Deep Neural Networks for Real-time Trajectory Planning

MSc Project Presentation

Amaury Francou — *Supervisor : Dr Dante Kalise*

Imperial College London
Department of Mathematics

Presented 15/09/2022

Imperial College London

Contents

Introduction

NeurIPS 2017

The 2017 NeurIPS conference Pieter Abbeel main talk $\rightarrow$ presented a video in which a robot was able to perform a *wide range of household tasks* [1].

Figure 1: 2017 Neural Information Processing Systems conference (NeurIPS), Pieter Abbeel talk : *Deep learning for robotics*

$\longrightarrow$ Robots have **required physical capabilities**

$\longrightarrow$ But limited by a lack of **intelligent embedded controls**.

AI & Optimal Control

- *Artificial intelligence* and *deep learning* : outperforms human operators on ever-increasing range of tasks [2].

- *Optimal control theory* : compute **continuous controls that dynamically optimize an objective function** $\rightarrow$ can describe self-controlled robots

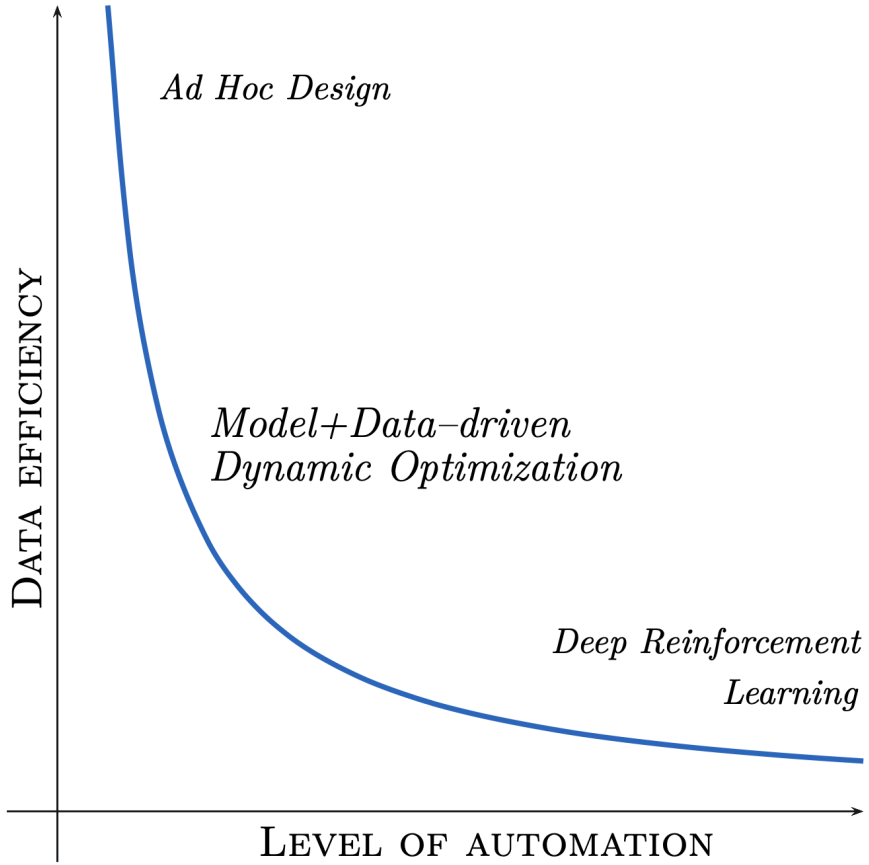


Figure 2: Automation level against data efficiency. Reprinted from [3] with permission.

Objective and Plan

- Objective : Build controller capable of navigating the agent from an initial to a final position $\rightarrow$ **trajectory planning problem**

- Criteria : Aiming for **time-energy optimality** $\rightarrow$ minimizing transportation time + magnitude of controls

- Plan : 1. Derive physics-informed *optimal control conditions*
  2. Effortlessly generate *synthetic dataset* based on optimal trajectories
  3. Use *deep learning model* to fit the data and synthesize a controller

The quadrotor drone setting

Obtain quantitative results + leverage physics-led intuitions

$\rightarrow$ focusing on **quadrotor drone** UAVs

Figure 3: Walkera QR X350 Quadcopter hovering [4]

- *Variety of applications such as* : infrastructure inspection - search and rescue operations - aerial photography - ...
- *Benefits from automation* : traffic management - aerial delivery - road maintenance - fire watch flights - ...

Model and optimal control problem

The planar-quadrotor

The planar-quadrotor model : 2-dimensional UAV moving in the $\mathbb{R}^2$ plane $\rightarrow$ simpler dynamics **yet capturing essence of control-motion relationship**.

