Real-Time Optimal Control Of Quadrocopters Using Deep Representations Of The Optimal State Feedback

Final Report

Authors: Shuo Li[1], Ekin Ozturk[2], Christophe De Wagter[1], Guido C. H. E. de Croon[1], Dario Izzo[2]
Affiliation: [1]Delft University of Technology, [2]ESA ACT

Date: 12 December 2019

Contacts:

Guido de Croon

Tel:	+31648255416
Fax:	-
e-mail:	g.c.h.e.decroon@tudelft.nl

Leopold Summerer (Technical Officer)

Tel:	+31(0)715654192
Fax:	+31(0)715658018
e-mail:	act@esa.int

Ariadna ID: 18/8501
Ariadna study type: Standard
Contract Number: 4000124422/18/NL/GLC/as

1 INTRODUCTION

A major challenge in the field of control is to achieve reliable, aggressive, high-speed control of autonomous vehicles. In space, this may involve spacecraft that need to land under harsh conditions, or even – in an extreme scenario – negotiating asteroid debris fields at high speeds. On earth, the exemplar task that draws most attention currently is high-speed autonomous flight of drones. Studying this problem on drones is of relevance to space as well, since drones can only carry very limited sensing and processing, resulting in restrictions that are similar to those of space systems.

Concerning high-speed control of drones, much research focuses on designing controllers which can track a reference guidance trajectory also when considering unmodeled dynamics, nonlinearities and disturbances which become significant when the maneuver of the drone gets aggressive [1], [2]. In terms of guidance, multiple methods varying from a simple setpoint to high order polynomial trajectory generation methods have shown their feasibility in guiding a quadrotor to the desired target including some time optimality principles.

Two fundamentally different approaches are used to obtain aggressive quadrotor trajectories. The first one is differential flatness based trajectory generation and control [3], [4]. This method is able to generate aggressive trajectories for quadrotors (based on a minimum-time polynomial guidance), and hence it is widely used in real quadrotor flights. However, the resulting trajectory can be far from being truly time optimal.

The second approach uses optimal control theory to find and fly a trajectory that incorporates the required optimality principles. Due to the time-consuming nature of this calculation, this method is unsuitable for an online implementation [5], [6]. Several methods have been proposed to address this, where the most common is to represent the system dynamics as a series of simpler linear systems with analytical solutions [7], [8]. Unfortunately, this simplification can lead to an inaccurate representation of the nonlinear response of the system and can thus negatively impact performance. An alternative approach is to find and use, on-board, a sub-optimal solution instead. For example, by using the result of the first iteration of a nonlinear programming (NLP) solver [9] which, although incomplete, is faster to compute.

In recent years, leveraging significant advances in machine learning techniques and in particular in artificial neural networks, a number of new methods have been proposed relevant to the aggressive control of quadrotor trajectories. Reference trajectories have been optimized using DNNs [10], waypoint tracking has been achieved by means of reinforcement learning [11] and trajectory tracking using RBFNN [12]. Tang et al. [13] combine both optimal control and machine learning. Their experimental results have shown that a trained neural network can predict an optimal trajectory to the target point, which can then be tracked using PID control. This work is an important step towards online optimal control, however the main computation is done on a workstation (i.e. not on-board) and, since a PID controller is introduced to track the reference, there are delays during the tracking as a result of which the controls may violate the constraints due to the feedback term. In a different context (i.e. spacecraft landing and mass optimal control) Sanchez et al. [14] successfully introduced the use of imitation learning of optimal controls to train DNNs capable of safely steering the system to desired target positions. Following that work, Tailor and Izzo [15] made an extensive study of the technique on simulated drone dynamics and Izzo et al. [16] introduced the term G&CNets (guidance and control networks) to refer to these networks and showed how to study the stability of the resulting trajectories analytically via differential algebraic techniques.

In this Ariadna study, we present an approach for the on-board optimal control problem of a quadrotor that does not need a PID controller to track the trajectory and we test it during real flights. In our approach, 250,000 optimal trajectories are generated offline. Then, a G&CNet---which is a neural network trained to learn this dataset---is computed. Instead of predicting an optimal trajectory as the work in [13], G&CNet predicts the required optimal thrust directly which will be transferred to the optimal pitch rate acceleration and sent to the controller, and thus can be seen in the context of non-Linear MPC. Since the work of [13] is difficult to reproduce, we made the comparison between G&CNet and the differential flatness based trajectory generation and control (DiffG&C) in simulation. The simulation results show that the proposed G&CNet can guide the drone to the target points much faster while satisfying optimality principles. Finally, the developed G&CNet and DiffG&C controllers are verified in real-world flight tests where the results show that the on-board G&CNet can guide the drone to the target with a resulting real-time trajectory that is very close to the theoretical optimal solution.

