Neural-Fly Reproduction

Zhiyang Zhou

Original paper:

Neural-Fly enables rapid learning for agile flight in strong winds https://arxiv.org/abs/2205.06908

Scripts

Filename	Description
`training-and-validation.ipynb`	Domain Adversarially Invariant Meta Learning (DAIML) algorithm, the offline learning process for Neural-Fly. This script trains a wind-invariant representation of the aerodynamic effects on a quadrotor. After training the model, some simple statistics and plots are generated which show the model performance fitting to the training and testing data.
`run_in_airsim_adaptive.py`	Running in airsim using adaptive control
`run_in_airsim_phi.py`	Running in airsim using Phi network

Get started

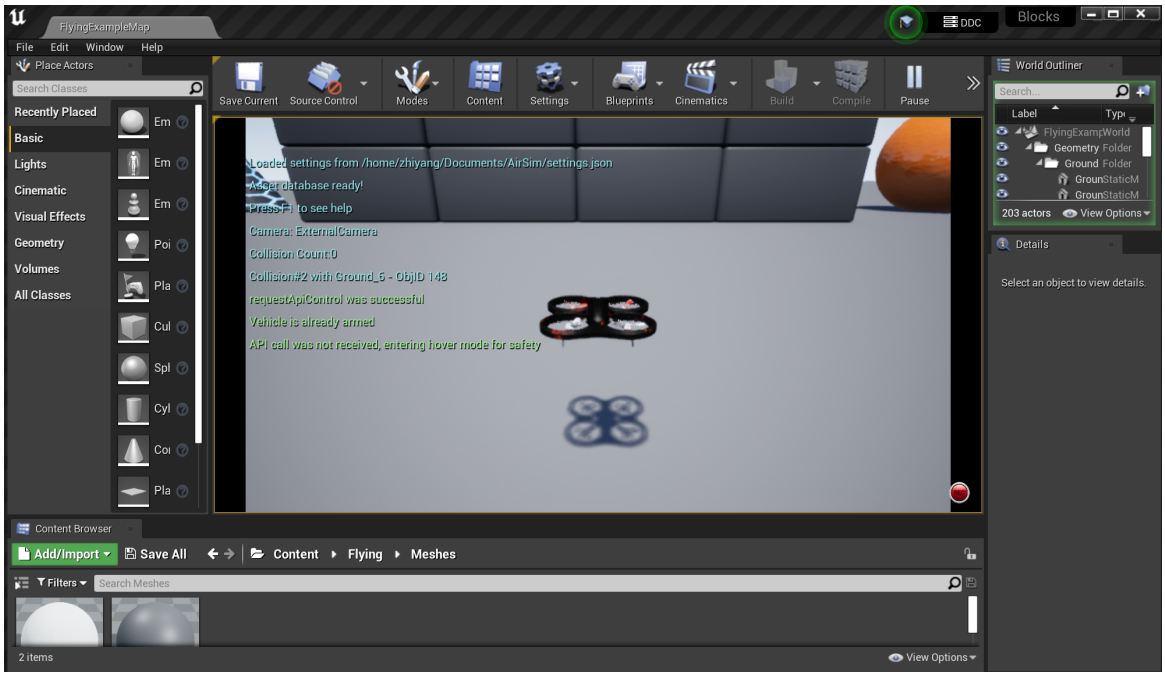
Reference: https://microsoft.github.io/AirSim/build_linux/

1. Register with Epic Games and link it with Github.
2. Check out 4.27 and download zip (git clone takes too muck time)
3. Follow this guide if you use ubuntu 22.04
   https://blog.csdn.net/weixin_58660639/article/details/147927822
4. After starting Unreal, choose blocks environment -> advanced options -> convert inplace
5. Put settings.json under ~/Documents/Airsim

```
{
  "SettingsVersion": 1.2,
  "SimMode": "Multirotor",
  "Vehicles": {
    "Drone1": {
      "VehicleType": "SimpleFlight",
      "AutoCreate": true
    }
  }
}
```

6. Click play

7. Run python code



Paper details

1.Input and labels (page 8)

Inputs (velocity, attitude quaternion, and motor speed PWM command)

- `vx, vy, vz`
- `quaternion`, not sure wxyz or xyzw
- rotor speed get from `getRotorStates() / 1000`

Label (offline calculation of aerodynamic residual force)

- `residual force` = actual force - nominal force

2.Phi Network

structure(page 23)

The output channel has 4 dimension(3 + 1bias), and is multiplied by linear coefficients (ax,ay,az which is also R4) to get 3 channels.

