POLITECNICO DI TORINO

Master's Degree in Mechatronic's Engineering

Master's Degree Thesis

Physics Guided Neural Networks for robust control of drones

Supervisors

Prof. Alessandro RIZZO

Candidate

Giuseppe SODERO

October 2022

Summary

Abstract

Throughout recent History Model-Based controllers dominated the scene of flight controller design and their effectiveness has been proven several times. Despite that, their reliance on the accuracy of the mathematical model used in order to represent real plant might lead to an explosion of the complexity of the problem in case of strongly non-linear systems such as UAVs. In the following work, we are going to propose an alternative approach for the design of flight controllers based on the use of ANN and their capabilities of being universal approximators in order to overcome some of the flaws of standard Model-Based controllers. We are going to investigate the effectiveness of a mixed approach of both Data and Information Driven techniques using Physics-Guided Neural Networks for the approximation of the real plant dynamics. Moreover, we are going to implement the Dynamics inversion of the plant using such techniques in order to design a flight controller based on feedback linearization. For the simulation of the plant and the data collection needed in order to train and validate the ML models we made use of Microsoft Arisim flight simulator while the ANN Development has been made through the Pytorch Framework. The Flight controller connection with the Quadrotor in the Airsim Environment uses the PX4 firmware and Mavlink communication protocol. The model we implemented is able to invert the Dynamics of the drone starting from a set of measurements and targets at given time instance(such as RPY angles, Velocities and accelerations in the body frame) and computes with good accuracy the set of forces applied on each vertex on the quad-rotor. Different Architectures has been tested, mainly LSTM and FeedForward, using both the standard Data-Driven procedure and Data-Information mixed procedure, and in each case the latter always improved the RMSE loss with respect to the counter-part of about 10%-20% with faster convergence.

Acknowledgements

A mia nonna Immacolata, so che sarai fiera di me, anche se non potrai esserci per gioire insieme.

Alla mia famiglia, che mi ha sempre supportato e incoraggiato, anche nei momenti più difficili.

Thank you Weibin, for the unbelievable support you gave me through this travel.

Table of Contents

List of Tables

List of Figures

Acronyms

AI
artificial intelligence

ANN
Artificial Neural Network

FFNN
FeedForward Neural Network

LSTM
Long-short term memory

ML
Machine Learning

MSE
Mean-square error

PGNN
Physics-Guided Neural Netwok

PINN
Physics-Infrofmed Neural Network

ReLu
Rectified Linear Unit

RF
Reference Frame

RMSE
Root Mean-Square error

Chapter 1

State of Art

Before the beginning of the analysis, a brief overview is given about the state of art relative to ANN implemented in UAV and Aircraft engineering literature, as discussed in [1]. The effectiveness of ANNs regarding the approximation of unknown model parameters, adaptive compensation of external disturbances and inversion error are documented extensively in the available literature.
In most of these cases ANNs are implemented alongside other controllers designed with traditional techniques to complement the control action, despite that, there are also instances of ANNs trained to mimic the control action of standard model-based controllers, and achieved acceptable results.
The aim of this work is also to investigate the effectiveness of data-infromation driven hybrid models for the implementation of an high level position control for UAVs.
A general overview about some practical implementations documented in literature for this class of hybrid models is given.

1.1 Feedback Linearization

In [2] 1997, ANN are used in order to implement adaptive control augmentation. Three feedforward networks consisting in 21 RBF units and 40 sigma-pi units complemented the control action of a fixed wing aircraft, resulting in the cancellation of the model inversion error without the full knowledge of the uncertanties related to high-g, fixed throttle manouvers. The algorithm used in order to ensure the convergence of the model's weights has been derived using Lyapunov theory, in order to ensure the boundedness of all the signals in the loop. Similarly, in [3] 2002 a two layer feed forward network with 5 hidden neurons has been used in order to perform adaptive control augmentation, although the performances where satisfactory the behaviour of the system has not been investigated under uncertainties and disturbances. Further improvements have been achieved in [4]2011 tackling a similar problem for adaptive augmentation for helicopters to reduce the model inversion error, but improving the convergence and tracking performances through Concurrent-learning. In [5] 2015 a two-layer feedforward network with 50 hidden neurons has been used in a similar fashion for adaptive control augmentation with remarkable results.

1.2 Optimal feedback control

In 2012 [6] a two layer FFNN has been trained offline in order to mimic the outputs produced by a controller implementing the optimal feedback control low. Since the model's training phase has been done in an offline fashion, it is highly dependent on the operating conditions in relative to the training set. Repeated training is needed if such conditions are considerably different.

1.3 Sliding mode control

In [7] 2015, the RBFNN architecture has been used in order to develop an uncertainty observer of an 8-rotor coaxial UAV, used for the reduction of external disturbances and inertia matrix uncertainties. This observer has been used alongside a Backstepping sliding mode controller. Another observer has been developed in [8]2016, used for the position estimation and attitude estimation, since the quadrotor controlled through a double-loop Integral Sliding Mode controller, was subjected to disturbances and also compensate parametric uncertainties.

1.4 Backstepping control

In [9] 2008 MLPs where used for the flight control of an helicopter, using these models for the estimation of unknown physical parameters of the model. Also in this case network weights were updated in an online fashion, using lyapunov stability theory to ensure convergence.

1.5 PGNN usage examples

In [10] 2018, PGNNs are used for the approximation PDE solution of the Scrhodiger equations, using a set of points generated by the integration of the following equations in order to observe measurements relative to the initial conditions across multiple points in space $\{h_i, x_i\}$, along with randomly generated points across the time periodic boundaries $\{t_i, x_i\}$. The goal is to approximate the complex valued solution, $h_{(t,x)}$ given the non-linear Schrodinger equations with periodic boundary conditions:

$$
\begin{aligned}
i\frac{\partial h}{\partial t} + 0.5\frac{\partial^2 h}{\partial^2 x} + |h|^2 h = 0 \\
h_{0,x} = 2sech(x) \\
h_{(t,-5)} = h_{(t,5)} \\
\frac{\partial h}{\partial x}_{(t,5)} = \frac{\partial h}{\partial x}_{(t,-5)}
\end{aligned}
$$

The complex valued neural network is able to approximate h, providing two outputs equal to the real and imaginary part of the solution, given x and t. This model is obtained by the minimization of the following loss function:

$$MSE = MSE_0 + MSE_b + MSE_f \tag{1.1}$$

Where MSE_0 is the mean square error with respect to the initial condition points, MSE_b is the mean square error relative to the points evaluated in the periodic boundaries and MSE_f is a penalty term related to the Schrodinger equation not being satisfied across randmly generated collocation points.