2 DESIGN OF THE G&CNET

2.1 The dynamic system

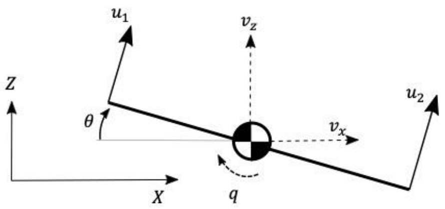


Fig. 1. Axis definition

Specifying the state of a quadrotor in the xoz plane as

$$\boldsymbol{x} = [x\ z\ v_x\ v_z\ \theta\ q] \tag{1}$$

as defined in Fig. 1, the dynamical model for which we compute the optimal control is:

$$f(\boldsymbol{x},\boldsymbol{u}) = \begin{bmatrix} \dot{x} = v_x \\ \dot{z} = v_z \\ \dot{v}_x = -\left[u_\Sigma \frac{\Delta F}{m} + 2\frac{\underline{F}}{m}\right]\sin\theta - \beta v_x \\ \dot{v}_z = -\left[u_\Sigma \frac{\Delta F}{m} + 2\frac{\underline{F}}{m}\right]\cos\theta - g_0 - \beta v_z \\ \dot{\theta} = q \\ \dot{q} = \frac{L}{I_{xx}}\Delta F(u_2 - u_1) \end{bmatrix} \tag{2}$$

where $\Delta F = \overline{F} - \underline{F} = 0.59N$ is the range of the thrust magnitude, $\overline{F} = 2.35N$ is the maximum thrust, $\underline{F} = 1.76N$ is the minimum thrust, $\beta = 0.5$ is the drag coefficient, $m = 0.389kg$ is the quadrotor mass, $L = 0.08m$ is the length of the quadrotor, $I_{xx} = 0.001242kgm^2$ is the moment of inertial about the x-axis, $u = [u_1, u_2] \in [0,1]$ are the left and right throttles respectively, and $u_\Sigma = (u_1 + u_2)$.

2.2 The optimization problem

The cost function we need to minimize for the optimal controls is:

$$J\big(\epsilon, t_f, \boldsymbol{u}(t)\big) = (1-\epsilon)t_f + \epsilon\int_0^{t_f}(u_1{}^2(t) + u_2{}^2(t))\mathrm{d}t \tag{3}$$

where $\epsilon \in [0,1]$ is a hybridization parameter. When $\epsilon = 0$, the cost function is exactly time-optimal, and when $\epsilon = 1$, the cost function is exactly power-optimal. With this parameter we are able to generate datasets from time-optimal to power-optimal continuously. Similar to the weighting factor of [17], we set ϵ close to zero ($\epsilon = 0.2$) to improve the numerical convergence of the problem and avoid difficult to track control profiles. We trained two networks for $\epsilon = 0.5$ and $\epsilon = 0.2$ in order to compare how well the quadrotor is able to track and execute the optimal controls with differing degrees of aggressiveness. As we are more interested in time-optimal guidance and control, the dataset and training details focus only on the $\epsilon = 0.2$ controller, but the same arguments and methods apply to the $\epsilon = 0.5$ controller.

$$\begin{gathered}\min_{\boldsymbol{u},t_f} J\left(\epsilon, t_f, \boldsymbol{u}(t)\right) \\ \mathit{subject\ to} \quad \dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x},\boldsymbol{u}), \forall t \\ \boldsymbol{x}(0) = \boldsymbol{x}_0 \\ \\ \boldsymbol{x}(t_f) = 0\end{gathered} \tag{4}$$

Using a direct transcription and collocation method (Hermite-Simpson transcription), the trajectory optimization problem is transformed into an NLP problem [17]. The AMPL modelling language was used to specify the NLP problem which was then solved via SNOPT, an SQP NLP solver. Solving for 250,000 trajectories with initial states, $\boldsymbol{x}_0$, drawn uniformly from $x_0 \in [-10,10]\, m$, $z_0 \in [-10,10]\, m$, $v_{x0} \in [-5,5]\, ms^{-1}$, $v_{z0} \in [-5,5]\, ms^{-1}$, $\theta_0 \in [-\pi/3, \pi/3]\, rad$, and $q_0 \in [-0.01,0.01]\, rads^{-1}$, we obtain a database of state-control pairs of the form:

$$\begin{gathered}k_i = \left(\boldsymbol{x}_j^{(i)}, \boldsymbol{u}_j^{(i)}\right)_{j=1}^{K} \quad \text{where} \\ \boldsymbol{x}_1^{(i)} = \boldsymbol{x}_0^{(i)}, \boldsymbol{x}_j^{(i)} = 0 \quad i = 1, \dots, M\end{gathered} \tag{5}$$

where i indexes the trajectories and $K = 81$ is the number of grid points in the Hermite-Simpson transcription [17]. We solved for 250,000 trajectories of which 214,210 converged, and following an 80-10-10 split, these trajectories were split into training, validation and test sets. Overall, this translates to 13,880,808 state-control pairs that the network was trained on, and 1,735,101 that the network was tested on.

