allocations δt_i within a compact set, every problem is solved for $t_m = 1$ seconds. This way, each δt_i can be interpreted as the fraction of time that the vehicle should spend between waypoints $\boldsymbol{P}_{i-1}$ and $\boldsymbol{P}_i$. Defining $\boldsymbol{\delta t}_m$ as the solution for $t_m = 1$, then $\boldsymbol{\delta t} = T \cdot \boldsymbol{\delta t}_m$ is the solution for an arbitrary $t_m = T$ (see Property 2 in Section III).

A naive way to populate the set $\mathbb{X}$ would be to fill it with x_i and y_i positions of every waypoint $\boldsymbol{P}_i = \begin{bmatrix} x_i & y_i \end{bmatrix}^T$. However, Property 3 from Section III states that the optimal time allocation $\boldsymbol{\delta t}$ is invariant to homogeneous transformations on the waypoints. By populating $\mathbb{X}$ with the waypoints $\boldsymbol{P}_i$, then one has to expect that the NN will "learn" the invariance property.

Instead, we find it more suitable to populate $\mathbb{X}$ with a waypoint representation that is agnostic to homogeneous transformations, such as the range-angle parametrization. The range vector $\boldsymbol{r} \triangleq \begin{bmatrix} r_1 & \cdots & r_m \end{bmatrix}^T$ is populated as the distance between two consecutive waypoints $r_i \triangleq \left\| \boldsymbol{P}_i + \boldsymbol{P}_{i-1} \right\|$, while the angle vector $\boldsymbol{a} \triangleq \begin{bmatrix} a_2 & \cdots & a_m \end{bmatrix}^T$ is populated as $a_i \triangleq \angle(\boldsymbol{V}_{i-1}, \boldsymbol{V}_i)$ where $\boldsymbol{V}_i \triangleq \boldsymbol{P}_i - \boldsymbol{P}_{i-1}$. The angles a_i are positive when they are the result of a positive rotation around $\boldsymbol{z} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$, and negative otherwise.

Writing the input set as range-angle pairs unloads the burden from the NN from having to learn the homogeneous invariance property. In order to keep the possible values of range within a compact set, all the ranges can be scaled by a common factor $\bar{\boldsymbol{r}} \triangleq \begin{bmatrix} \bar{r}_0 & \cdots & \bar{r}_{m-1} \end{bmatrix}^T = s \cdot \boldsymbol{r}$, with $s > 0$, such that $0 < \bar{r}_i \leq 1, \forall i \in \{0, \cdots, m-1\}$. Since the optimal time allocation $\boldsymbol{\delta t}$ is also invariant to scaling on the waypoints, the optimal values of $\boldsymbol{\delta t}$ are not affected by this transformation.

Thus, for a given set j of $m+1$ waypoints, the inputs are defined as $\mathbb{X}_j = \begin{bmatrix} \bar{\boldsymbol{r}}^T & \boldsymbol{a}^T \end{bmatrix}^T$.

D. Neural Network Architecture

In order to train the neural network, we use the input-output pairs obtained through the method described in Section IV-C. We apply the rectified linear unit (ReLU) nonlinearity on the output of each of the hidden layers [22]. Using ReLU over other activation functions like sigmoid makes the network train faster as it does not require exponential computations. The gradient of ReLU also does not saturate

for high input values. A softmax activation function on the output layer ensures the outputs to lie within the domain $0 \leq \delta t_i \leq 1$ and $\sum \delta t_i = 1$.

The error was calculated based on the cross entropy loss function [23]. The network was trained in a supervised mode and the weights and biases of the hidden layers and the weights of the output layer were initialized using the Xavier's initialization method [24]. The biases of the output layer were initialized at a constant of 0. We used the Adam optimizer [25] with $\beta_1 = 0.9$ and $\beta_2 = 0.999$. The learning rate was reduced by a factor of 5 after every $10k$ iterations. The training was stopped when the error on the validation set started increasing as a way to prevent "overfitting".