Another interesting use of PGNNs is the data-driven discovery PDEs, e.g. in the same paper such networks are used for the approximation of a latent function $\psi_{(t,x,y)}$, such that $u = \frac{\partial \psi}{\partial y}$, $v = -\frac{\partial \psi}{\partial x}$ and the pressure $p_{t,x,y}$ relative to the 2-dimensional Navier-Stokes equations, with u and v being the x and y components of the fluid velocity field.

$$\frac{\partial u}{\partial t} + \lambda_1(u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y}) = -\frac{\partial p}{\partial x} + \lambda_2(\frac{\partial^2 u}{\partial^2 x} + \frac{\partial^2 u}{\partial^2 y})$$
$$\frac{\partial v}{\partial t} + \lambda_1(u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y}) = -\frac{\partial p}{\partial x} + \lambda_2(\frac{\partial^2 v}{\partial^2 x} + \frac{\partial^2 v}{\partial^2 y})$$

Given Noisy measurements $\{t_i, x_i, y_i, u_i, v_i\}_{i=1}^N$, the network is optimized minimizing the MSE relative to the partial derivatives of the the output ψ and the expected values given t_i, x_i, y_i, moreover, a additional penalties are added to the overall loss function, depending on the consistency of the predicted outputs with the Nevier-Stokes equations discussed previously.

In both instances these networks showed remarkable results and the effectiveness in both the improvement of performances and avoiding overfitting led are object of further research.

Chapter 2

Quadrotor Kinematics and Dynamics

This chapter describes the coordinates systems and reference frames used in order to represent the quadrotor's[11] kynematic and dynamic Quantities used for the analisys made in this thesis.

- The Newton's equations of motion are given in the non inertial frame positioned in the quadrotor's center of mass.
- The aerodynamic forces and torques are expressed in the inertial frame.

Section 2.1 refers to the coordinates systems, the reference frames and transformations related to the drone position and attitude in space. In section 2.2 a description of the quadrotor kynematics is given using RFs used in section 2.1. The analysis of the quadrotor's Dynamics is made in section 2.3.

2.1 Quadrotor coordinates Systems

Given a point in space, its position can be represented with respect to different reference frames. In figure 2.1, e.g. given $\mathcal{R}_1 = \{\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}, O_1\}$, with $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w} \in \mathbb{R}^3$ unitary vectors and $\mathcal{R}_2 = \{\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}, O_2\}$, with $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in \mathbb{R}^3$ unitary vectors, the position vector of point $\boldsymbol{P}_i$ is different depending on which reference frame it is represented in. A reference frame can be represented with respect to another, through the homogeneous transformation that transform the first frame in the second one.
This transformation is a rototranslation, which can be expressed using an analytic form i.e. six-dimensional vector using the centers relative position and Euler's angles, or a geometric form, i.e. a 4x4 rototranslation matrix. Considering the

frames, e.g. $\mathcal{R}_1$ and $\mathcal{R}_2$, if the second one is represented with respect the first, the resulting vector is $P_{12} := [\boldsymbol{t}_{\mathbf{12}}^T, \boldsymbol{\Phi}_{12}^T]^T \in \mathbb{R}^6$, where $\boldsymbol{t}_{\mathbf{12}} := [t_u, t_v, t_w,]^T$ is the distance $\overline{\mathrm{O_1O_2}}$ and $\boldsymbol{\Phi}_{12} := [\alpha, \beta, \gamma]^T$ is the set of Euler's angles for the attitude of the second frame.
As shown in figure 2.1, the position ofP_i can be represented in both $\mathcal{R}_1$ and $\mathcal{R}_2$ with two different set of coordinates, $\boldsymbol{p}_i^1 := [u_i, v_i, w_i]^T$ in frame $\mathcal{R}_1$ and $\boldsymbol{p}_i^2 := [x_i, y_i, z_i]^T$ in frame $\mathcal{R}_2$.
Given these two reference frames, through the transformation defined previously i.e. T_1^2, it is possible to transform the homogeneous vector $\boldsymbol{P}_i^1 := [u_i, v_i, w_i, 1]^T = [\boldsymbol{p}_i^{1T}, 1]^T$ into the homogeneous vector $\boldsymbol{P}_i^2 := [x_i, y_i, z_i, 1]^T = [\boldsymbol{p}_i^{2T}, 1]^T$ according to the equation:

$$\boldsymbol{P}_i^1 = \boldsymbol{T}_1^2 \boldsymbol{P}_i^2 \tag{2.1}$$

where $\boldsymbol{T}_1^2 \in \mathbb{R}^{4\times4}$ is the homogeneous Roto-Translation matrix defined as:

$\boldsymbol{T}_1^2 = \begin{bmatrix} \boldsymbol{R}_{1(\boldsymbol{\Phi})}^2 & \boldsymbol{t}_{\mathbf{12}} \\ \mathbf{0}^T & 1 \end{bmatrix}$ with $\boldsymbol{R}_{1(\Phi)}^2 \in \mathbb{R}^{3\times3}$ as the rotation matrix obtained from the set of Euler's angles Φ_{12} and $\mathbf{0} := [0, 0, 0]^T \in \mathbb{R}^3$.

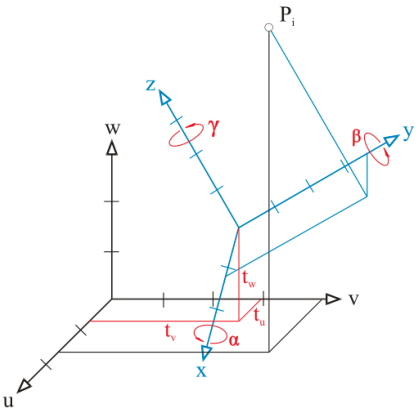

Figure 2.1: Representation of RFs $\mathcal{R}_1$ and $\mathcal{R}_2$ in space

For the purposes of this work, all the Kynematic and Dynamic quantities analyzed in the following chapters are going to be represented it two RFs:

- The envoironment Reference Frame $\mathcal{R}_e$, which is a fixed reference frame in space used in order to represent drone quantities as seen by an external observer in a determined position. This frame is a right-handed frame oriented according to NED convention (north-east-down), where the x axis of this frame is oriented toward the north direction, the y axis towards the east direction and the z axis complements the set.

