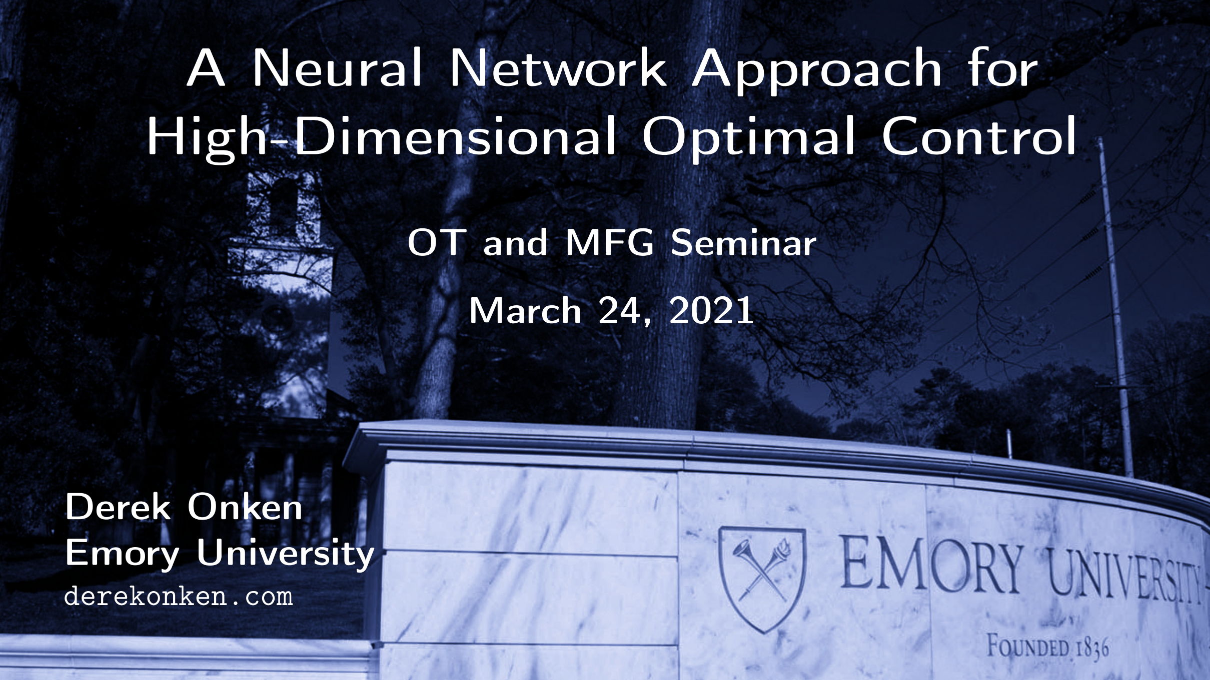
A Neural Network Approach for High-Dimensional Optimal Control

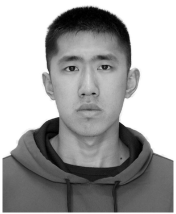
OT and MFG Seminar

March 24, 2021

Derek Onken
Emory University
derekonken.com



Collaborators and Acknowledgments



Xingjian Li
Emory

Lars Ruthotto
Emory

Samy Wu Fung
UCLA

Levon Nurbekyan
UCLA

Stan Osher
UCLA

Funding: UNITEDHEALTH GROUP®

Special thanks: Organizers and staff of IPAM Long Program MLP 2019 and NVIDIA.

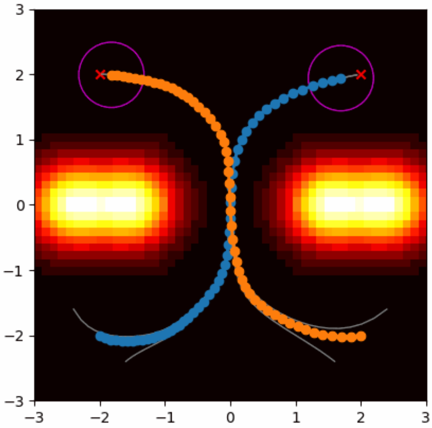
Overview

- **Background**
  - Problem
  - Pontryagin Maximum Principle (PMP)
  - Hamilton–Jacobi–Bellman Partial Differential Equation (HJB)
- **Mathematical Formulation**
  - Shock-Robustness
  - HJB Penalizers
- **Neural Networks (NNs)**
  - Model Formulation
  - Numerics
- **Results**
  - 150-Dimensional Swarm Trajectory Planning
  - Quadcopter with Complicated Dynamics
- **Conclusion**

Optimal Control (OC) Problem

Corridor Problem

Consider two *centrally-controlled* agents that navigate through a corridor/valley between two hills to fixed targets



Assume

- We have control over the agents' velocities (the *control*)

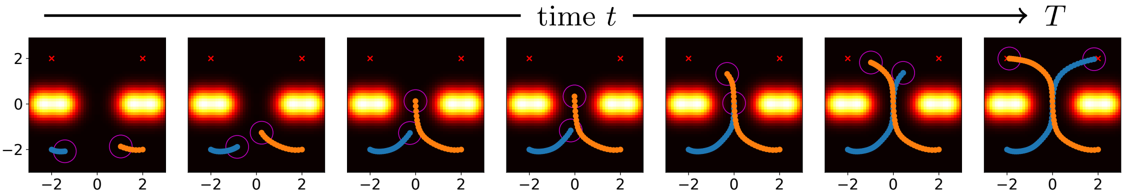
Want

- Shortest paths, e.g. the geodesics (*optimality*)
- No collisions
- Agents to reach targets at final time

Multi-Agent Formulation

Consider n agents initially at $x_1, \ldots, x_n \in \mathbb{R}^q \quad \Rightarrow \boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{R}^d$

Agents follow trajectories $\boldsymbol{z}_{\boldsymbol{x}}(t)$ during time $t \in [0, T]$



Initial

$$\boldsymbol{z}_{\boldsymbol{x}}(0) = \boldsymbol{x} = \begin{bmatrix} -2 \\ -2 \\ 2 \\ -2 \end{bmatrix} \begin{matrix} \left.\vphantom{\begin{matrix} -2 \\ -2 \end{matrix}}\right\} \text{agent } 1 \\ \left.\vphantom{\begin{matrix} 2 \\ -2 \end{matrix}}\right\} \text{agent } 2 \end{matrix}$$