2.3 *Network architecture and training*

We construct neural network architectures in the same manner as [17] with 3 layers, 100 hidden units with softplus activation functions, and sigmoid activation functions for the output controls.

Thus, we train on the loss function:

$$l = ||N(\boldsymbol{x}) - \boldsymbol{u}^*||^2 \tag{6}$$

with a minibatch size of 256 and a starting learning rate of 10^{-3} using the Adam optimizer. For further details on network training and construction, refer to [17]. From this training, the $\epsilon = 0.2$ network was able to achieve a mean absolute error (MAE) of 0.0105 for u_1 and 0.0107 for u_2 on the training set, and a MAE of 0.0108 for u_1 and 0.0109 for u_2 on the test set.

3 SIMULATION RESULT AND ANALYSIS

In this section, we analyse the theoretical performance of the proposed optimal controller. First we discuss the simulated stability characteristics of the G&CNet($\epsilon = 0.2$) controller. Then we introduce the aforementioned DiffG&C as a benchmark controller. Finally, we detail the simulation of both methods and present a comparison between simulations.

3.1 Stability of Neural Network Controller

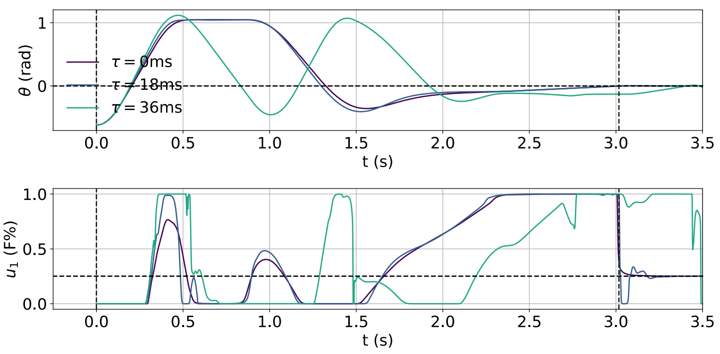


Fig. 2. Pitch (top) and the left thrust (bottom) during a G&CNet driven trajectory simulated with control delays of 0ms, 18ms and 36ms. The vertical dashed lines show the initial and final time of the true optimal trajectory. The horizontal dashed lines show the target final states: $\theta(t_f) = 0.0$and $u_1(t_f) = u_{hover}$.

One of the foremost important things is the stability of any controller used on the quadrotor as an unstable controller can lead to failure. The primary stability concerns arise due to the fact that in a real quadrotor there is a measurable delay between the computation of the controls, the state given to the controller and the controller response which arises due to factors such as the time taken to compute the state, and the inertia of the rotors. This delay can be modeled by a fixed time between the command and the execution of the control command:

$$\boldsymbol{u}(t) = N(\boldsymbol{x}(t - \tau)) \tag{7}$$

where τ is the time delay. Using the methods developed in [16], we find that the stability margin of the G&CNet($\epsilon = 0.2$) controller is $\tau_s = 63.8ms$. Although this stability margin is high, it mostly provides information as to the hovering stability of the quadrotor, but we are more interested in the general stability during flight. Fig. 2 shows the effect of an increasing time delay on the G&CNet($\epsilon = 0.2$) controller left thrust and pitch for delays of $\tau = 0ms$, $\tau = 18ms$ and $\tau = 36ms$. Here we see that, as the delay increases, the controller becomes increasingly unstable up to the point where it is no longer able to track the optimal trajectory nor hover in the final state.