- The body Reference Frame $\mathcal{R}_b$, which is a mobile non-inertial frame attached to the drone's center of mass. The attitude of this frame is represented in $\mathcal{R}_e$ using ϕ(roll), θ(pitch), and ψ(yaw) angles. The roll angle expresses a counterclockwise rotation along the x axis, the pitch angle a counterclockwise rotation along the y axis and the yaw angle expresses a counterclockwise rotation along the z axis, such that:

$$R^b_{e(\phi,\theta,\psi)} = R_{x(\phi)} R_{y(\theta)} R_{z(\psi)} \tag{2.2}$$

2.2 Quadrotor Kynematics

Using the RFs that has been introduced in section 2.1 the following set of state variables can be chosen in order to represent the Kynematic state of the quadrotor:

- x_e the quadrotor's x position in $\mathcal{R}_e$
- y_e the quadrotor's y position in $\mathcal{R}_e$
- z_e the quadrotor's z position in $\mathcal{R}_e$
- u the quadrotor's velocity along the x axis in $\mathcal{R}_b$
- v the quadrotor's velocity along the y axis in $\mathcal{R}_b$
- w the quadrotor's velocity along the z axis in $\mathcal{R}_b$
- ϕ is roll angle defined in section 2.1
- θ is pitch angle defined in section 2.1
- ψ is yaw angle defined in section 2.1
- p is the roll rate represented in frame$\mathcal{R}_b$
- q is the pitch rate represented in frame$\mathcal{R}_b$
- r is the yaw rate represented in frame$\mathcal{R}_b$

Given this set of state variables we can put in relation the position state variables expressed in the inertial frame $\mathcal{R}_e$, $\boldsymbol{p}_e := [x_e, y_e, z_e]^T$ and the velocity state variables expressed in the body frame $\mathcal{R}_b$, $\boldsymbol{v}_b = [u, v, w]^T$ according to equation 2.3:

$$\frac{d\boldsymbol{p}_e}{dt} = R^e_b v_b = R^{b^T}_e \boldsymbol{v}_b = (R_{x(\phi)} R_{y(\theta)} R_{z(\psi)})^T v_b \tag{2.3}$$

Considering also the angular quantities, the rpy angles $\Phi := [\phi, \theta, \psi]^T$ are quantities expressed in $\mathcal{R}_e$ meanwhile, angular rates $\boldsymbol{\omega}_b := [p, q, r]^T$ are expressed in $\mathcal{R}_b$.

$$\boldsymbol{\omega}_b = R_{x(\dot{\phi})}[\dot{\phi},0,0]^T + R_{x(\phi)}R_{y(\dot{\theta})}[0,\dot{\theta},0]^T + R_{x(\phi)}R_{y(\theta)}R_{z(\dot{\psi})}[0,0,\dot{\psi}]^T \tag{2.4}$$

considering that $R_{x(\dot{\phi})} \approx R_{y(\dot{\theta})} \approx R_{z(\dot{\psi})} \approx I_{3\times3}$:

$$\boldsymbol{\omega}_b \approx [\dot{\phi},0,0]^T + R_{x(\phi)}[0,\dot{\theta},0]^T + R_{x(\phi)}R_{y(\theta)}[0,0,\dot{\psi}]^T \tag{2.5}$$

2.3 Quadrotor Dynamics

Given The Newton's second law, the quadrotor's acceleration $\boldsymbol{a}_{e(t)}$ in $\mathcal{R}_e$ can be computed according to:

$$M\boldsymbol{a}_e = \frac{Md\boldsymbol{v}_e}{dt} = \boldsymbol{F}_e \tag{2.6}$$

where M is the total mass of the drone and $\boldsymbol{F}_e = \sum_{i=0}^{N} \boldsymbol{f}_{ie}$, with $\boldsymbol{f}_{ie}$ as an external force represented in $\mathcal{R}_e$ applied to the drone and N is total number of external forces applied to the drone. Since $\boldsymbol{v}_b$ is a state variable defined in section 2.2, using the Coriolis equation and equation 2.6 a relation can be obtained:

$$\boldsymbol{a}_b = \frac{d\boldsymbol{v}_b}{dt} = \frac{1}{M}\boldsymbol{F}_b + [rv - qw, pw - ru, qu - pv]^T \tag{2.7}$$

Considering rotational motion, the general form of Euler's equation can be used:

$$\boldsymbol{M} = I\dot{\boldsymbol{\omega}}_b + \boldsymbol{\omega_b} \times \boldsymbol{L} = I\dot{\boldsymbol{\omega}}_b + \boldsymbol{\omega}_b \times (I\boldsymbol{\omega}_b) \tag{2.8}$$

With **M** as the total external torque applied to the body, **L** is the quadrotor's total angular momentum and I is the central Inertia matrix of the quadrotor which, considering its symmetry and the masses condensed in the center and the vertexes:

$$I = \begin{bmatrix} \frac{2M_0r^2}{5} + 2ml^2 & 0 & 0 \\ 0 & \frac{2M_0r^2}{5} + 2ml^2 & 0 \\ 0 & 0 & \frac{2M_0r^2}{5} + 2ml^2 \end{bmatrix} \tag{2.9}$$

and so,

$$I^{-1} = \begin{bmatrix} \frac{1}{\frac{2M_0r^2}{5}+2ml^2} & 0 & 0 \\ 0 & \frac{1}{\frac{2M_0r^2}{5}+2ml^2} & 0 \\ 0 & 0 & \frac{1}{\frac{2M_0r^2}{5}+2ml^2} \end{bmatrix} \tag{2.10}$$

with M_1 as the mass in the center, R as the centerbox size and l as the arm length. Equation 2.8 can rearrange in order to get $\dot{\boldsymbol{\omega}}_b$ as follows:

$$\dot{\boldsymbol{\omega}}_b = \begin{pmatrix} \frac{1}{J_x} & 0 & 0 \\ 0 & \frac{1}{J_y} & 0 \\ 0 & 0 & \frac{1}{J_z} \end{pmatrix} \left[\begin{pmatrix} 0 & r & -q \\ -r & 0 & p \\ q & -p & 0 \end{pmatrix} \begin{pmatrix} J_x & 0 & 0 \\ 0 & J_y & 0 \\ 0 & 0 & J_z \end{pmatrix} \boldsymbol{\omega}_b + \boldsymbol{\tau} \right] \tag{2.11}$$

with, $J_x = J_y = J_z = \frac{2M_0r^2}{5} + 2ml^2$

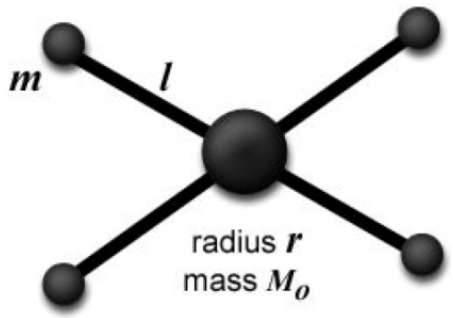

Figure 2.2: Quadrotor mass distribution model

2.4 Forces and Torques

Following the discussions made in section 2.4 the quadrotor dynamics is now analyzed. Since there are no aerodynamic lifting sources for sake of simplcity, the forces and torques considered for this model are the ones due to gravity and propellers.