Target

$$\boldsymbol{y} = \begin{bmatrix} 2 \\ 2 \\ -2 \\ 2 \end{bmatrix}$$

Terminal Cost

$$G\big(\boldsymbol{z}_{\boldsymbol{x}}(T)\big) = \frac{\alpha_1}{2} \|\boldsymbol{z}_{\boldsymbol{x}}(T) - \boldsymbol{y}\|^2$$

for multiplier $\alpha_1 \in \mathbb{R}$

Trajectories Governed by Differential Equation

The state $\boldsymbol{z_x}$ depends on the control $\boldsymbol{u_x}$ and previous state via the system

$$\partial_t \boldsymbol{z_x}(t) = f\big(t, \boldsymbol{z_x}(t), \boldsymbol{u_x}(t)\big), \quad \boldsymbol{z_x}(0) = \boldsymbol{x} \tag{1}$$

For Corridor: $= \boldsymbol{u_x}(t)$ (the velocity)

where

- time $t \in [0, T]$
- initial state $\boldsymbol{x} \in \mathbb{R}^d$
- admissible controls $U \subset \mathbb{R}^a$
- $f\colon [0,T] \times \mathbb{R}^d \times U \to \mathbb{R}^d$ models the evolution of the state $\boldsymbol{z_x}\colon [0,T] \to \mathbb{R}^d$ in response to the control $\boldsymbol{u_x}\colon [0,T] \to U$

Running Cost

Running costs where z_i and u_i are the state and control for the ith agent, respectively

$$L\big(t, \boldsymbol{z}(t), \boldsymbol{u}(t)\big) = E\big(\boldsymbol{z}(t), \boldsymbol{u}(t)\big) + \alpha_2 Q\big(\boldsymbol{z}(t), \boldsymbol{u}(t)\big) + \alpha_3 W\big(\boldsymbol{z}(t), \boldsymbol{u}(t)\big)$$

$$= \sum_{i=1}^{n} \underbrace{E_i\big(z_i(t), u_i(t)\big)}_{\text{For Corridor: } \frac{1}{2}\|u_i(t)\|^2} + \alpha_2 \sum_{i=1}^{n} \underbrace{Q_i\big(z_i(t), u_i(t)\big)}_{\text{sum of Gaussians}} + \alpha_3 \sum_{j \neq i} \underbrace{W_{ij}\big(z_i(t), z_j(t)\big)}_{\text{piecewise Gaussian repulsion}}$$

for multipliers $\alpha_2, \alpha_3 \in \mathbb{R}$ and

- E_i is the energy of an agent,
- Q_i represents any obstacles or terrain,
- W_{ij} are the interaction costs between homogeneous agents i and j with radius r

$$W_{ij}(z_i, z_j) = \begin{cases} \exp\left(-\frac{\|z_i - z_j\|_2^2}{2r^2}\right), & \|z_i - z_j\|_2 < 2r \\ 0, & \text{otherwise} \end{cases}$$

Optimal Control (OC) Problem

Running Cost: $L(s,\cdot) = E(\cdot) + \alpha_2 Q(\cdot) + \alpha_3 W(\cdot)$
Terminal Cost: $G\big(\boldsymbol{z_x}(T)\big) = \frac{\alpha_1}{2}\|\boldsymbol{z_x}(T) - \boldsymbol{y}\|^2$

Goal: Find the control that incurs minimal cost[1]

$$\Phi(t,\boldsymbol{x}) = \inf_{\boldsymbol{u_x}} \left\{ \int_t^T L\big(s, \boldsymbol{z_x}(s), \boldsymbol{u_x}(s)\big)\, \mathrm{d}s + G\big(\boldsymbol{z_x}(T)\big) \right\} \tag{2}$$

- $\Phi(t,\boldsymbol{x}) \in \mathbb{R}$ is the *value function* (i.e., optimal cost-to-go)
- solution $\boldsymbol{u}_{\boldsymbol{x}}^*$ is the *optimal control*
- *optimal trajectory* $\boldsymbol{z}_{\boldsymbol{x}}^*$ dictated by $\boldsymbol{u}_{\boldsymbol{x}}^*$

[1] Fleming and Soner. *Controlled Markov Processes and Viscosity Solutions*. 2006.

Pontryagin Maximum Principle (PMP)

Existing Approach

Solve the forward-backward system[2] for $0 \le t \le T$

$$\begin{cases} \partial_t \boldsymbol{z}_{\boldsymbol{x}}^*(t) = -\nabla_{\boldsymbol{p}} H\big(t, \boldsymbol{z}_{\boldsymbol{x}}^*(t), \boldsymbol{p}_{\boldsymbol{x}}(t)\big), \\ \partial_t \boldsymbol{p}_{\boldsymbol{x}}(t) = \nabla_{\boldsymbol{x}} H\big(t, \boldsymbol{z}_{\boldsymbol{x}}^*(t), \boldsymbol{p}_{\boldsymbol{x}}(t)\big), \\ \boldsymbol{z}_{\boldsymbol{x}}^*(0) = \boldsymbol{x}, \quad \boldsymbol{p}_{\boldsymbol{x}}(T) = \nabla G\big(\boldsymbol{z}_{\boldsymbol{x}}^*(T)\big), \end{cases} \tag{3}$$

where

- Hamiltonian $H(t, \boldsymbol{x}, \boldsymbol{p}_{\boldsymbol{x}}) = \sup_{\boldsymbol{u}_{\boldsymbol{x}} \in U} \{-\boldsymbol{p}_{\boldsymbol{x}} \cdot f(t, \boldsymbol{x}, \boldsymbol{u}_{\boldsymbol{x}}) - L(t, \boldsymbol{x}, \boldsymbol{u}_{\boldsymbol{x}})\}$
- adjoint $\boldsymbol{p}_{\boldsymbol{x}} \colon [0, T] \to \mathbb{R}^d$

then notation-wise, we have $\boldsymbol{u}_{\boldsymbol{x}}^*(t) = \boldsymbol{u}^*\big(t, \boldsymbol{z}_{\boldsymbol{x}}^*(t), \boldsymbol{p}_{\boldsymbol{x}}(t)\big)$

[2] Pontryagin et al. *The Mathematical Theory of Optimal Processes*. 1962.