3.2 Differential flatness based aggressive trajectory generation and control (DiffG&C)

A commonly used aggressive trajectory generation method is to use high order polynomials $P(t) = \boldsymbol{p}^{\mathrm{T}}\boldsymbol{t}$ to connect the initial point, the waypoints and the final point [3], [4]. Thanks to the differential flatness properties of the quadrotor, the thrust on each rotor can be directly related to the 4th order derivative of the position curves $\boldsymbol{u} = f(\boldsymbol{p}, t)$ [3], [18]. In particular, in this method, we use the same kinematics model as the reference [18] with Bebop's drag coefficient $D = \mathrm{diag}(-\beta, -\beta, -\beta)$), mass $m = 0.389$kg and length $L = 0.08$m.

$$\begin{cases} \dot{\boldsymbol{x}} = \boldsymbol{v} \\ \dot{\boldsymbol{v}} = \boldsymbol{g} + \boldsymbol{T} + \boldsymbol{R}^T \boldsymbol{D} \boldsymbol{R} \boldsymbol{v} \\ \dot{\boldsymbol{\Phi}} = \boldsymbol{R}' \boldsymbol{q} \end{cases} \tag{8}$$

where thrust $\boldsymbol{T} = [0,0,T]^{\mathrm{T}}$ and body rate $\boldsymbol{q} = [p,q,r]^{\mathrm{T}}$ are the inputs of the system with the assumption that the low-level acceleration controller and rate controller can track the reference well. Equation 9 is used to check the feasibility of the thrust each rotor can provide.

$$\dot{\mathbf{q}} = \boldsymbol{I}^{-\mathbf{1}}(\tau - \boldsymbol{q} \times \boldsymbol{I}\boldsymbol{q}) \tag{9}$$

From the computed polynomial trajectory, the body rate $\boldsymbol{q}$ and the rotor thrusts can be determined. For a given arrival time t_f, the best trajectory connecting two states is the one with minimal snap. By decreasing the arrival time t_f until the constraints are violated, the polynomial trajectory with minimum arrival time and minimal snap can be found.

$$\min_{t_f} \left\{ \min_{\boldsymbol{P}} \int_0^{t_f} P^{(4)}(t) \mathrm{d}t \right\} = \min_{t_f} \left\{ \min_{\boldsymbol{P}} \boldsymbol{P}^{\mathrm{T}} \boldsymbol{Q} \boldsymbol{P} \right\} \tag{10}$$

$$s.t. \boldsymbol{A}\boldsymbol{P} = \boldsymbol{b} \tag{11}$$

$$\boldsymbol{f}(\boldsymbol{P}, t) < \boldsymbol{c} \tag{12}$$

where (10) is the optimization target, the integral of the 4th order derivative of the polynomial which can be written as a quadratic form. Equation 11 is the constraints of the polynomial and (12) gives the input constraints. The readers are referred to [4] for the detailed derivation of matrix $\boldsymbol{Q}$.

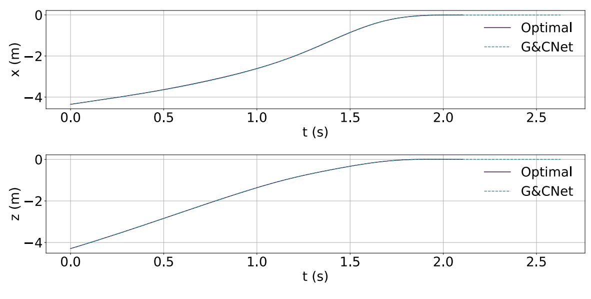
The feed-forward control inputs are computed from the polynomial trajectories and a feedback PID controller is used to compensate for disturbance. The readers are referred to [18] for further details on the controller implementation.

We only investigate the movement in the xoz plane by setting any movement in the y direction to 0. This way, the model given by (8) can be simplified to the model in (2).

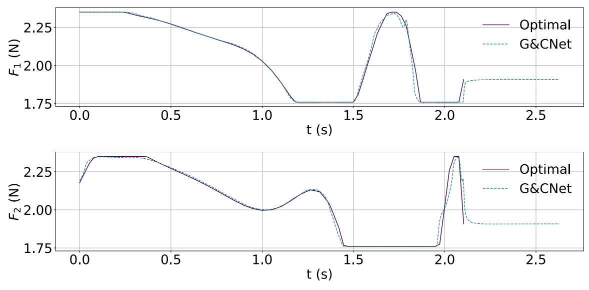
3.3 Simulation of the G&CNet Controller

In this simulation, we use the model from (2) as our dynamical model with the rate acceleration $\dot{q}$ and total thrust T as the inputs. The reason is that on the real drone, there are different low-level controllers which can track the thrust and the rate acceleration accurately, one of which is the incremental nonlinear dynamic inversion controller (INDI) [1]. We calculate the desired thrust and rate acceleration command from the G&CNet controller outputs using Eq. (13)

$$\begin{cases} \dot{q}_{cmd} = \frac{(u_1 - u_2)\Delta FL}{I_{xx}} \\ T_{cmd} = \frac{(u_1 + u_2)\Delta F}{m} \end{cases} \tag{13}$$



(a) A simulated trajectory using G&CNet($\epsilon = 0.2$).