At the time of this paper, we have trained 28 neural networks, each of them corresponding to the number of input waypoints in the range from 3 waypoints to 30 waypoints. Based on need, it is also possible to train networks for more than 30 waypoints. It took us over a month to obtain the BGD solutions to all data sets, and about a week for training the neural networks (Intel i5-4690k CPU with Nvidia GTX 1080 GPU).

E. Algorithm Pipeline

The pipeline in Figure 4 summarizes the proposed algorithm for a trained NN. The input to the systems are the waypoints $\boldsymbol{P}_k, k \in \{0,1,\cdots m\}$ and the desired final time T. The waypoints are transformed in range-angle, and the ranges are normalized. The normalized ranges and the angles are the inputs for the trained NN, which outputs the time fractions $\boldsymbol{\delta t}_m$. The output of the algorithm is then $\boldsymbol{\delta t} = T \cdot \boldsymbol{\delta t}_m$, which can be used as a prior for minimum snap.

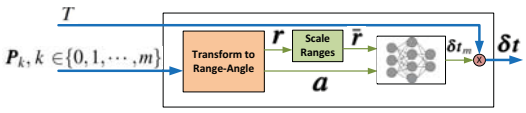

Fig. 4. Algorithm Pipeline Schematic Illustration.

V. Results and Analysis

To show proof of concept, we trained a NN that is able to take 30 waypoints as inputs, and produce 29 δt's as outputs. As mentioned before, the SIGN dataset was used for training (85% of the data) and validation (15% of the data). Every gesture from the SIGN dataset that had more than 30 points was decimated down to get 30 waypoints. The gestures that had less than 30 points were discarded, amounting to 29274 trajectories in the train-validation dataset. We construct a fully connected feed-forward neural network of size $57 \times 84 \times 72 \times 64 \times 48 \times 29$ with weights and biases.

In order to test the outcome of the trained NN, the ILGDB dataset was used (none of the trajectories in ILGDB were used for training). Again, the gestures from the dataset that had more than 30 points were decimated down to get 30 waypoints and gestures that had less than 30 points were discarded, amounting to 6259 trajectories in the ILGDB dataset. For every set j of 30 waypoints, we solve the minimum snap problem using Backtracking Gradient Descent, getting the locally optimal cost $J_{BGD}(j)$.The cost $J_{NN}(j)$ is obtained when solving minimum snap with fixed time allocations using the $\boldsymbol{\delta t}$'s that were given by the NN. For the sake of comparison with other methods in the literature [9], we also get the cost $J_{TVP}(j)$ by solving minimum snap with fixed time allocations with $\boldsymbol{\delta t}$'s given by a trapezoidal velocity profile [9] with maximum velocity $5m/s$ and maximum acceleration $2.5m/s^2$. The costs $J_{BGD}(j)$, $J_{NN}(j)$ and $J_{TVP}(j)$ are obtained for the same final time $t_m = T$ dictated by the final time calculated by the TVP method.

Because the minimum snap cost is sensitive to the 7-th power of the time allocations (see Property 2 in Section III), the optimal costs are normalized by a 7-th root:

$$\bar{J}_{BGD}(j) \triangleq J^{\frac{1}{7}}_{BGD_i}(j), \quad \bar{J}_{NN}(j) \triangleq J^{\frac{1}{7}}_{NN}(j), \quad \bar{J}_{TVP}(j) \triangleq J^{\frac{1}{7}}_{TVP}(j)$$

The Percent Relative Error (PRE) between NN and the BGD method is defined in (10). Similarly, the PRE between TVP and the BGD method is defined in (11). Note that $E^{rel}_{a,b}$ is positive when method b outperforms method a, and negative otherwise.

$$E^{rel}_{NN,BGD}(j) = 100 \cdot \left[\bar{J}_{NN}(j) - \bar{J}_{BGD}(j)\right] \cdot \bar{J}^{-1}_{BGD}(j) \tag{10}$$

$$E^{rel}_{TVP,BGD}(j) = 100 \cdot \left[\bar{J}_{TVP}(j) - \bar{J}_{BGD}(j)\right] \cdot \bar{J}^{-1}_{BGD}(j) \tag{11}$$