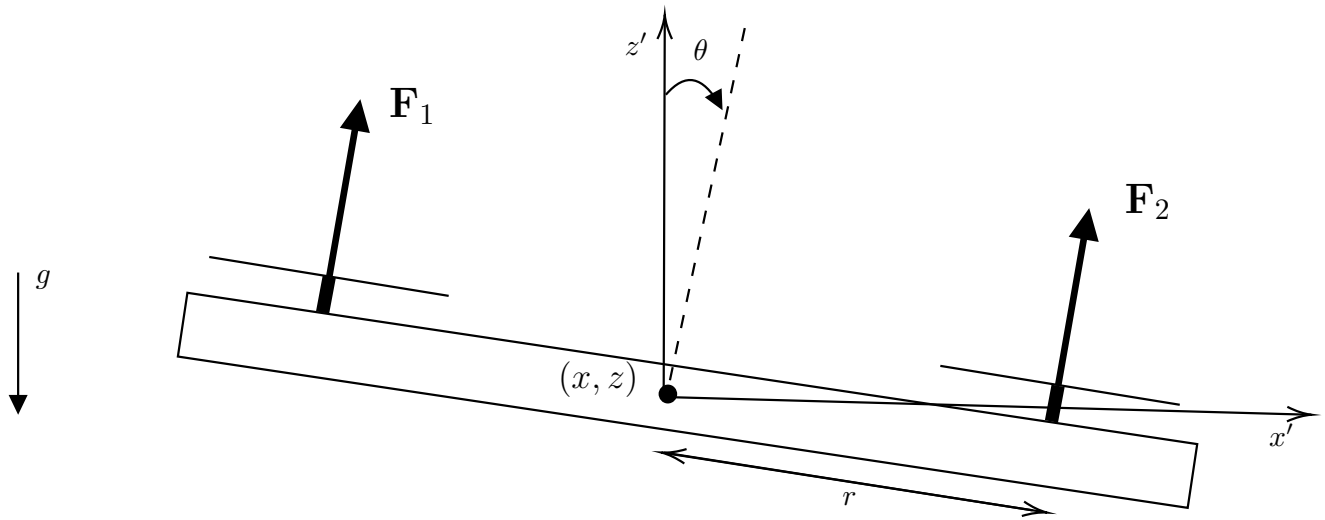


Figure 4: The planar-quadrotor model.

The model is extensively studied in the related literature [5, 6, 7, 8, 9].

Dynamics I

The dynamics governing the UAV's motion are given by Newton's laws of motion.

$$\begin{cases} m\ddot{x} = (F_1 + F_2)\sin\theta \\ m\ddot{z} = (F_1 + F_2)\cos\theta - mg \\ I\ddot{\theta} = r(F_1 - F_2) \end{cases} \tag{1}$$

Our general control is set as in the following.

$$\mathbf{u} := (u_T, u_R)^\top, \text{ where } u_T := \frac{F_1 + F_2}{m} \text{ and } u_R := \dot{\theta} \tag{2}$$

The quadrotor UAV is characterized by the *state vector* given hereby.

$$\mathbf{q} := (x, \dot{x}, z, \dot{z}, \theta)^\top \tag{3}$$

Dynamics II

The controlled dynamics are defined in the compact following way.

$$\dot{\mathbf{q}} = \boldsymbol{f}(\mathbf{q}, \mathbf{u}), \text{ where } \boldsymbol{f} \text{ is given by } \boldsymbol{f}(\mathbf{q}, \mathbf{u}) := \begin{pmatrix} \dot{x} \\ u_T \sin\theta \\ \dot{z} \\ u_T \cos\theta - g \\ u_R \end{pmatrix} \tag{4}$$

- Planar-quadrotor model : **arises with 5 equations dynamics**
- **The curse of dimensionality** $\rightarrow$ computing numerical optimal solutions becomes intractable as the number of equations and parameters grows
- Bypass this downfall using deep learning to compute controls **in real-time and on the edge**

Bolza problem

We consider the following *Bolza problem*.

$$\begin{aligned} \min_{\mathbf{u}(\cdot)} \quad & J[\mathbf{u}(\cdot)] := \ t_f + \int_0^{t_f} \|\mathbf{u}\|^2 dt \\ \text{s.t.} \quad & \mathbf{q}(t = t_f) = \mathbf{q}_f, \\ & \mathbf{q}(t = 0) = \mathbf{q}_0, \\ & \dot{\mathbf{q}} = \boldsymbol{f}(\mathbf{q}, \mathbf{u}), \\ & \mathbf{u} \in U, \\ & t \in [0, t_f] \end{aligned} \tag{OCP}$$

- Admissible controls in U are such that $\underline{u_T} \leq u_T(t) \leq \overline{u_T}$ and $|u_R(t)| \leq \overline{u_R}$
- Open loop controls $\rightarrow$ solves the (OCP) equation solely based on the initial state $\mathbf{q}_0$, in the form $\mathbf{u}^\star = \mathbf{u}^\star(t, \mathbf{q}_0)$
- **Closed loop controls** $\rightarrow$ instant measured state is used to condition the upcoming controls, in the form $\mathbf{u}^\star = \mathbf{u}^\star(t, \mathbf{q})$

Pontryagin's minimum principle

- We denote $\boldsymbol{\lambda}$ the **costate vector** and H the **Hamiltonian**
- An optimal feedback control $\mathbf{u}^\star = \mathbf{u}^\star(t, \mathbf{q}, \boldsymbol{\lambda})$ verifies the following **Pontryagin's minimum principle equations**
- The (PMP) **boundary value problem provides a necessary condition for optimality**[1]

$$\begin{cases} \mathbf{u}^\star = \arg\min_{\mathbf{u}(\cdot)\in U} H(t, \mathbf{q}, \boldsymbol{\lambda}, \mathbf{u}) \\ \dot{\mathbf{q}} = \frac{\partial H}{\partial \boldsymbol{\lambda}} = \boldsymbol{f}(\mathbf{q}, \mathbf{u}^\star), \quad \mathbf{q}(0) = \mathbf{q}_0, \quad \mathbf{q}(t_f) = \mathbf{q}_f \qquad \text{(PMP)} \\ -\dot{\boldsymbol{\lambda}}(t) = \frac{\partial H}{\partial \mathbf{q}}\Big(t, \mathbf{q}, \boldsymbol{\lambda}, \mathbf{u}^\star(t, \mathbf{q}, \boldsymbol{\lambda})\Big) \qquad \text{(adjoint equation)} \end{cases}$$

[1] See manuscript for Hamilton-Jacobi-Bellman equation approach and relationship to PMP conditions

Solving controls

Finding optimal controls and trajectories :

1. Specifying the Hamiltonian

$$H(t,\mathbf{q},\boldsymbol{\lambda},\mathbf{u})=1+\lambda_1\dot{x}+\lambda_3\dot{z}-\lambda_4 g+\left(\lambda_5 u_R+u_R^2\right)+\left((\lambda_2\sin\theta+\lambda_4\cos\theta)u_T+u_T^2\right) \tag{5}$$