In `training-and-validation.ipynb` [block3] the output dimension is 3 which is wrong.

loss function(page 18)

zero-sum max-min game

To solve the domain shift problem, inspired by [51], we propose the following adversarial optimization framework:

$$\max_{h} \min_{\phi, a_1, \cdots, a_K} \sum_{k=1}^{K} \sum_{i=1}^{N_k} \left(\left\| y_k^{(i)} - \phi(x_k^{(i)}) a_k \right\|^2 - \alpha \cdot \text{loss} \left(h(\phi(x_k^{(i)})), k \right) \right), \tag{5}$$

where h is another DNN that works as a discriminator to predict the environment index out of K wind conditions, $\text{loss}(\cdot)$ is a classification loss function (e.g., the cross entropy loss), $\alpha \geq 0$ is a hyperparameter to control the degree of regularization, k is the wind condition index, and (i) is the input-output pair index. Intuitively, h and ϕ play a zero-sum max-min game: the goal of h is to predict the index k directly from $\phi(x)$ (achieved by the outer $\max$); the goal of ϕ is to approximate the label $y_k^{(i)}$ while making the job of h harder (achieved by the inner $\min$). In other words, h is a learned regularizer to remove the environment information contained in ϕ. In our experiments, the output of h is a $K-$dimensional vector for the classification probabilities of K conditions, and we use the cross entropy loss for $\text{loss}(\cdot)$, which is given as

$$\text{loss} \left(h(\phi(x_k^{(i)})), k \right) = -\sum_{j=1}^{K} \delta_{kj} \log \left(h(\phi(x_k^{(i)}))^\top e_j \right) \tag{6}$$

where $\delta_{kj} = 1$ if $k = j$ and $\delta_{kj} = 0$ otherwise and e_j is the standard basis function.

training(page 19)

'(i) The adaptation step (Line 4-6) solves an least squares problem as a function of φ on the adaptation set Ba. (ii) The training step (Line 7) updates the learned representation φ on the training set B, based on the optimal linear coefficient a∗ solved from the adaptation step. (iii) The regularization step (Line 8-9) updates the discriminator h on the training set.'

3.Dynamics (page 16)

q is a 3 dimension vector

We consider a general robot dynamics model:

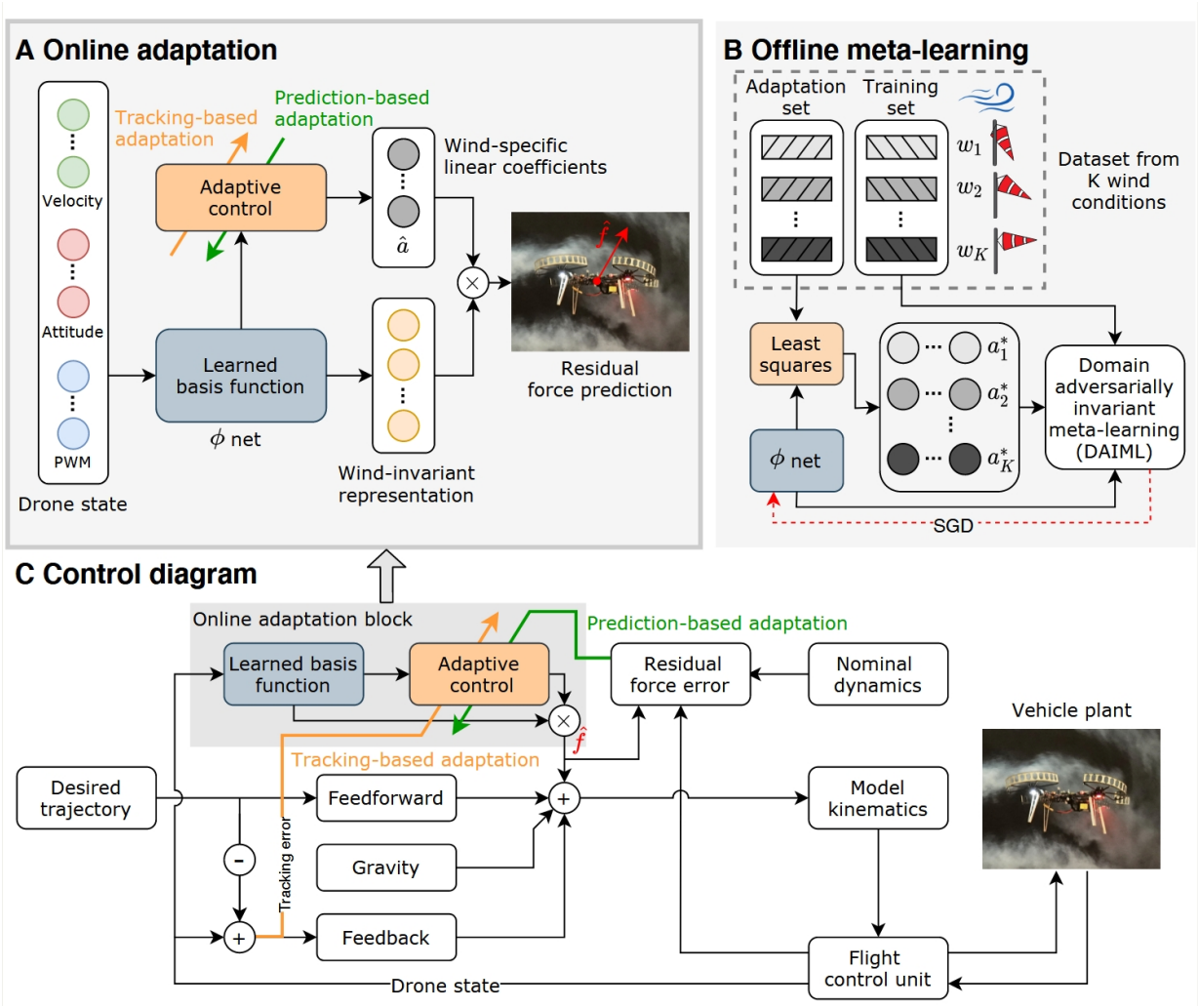
$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + g(q) = u + f(q, \dot{q}, w) \tag{1}$$

where $q, \dot{q}, \ddot{q} \in \mathbb{R}^n$ are the n dimensional position, velocity, and acceleration vectors, $M(q)$ is the symmetric, positive definite inertia matrix, $C(q, \dot{q})$ is the Coriolis matrix, $g(q)$ is the gravitational force vector and $u \in \mathbb{R}^n$ is the control force. Most importantly, $f(q, \dot{q}, w)$ incorporates unmodeled dynamics, and $w \in \mathbb{R}^m$ is an unknown hidden state used to represent the underlying environmental conditions, which is potentially time-variant. Specifically, in this article, w represents the wind profile (for example, the wind profile in Fig. 1), and different wind profiles yield different unmodeled aerodynamic disturbances for the UAV.

Neural-Fly can be broken into two main stages, the offline meta-learning stage and the online adaptive control stage. These two stages build a model of the unknown dynamics of the form

$$f(q, \dot{q}, w) \approx \phi(q, \dot{q}) a(w), \tag{2}$$

4.Control Diagram (page 4)



controller design(page 20)

Our online adaptive control algorithm can be summarized by the following control law, adaptation law, and covariance update equations, respectively.

$$u_{\mathrm{NF}} = \underbrace{M(q)\ddot{q}_r + C(q,\dot{q})\dot{q}_r + g(q)}_{\text{nominal model feedforward terms}} \underbrace{-Ks}_{\text{PD feedback}} \underbrace{-\phi(q,\dot{q})\hat{a}}_{\text{learning-based feedforward}} \tag{7}$$

$$\dot{\hat{a}} = \underbrace{-\lambda\hat{a}}_{\text{regularization term}} \underbrace{-P\phi^\top R^{-1}(\phi\hat{a} - y)}_{\text{prediction error term}} \underbrace{+P\phi^\top s}_{\text{tracking error term}} \tag{8}$$

$$\dot{P} = -2\lambda P + Q - P\phi^\top R^{-1}\phi P \tag{9}$$

where u_{NF} is the control law, $\dot{\hat{a}}$ is the online linear-parameter update, P is a covariance-like matrix used for automatic gain tuning, $s = \dot{\tilde{q}} + \Lambda\tilde{q}$ is the composite tracking error, y is the measured aerodynamic residual force with measurement noise ϵ, and K, Λ, R, Q, and λ are gains. The structure of this control