Figure 5 (a) shows a histogram distribution for $E^{rel}_{NN,BGD}$. On average, the NN method is 12% worse than BGD, with a standard deviation of $\sigma_{NN} = 11\%$. The NN method outperformed BGD 7.5% of the time[3]. In contrast, Figure 5 (b) shows a histogram distribution for $E^{rel}_{TVP,BGD}$. In average, the TVP method provides a guess that is 192% worse than BGD_1, with a standard deviation of $\sigma_{TVP} = 140\%$. Within those trials, the NN method provided a better initial guess than TVP 99.14% of the time, demonstrating that NN is more powerful than techniques that are currently used in the literature to guess the optimal time allocation.

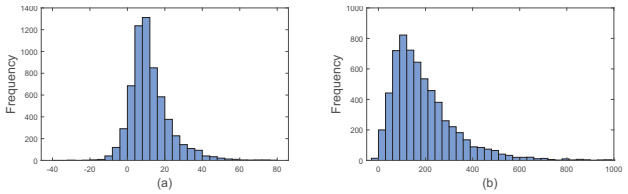

Fig. 5. Histograms of PREs $E^{rel}_{NN,BGD}$ (a), $E^{rel}_{TVP,BGD}$ (b).

Figure 6 presents some minimum snap results obtained for time allocations given by the NN method (red dotted lines) and the TVP method (black dotted lines). Figure 6 shows that the NN calculated trajectories are quite close to the original ones, while the TVP time allocations are clearly less optimal for when the trajectories have sharp turns, as in Figs. 6-(b),(e),(h), and seems to perform slightly worse for the smooth trajectories in Figs. 6-(c),(g). Table I compares the relative errors between NN and TVP for the sample trajectories of Fig. 6. Overall, TVP's performance is much farther from the BGD optimals than NN, except for the case

[3]The reason for NN to outperform BGD is given by the fact that BGD converged to a non-global minimum in those examples, while NN learned a better solution for those cases. This result is surprising to us, since we were expecting BGD to always outperform NN.

in 6(d), in which TVP is slightly better. It is also worth mentioning that the trajectories in Figs. 6-(e),(f),(g),(h) were not used to train the NN.

TABLE I

PRE FOR NN AND TVP FOR SOME SAMPLE TRAJECTORIES

	(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)
$E^{rel}_{NN,BGD}$	15.3	5.62	21.0	-2.27	-7.37	29.8	-5.61	4.22
$E^{rel}_{TVP,BGD}$	37.6	167	166	-6.67	85.9	157	102	219

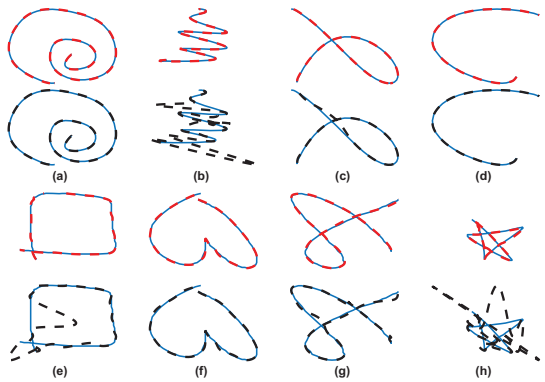

Fig. 6. Minimum snap with time allocations obtained from the NN (red dotted curves) and TVP (black dotted curves) compared with original hand-drawn trajectories (solid blue curves).