Pontryagin Maximum Principle (PMP)

Existing Approach

Solve the forward-backward system[2] for $0 \le t \le T$

$$\begin{cases} \partial_t \boldsymbol{z}_{\boldsymbol{x}}^*(t) = -\nabla_{\boldsymbol{p}} H\big(t, \boldsymbol{z}_{\boldsymbol{x}}^*(t), \boldsymbol{p}_{\boldsymbol{x}}(t)\big), \\ \partial_t \boldsymbol{p}_{\boldsymbol{x}}(t) = \nabla_{\boldsymbol{x}} H\big(t, \boldsymbol{z}_{\boldsymbol{x}}^*(t), \boldsymbol{p}_{\boldsymbol{x}}(t)\big), \\ \boldsymbol{z}_{\boldsymbol{x}}^*(0) = \boldsymbol{x}, \quad \boldsymbol{p}_{\boldsymbol{x}}(T) = \nabla G\big(\boldsymbol{z}_{\boldsymbol{x}}^*(T)\big), \end{cases} \tag{3}$$

where

- Hamiltonian $H(t, \boldsymbol{x}, \boldsymbol{p}_{\boldsymbol{x}}) = \sup_{\boldsymbol{u}_{\boldsymbol{x}} \in U} \{-\boldsymbol{p}_{\boldsymbol{x}} \cdot f(t, \boldsymbol{x}, \boldsymbol{u}_{\boldsymbol{x}}) - L(t, \boldsymbol{x}, \boldsymbol{u}_{\boldsymbol{x}})\}$
- adjoint $\boldsymbol{p}_{\boldsymbol{x}} : [0, T] \to \mathbb{R}^d$

then notation-wise, we have $\boldsymbol{u}_{\boldsymbol{x}}^*(t) = \boldsymbol{u}^*\big(t, \boldsymbol{z}_{\boldsymbol{x}}^*(t), \boldsymbol{p}_{\boldsymbol{x}}(t)\big)$

Comments

- *Local* solution method
  - Solved for a single $\boldsymbol{x}$
  - For a new $\boldsymbol{x}$, need to resolve (3)
- Solving the system is difficult and depends on the initial guess $\boldsymbol{p}_{\boldsymbol{x}}(0)$ (if using a shooting method)

[2]Pontryagin et al. *The Mathematical Theory of Optimal Processes*. 1962.

Hamilton-Jacobi-Bellman (HJB)

Existing Approach

Solve the HJB PDE[3]
(also called *dynamic programming* equations)

$$\begin{cases} -\partial_t \Phi(t, \boldsymbol{x}) = -H\big(t, \boldsymbol{x}, \nabla\Phi(t, \boldsymbol{x})\big), \\ \Phi(T, \boldsymbol{x}) = G(\boldsymbol{x}) \end{cases} \tag{4}$$

arises from correspondence

$$\boldsymbol{p}_{\boldsymbol{x}}(t) = \nabla\Phi\big(t, \boldsymbol{z}^*_{\boldsymbol{x}}(t)\big) \tag{5}$$

[3] Bellman. *Dynamic Programming*. 1957.

Hamilton-Jacobi-Bellman (HJB)

Existing Approach

Solve the HJB PDE[3]
(also called *dynamic programming* equations)

$$\begin{cases} -\partial_t \Phi(t, \boldsymbol{x}) = -H\big(t, \boldsymbol{x}, \nabla \Phi(t, \boldsymbol{x})\big), \\ \Phi(T, \boldsymbol{x}) = G(\boldsymbol{x}) \end{cases} \tag{4}$$

arises from correspondence

$$\boldsymbol{p}_{\boldsymbol{x}}(t) = \nabla \Phi\big(t, \boldsymbol{z}^*_{\boldsymbol{x}}(t)\big) \tag{5}$$

Comments

- *Global* solution method
  - Solved for all $\boldsymbol{x}$
  - For a new $\boldsymbol{x}$, no recomputation
- Need grids to solve (4), which scale poorly to high-dimensions

[3] Bellman. *Dynamic Programming*. 1957.

Our Approach

Motivation

Want:

- Semi-global solution method (from HJB)

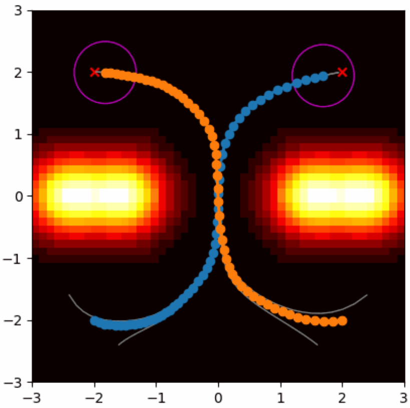
  $\Rightarrow$ one model useful for many initial conditions

  $\Rightarrow$ method is robust to shocks/disturbances

- High-dimensional (from PMP)

  $\Rightarrow$ multi-agent problems provide high dimensionality and are easy to visualize

Corridor Problem



Semi-Global Solution Method

Robust to Shocks

Want: semi-global Φ (value function)

How to obtain:

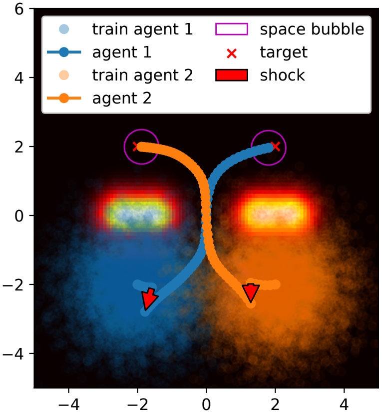
- Solve for Hamiltonian H
- Replace adjoint $\boldsymbol{p}$ with $\nabla\Phi$ using (5)
- Use initial states sampled from Gaussian distribution
- Solve

$$\min_{\Phi} \mathbb{E}_{\boldsymbol{x}\sim\mathcal{N}(\mu,\boldsymbol{\Sigma})} \left\{ \int_0^T L\big(s, \boldsymbol{z_x}(s), \boldsymbol{u_x}(s)\big)\, \mathrm{d}s + G\big(\boldsymbol{z_x}(T)\big) \right\}$$

s.t.

$$\partial_t \boldsymbol{z_x}(t) = -\nabla_{\boldsymbol{p}} H\big(t, \boldsymbol{z_x}(t), \nabla\Phi(t, \boldsymbol{z_x}(t))\big) = -\nabla\Phi(t, \boldsymbol{z_x}(t))$$

For Corridor



Example:

$$\mu = \begin{bmatrix} -2 \\ -2 \\ 2 \\ -2 \end{bmatrix}, \quad \boldsymbol{\Sigma} = \boldsymbol{I}$$

Penalizers

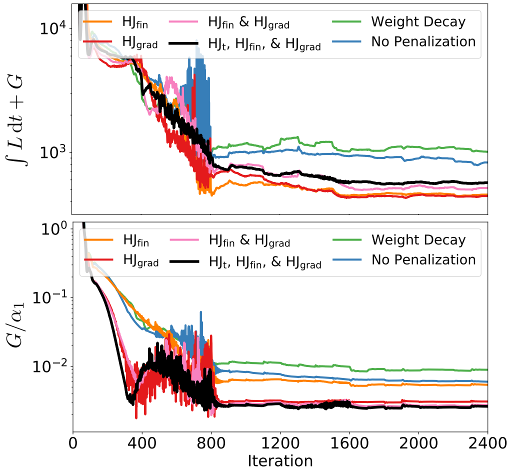
Recall the HJB equations

$$-\partial_t \Phi\big(t, \boldsymbol{z_x}(t)\big) = -H\big(t, \boldsymbol{z_x}(t), \nabla\Phi(t, \boldsymbol{z_x}(t))\big),$$
$$\Phi\big(T, \boldsymbol{z_x}(T)\big) = G\big(\boldsymbol{z_x}(T)\big)$$