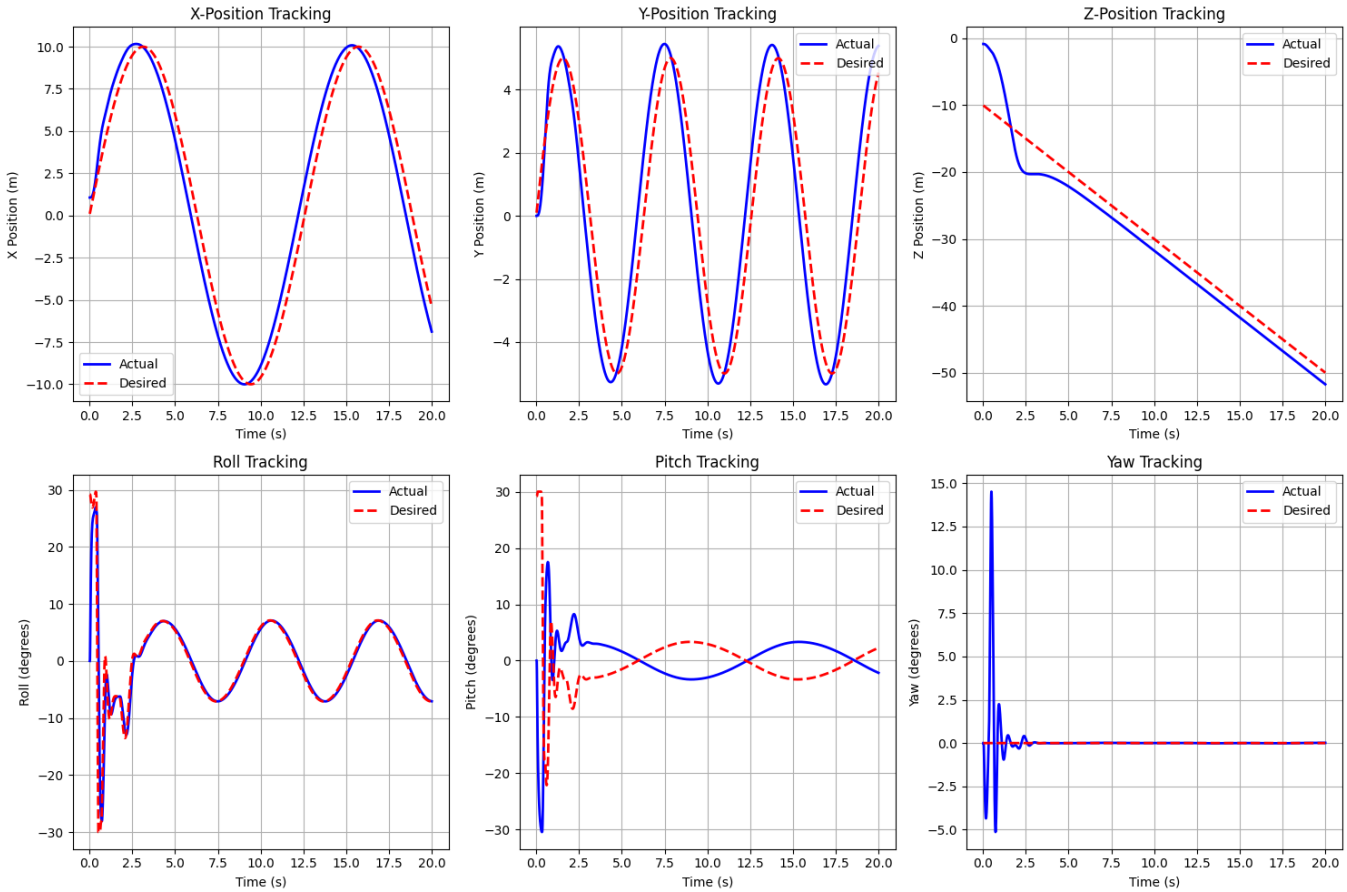
5.Implementation (page 9)

'We implemented our control algorithm and the baseline control methods in the position control loop in Python, and run it on the onboard Linux computer at 50 Hz. The PX4 was set to the offboard flight mode and received thrust and attitude commands from the position control loop.'