As represented in figure 2.3 the total force, or thrust, generated by the propellers is $\boldsymbol{F_p} = \boldsymbol{F_1} + \boldsymbol{F_2} + \boldsymbol{F_3} + \boldsymbol{F_4}$, which are always negative and parallel to the $\boldsymbol{z}_B$ axis. Propellers aside, the other external force applied on the drone is the gravity force $\boldsymbol{F_g} = m\boldsymbol{g}$, always negative and parallel to $\boldsymbol{z}_e$ axis.

$$\boldsymbol{F_b} = \boldsymbol{F_p} + R_b^e \boldsymbol{F_g} \tag{2.12}$$

Considering the torques acting on the Drone, as expressed in figure 2.3:

$$L = F_1 l - F_2 l - F_3 l + F_4 l \tag{2.13}$$

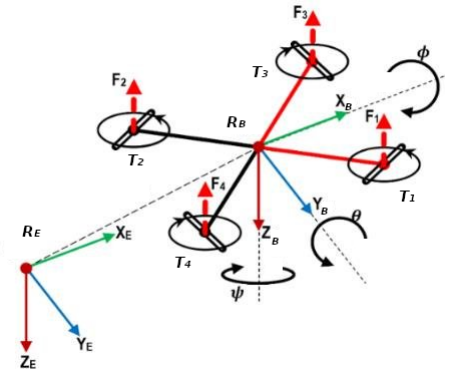

Figure 2.3: Forces and Torques acting on quadrotor

$$M = -F_1 l + F_2 l - F_3 l + F_4 l \tag{2.14}$$

$$N = -T_1 - T_2 + T_3 + T_4 \tag{2.15}$$

$$\boldsymbol{\tau} = (L, M, N)^T \tag{2.16}$$

Considering equations 2.7 and 2.11 in section 2.3:

$$\boldsymbol{a}_b = \frac{d\boldsymbol{v}_b}{dt} = \frac{1}{M}\left(\boldsymbol{F_p} + R_b^e \boldsymbol{F_g}\right) + \begin{pmatrix} rv - qw \\ pw - ru \\ qu - pv \end{pmatrix} \tag{2.17}$$

$$\dot{\boldsymbol{\omega}}_b = \begin{pmatrix} \frac{1}{J_x} & 0 & 0 \\ 0 & \frac{1}{J_y} & 0 \\ 0 & 0 & \frac{1}{J_z} \end{pmatrix} \left[\begin{pmatrix} (Iz - Iy)qr \\ (Ix - Iz)pr \\ (Iy - Ix)pq \end{pmatrix} \boldsymbol{\omega}_b + \begin{pmatrix} L \\ M \\ N \end{pmatrix} \right] \tag{2.18}$$

Chapter 3

Artificial Neural Networks

Artificial Neural Networks[12] have been developed in order to generalize mathematical models of biological nervous systems. This complex models are used as universal approximators and their capabilities for handling complex task has been proved multiple times. This kind of models can be used in order to tackle both classification and regression problems but the computation of the correct parameters needed for the approximation or the estimation of an unknown function cannot be computed in closed form. In order to get a good approximation of such parameters, the initial problem can be converted into an optimization problem relative to a specific objective function, using a large dataset of information relative to the function itself that we are trying to approximate or estimate. Such objective function elaborate the outputs that the model produce given each specific input in a given optimization step. This class of Data-driven models are called Machine-Learning[12] (ML) models, as in each step of the optimization process, the model "learns" how to improve itself in order to accomplish a specific task. Both supervised and unsupervised learning problems can be tackled using this models, but, for the purposes of this work, models optimized through supervised learning are going to be considered.
In this scenario, the expected output the model shall produce given each specific input in the dataset collected, is known. The objective function to be optimized in this case is typically a loss function related to the error between the output of the model and the expected result. Different Loss function can be used, depending on the kind of problem that has been considered, for example the MSE loss[12] or the L1 loss[12] for regression models or cross entropy for classification problems. In section 3.1 the basic elements of the ANN model are analyzed, starting from the perceptron [12] model. In section 3.2 the Feed-forward[12] architecture is summarized, which is the simplest model of ANN. In the following sections some more advanced architectures are discussed, such as the LSTM[13] architecture in section 3.3 and an hybrid model (information and data-driven) such as PINN in

section 3.4

3.1 Perceptron

The perceptron[12] is the basic processing element for ANNs. Given a set of input features $\boldsymbol{D} = \{\boldsymbol{X}_1^n, \boldsymbol{X}_2^n, ..., \boldsymbol{X}_N^n\}$ such that $\boldsymbol{X}_i^n \in \boldsymbol{D} \subset \mathbb{R}^n$ and a set of expected results $\boldsymbol{Y} = \{\boldsymbol{Y}_1^m, \boldsymbol{Y}_2^m, ..., \boldsymbol{Y}_N^m\}$ such that $\boldsymbol{Y}_i^m \in \boldsymbol{B}^m = \{0,1\}^m$, this mathematical model is a non linear function $P(\boldsymbol{X}_i^N) : \mathbb{R}^n \to \boldsymbol{B}^m$ which maps the input tensor an output. This model is practically equivalent to a single layer FFNN, which is going to be analyzed in section 3.2. This models can be used for classification problems although it is able to perform accurately for problems in which the classes are linearly separable.
For the sake of simplicity numeric features are considered in this section, nevertheless , the following analysis can easily be extended to other kind of inputs, such as text based features. The function is composed by three elements:

- A tensor of weights $\boldsymbol{w} \in \mathbb{R}^{m \times n}$ which constains different scale parameters relative to each element in the input tensor $\boldsymbol{X}_i^n$ with respect to each output in the expected result tensor $\boldsymbol{Y}^n$.
- The bias b is an offset parameter.
- An activation function f(): $\mathbb{R}^m \to \mathbb{R}^m$ which is going to provide the perceptron's output. Different activation functions can be chosen(hyperbolic tangent, Rectified linear unit, sigmoid etc...) according to the kind of problem the perceptron is going to be optimized for.