Make penalizers

$$c_{\mathrm{HJt},\boldsymbol{x}}(t) =$$
$$\int_0^t \Big| \partial_s \Phi(s, \boldsymbol{z_x}(s)) - H\big(s, \boldsymbol{z_x}(s), \nabla\Phi(s, \boldsymbol{z_x}(s))\big) \Big| \, \mathrm{d}s$$
$$c_{\mathrm{HJfin},\boldsymbol{x}} = \Big| \Phi(T, \boldsymbol{z_x}(T)) \; - \; G(\boldsymbol{z_x}(T)) \Big|$$
$$c_{\mathrm{HJgrad},\boldsymbol{x}} = \Big| \nabla\Phi(T, \boldsymbol{z_x}(T)) \; - \; \nabla G(\boldsymbol{z_x}(T)) \Big|$$

Empirically Effective in Training



HJt penalizer ⇒ few time steps[4,5]

[4] Yang and Karniadakis. "Potential Flow Generator with L_2 Optimal Transport . . . ". 2020.
[5] Onken et al. "OT-Flow: Fast and Accurate Continuous Normalizing Flows via Optimal Transport". 2020.

Formulation

Rewrite time-integrals as part of the ODE

$$\min_{\Phi} \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathcal{N}(\mu,\boldsymbol{\Sigma})} c_{\mathrm{L},\boldsymbol{x}}(T) + G(\boldsymbol{z}_{\boldsymbol{x}}(T)) + \beta_1 c_{\mathrm{HJt},\boldsymbol{x}}(T) + \beta_2\, c_{\mathrm{HJfin},\boldsymbol{x}} + \beta_3\, c_{\mathrm{HJgrad},\boldsymbol{x}}, \tag{6}$$

subject to

$$\partial_t \begin{pmatrix} \boldsymbol{z}_{\boldsymbol{x}}(t) \\ c_{\mathrm{L},\boldsymbol{x}}(t) \\ c_{\mathrm{HJt},\boldsymbol{x}}(t) \end{pmatrix} = \begin{pmatrix} -\nabla_{\boldsymbol{p}} H\big(t, \boldsymbol{z}_{\boldsymbol{x}}(t), \nabla\Phi(t, \boldsymbol{z}_{\boldsymbol{x}}(t))\big) \\ L_{\boldsymbol{x}}(t) \\ \Big| \partial_t \Phi(t, \boldsymbol{z}_{\boldsymbol{x}}(t)) - H\big(t, \boldsymbol{z}_{\boldsymbol{x}}(t), \nabla\Phi(t, \boldsymbol{z}_{\boldsymbol{x}}(t))\big) \Big| \end{pmatrix}, \quad \begin{pmatrix} \boldsymbol{z}_{\boldsymbol{x}}(0) \\ c_{\mathrm{L},\boldsymbol{x}}(0) \\ c_{\mathrm{HJt},\boldsymbol{x}}(0) \end{pmatrix} = \begin{pmatrix} \boldsymbol{x} \\ 0 \\ 0 \end{pmatrix}.$$

where, by the envelope formula,

$$L_{\boldsymbol{x}}(t) = \nabla\Phi(t, \boldsymbol{z}_{\boldsymbol{x}}(t)) \cdot \nabla_{\boldsymbol{p}} H\big(t, \boldsymbol{z}_{\boldsymbol{x}}(t), \nabla\Phi(t, \boldsymbol{z}_{\boldsymbol{x}}(t))\big) - H\big(t, \boldsymbol{z}_{\boldsymbol{x}}(t), \nabla\Phi(t, \boldsymbol{z}_{\boldsymbol{x}}(t))\big)$$

Scalars $\beta_1, \beta_2, \beta_3$ are weighted multipliers (NN hyperparameters)

How do we solve this PDE-constrained optimization problem?

How do we solve this PDE-constrained optimization problem?

Blend Neural Networks and Differential Equations

Choose your buzzword: Neural ODEs, Physics-Informed Neural Networks, etc.

Neural Network (NN) Basics

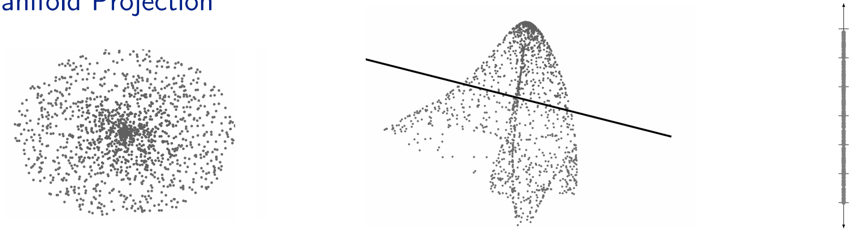
Consider a parameterized function:

$$C = g(\boldsymbol{z}; \boldsymbol{\theta})$$

where

- $\boldsymbol{z} \in \mathbb{R}^d$ is an input item (e.g., the state of the system)
- $C \in \mathbb{R}$ is the corresponding output (e.g., the value from Φ)
- $\boldsymbol{\theta} \in \mathbb{R}^p$ are the parameters/weights of the model g

Think: Manifold Projection



Input Features — Transformed (Hidden) Features — Output

Single-Layer Example

d - # features

m - width

Features

$\boldsymbol{z} \in \mathbb{R}^d$

Weights ($\boldsymbol{\theta}$)

$\boldsymbol{K} \in \mathbb{R}^{m \times d}$

$\boldsymbol{w} \in \mathbb{R}^m$

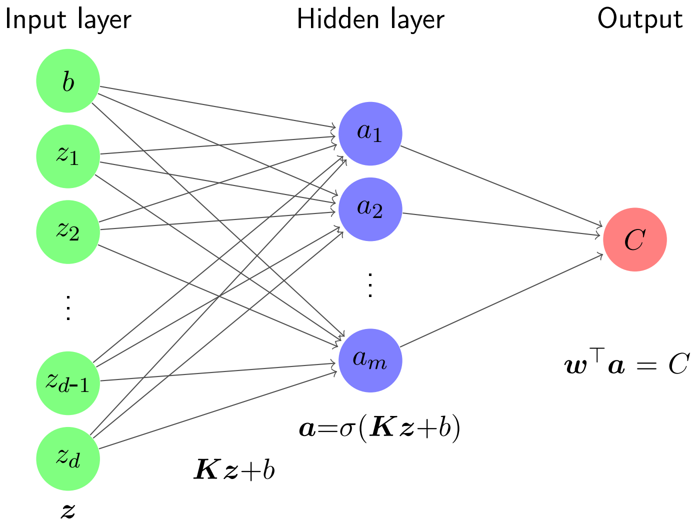
bias $b \in \mathbb{R}$

Outputs

$C \in \mathbb{R}$

Nonlinearity σ

$\tanh$, sigmoid, etc.