Results

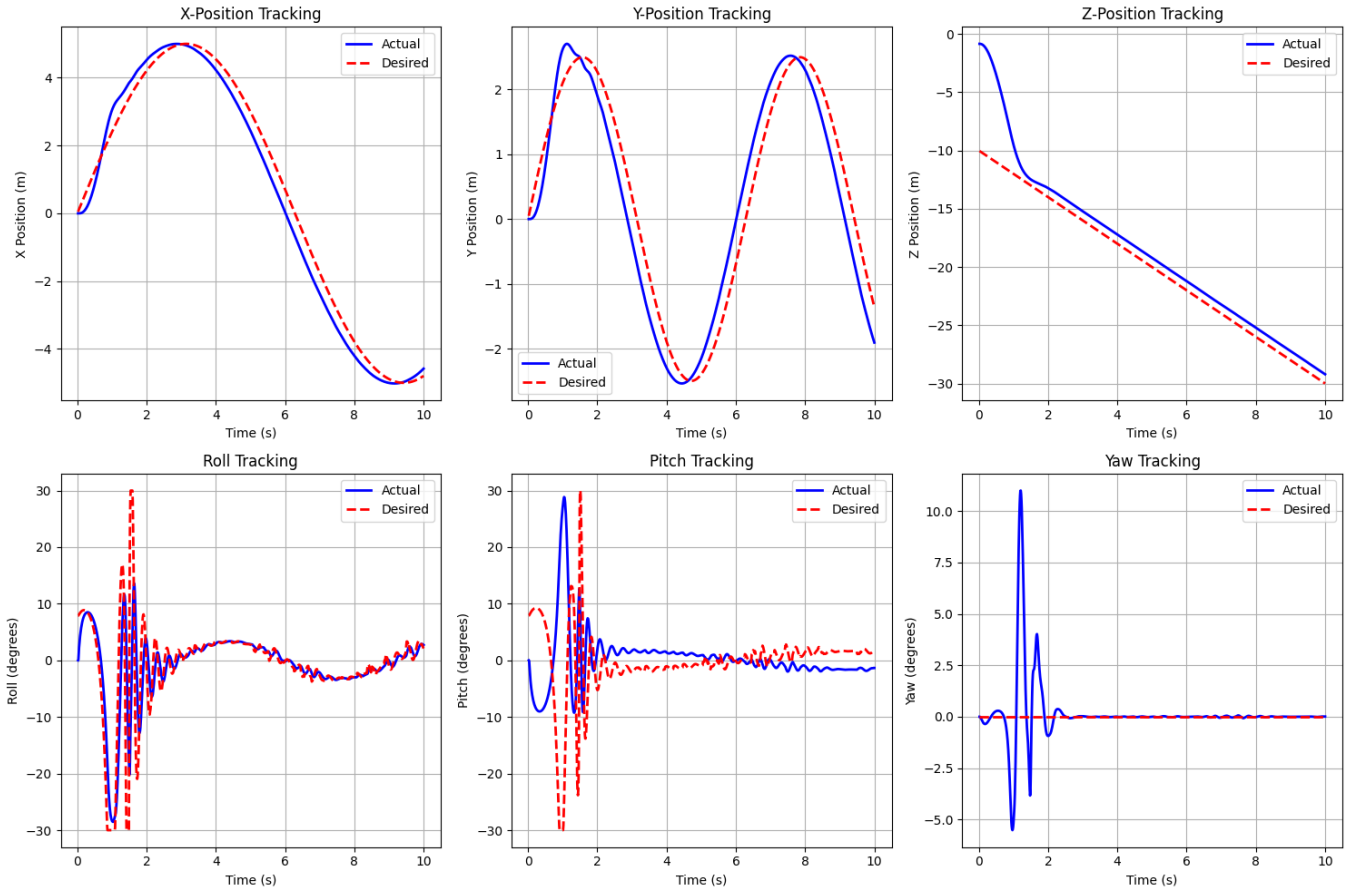
Added baseline -- traditional adaptive control

Actual position go ahead of desired, which is strange.