(b) Force of the front rotors and the rear rotors along the simulated trajectory.

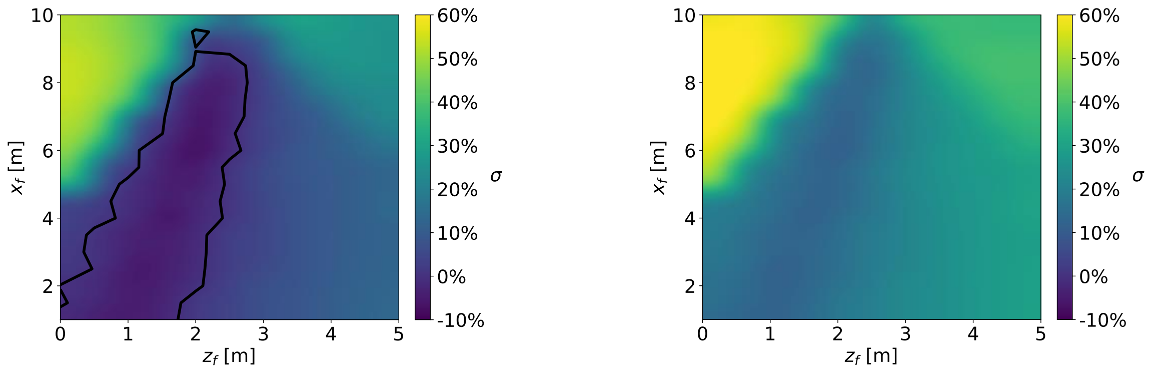
Fig. 3. An example simulation of G&CNet($\epsilon = 0.2$). In each simulation step, the controller receives $x, z, v_x, v_z, \theta, q$ and outputs the thrust command of the front rotors and the rear rotors. The desired total thrust T and rate acceleration $\dot{q}$ are calculated by (13) and sent to model 2 for integration.

3.4 Comparison between DiffG&C and G&CNet

In this section, a comparison is made in simulation between DiffG&C and G&CNet. The time required by the drone to reach the target is used to derive a performance index. In each trial, the initial position of the drone is set to be $[x_0, z_0] = [0m, 2.5m]$ and the same target $x_f \in [1,10]$, $z_f \in [0,5]$ is set for both controllers. To quantify the performance of a method, we introduce an index σ:

$$\sigma = \frac{t_f^{DiffG\&C} - t_f^{G\&CNet}}{t_f^{DiffG\&C}} \tag{14}$$

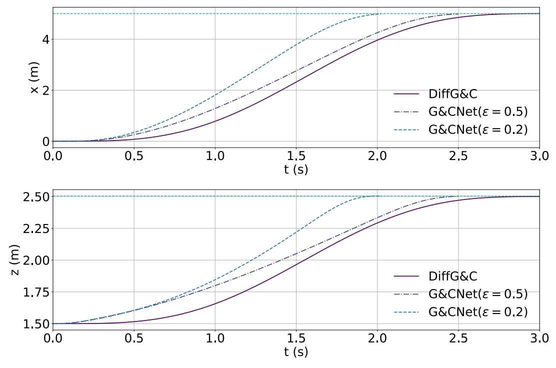
where t_f^* is the arrival time of each controller. When $\sigma > 0$, the G&CNet controller is faster than DiffG&C and vice versa. Fig. 4 gives the simulation results of multiple target points with $\epsilon = 0.2$ and $\epsilon = 0.5$. From Fig. 4(a), it can be seen that, in most cases, G&CNet($\epsilon = 0.5$) has a shorter arrival time than DiffG&C outside the region delineated by the black border, and in this region arrival time is always shorter and up to 60% faster than DiffG&C.



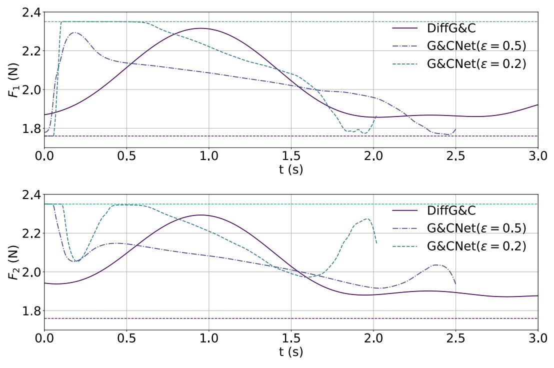
(a) $\epsilon = 0.5$ (b) $\epsilon = 0.2$

Fig. 4. Comparison of arrival time between DiffG&C and G&CNet. Despite power optimality being weighted equally to time optimality, G&CNet($\epsilon = 0.5$) can, in most cases, steer the drone to the target points in less time than DiffG&C (the black line shows the region border where G&CNet outperforms DiffG&C). On the other hand, G&CNet($\epsilon = 0.2$) is always faster than DiffG&C.