Our Network

A Brief Look Under the Hood

We parameterize the value function

$$\boldsymbol{a}_0 = \sigma(\boldsymbol{K}_0 \boldsymbol{s} + \boldsymbol{b}_0),$$

- space-time inputs $\boldsymbol{s} = (\boldsymbol{x}, t) \in \mathbb{R}^{d+1}$

[6] He et al. "Deep Residual Learning for Image Recognition". 2016.

Our Network

A Brief Look Under the Hood

We parameterize the value function

where $N(\boldsymbol{s}) = \boldsymbol{a}_0 + \sigma(\boldsymbol{K}_1 \boldsymbol{a}_0 + \boldsymbol{b}_1)$,

$$\boldsymbol{a}_0 = \sigma(\boldsymbol{K}_0 \boldsymbol{s} + \boldsymbol{b}_0),$$

and

- space-time inputs $\boldsymbol{s}=(\boldsymbol{x}, t) \in \mathbb{R}^{d+1}$
- $N(\boldsymbol{s})\colon \mathbb{R}^{d+1} \to \mathbb{R}^m$ is a residual neural network (ResNet)[6]
- element-wise activation function $\sigma(\boldsymbol{x}) = \log(\exp(\boldsymbol{x}) + \exp(-\boldsymbol{x}))$

[6] He et al. "Deep Residual Learning for Image Recognition". 2016.

Our Network

A Brief Look Under the Hood

We parameterize the value function with

$$\Phi(\boldsymbol{s};\boldsymbol{\theta}) = \boldsymbol{w}^\top N(\boldsymbol{s}) + \frac{1}{2}\boldsymbol{s}^\top(\boldsymbol{A}^\top\boldsymbol{A})\boldsymbol{s} + \boldsymbol{b}^\top\boldsymbol{s} + c, \qquad \text{for} \quad \boldsymbol{\theta} = (\boldsymbol{w}, \boldsymbol{A}, \boldsymbol{b}, c, \boldsymbol{K}_0, \boldsymbol{K}_1, \boldsymbol{b}_0, \boldsymbol{b}_1)$$

where $N(\boldsymbol{s}) = \boldsymbol{a}_0 + \sigma(\boldsymbol{K}_1\boldsymbol{a}_0 + \boldsymbol{b}_1)$,

$$\boldsymbol{a}_0 = \sigma(\boldsymbol{K}_0\boldsymbol{s} + \boldsymbol{b}_0),$$

and

- space-time inputs $\boldsymbol{s}{=}(\boldsymbol{x}, t) \in \mathbb{R}^{d+1}$
- $N(\boldsymbol{s})\colon \mathbb{R}^{d+1} \to \mathbb{R}^m$ is a residual neural network (ResNet)[6]
- element-wise activation function $\sigma(\boldsymbol{x}) = \log(\exp(\boldsymbol{x}) + \exp(-\boldsymbol{x}))$
- $\boldsymbol{\theta}$ contains the trainable weights: $\boldsymbol{w} \in \mathbb{R}^m$, $\boldsymbol{A} \in \mathbb{R}^{10\times(d+1)}$, $\boldsymbol{b} \in \mathbb{R}^{d+1}$, $c \in \mathbb{R}$, $\boldsymbol{K}_0 \in \mathbb{R}^{m\times(d+1)}$, $\boldsymbol{K}_1 \in \mathbb{R}^{m\times m}$, and $\boldsymbol{b}_0, \boldsymbol{b}_1 \in \mathbb{R}^m$.

[6] He et al. "Deep Residual Learning for Image Recognition". 2016.

Differential Equations

Recall: We are solving

$$\min_{\Phi} \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathcal{N}(\mu,\boldsymbol{\Sigma})} c_{\mathrm{L},\boldsymbol{x}}(T) + G(\boldsymbol{z}_{\boldsymbol{x}}(T)) + \beta_1 c_{\mathrm{HJt},\boldsymbol{x}}(T) + \beta_2\, c_{\mathrm{HJfin},\boldsymbol{x}} + \beta_3\, c_{\mathrm{HJgrad},\boldsymbol{x}},$$

subject to

$$\partial_t \begin{pmatrix} \boldsymbol{z}_{\boldsymbol{x}}(t) \\ c_{\mathrm{L},\boldsymbol{x}}(t) \\ c_{\mathrm{HJt},\boldsymbol{x}}(t) \end{pmatrix} = \begin{pmatrix} -\nabla_{\boldsymbol{p}} H\big(t, \boldsymbol{z}_{\boldsymbol{x}}(t), \nabla\Phi(t, \boldsymbol{z}_{\boldsymbol{x}}(t))\big) \\ L_{\boldsymbol{x}}(t) \\ \Big| \partial_t \Phi(t, \boldsymbol{z}_{\boldsymbol{x}}(t)) - H\big(t, \boldsymbol{z}_{\boldsymbol{x}}(t), \nabla\Phi(t, \boldsymbol{z}_{\boldsymbol{x}}(t))\big) \Big| \end{pmatrix}, \quad \begin{pmatrix} \boldsymbol{z}_{\boldsymbol{x}}(0) \\ c_{\mathrm{L},\boldsymbol{x}}(0) \\ c_{\mathrm{HJt},\boldsymbol{x}}(0) \end{pmatrix}, = \begin{pmatrix} \boldsymbol{x} \\ 0 \\ 0 \end{pmatrix}.$$

Differential Equations

Which is the same as training the neural ODE

$$\min_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathcal{N}(\mu,\boldsymbol{\Sigma})} c_{\mathrm{L},\boldsymbol{x}}(T) + G(\boldsymbol{z}_{\boldsymbol{x}}(T)) + \beta_1 c_{\mathrm{HJt},\boldsymbol{x}}(T) + \beta_2\, c_{\mathrm{HJfin},\boldsymbol{x}} + \beta_3\, c_{\mathrm{HJgrad},\boldsymbol{x}},$$

subject to

$$\partial_t \begin{pmatrix} \boldsymbol{z}_{\boldsymbol{x}}(t) \\ c_{\mathrm{L},\boldsymbol{x}}(t) \\ c_{\mathrm{HJt},\boldsymbol{x}}(t) \end{pmatrix} = F\big(t,\, \boldsymbol{z}_{\boldsymbol{x}}(t),\, \nabla\Phi(t, \boldsymbol{z}_{\boldsymbol{x}}(t)\,;\boldsymbol{\theta})\big), \quad \begin{pmatrix} \boldsymbol{z}_{\boldsymbol{x}}(0) \\ c_{\mathrm{L},\boldsymbol{x}}(0) \\ c_{\mathrm{HJt},\boldsymbol{x}}(0) \end{pmatrix}, = \begin{pmatrix} \boldsymbol{x} \\ 0 \\ 0 \end{pmatrix}.$$