2. Computing the costate variables using adjoint equation

→ The costate variables can be **parametrized by 4 constants** $c_1, c_2, c_3, c_4 \in \mathbb{R}^4$ and vary with time and state

3. Minimizing H with respect to controls

$$u_T^\star(t)=\min\left(\max\left(a(t),\underline{u_T}\right),\overline{u_T}\right) \quad \text{and} \quad u_R^\star(t)=\min\left(\max\left(b(t),-\overline{u_R}\right),\overline{u_R}\right) \tag{6}$$

→ The regulating function a is fully determined by the state and **the 4 constants**

→ The regulating function b is determined as an **initial value problem** parameterized by **the 4 constants**

Augmented system

- We build an **augmented system** using a vector $\mathbf{y} := (x, \dot{x}, z, \dot{z}, \theta, b)^\top$
- We solve the related augmented **initial value problem** to generate an appropriate trajectory :
  - State components are computed according to the dynamics
  - The controls u_T and u_R are computed using (6) $\rightarrow$ Using a and b

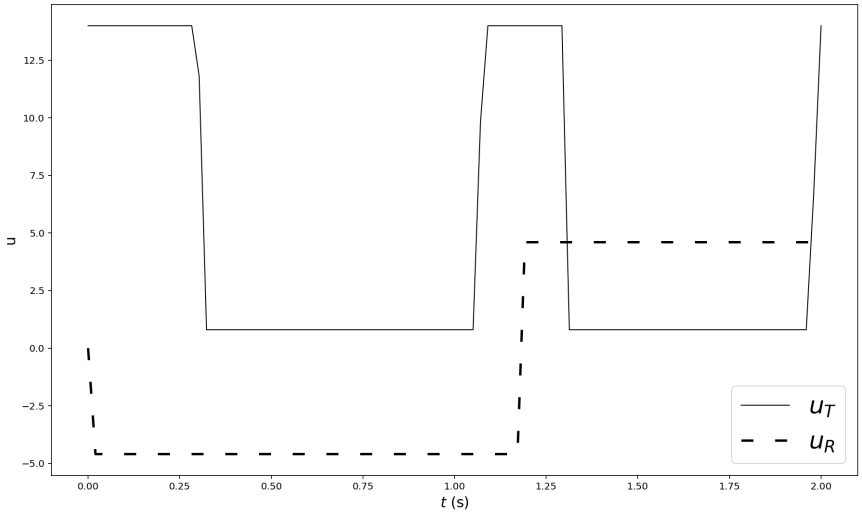
$$\dot{\mathbf{y}} = \boldsymbol{g}(\mathbf{y}, \mathbf{u}) := \begin{pmatrix} \dot{x} \\ u_T \sin\theta \\ \dot{z} \\ u_T \cos\theta - g \\ u_R \\ \dot{b}(t) := \left(\frac{(c_2 - c_1 t)\cos\theta - (c_4 - c_3 t)\sin\theta}{2} \right) u_T \end{pmatrix}, \quad \mathbf{y}(0) = (\mathbf{q}_0, 0)^\top \tag{7}$$

- **Any choice of 4 constants c_1, c_2, c_3, c_4 fully determines a trajectory**
- **There exists a latent space of optimal trajectories included in $\mathbb{R}^4$**

Generating dataset

Example trajectory I

- What is the latent space structure ?
- We generate a sizeable amount of trajectories and compare to **solver** obtained ones (ICLOCS2/IPOPT)
- We present an example starting at $\mathbf{q}_0 = \left(9.48,\ -1.12,\ -4.29,\ 7.14,\ 0.86\right)^\top$ and arriving at $\mathbf{q}(t_f = 2) = \left(10.64,\ 1.87,\ -3.78,\ -10.12,\ -0.87\right)^\top$