Fig. 5 shows a comparison plot of the trajectories and controls of DiffG&C, G&CNet($\epsilon = 0.5$) and G&CNet($\epsilon = 0.2$). It can be seen that all three controllers reach the target, but the control profiles and arrival times differ significantly. With DiffG&C, due to the polynomial representation of trajectories, the quadrotor inputs cannot be fully utilised, and thus the time-optimality cannot be guaranteed. On the other hand, G&CNets are able to saturate the inputs and arrive at a similar or smaller time.



(a)



(b)

Fig. 5. An example of comparison between DiffG&C and G&CNet($\epsilon = 0.5$) when $x_f = 5$, $z_f = 2.5$.

4 EXPERIMENT SETUP AND RESULT

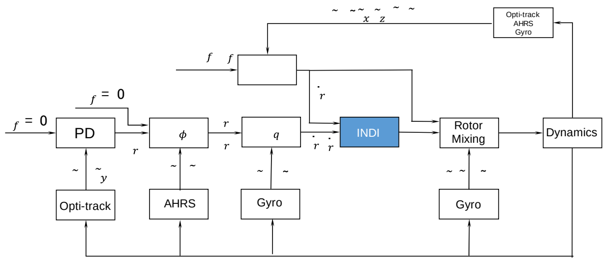
In this section we show the experimental setup for real-world flights and the flight performance of each method.

4.1 Experiment Setup

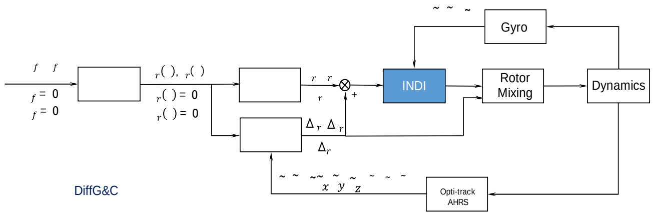
To verify the proposed G&CNet, we use a commercial Parrot Bebop 1 as our flying platform (Fig. 6). The This fully replaced by an open-source autopilot, Paparazzi-UAV. This autopilot provides full access to the raw sensor data and rotor commands.

Fig. 6. A Parrot Bebop 1 is used as the flying platform. The original autopilot is fully replaced by an open-source autopilot called Paparazzi UAV.

In this experiment, the position and velocity feedback are from Opti-track motion capture system. The attitude estimation is from an on-board complementary filter, which is inevitably biased. The angular rate estimation is from the on-board gyroscope. The control architecture is shown in Fig. 7. For G&CNet, the lateral movement and heading are controlled by the original outer-loop PID controller and inner-loop INDI controller to keep $y = 0$ and $\psi = 0^{\circ}$. The maneuver on the vertical plane is taken over by the proposed G&CNet. In each control update, G&CNet receives the state estimations and outputs the desired pitch acceleration $\dot{q}$ and the thrust T. For the benchmark DiffG&C, after the trajectory is generated, the desired angular rate $\boldsymbol{q}$ can be directly calculated. Then a feedback controller is used to compensate the deviation caused by the model inaccuracy, and the state estimation bias.



(a) The control structure of G&CNet. A PD controller and an INDI controller are used to keep the quadrotor at $y = 0$ and $\psi = 0^{\circ}$.



(b) The control structure of DiffG&C. The feed-forward signal is directly computed from generated trajectories. A feedback controller is used to correct for deviations.

Fig. 7. The control structure of the proposed G&CNet and the benchmark DiffG&C.

In the real-world flight tests, we test 3 controllers which are DiffG&C, G&CNet($\epsilon = 0.5$) and G&CNet($\epsilon = 0.2$) respectively. For each controller, the start position is set to be $\boldsymbol{x}_0 = [0\text{m}, -1.5\text{m}]^{\mathrm{T}}$ and 3 targets which are$\boldsymbol{x}_f^1 = [5\text{m}, -2.5\text{m}]^{\mathrm{T}}$, $\boldsymbol{x}_f^2 = [5\text{m}, -1.5\text{m}]^{\mathrm{T}}$ and $\boldsymbol{x}_f^3 = [5\text{m}, -0.5\text{m}]^{\mathrm{T}}$ are set to be tested. For each target, we have 10 independent flights. To evaluate the performance of one controller, we have 2 indices which are average arrival time $\Delta\bar{t}_*$ and average tracking error $\Delta\bar{x}_*$ defined by