Training and Numerics

Solving the Minimiziation / Training the Neural ODE:

Iterate through

1. Solve the ODE
2. Compute the loss function
3. Backpropagate
4. Update parameters $\boldsymbol{\theta}$

Training and Numerics

Solving the Minimiziation / Training the Neural ODE:

Iterate through

1. Solve the ODE
2. Compute the loss function
3. Backpropagate
4. Update parameters $\boldsymbol{\theta}$

ODE solver:

Runge-Kutta 4 $\Rightarrow$ efficient and accurate

Discretize-then-Optimize Approach:[7,8]

First, discretize the ODE at time points, then optimize over that discretization

As opposed to optimize-then-discretize, e.g., solve Karush-Kuhn-Tucker then discretize

[7] Gholaminejad, Keutzer, and Biros. "ANODE: Unconditionally Accurate Memory-Efficient . . . ". 2019.

[8] Onken and Ruthotto. "Discretize-Optimize vs. Optimize-Discretize for Time-Series . . . ". 2020.

Training and Numerics

Solving the Minimiziation / Training the Neural ODE:

Iterate through

1. Solve the ODE
2. Compute the loss function
3. Backpropagate
4. Update parameters $\boldsymbol{\theta}$

Loss / Objective Function:

$$J(\boldsymbol{\theta}) = \mathop{\mathbb{E}}_{\boldsymbol{x} \sim \mathcal{N}(\mu, \boldsymbol{\Sigma})} c_{\mathrm{L},\boldsymbol{x}}(T) + G(\boldsymbol{z}_{\boldsymbol{x}}(T)) + \beta_1 c_{\mathrm{HJt},\boldsymbol{x}}(T) + \beta_2\, c_{\mathrm{HJfin},\boldsymbol{x}} + \beta_3\, c_{\mathrm{HJgrad},\boldsymbol{x}}$$

Training and Numerics

Solving the Minimiziation / Training the Neural ODE:

Iterate through

1. Solve the ODE
2. Compute the loss function
3. Backpropagate
4. Update parameters $\boldsymbol{\theta}$

Compute gradient with respect to parameters (chain rule)

Use automatic differentiation[9] to compute $\nabla_{\boldsymbol{\theta}} J$

[9] Nocedal and Wright. *Numerical Optimization*. 2006.

Training and Numerics

Solving the Minimiziation / Training the Neural ODE:

Iterate through

1. Solve the ODE
2. Compute the loss function
3. Backpropagate
4. Update parameters $\boldsymbol{\theta}$

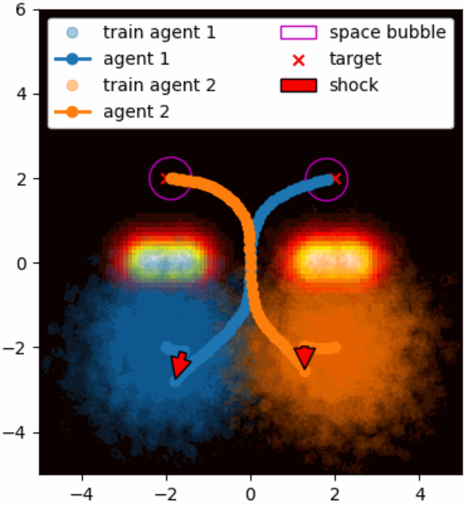
Use ADAM[10]

A stochastic subgradient method with momentum

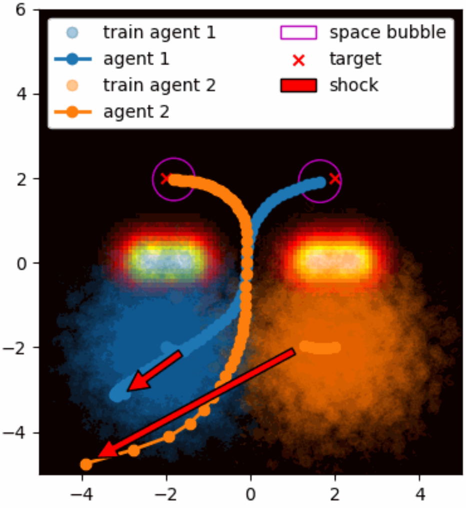
Empirically, ADAM works well in noisy high-dimensional spaces

[10] Kingma and Ba. "Adam: A Method for Stochastic Optimization". 2015.

Results



Small Shock



Large Shock

Baseline

Corridor

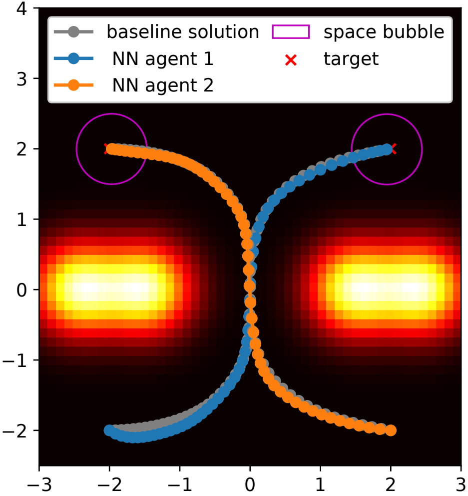
Running Cost: $L(t,\cdot) = E(\cdot) + \alpha_2 Q(\cdot) + \alpha_3 W(\cdot)$
Terminal Cost: $G(\boldsymbol{z}) = \frac{\alpha_1}{2}\|\boldsymbol{z}-\boldsymbol{y}\|^2$

Discrete optimization approach via forward Euler

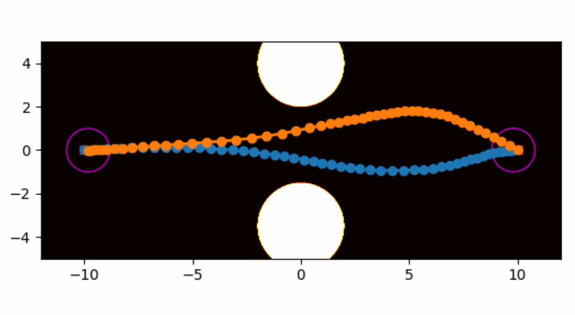
$$\min_{\{\boldsymbol{u}^{(k)}\}} \quad G\left(\boldsymbol{z}^{(n_t)}\right) + h\sum_{k=0}^{n_t-1} L\left(t^{(k)}, \boldsymbol{z}^{(k)}, \boldsymbol{u}^{(k)}\right)$$
$$\text{s.t.} \quad \boldsymbol{z}^{(k+1)} = \boldsymbol{z}^{(k)} + h\, f\left(t^{(k)}, \boldsymbol{z}^{(k)}, \boldsymbol{u}^{(k)}\right),$$
$$\boldsymbol{z}^{(0)} = \boldsymbol{x}$$

where $h{=}T/n_t$. We use $T{=}1$ and $n_t{=}50$.