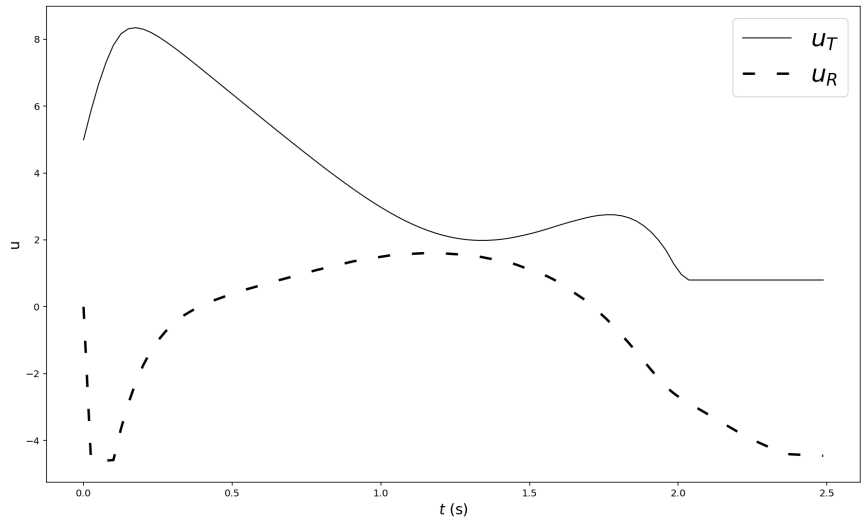
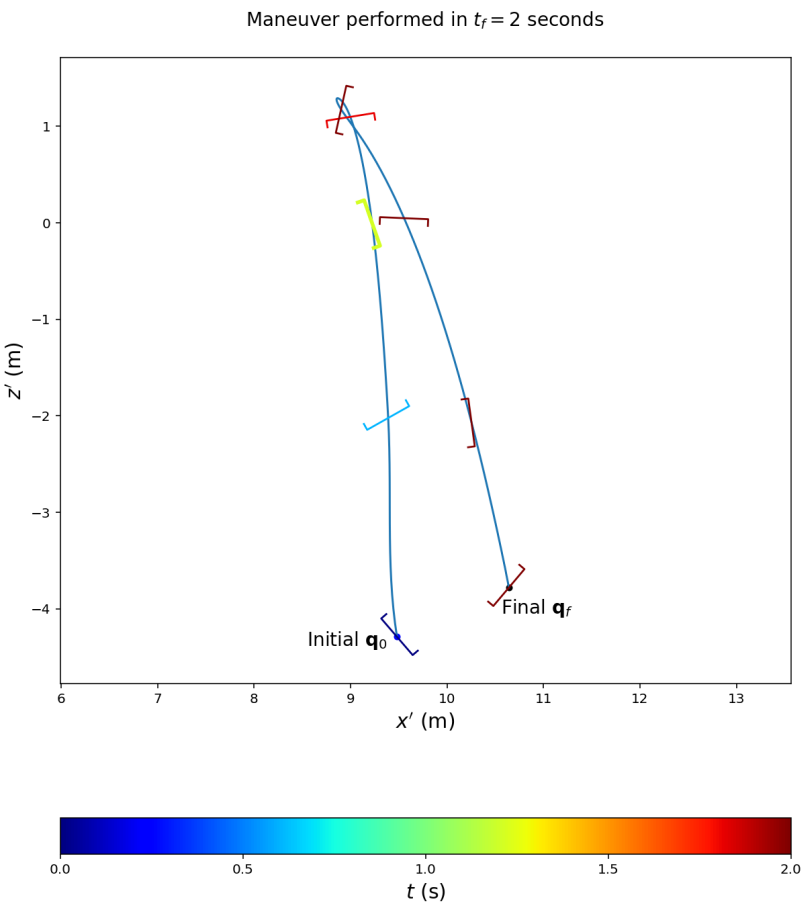


Figure 5: (a) Theoretical controls (b) Solver-based controls

Example trajectory II



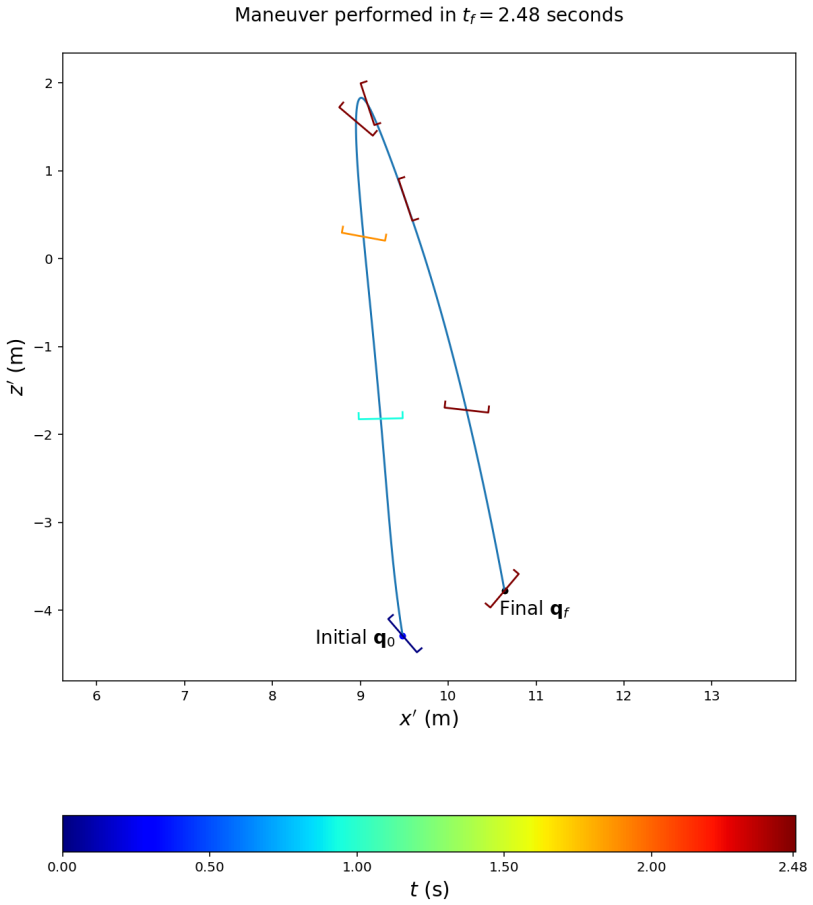


Figure 6: (a) Theoretical trajectory

(b) Solver-based trajectory

Example trajectory III

	Theoretically generated	Solver-based
t_f	**2.00**	2.48
E	65.42	**57.36**
J	67.42	**59.84**

Table 1: Cost functionals and contributions associated with the example trajectory.

- Theoretically generated trajectory and controls are slightly more **time-optimal**
- **Energy consumption** is lower for the solver-based controls
- The cost functional J is roughly 12.5% higher for the theoretically generated trajectory $\rightarrow$ Near-optimal

- **The latent space is a strict subset of** $\mathbb{R}^4$
- **Trajectories generated from any** $(c_1, c_2, c_3, c_4) \in \mathbb{R}^4$ **will be considered as optimal**

Data representivity

- We can **shape the data** according to our needs :
  - The constants + initial states + time horizons are **randomly sampled** (combination of Normal / $\log$-Normal / custom uniform distributions)
  - We **require 'evenly' distributed trajectories** (relative position of initial and final locations - relative orientation - distance traveled - ...)