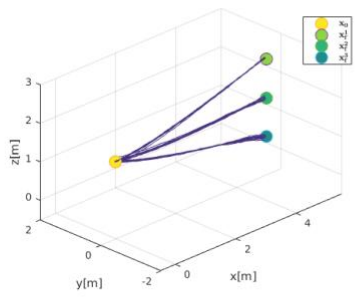
$$\Delta\bar{t}_* = \frac{\sum_i^N \Delta t_*^i}{N} \tag{15}$$

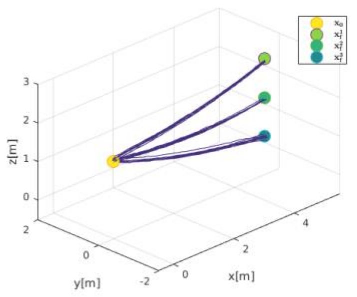
$$\Delta\bar{x}_* = \frac{\sum_{i=1}^N \sum_{j=1}^{n_i} \left\| \bar{\boldsymbol{x}}_*^{i,j} - \boldsymbol{x}_{r,*}^{i,j} \right\|}{\sum_{i=1}^N n_i} \tag{16}$$

where Δt_*^i is the arrival time of ith flight of method *. N is the number of the flight of one controller, which is 10 in our case. $\bar{\boldsymbol{x}}_*^{i,j}$ is the position of ith flight's jth sample measured by the Opti-track system. $\boldsymbol{x}_{r,*}^{i,j}$ is the corresponding position reference. It should be noted that in DiffG&C, $\boldsymbol{x}_r$ is the reference trajectory while in G&CNet, it is the simulated trajectory.

4.2 Experiment Result

The experiment is set up as described in the previous section and we have 90 flights in total (3 controllers × 3 targets × 10 flights, depicted in Fig. 8). The average arrival time is listed in Table I and the average tracking error is listed in Table II.


(a) DiffG&C


(b) G&CNet($\epsilon = 0.5$)

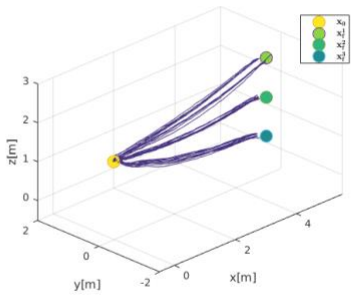

(c) G&CNet($\epsilon = 0.2$)

Fig. 8. The real-world flight data of different controllers to different targets

Table I. Average arrival time $\Delta \bar{t}_*$ to targets $\boldsymbol{x}_f^i$

Controller	$\boldsymbol{x}_f^1$	$\boldsymbol{x}_f^2$	$\boldsymbol{x}_f^3$
DiffG&C	$2.63s \mp 0.05s$	$2.18s \mp 0.02s$	$2.10s \mp 0.04s$
G&CNet($\epsilon = 0.5$)	$236s \mp 0.02s$	$2.20s \mp 0.02s$	$2.13s \mp 0.01s$
G&CNet($\epsilon = 0.2$)	$1.96s \mp 0.03s$	$1.88s \mp 0.03s$	$2.91s \mp 0.04s$

Table II. Average tracking error $\Delta \bar{x}_*$ to targets

Controller	$\boldsymbol{x}_f^1$	$\boldsymbol{x}_f^2$	$\boldsymbol{x}_f^3$
DiffG&C	0.06m	0.07m	0.07m
G&CNet($\epsilon = 0.5$)	0.13m	0.09m	0.10m
G&CNet($\epsilon = 0.2$)	0.17m	0.15m	0.28m

From Table I, it can be seen that when the target is set to $\boldsymbol{x}_f^1$, G&CNet($\epsilon = 0.5$) reaches the target in a shorter time DiffG&C, whereas for targets $\boldsymbol{x}_f^2$ and $\boldsymbol{x}_f^3$, it is on par with the benchmark. On the other hand, G&CNet($\epsilon = 0.2$) always reaches the target in faster time. These experimental results confirm the simulation results that were obtained in Section 3.

In terms of tracking error, DiffG&C has the smallest tracking error $\Delta \bar{x}$ followed by G&CNet($\epsilon = 0.5$), and finally G&CNet($\epsilon = 0.2$). We find that G&CNet($\epsilon = 0.5$) outperforms G&CNet($\epsilon = 0.2$) in terms of the tracking error. This can be attributed to the fact that a lower ϵ corresponds to a more aggressive trajectory and, in turn, a high-frequency high amplitude changes of the inputs. As mentioned in Section 2, this is difficult for the quadrotor to track due to the inertial properties of its rotors.