This elements are combined according to following equation:

$$P(\boldsymbol{X_i^n}) = h(f(\boldsymbol{w}\boldsymbol{X_i^n} + \boldsymbol{b})) = \hat{\boldsymbol{Y}}_i^m, \tag{3.1}$$

Were h() is a multidimensional step function centered in the origin.

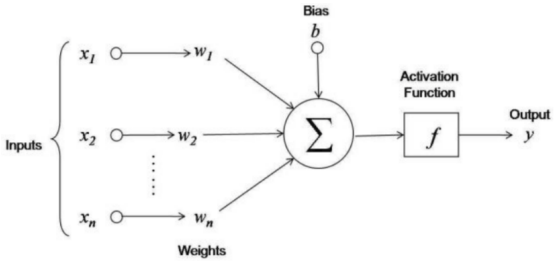

Figure 3.1: perceptron model

Considering this structure, the **w** tensor and **b** are parameters of the optimization problem. In each step the loss function can be evaluated according to the output of the model $\hat{\boldsymbol{Y}}_i^m$ and the expected result $\boldsymbol{Y}_i^n$ given the input tensor X_i^n and the weights can be updated according to the following rule :

$$\delta \boldsymbol{w}_i = \eta(\hat{\boldsymbol{Y}}_i^m - \boldsymbol{Y}_i^n)\boldsymbol{X}_i^{n^T} \tag{3.2}$$

Please note that equation 3.2 implies that the weights are updated only if the expected result and the predicted output don't match. The parameter $\eta \in \mathbb{R}^+$ is an hyper parameter[12] denominated learning rate. This Hyper parameter defines the convergence speed to the optimization problem's final solution and must be accurately tuned in order to regulate the speed of convergence and the error with respect to the optimal solution for the optimization problem.

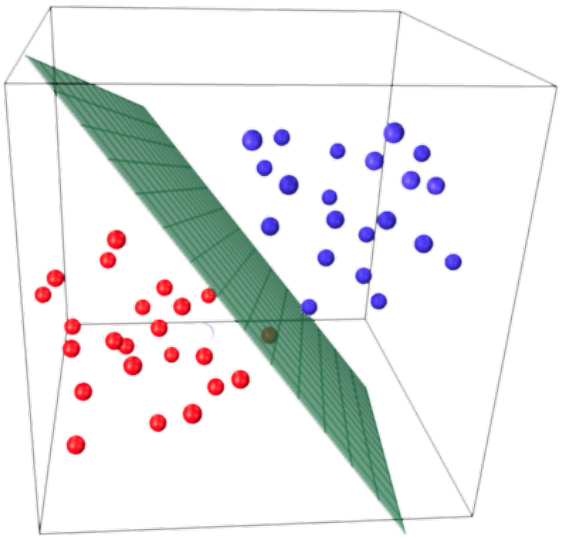

Figure 3.2: Linearly separable classes

3.2 Neural Networks and Feedforward Neural Networks

The fundamental element of NNs is the neuron: It can be considered as a mathematical abstraction of the biological neuron which, in a similar fashion as what has been described in section 3.1, provides a response based on the input feed into it. These artificial Neurons are connected and organized in layers, where every layer collects the outputs from the preceding one and providing its response as input to the next one. The connection between each neuron is represented by

the Matrix of weights related to each output of the previous layer that, added to a specific bias, are feed into each neuron's activation function: As described in section 3.1 the perceptron model can be considered as a single layer neural network. The basic architecture of ANNs consists of three types of layers: input, hidden and output layers. In Feedforward Neural Networks (FFNN) the propagation of each the model's input information is strictly performed in a feedforward fashion, so the output of each layer cannot be propagated towards the preceding layers, in constrast with what happens in RNNs and LSTM described in section 3.3. In a Similar fashion to section 3.1, the model's parameter for each layer $\boldsymbol{l}_i$ are the matrix of weight $\boldsymbol{w}_i$ and the vector of biases $\boldsymbol{b}_i$. The supervised training procedure is performed using an appropriate loss function (for example MSE in regression problems) in order to compute a good approximation of the optimal values for the model's parameters. The loss function $L() : \mathbb{R}^m \to \mathbb{R}$ is evaluated through each step of the optimization algorithm in order to compute $\nabla_{w,b} L(net(\boldsymbol{X}_i^n))$.
Since the overall network function is the combination of all the hidden layers inputs and outputs, such gradient can be computed using the chain rule for multivariate functions:
Given h($\mathbf{x}$) = f(g($\mathbf{x}$)): $\mathbb{R}^{n1} \to \mathbb{R}$, f(y) $\mathbb{R}^{n2} \to \mathbb{R}$, g(x) $\mathbb{R}^{n1} \to \mathbb{R}^{n2}$, $\boldsymbol{x} \in \mathbb{R}^{n1}$

$$\frac{\delta h}{\delta x_i} = \sum_{j=0}^{n_2} \frac{\delta f}{\delta g_j} \frac{g_j}{x_i} \tag{3.3}$$

where $\frac{\delta f}{\delta g_j}$ is the j-th partial derivative of f with respect to g. This rule can be used recursively considering the dependencies between each layer's outputs and inputs in order to compute the loss function's gradient with respect to each layer's parameters. This algorithm computes the gradients starting from the last layer and uses the result in order to compute the loss function's gradient for the layer that precede it, iterating up to the input layer, for this reason it's named backpropagation[12]. Once the loss function's gradient are computed for each layer, all the model's parameter can be updated through an optimization rule, for example:
Given $\boldsymbol{w}^{nk}$ wich is the weights matrix for the k-th layer in the n-th iteration, for each element $\boldsymbol{w}_{ij}^{nk}$

$$\Delta w_{ij}^{n+1k} = -\eta \frac{\delta L}{\delta w_{ij}^{nk}} + \alpha \Delta w_{ij}^{nk} \tag{3.4}$$

with η as learning rate, and α as momentum, a parameter which represent the influence of the increment in the current optimization step of the previous one. This training procedure can be performed also in batch mode, where the update of the parameters is performed with the average of the gradients evaluated with multiple samples. The batch size is another hyper parameter which must be tuned depending on the specific dataset the network is trained with.