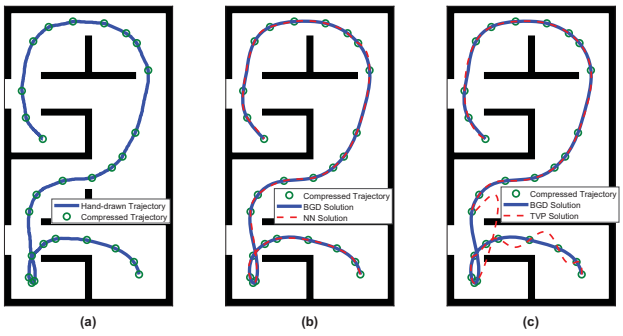

Fig. 7. Human-machine interface example with a hypothetical map. Image (a) displays a hand-drawn trajectory on the map and the compressed waypoints. Image (b) displays the compressed waypoints and the minimum snap solutions obtained through BGD and NN. Image (c) displays the compressed waypoints and the minimum snap solutions obtained through BGD and TVP.

Fig. 7 shows an example of the utilization of a hand-drawn trajectory to interpolate a minimum snap trajectory on a map. The user draws a desired trajectory (Fig. 7-(a) - trajectory was drawn from the bottom to the top of the map), which is compressed into a smaller set of waypoints (green circles). Fig. 7-(b) compares the solutions obtained through BGD and NN, demonstrating that NN obtains a decent approximation for the optimal time allocations ($E^{rel}_{NN,BGD} = 18.9\%$ in this case). On the other hand, the time allocation suggested by TVP leads to collision in the first phase of the trajectory, as seen in Fig. 7-(c) ($E^{rel}_{TVP,BGD} = 136.7\%$ in this case). The collision aspect can be remedied by utilizing the method described in [8], but would require multiple minimum-snap solutions, whereas the same was not needed in BGD or NN for the given example. In terms of computational time, it took 3.23s to obtain the BGD solution, while it takes less than 0.05s to obtain trajectories in NN and TVP (see Figure 1 for a comparison of time improvement when abandoning BGD into the use of NN and TVP for initial time guess).

VI. FINAL CONSIDERATIONS

The properties that were used to design the input/output pairs of the NN rely on the assumption that (i) the problem has only position equality constraints in the waypoints, (ii) it has no inequality constraints, and (iii) that the vehicle starts from rest and finishes at rest. Although these are reasonable for tablet-based human commands, one might argue that these assumptions have a strong negative impact in other applications, such as in motion planning.

In practice, most works in the literature solve minimum snap with only a set of position waypoints (no velocity, acceleration or jerk equality constraints) [8][9][26], so (i) really is not a commonly prohibitive assumption. Also, assumption (ii) is typical in many works [7][12], even when there are maximum velocity and acceleration constraints. A common workaround for assumption (ii) is to extend final time T until these quantities are within desired bounds, which is an effective method due to Property 1 presented in Section III.

As opposed to assumptions (i) and (ii), assumption (iii) can be limiting in autonomous navigation applications. This would require the vehicle to always start from rest, but autonomous systems are typically recalculating paths many times per second while in motion. This assumption can potentially be circumvented by adding an extra input to the NN with the vehicle's initial velocity, and retraining it.

VII. CONCLUSION

This paper presents a methodology for employing a Neural Network as a tool for obtaining a guess of time allocations when generating minimum snap trajectories for quadrotors. The results demonstrate that using a trained NN lead to a better initial guess of time allocations than the tools that are available in current literature. In addition, the NN approach produces a result two orders of magnitude faster than gradient descent when solving a 30 waypoint problem for 2D rest-to-rest maneuvers.

For future work, we expect to train more neural networks for problems with higher number of waypoints and for different initial/terminal velocities. The ideas herein presented can also be extended to other motion planners, such as minimum acceleration, or minimum jerk.

Another interesting line of future research would be to combine the outputs of the NN method with the minimum snap solutions presented in [12] and [13]. Instead of setting a waypoint as an equality constraint, both these methods specify waypoints through inequalities, allowing the waypoint to be anywhere within a safe convex region, which implicitly optimize allocation time as well. In addition, [12] provides a fast refinement method for dealing with intermediate large velocities, acceleration or jerk, allowing to improve trajectories without performing BGD.

REFERENCES

[1] P. Abbeel, *Apprenticeship learning and reinforcement learning with application to robotic control*. ProQuest, 2008.