- **Choosing the predictor** $\hat{\mathbf{q}}(t) := \mathbf{q}_f - \mathbf{q}(t)$, with $\hat{\theta}(t) \equiv \theta_f - \theta(t) \mod 2\pi$ and transform the angle according to $\hat{\theta} \to (\cos\hat{\theta}, \sin\hat{\theta})$

We build the dataset $\mathcal{D} := \left\{ \left(\hat{\mathbf{q}}_{(i)}, \mathbf{u}_{(i)} \right) \right\}_{i=1}^{M} \subset X \times U$ using a Runge-Kutta method of order 4 implemented in `Python`, discarding over-represented trajectories

- Very **efficient method** $\to$ built a dataset composed of $123\,750\,000$ points
- We use a $85\%/15\%$ split to divide the dataset into a **training set** and a **validation set**

Deep neural network for near-optimal closed loop controls

Model

The deep neural network model is composed of :

- An input layer made of 6 neurons
- 5 hidden layers made of 900 neurons each and activated using ReLU functions ($x \to \max(0, x)$)
- All connections between hidden layers are subject to dropout with a probability of 25%
- An output layer made of 2 neurons, respectively activated using **modified sigmoids** $\sigma_T(x) := (\overline{u_T} - \underline{u_T}) \frac{1}{1+e^{-x}} + \underline{u_T}$ and $\sigma_R(x) := \overline{u_R} \left(\frac{2}{1+e^{-x}} - 1 \right)$

The network is composed of $3\,251\,702$ trainable parameters. The DNN is optimized using **stochastic gradient descent** and the training is made performing early stopping

From $z = 0$ to $z = 1$ I

- We present the 'take-off' trajectory starting at $\mathbf{q}_0 = \left(0,\, 0,\, 0,\, 0,\, 0\right)^\top$ and **targeting** $\mathbf{q}_f = \left(0,\, 0,\, 1,\, 0,\, 0\right)^\top \rightarrow$ The trajectory is now **closed-loop**

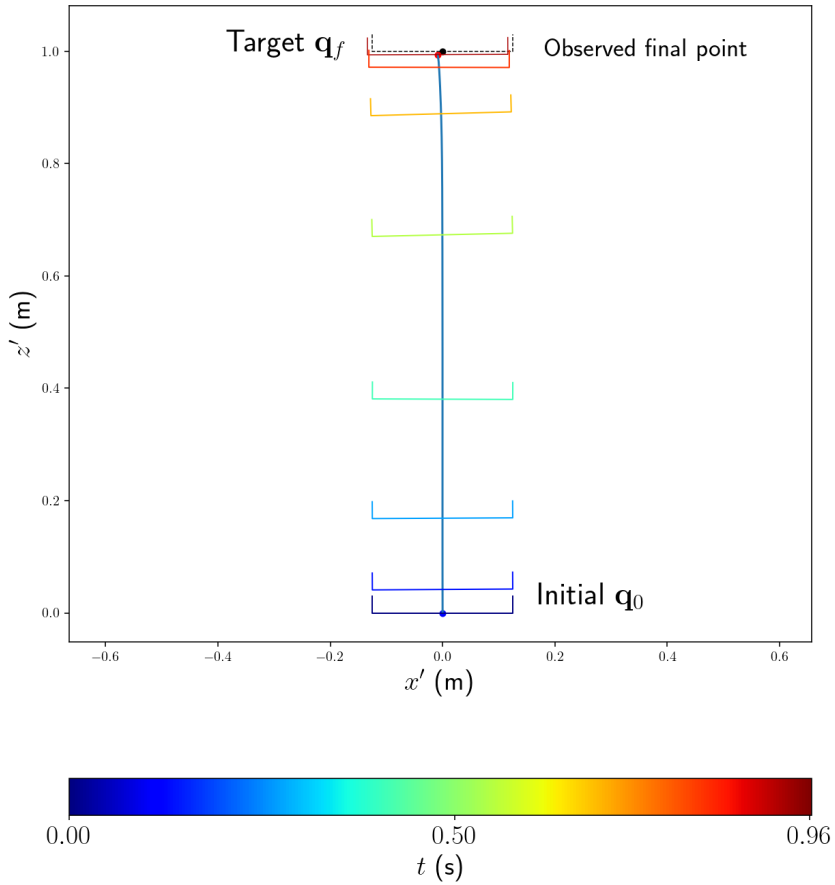


Figure 7: Closed loop quadrotor take-off trajectory.

From $z = 0$ to $z = 1$ II

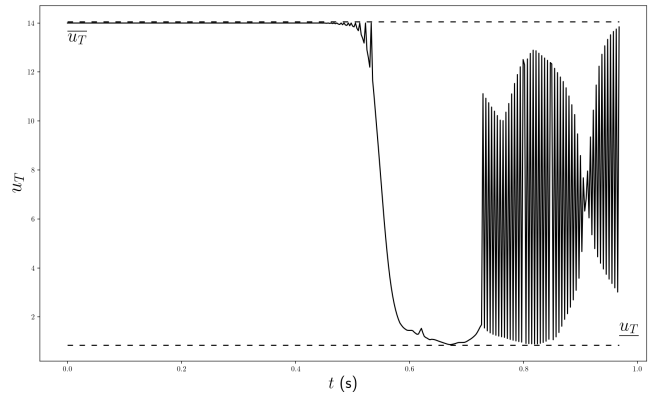
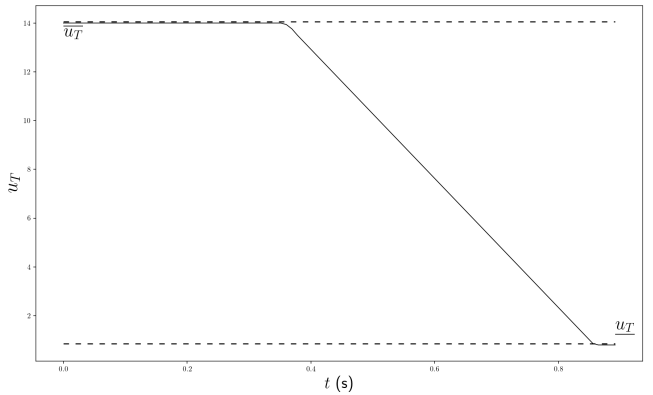