Neural-fly

still debugging



Todo

- check the calculation of desired roll & pitch
- check the order of quaternions
- add wind.....

Code details

line56: define neural_fly_controller

- line73: calculate velocity and acceleration with one-order euler method
- line82: control parameters
- line90: calculate tracking error s
- line119: use phi network
- line144-165: controller update
- line179: calculate desired thrust and angle

line225: class AirSimNeuralFlyController

- line358: define run_simulation

line515: main loop

line539: desire trajectory

Dataset filenaming scheme

Filenames are structured as

```
<VEHICLE>_<TRAJECTORY>_<METHOD>_<CONDITION>.csv
```

For specific details please refer to the article.

Field	Description
`<VEHICLE>`	• `custom`: drone built using consumer off-the-shelf components with PX4 flight controller • `intel`: Intel-Aero RTF Drone, used for data collection for Neural-Fly-Transfer controller
`<TRAJECTORY>`	• `random3`: randomized trajectory created by randomly sampling 2 waypoints and generating a smooth spline from the current position through both waypoints; continuous derivatives through snap • `random2`: similar to `random3` except only one random waypoint is generated • `figure8`: a lemniscate trajectory given by `(x(t),y(t),z(t)) = (1.25 * sin(t), 0, 0.75 * sin(2 * t)`
`<METHOD>`	• `'baseline'`: nonlinear baseline control • `'indi'`: incremental nonlinear dynamics inversion control • `'L1'`: L1 adaptive control • `'NF-C'`: Neural-Fly-Constant, our adaptive controller without any learning • `'NF-T'`: Neural-Fly-Transfer, our learning based adaptive control with the ML model trained on data from the Intel-Aero drone • `'NF'`: Neural-Fly, our learning based adaptive control with the ML model trained on data collected with the custom drone ,
`<CONDITION>`	wind condition for experiments, where the number corresponds to the fan array duty cycle and converts to • `nowind` = 0 m/s • `10wind` = 1.3 m/s • `20wind` = 2.5m/s • `30wind` = 3.7 m/s • `35wind` = 4.2 m/s • `40wind` = 4.9 m/s • `50wind` = 6.1 m/s • `70wind` = 8.5 m/s • `70p20sint` = 8.5+2.4sin(t) m/s • `100wind` = 12.1 m/s

Experiment data

Additionally, the data from the experiment results present in the paper is provided. To load the data, run the following in python

```
import utils
Data = utils.load_data(folder='data/experiment')
```

This will load all of the experiment data as a list of dictionaries. The `i`th experiment, field `field`, at the `j`th timestep, can be accessed with `Data[i][field][j]`. Most fields are ndarrays except the metadata fields, pulled from the filename. Available fields are given in the following table.

`field`	**description**
`'t'`	time in seconds
`'p'`	position vector in meters
`'p_d'`	desired position vector in meters
`'v'`	velocity vector in m/s
`'v_d'`	desired velocity in m/s
`'q'`	attitude represented as a unit quaternion
`'R'`	rotation matrix (body frame to world frame)
`'w'`	(this is a misnomer - should be ω) angular velocity in rad/s
`'T_sp'`	thrust setpoint sent to the flight controller
`'q_sp'`	attitude command sent to the flight controller
`'hover_throttle'`	throttle at hover computed as a best fit function of the battery voltage
`'fa'`	aerodynamic residual force computed using numerical differentiation of `v` and `T_sp`, `q`, and `hover_throttle`
`'pwm'`	motor speed commands from the flight controller
`'vehicle'`	`<VEHICLE>` field from filename
`'trajectory'`	`<TRAJECTORY>` field from filename
`'method'`	`<METHOD>` field from filename
`'condition'`	`<CONDITION>` field from filename