[2] D. Mellinger, N. Michael, and V. Kumar, "Trajectory generation and control for precise aggressive maneuvers with quadrotors," *The International Journal of Robotics Research*, 2012.

[3] S. Lupashin, A. Schöllig, M. Sherback, and R. D'Andrea, "A simple learning strategy for high-speed quadrocopter multi-flips," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE, 2010, pp. 1642–1648.

[4] J. H. Gillula, H. Huang, M. P. Vitus, and C. J. Tomlin, "Design of guaranteed safe maneuvers using reachable sets: Autonomous quadrotor aerobatics in theory and practice," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE, 2010, pp. 1649–1654.

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

[6] M. Van Nieuwstadt, M. Rathinam, and R. Murray, "Differential flatness and absolute equivalence of nonlinear control systems," *SIAM Journal on Control and Optimization*, vol. 36, no. 4, pp. 1225–1239, 1998.

[7] M. Cutler and J. P. How, "Analysis and control of a variable-pitch quadrotor for agile flight," *Journal of Dynamic Systems, Measurement, and Control*, vol. 137, no. 10, 2015.

[8] 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.

[9] S. Liu, M. Watterson, S. Tang, and V. Kumar, "High speed navigation for quadrotors with limited onboard sensing," in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE, 2016, pp. 1484–1491.

[10] S. G. Johnson, "The nlopt nonlinear-optimization package," 2014.

[11] R. Allen and M. Pavone, "A real-time framework for kinodynamic planning with application to quadrotor obstacle avoidance," in *AIAA Guidance, Navigation, and Control Conference*, 2016, p. 1374.

[12] S. Liu, M. Watterson, K. Mohta, K. Sun, S. Bhattacharya, C. J. Taylor, and V. Kumar, "Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-d complex environments," *IEEE Robotics and Automation Letters*, vol. 2, no. 3, pp. 1688–1695, 2017.

[13] J. Chen, T. Liu, and S. Shen, "Online generation of collision-free trajectories for quadrotor flight in unknown cluttered environments," in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE, 2016, pp. 1476–1483.

[14] S. Li, M. Okuda, and S. Takahashi, "Embedded key-frame extraction for cg animation by frame decimation," in *Multimedia and Expo, 2005. ICME 2005. IEEE International Conference on*. IEEE, 2005, pp. 1404–1407.

[15] M. M. de Almeida and M. Akella, "New numerically stable solutions for minimum-snap quadcopter aggressive maneuvers," in *American Control Conference (ACC), 2017*. IEEE, 2017, pp. 1322–1327.

[16] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[17] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[18] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the rrt." Institute of Electrical and Electronics Engineers, 2011.

[19] F. Hauer and P. Tsiotras, "Deformable rapidly-exploring random trees," in *Proceedings of the Robotics: Science and Systems Conference, Boston, MA*, 2017, p. P08.

[20] A. Almaksour, E. Anquetil, S. Quiniou, and M. Cheriet, "Personalizable pen-based interface using lifelong learning," in *Frontiers in Handwriting Recognition (ICFHR), 2010 International Conference on*. IEEE, 2010, pp. 188–193.

[21] N. Renau-Ferrer, P. Li, A. Delaye, and E. Anquetil, "The ilgdb database of realistic pen-based gestural commands," in *Pattern Recognition (ICPR), 2012 21st International Conference on*. IEEE, 2012, pp. 3741–3744.

[22] 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.

[23] R. A. Dunne and N. A. Campbell, "On the pairing of the softmax activation and cross-entropy penalty functions and the derivation of the softmax activation function," in *Proc. 8th Aust. Conf. on the Neural Networks, Melbourne*, vol. 181, 1997, p. 185.

[24] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 2010, pp. 249–256.

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

[26] Y. Lin, F. Gao, T. Qin, W. Gao, T. Liu, W. Wu, Z. Yang, and S. Shen, "Autonomous aerial navigation using monocular visual-inertial fusion," *Journal of Field Robotics*, vol. 35, no. 1, pp. 23–51, 2018.