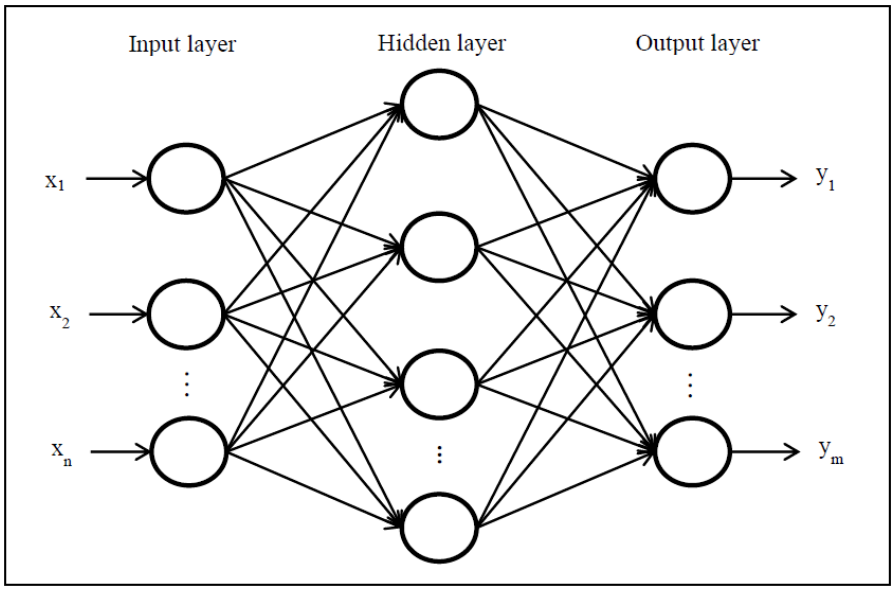

Figure 3.3: Feedforward Neural Network scheme

3.3 Recurrent Neural Networks and Long Short Term Memory Networks

In opposite fashion with FFNNs, the RNN[12] architecture's core element is the capacity to keep memory of the previous outputs the network produced. This design is extremely suited for time series tasks such as forecasting and for simulation of dynamical systems. The RNN cell output is both function of the input and its hidden state, function of the previous output. The parameters in this architecture are:

- The input weight matrix $\boldsymbol{w}_i$
- The output bias vector $\boldsymbol{b}_y$
- The hidden state weight matrix $\boldsymbol{w}_h$
- The output weight matrix $\boldsymbol{w}_y$
- The hidden state bias vector $\boldsymbol{b}_h$
- The hidden state activation function $\sigma_h()$
- The output activation function $\sigma_y()$

The RNN cell produces its output and hidden layer vector according to equation:

$$\boldsymbol{h}_t = \sigma_h(\boldsymbol{w}_i\boldsymbol{x}_t + \boldsymbol{w}_h\boldsymbol{h}_{t-1} + \boldsymbol{b}_i) \tag{3.5}$$

$$\boldsymbol{y}_t = \sigma_y(\boldsymbol{w}_y\boldsymbol{h}_t + \boldsymbol{b}_y) \tag{3.6}$$

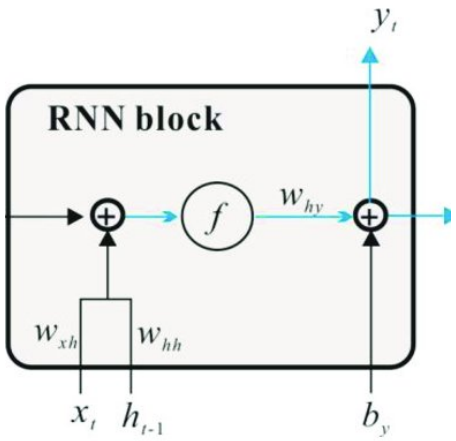

Figure 3.4: RNN cell

Multiple hidden RNN cells can be stacked in order to process sequences of date over a predefined time window, initializing the first hidden state with zeros or random values. This is architecture is effective but not free from some flaws, such as the problem of vanishing gradients during the training phase, especially for very long sequences of data.

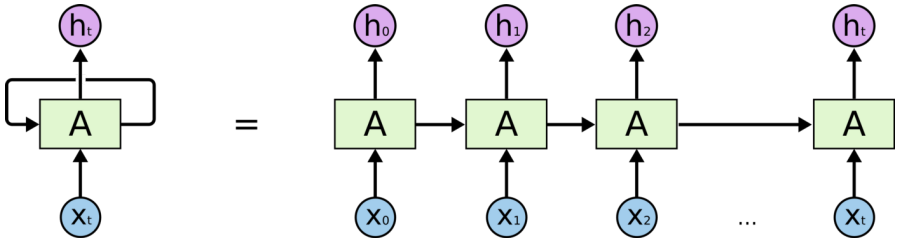

Figure 3.5: Recurrent neural network

Different architectures evolved from the basic RNN and one of the most successful model is the Long Short Term Memory Network[12]. This networks improved the RNN's basic architecture mainly by introducing internal gates that are able to stop the data flows inside the cell. There are three different gates, the input, forget

and output gates. This gates are implemented through sigmoid activation function which are multiplied element-wise with the data flaw. The LSTM's input output and hidden layer can be expressed as:

$$\boldsymbol{f}_t = \sigma_g(\boldsymbol{w}_f \boldsymbol{x}_t + \boldsymbol{U}_f \boldsymbol{h}_{t-1} + \boldsymbol{b}_f) \tag{3.7}$$

$$\boldsymbol{i}_t = \sigma_g(\boldsymbol{w}_i \boldsymbol{x}_t + \boldsymbol{U}_i \boldsymbol{h}_{t-1} + \boldsymbol{b}_i) \tag{3.8}$$

$$\boldsymbol{o}_t = \sigma_g(\boldsymbol{w}_o \boldsymbol{x}_t + \boldsymbol{U}_o \boldsymbol{h}_{t-1} + \boldsymbol{b}_o) \tag{3.9}$$

$$\tilde{\boldsymbol{c}}_t = \sigma_h(\boldsymbol{w}_c \boldsymbol{x}_t + \boldsymbol{U}_c \boldsymbol{h}_{t-1} + \boldsymbol{b}_c) \tag{3.10}$$

$$\boldsymbol{c}_t = \tilde{\boldsymbol{c}}_t \odot \boldsymbol{i}_t + \boldsymbol{c}_{t-1} \odot \boldsymbol{f}_t \tag{3.11}$$