Figure 8: (a) Feedback u_T control



(b) Solver-based u_T control

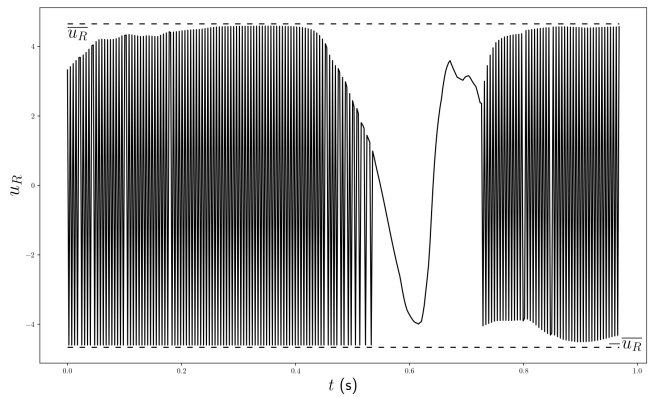


Figure 9: (a) Feedback u_R control

(b) Solver-based u_R control

Randomly selected trajectory I

- We present a randomly drawn closed loop quadrotor trajectory $\rightarrow$ starting at $\mathbf{q}_0 = \big(-0.69,\ -1.29,\ 4.07,\ 7.40,\ 6.21\big)^\top$, targeting final state $\mathbf{q}_f = \big(-6.51,\ -8.47,\ 10.38,\ 2.78,\ 6.25\big)^\top$

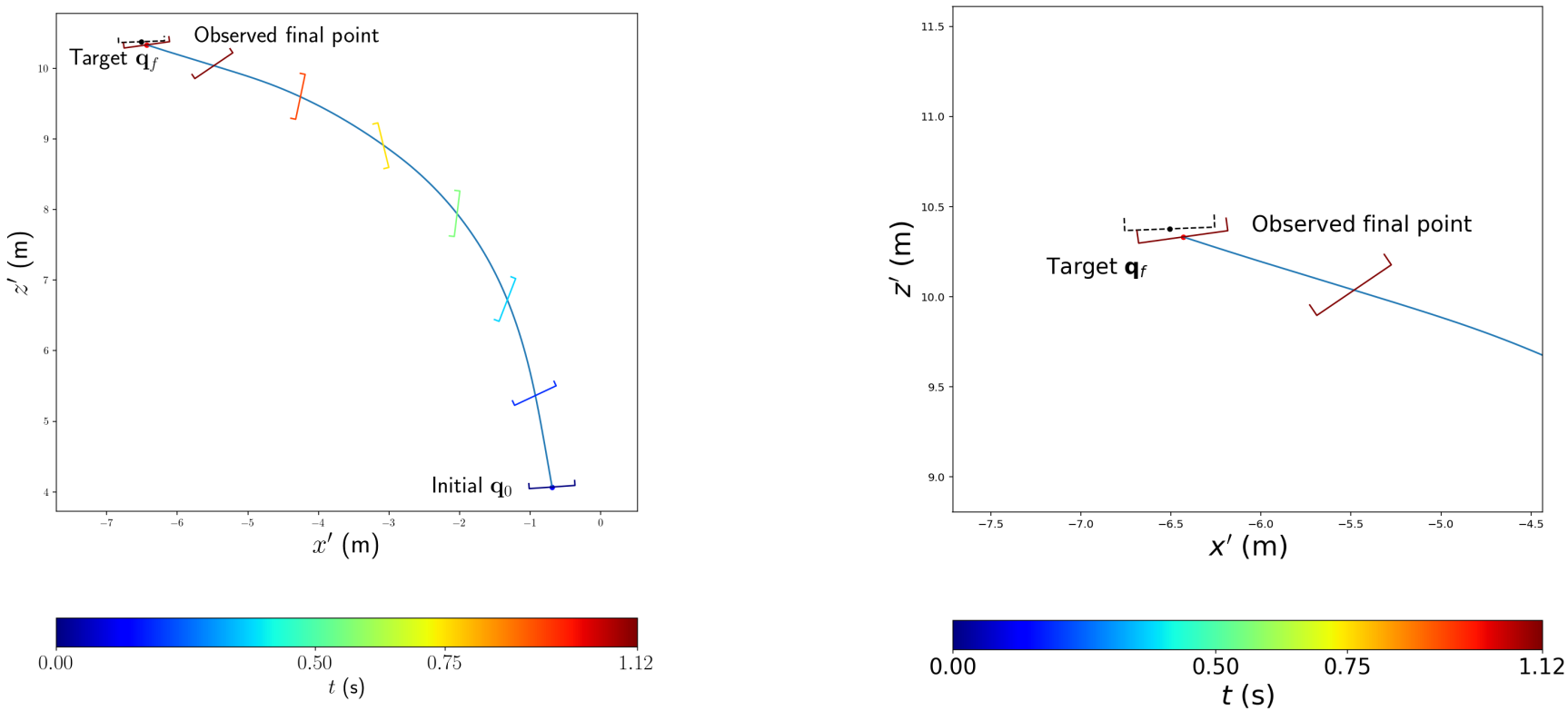
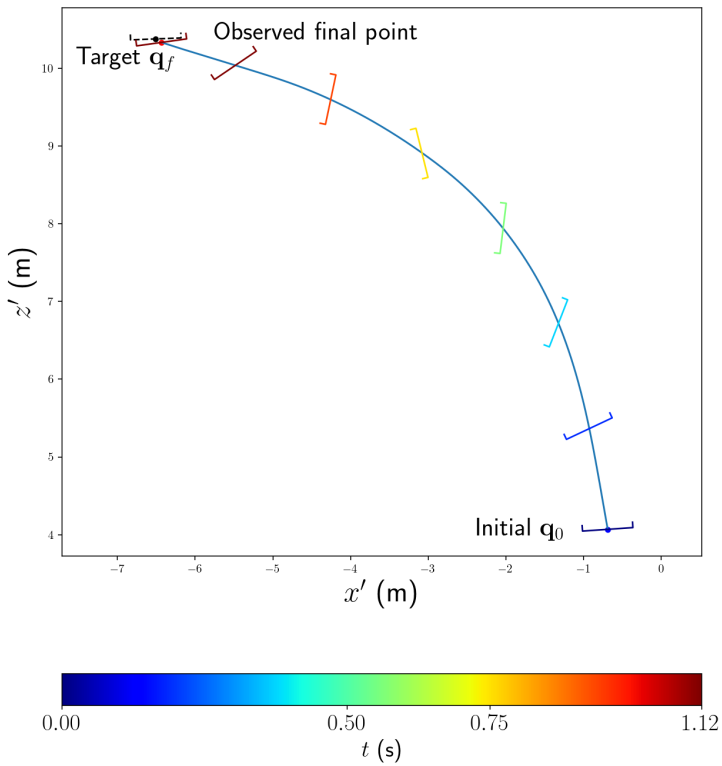