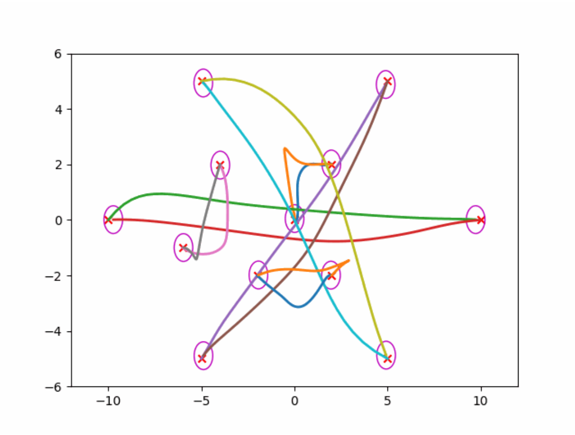
This is a *local* approach, whereas the NN is *global*



Swap Experiments



Two agents swap positions with hard corridor[11]



Twelve agents swap positions[11]

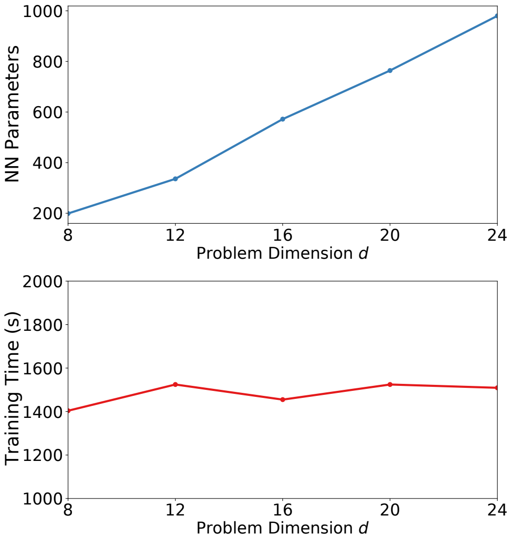
[11] Mylvaganam, Sassano, and Astolfi. "A Differential Game Approach to Multi-Agent Collision Avoidance". 2017.

Addressing Curse of Dimensionality[12]

Setup:

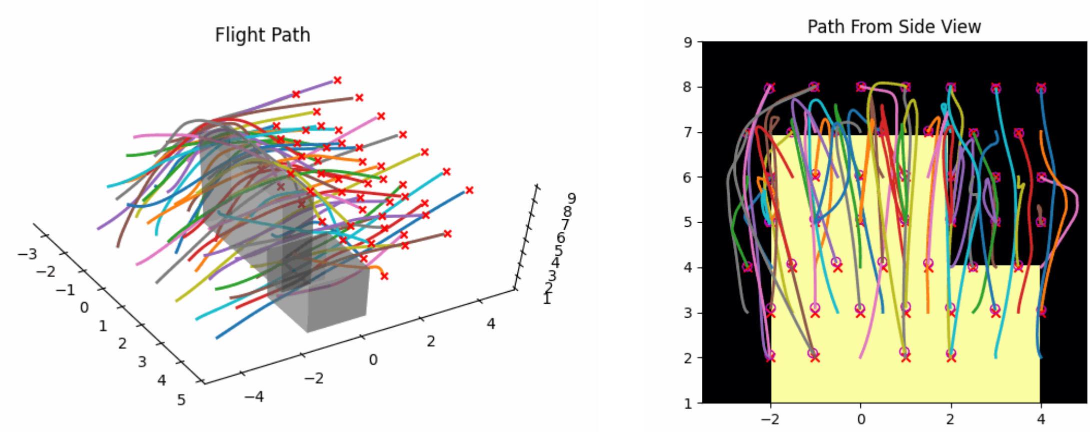
- Take subproblems of the 12-agent swap experiment (2, 3, 4, 5, and 6 pairs of agents)
- Train the smallest NN we can that achieves a fixed suboptimality (relative to baseline)

The number of parameters grows linearly with problem dimension d



[12] Bellman. *Dynamic Programming*. 1957.

Swarm Trajectory Planning



50 3-dimensional agents with obstacles[13]

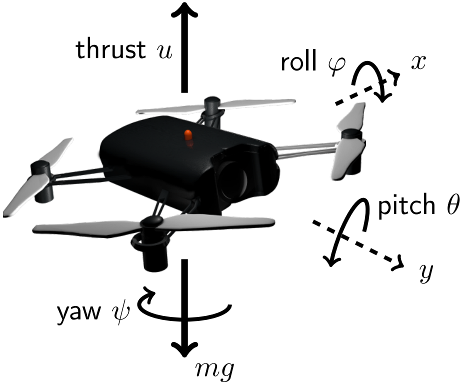
[13] Hönig et al. "Trajectory Planning for Quadrotor Swarms". 2018.

Quadcopter Problem

More complicated dynamics[14]

Controls: thrust u, torques $\tau_\psi, \tau_\theta, \tau_\varphi$

$$\dot{\boldsymbol{z}} = f(\boldsymbol{x}, \boldsymbol{u}) \implies \begin{cases} \dot{x} = v_x \\ \dot{y} = v_y \\ \dot{z} = v_z \\ \dot{\psi} = v_\psi \\ \dot{\theta} = v_\theta \\ \dot{\varphi} = v_\varphi \\ \dot{v}_x = \frac{u}{m} f_7(\psi, \theta, \varphi) \\ \dot{v}_y = \frac{u}{m} f_8(\psi, \theta, \varphi) \\ \dot{v}_z = \frac{u}{m} f_9(\theta, \varphi) - g \\ \dot{v}_\psi = \tau_\psi \\ \dot{v}_\theta = \tau_\theta \\ \dot{v}_\varphi = \tau_\varphi \end{cases}$$