$$\boldsymbol{h}_t = \boldsymbol{c}_t \odot \boldsymbol{o}_t \tag{3.12}$$

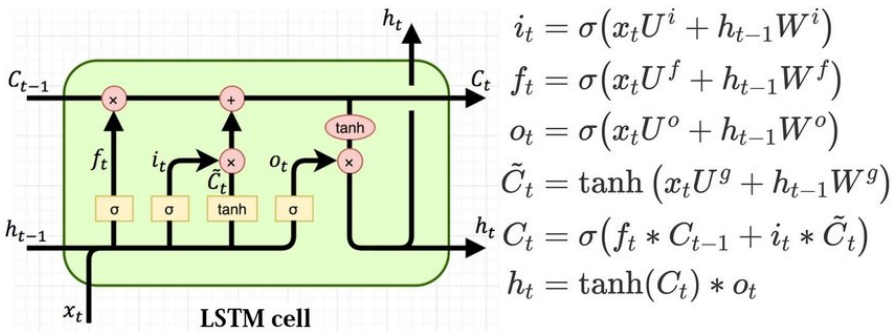

Figure 3.6: Long-Short Term Memory cell

3.4 Physics Informed Neural Networks

All the models described in the previous sections are mainly data driven models, optimized for simulation and prediction purposes, mainly based on the information that can be extrapolated from the dataset used for the model's training phase. This approach is effective, but, the training process could be incredibly Data-Hungry and could also lead to very slow convergence to the model parameter's optimal value. Overcoming this flaws is possible using an Hybrid

approach, which introduces in the model some prior knowledge related to the system's behaviour. This approach is implemented in Physics-Guided Nerual Networks[10], which introduce in the model the constraints related to the physical laws involved in the system which is source of the Dataset used in the training phase. This models are used in different field of physics and the evidence shows that such model improved both from the number of samples required for the convergence of the model and also in the accuracy of the resulting model. The key for the development for this class of models resides in the construnction of a physics based loss function, which is then used for the training of the model itself. The physics based loss function is built through the introduction of physcal constraints that can involve both inputs and the outputs generated by the model, and also their derivatives. The typical physics based loss function's structure can be expressed as:

$$Loss(\hat{\boldsymbol{Y}}_i^N, \boldsymbol{Y}_i^N) = Loss_{emp}(\hat{\boldsymbol{Y}}_i^n, \boldsymbol{Y}_i^n) + \lambda_{phy} Loss_{phy}(\hat{\boldsymbol{Y}}_i^n, \boldsymbol{X}_i^n) + \lambda R(\boldsymbol{w}_{net}) \quad (3.13)$$

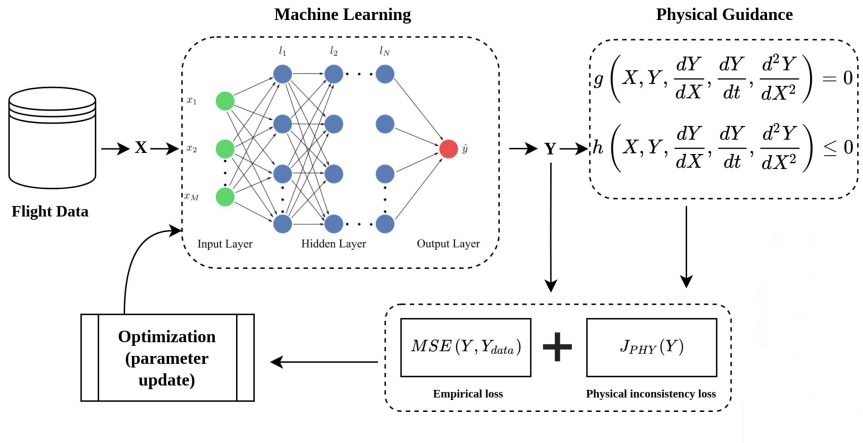

Figure 3.7: Physics guided neural network optimiztion

Chapter 4

Quadrotor PGNN model

The focus of this work is the design of an high level position control based on model inversion through an ANN, using IMU data collected from the drone's onboard sensors. Through the position error, the controller is going to generate reference values for the acceleration measured in the body frame $\boldsymbol{a}_b^* \in \mathbb{R}^3$.
The reference $\boldsymbol{a}_b^*$ is then fed along with the current kynematic state to the ANN, which provides as output the reference value $\boldsymbol{u}^* \in \mathbb{R}^4$ for the actuators in each vertex. The references provided by the Network are then fed into a low level control algorithm. In this chapter the ANN's design is going to be analyzed deeply.

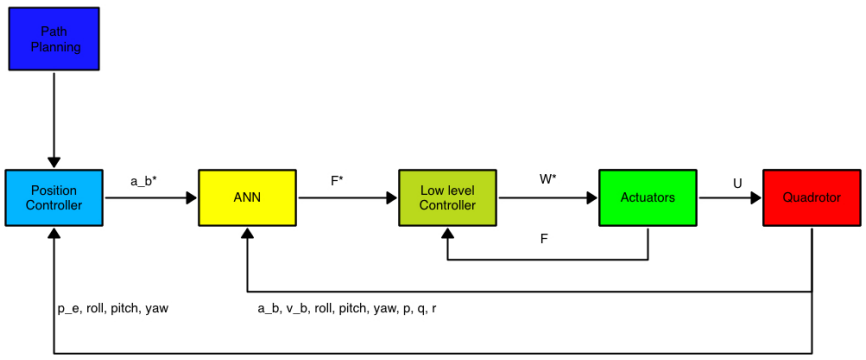

Figure 4.1: Quadrotor control scheme

4.1 Drone Model

According to the analysis made in chapter 2 the following state space model can be defined.

$$\boldsymbol{u}_{(t)} = \begin{bmatrix} F_{1(t)} \\ F_{2(t)} \\ F_{3(t)} \\ F_{4(t)} \end{bmatrix}, \boldsymbol{X}_{(t)} = \begin{bmatrix} u_{(t)} \\ v_{(t)} \\ w_{(t)} \\ \phi_{(t)} \\ \theta_{(t)} \\ \psi_{(t)} \\ p_{(t)} \\ q_{(t)} \\ r_{(t)} \end{bmatrix}, \dot{\boldsymbol{X}}_{(t)} = \begin{bmatrix} \dot{u}_{(t)} \\ \dot{v}_{(t)} \\ \dot{w}_{(t)} \\ \dot{\phi}_{(t)} \\ \dot{\theta}_{(t)} \\ \dot{\psi}_{(t)} \\ \dot{p}_{(t)} \\ \dot{q}_{(t)} \\ \dot{r}_{(t)} \end{bmatrix}, \boldsymbol{Y}_{(t)} = \begin{bmatrix} \phi_{(t)} \\ \theta_{(t)} \\ \psi_{(t)} \end{bmatrix} \tag{4.1}$$