Figure 10: Randomly sampled closed loop quadrotor trajectory

Randomly selected trajectory II

- We compare the randomly sampled trajectory with solver-based one



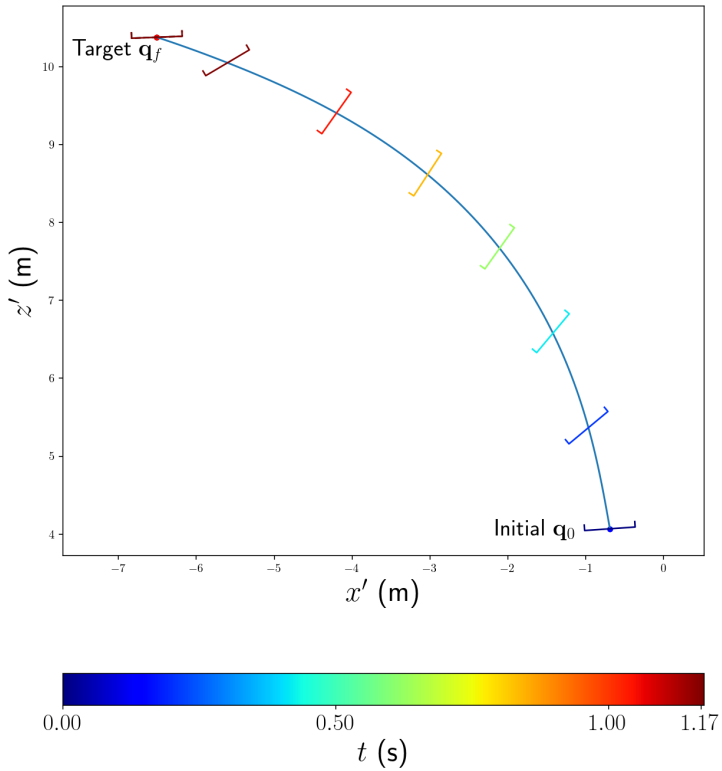
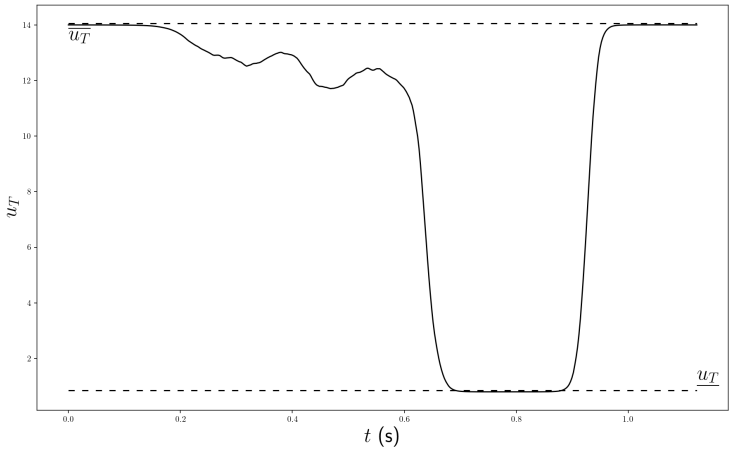