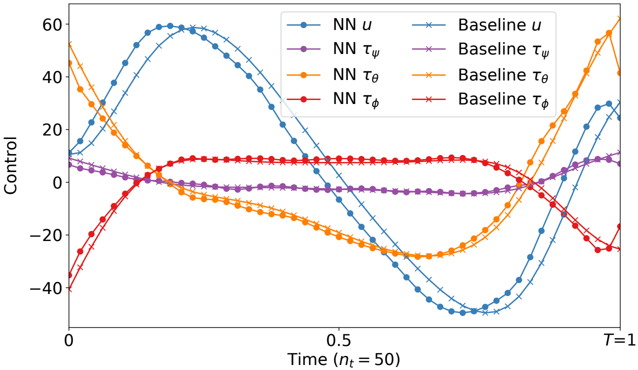


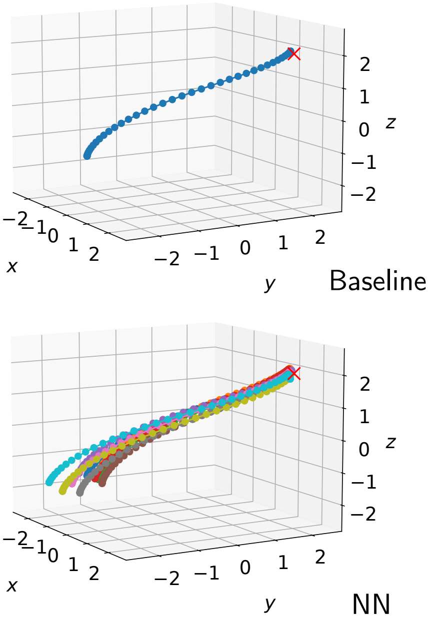
where

$$\begin{cases} f_7(\psi, \theta, \varphi) & = \sin(\psi)\sin(\varphi) + \cos(\psi)\sin(\theta)\cos(\varphi), \\ f_8(\psi, \theta, \varphi) & = -\cos(\psi)\sin(\varphi) + \sin(\psi)\sin(\theta)\cos(\varphi), \\ f_9(\theta, \varphi) & = \cos(\theta)\cos(\varphi). \end{cases}$$

[14] Carrillo et al. "Modeling the Quad-Rotor Mini-Rotorcraft". 2013.

Quadcopter Comparison with Baseline





Review

- **Want to solve**
  - High-Dimensional Control Problems
  - Semi-Globally
- **Combine Pontryagin Maximum Principle and Hamilton-Jacobi-Bellman approaches**
- **Parameterize the value function Φ with a neural network**
- **Solve trajectory problem in 150 dimensions**
- **Solve quadcopter problem with complicated dynamics**
- **Demonstrate shock-robustness**

Conclusions

- **Parameterizing Φ**
  $\Rightarrow$ extrapolation capabilities
- **HJB penalizers improve training**
- **Lagrangian coordinates (no grids) help scalability**

DO, L Nurbekyan, X Li, S Wu Fung, S Osher, L Ruthotto
A Neural Network Approach Applied to Multi-Agent Optimal Control
2021 European Control Conference
arXiv:2011.04757, 2020

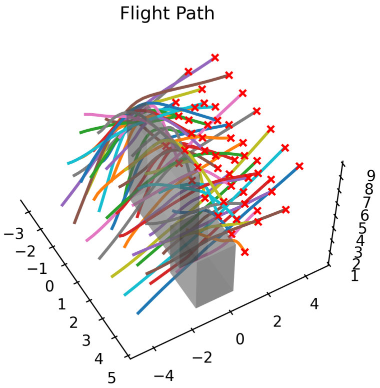
Coming Soon:

DO, L Nurbekyan, X Li, S Wu Fung, S Osher, L Ruthotto
A Neural Network Approach for High-Dimensional Optimal Control

Code: `github.com/donken/NeuralOC`
Simulations: `imgur.com/a/eWr6sUb`

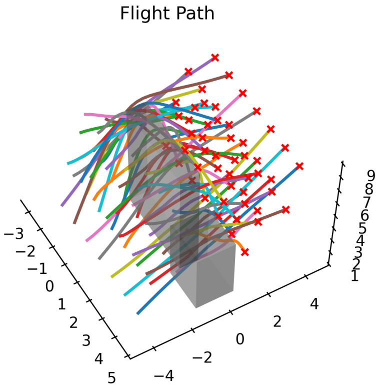
Future Work

- **More rigorous experiments with many 12-d quadcopters**
- **Deployment on actual quadcopters**
- **Combination with existing methods and sensors**



Future Work

- More rigorous experiments with many 12-d quadcopters
- Deployment on actual quadcopters
- Combination with existing methods and sensors



Questions?

References I

Bellman, Richard (1957). *Dynamic Programming*. Princeton University Press, Princeton, N. J., pp. xxv+342.

Carrillo, Luis Rodolfo García et al. (2013). "Modeling the Quad-Rotor Mini-Rotorcraft". In: *Quad Rotorcraft Control*. Springer, pp. 23–34.

Fleming, Wendell H. and H. Mete Soner (2006). *Controlled Markov Processes and Viscosity Solutions*. Second. Vol. 25. Stochastic Modelling and Applied Probability. Springer, New York, pp. xviii+429. ISBN: 978-0387-260457; 0-387-26045-5.

Gholaminejad, Amir, Kurt Keutzer, and George Biros (2019). "ANODE: Unconditionally Accurate Memory-Efficient Gradients for Neural ODEs". In: *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 730–736.

He, Kaiming et al. (2016). "Deep Residual Learning for Image Recognition". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778.

Hönig, Wolfgang et al. (2018). "Trajectory Planning for Quadrotor Swarms". In: *IEEE Transactions on Robotics* 34.4, pp. 856–869.

References II

Kingma, Diederik P. and Jimmy Ba (2015). "Adam: A Method for Stochastic Optimization". In: *International Conference on Learning Representations (ICLR)*.

Mylvaganam, Thulasi, Mario Sassano, and Alessandro Astolfi (2017). "A Differential Game Approach to Multi-Agent Collision Avoidance". In: *IEEE Transactions on Automatic Control* 62.8, pp. 4229–4235.

Nocedal, Jorge and Stephen Wright (2006). *Numerical Optimization*. Springer Science & Business Media.

Onken, Derek and Lars Ruthotto (2020). "Discretize-Optimize vs. Optimize-Discretize for Time-Series Regression and Continuous Normalizing Flows". In: *arXiv:2005.13420*.

Onken, Derek et al. (2020). "OT-Flow: Fast and Accurate Continuous Normalizing Flows via Optimal Transport". In: *AAAI*.

Pontryagin, L. S. et al. (1962). *The Mathematical Theory of Optimal Processes*. Translated by K. N. Trirogoff; edited by L. W. Neustadt. Interscience Publishers John Wiley & Sons, Inc. New York-London, pp. viii+360.

References III

Yang, Liu and George Em Karniadakis (2020). "Potential Flow Generator with L_2 Optimal Transport Regularity for Generative Models". In: *IEEE Transactions on Neural Networks and Learning Systems*.