5 CONCLUSIONS

We have proposed G&CNet as a novel online optimal controller for quadrotors that removes the need for expensive real-time optimal trajectory generation by learning a deep neural representation of the optimal state-control mapping. We have demonstrated, both in simulation and with real-world flight tests, that G&CNets are not only feasible for this purpose, but also competitive with a commonly used method, DiffG&C. Our results indicate that a G&CNet weighting equally power and time optimality ($\epsilon = 0.5$) is, at worst, 10% slower than DiffG&C and faster most of times while a G&CNet aggressively biased towards time optimality ($\epsilon = 0.2$) is always considerably faster by up to 60%.

The findings of the current study are highly relevant for space applications as well. The employed G&CNet was first proposed by the European Space Agency for spacecraft landing tasks. The main efforts in the current study, and the main novel findings, are on how to close the reality gap between a G&CNet trained based on a theoretically very "clean" model and a real-world system – in our case a drone. Taking into account saturations, delays, and employing high-speed low-level controllers have proven to be successful in closing this gap. These findings generalize to space systems that may employ the developed methodology. The presented study will allow future space systems to perform high-speed optimal control with limited onboard resources. It will further have spinoffs on earth, starting with high-speed drone flight, but with the potential to expand to any type of robot and type of locomotion.

There are many avenues of exploration available. Future work can focus on adding the actuator model into the optimal control problem thus eliminating the issue of difficult to track bang-bang controls for the rotors. A further extension of our work would be to implement the optimal control problem in the full 3-dimensional model thus potentially adding more interesting maneuver capabilities to the quadrotor. Additionally, the network could be trained to achieve a nonzero velocity in the final state in preparation for consecutive maneuvers.

REFERENCES

[1] E. J. Smeur, Q. Chu, and G. C. de Croon, "Adaptive incremental nonlinear dynamic inversion for attitude control of micro air vehicles," Journal of Guidance, Control, and Dynamics, vol. 38, no. 12, pp. 450–461, 2015.

[2] E. Tal and S. Karaman, "Accurate tracking of aggressive quadrotor trajectories using incremental nonlinear dynamic inversion and differential flatness," in2018 IEEE Conference on Decision and Control (CDC), pp. 4282–4288, IEEE, 2018.

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

[4] A. Bry, C. Richter, A. Bachrach, and N. Roy, "Aggressive flight of fixed-wing and quadrotor aircraft in dense indoor environments," The International Journal of Robotics Research, vol. 34, no. 7, pp. 969–1002, 2015.

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

[6] F. Morbidi, R. Cano, and D. Lara, "Minimum-energy path generation for a quadrotor uav," in 2016 IEEE International Conference on Robotics and Automation (ICRA), pp. 1492–1498, IEEE, 2016.

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

[8] O. Santos, H. Romero, S. Salazar, O. Garcia-Perez, and R. Lozano, "Optimized discrete control law for quadrotor stabilization: Experimental results," Journal of Intelligent & Robotic Systems, vol. 84, no. 1-4, pp. 67–81, 2016.

[9] M. Geisert and N. Mansard, "Trajectory generation for quadrotor based systems using numerical optimal control," in 2016 IEEE international conference on robotics and automation (ICRA), pp. 2958–2964, IEEE, 2016.

[10] 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 (ICRA), pp. 5183–5189, IEEE, 2017.

[11] 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.

[12] S. Li, Y. Wang, J. Tan, and Y. Zheng, "Adaptive rbfnns/integral sliding mode control for a quadrotor aircraft," Neurocomputing, vol. 216, pp. 126–134, 2016.

[13] 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 (IROS), pp. 3620–3625, IEEE, 2018.

[14] C. Sanchez-Sanchez and D. Izzo, "Real-time optimal control via deep neural networks: study on landing problems," Journal of Guidance, Control, and Dynamics, vol. 41, no. 5, pp. 1122–1135, 2018.

[15] D. Tailor and D. Izzo, "Learning the optimal state-feedback via supervised imitation learning," Astrodynamics, Sep 2019.

[16] D. Izzo, D. Tailor, and T. Vasileiou, "On the stability analysis of deep neural network representations of an optimal state-feedback," arXive-prints, p. arXiv:1812.02532, Dec 2018.

[17] D. Tailor and D. Izzo, "Learning the optimal state-feedback via supervised imitation learning," arXiv e-prints, p. arXiv:1901.02369, Jan 2019.

[18] M. Faessler, A. Franchi, and D. Scaramuzza, "Differential flatness of quadrotor dynamics subject to rotor drag for accurate tracking of high-speed trajectories," IEEE Robotics and Automation Letters, vol. 3, no. 2, pp. 620–626, 20