Figure 11: (a) Closed loop trajectory

(b) Solver-based trajectory

Randomly selected trajectory III



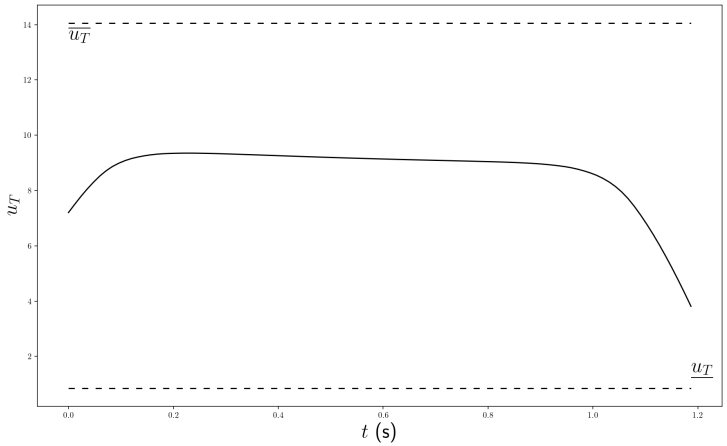
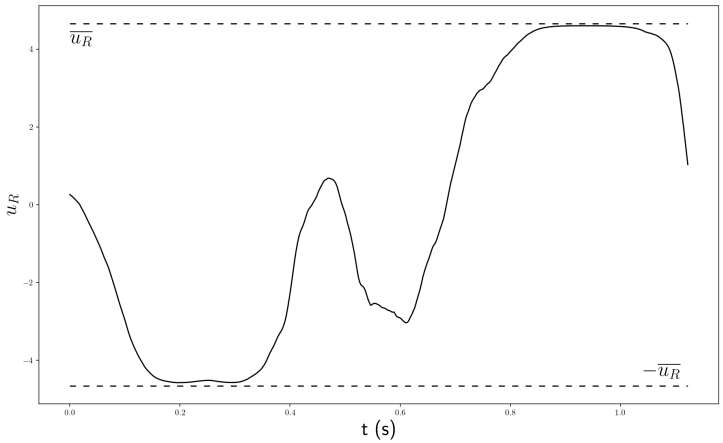


Figure 12: (a) Feedback u_T control

(b) Solver-based u_T control



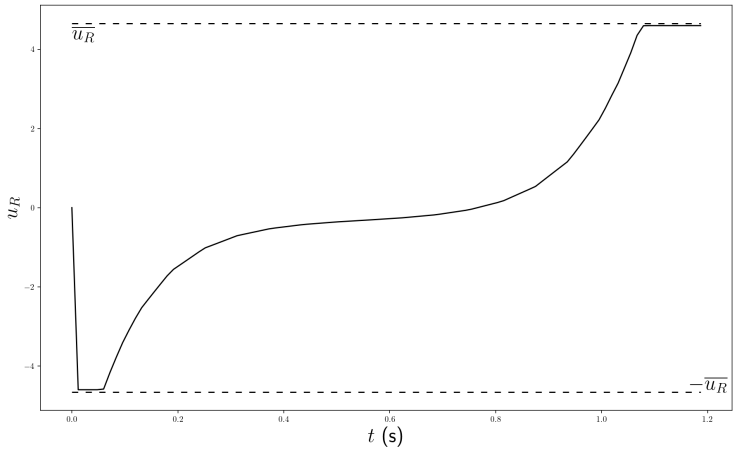


Figure 13: (a) Feedback u_R control

(b) Solver-based u_R control

Randomly selected trajectory IV

	DNN-generated	Solver-based
t_f	**1.12**	1.17
E	116.25	**97.12**
J	117.37	**98.29**

Table 2: Cost functionals and contributions associated with the randomly sampled trajectory.

- The closed loop controller performs the navigation in $t_f = 1.12$ seconds
- More time-optimal closed loop controls but less energy optimal $\rightarrow$ J is 19 % higher in this case
- Closed loop controls have a tendency to vary too strongly at first, before correcting in a second time
- Yet, they can be applied in real-time on the edge

Conclusion

Main results

- **A sizeable dataset of near-optimal trajectories was generated efficiently**
- A deep neural network was designed and trained effectively
- **The quadrotor joins $\mathbf{q}_f$ within a reasonable tolerance autonomously for a wide range of drawn initial and final states**

- Some 'atypical' final states remain un-reached by the UAV $\rightarrow$ we propose a set of improvements

Future research

Several future axis can be considered :

- Separate the controller in two different networks
- Consider augmented predictors including the time variable, in the form of $\hat{\mathbf{q}} = (x_f - x,\, \dot{x}_f - \dot{x},\, z_f - z,\, \dot{z}_f - \dot{z},\, \cos(\theta_f - \theta),\, \sin(\theta_f - \theta),\, t)^\top$
- Take into account progressiveness and temporal dependence of the state control relationship $\rightarrow$ Use long short-term memory (LSTM) with built-in feedback loops
- Explore latent space $\rightarrow$ A linear relationship exists between constants c_1, c_2, c_3, c_4
- Optimal control boosted deep reinforcement learning $\rightarrow$ Use deep deterministic policy gradient method (DDPG) with our kickstarted model
- Assume bang-bang controls $\rightarrow$ Go from regression to classification

References I

[1] Pieter Abbeel. Deep learning for robotics. Neural Information Processing Systems - NeurIPS, conference, 2017.

[2] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[3] Dante Kalise. Week 11: Dynamic optimization, lecture notes. Optimization, MATH70005. Imperial College London, delivered 21 March 2022.

[4] Wikimedia Commons. Walkera QR X350 Quadcopter Hovering. `https://commons.wikimedia.org/wiki/File:Walkera_QR_X350_Quadcopter_Hovering.jpg`, 2013. Accessed: 03-07-22.

[5] Markus Hehn, Robin Ritz, and Raffaello D'Andrea. Performance benchmarking of quadrotor systems using time-optimal control. *Autonomous Robots*, 33(1):69–88, 2012.

References II

[6] Robin Ritz, Markus Hehn, Sergei Lupashin, and Raffaello D'Andrea. Quadrocopter performance benchmarking using optimal control. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5179–5186. IEEE, 2011.

[7] Zhikun Wang, Roderich Groß, and Shiyu Zhao. Aerobatic tic-toc control of planar quadcopters via reinforcement learning. *IEEE Robotics and Automation Letters*, 7(2):2140–2147, 2022.

[8] Praveen Venkatesh, Sanket Vadhvana, and Varun Jain. Analysis and control of a planar quadrotor. *arXiv preprint arXiv:2106.15134*, 2021.

[9] Teodor Tomić, Moritz Maier, and Sami Haddadin. Learning quadrotor maneuvers from optimal control and generalizing in real-time. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1747–1754. IEEE, 2014.

Thank you !