Note that the position state is not relevant for the design of the dynamic inversion model since the drone is not dependent on the actual drone's position, so, for sake of simplicity, it is omitted. Using the equations described in chapter 2 it is possible obtain $\dot{X}_{(t)}$. These equations don't take into account the contributions relative to non-linear forces and torques such as the aerodynamic effects related to turbulence, wind and drag. Taking into account these factors, it is possible to express the forces and torques applied on the quadrotor as:
$\boldsymbol{X}_t := \boldsymbol{X}_{(t)}$, $\boldsymbol{u}_t := \boldsymbol{u}_{(t)}$

$$\boldsymbol{a}_b = \frac{d\boldsymbol{v}_b}{dt} = \frac{1}{M}\left(\boldsymbol{F_p} + R_b^e \boldsymbol{F_g}\right) + \begin{pmatrix} rv - qw \\ pw - ru \\ qu - pv \end{pmatrix} + \boldsymbol{\mathcal{B}_{1(X,u)}} \tag{4.2}$$

$$\dot{\boldsymbol{\omega}}_b = \begin{pmatrix} \frac{1}{J_x} & 0 & 0 \\ 0 & \frac{1}{J_y} & 0 \\ 0 & 0 & \frac{1}{J_z} \end{pmatrix} \left[\begin{pmatrix} (Iz - Iy)qr \\ (Ix - Iz)pr \\ (Iy - Ix)pq \end{pmatrix} \boldsymbol{\omega}_b + \begin{pmatrix} L \\ M \\ N \end{pmatrix} \right] + \boldsymbol{\mathcal{B}_{2(X,u)}} \tag{4.3}$$

with, $\boldsymbol{\mathcal{B}_{1(X,u)}}$ and $\boldsymbol{\mathcal{B}_{2(X,u)}}$ representing the non linear forces and torques contributions that have been neglected in the analysis in chapter 2. The overall function representing the model state derivative $\dot{\boldsymbol{X}}_{(t)}$ is:

$$\dot{\boldsymbol{X}}_{(t)} = \mathcal{N}_{(\boldsymbol{X}_t, \boldsymbol{u}_t)} \tag{4.4}$$

where $\mathcal{N}_{(\boldsymbol{X}_t, \boldsymbol{u}_t)}$ is a generic unknown non-linear function dependent on the current state and the current control input.
This model can be sampled in order to generate a large dataset used for the ANN's training. For each sample equation 4.5 is valid,since using 1-st order taylor's polynomial, the state in $t + \Delta t$ is:

$$\boldsymbol{X}_{t+\Delta t} \simeq \boldsymbol{X}_t + \mathcal{N}_{(\boldsymbol{X}_t, \boldsymbol{u}_t)} \Delta t \tag{4.5}$$

4.2 Neural Network Inversion Model

Based on the control architecture discussed previously the ANN that has to be designed, given measurements relative to the current state of the system and a target acceleration, it has to provide the desired control input. Using the model described in section 4.1 it can be assumed that a function f() that performs the inversion of the dynamics of the system given $\boldsymbol{X}_t$, $\boldsymbol{a}_t$ exists:

$$f(\boldsymbol{X}_t, \boldsymbol{a}_t) = \boldsymbol{u}_t \tag{4.6}$$

since the objective is to compute the control input for instant t+Δt, equation 4.6 can be used imposing $\boldsymbol{a}_{bt+\Delta T} = \boldsymbol{a}_b^*$:

$$f(\boldsymbol{X}_{t+\Delta T}, \boldsymbol{a}_b^*) = \boldsymbol{u}_{t+\Delta t} = \boldsymbol{u}^* \tag{4.7}$$

and then, using eq. 4.5:

$$f(\boldsymbol{X}_t + \mathcal{N}_{(\boldsymbol{X}_t, \boldsymbol{u}_t)}\Delta t, \boldsymbol{a}_b^*) = \boldsymbol{u}^* \tag{4.8}$$

This function must be approximated through the ANN model that is going to be trained, but it's still dependent on the actuation provided in the current timestamp. This is not acceptable for the ANN training, since for small values of Δt some of the input features ($\boldsymbol{u}_t$) are very close to the expected values ($u_{t+\Delta t}$), so in order to complete an effective training, it is mandatory to provide alternative features which allow the network to reach a good level of approximation. so, using eq. 4.6 again:

$$\tilde{\mathcal{N}}(\boldsymbol{X}_t, \boldsymbol{a}_b^*, \boldsymbol{a}_{bt}, \Delta t) := f(\boldsymbol{X}_t + \mathcal{N}_{(\boldsymbol{X}_t, f(\boldsymbol{X}_t, \boldsymbol{a}_{bt}))}\Delta t, \boldsymbol{a}_b^*) = \ \boldsymbol{u}^* \tag{4.9}$$

where $\tilde{\mathcal{N}}$ is the non-linear function the ANN must be trained to approximate. Note that this function is only dependent on the the state, its derivative, the time difference between the samples, the target and the current accelerations.

4.3 Neural Network Architectures

After the model's definition, a good dataset must be collected for the model's training phase. In order to achieve this goal, the AirSim simulator has been used, collecting ground truth data from different quadrotor flights. Three different flights have been collected, with duration of about 20 minutes each:

- Flight $\mathcal{F}_e$, with some hover sections and gentile maneuvers executed at low speed.
- Flight $\mathcal{F}_m$, with sections of hover, gentle and harsh maneuvers.
- Flight $\mathcal{F}_h$, with harsh maneuvers executed at high speed.

each flight has been sampled using the following structure:

$$\mathcal{F}_{(n)} = \Big[t, a_{bx}, a_{by}, a_{bz}, u, v, w, \phi, \theta, \psi, p, q, r, F_{1_i}, F_{2_i}, F_{3_i}, F_{4_i}\Big] \tag{4.10}$$

from each flight, three different datasets $\mathcal{D}$ have been constructed , replicating inputs and outputs of the function $\tilde{\mathcal{N}}$ described in section4.2.
$\mathcal{D} = \{\mathcal{X}, \mathcal{Y}\}$

$$\mathcal{X}_{(n)} = [\Delta t, a_{bx_n}, a_{by_n}, a_{bz_n}, a_{bx_{n+1}}, a_{by_{n+1}}, a_{bz_{n+1}}, u_n, v_n, w_n, \phi_n, \theta_n, \psi_n, p_n, q_n, r_n] \tag{4.11}$$

$$\mathcal{Y}_{(n)} = [F_{1_{n+1}}, F_{2_{n+1}}, F_{3_{n+1}}, F_{4_{n+1}}] \tag{4.12}$$

These dataset is used to solve a supervised learning regression problem, where the Network has to provide for each sample $\mathcal{X}_{(n)}$ the correct output $\mathcal{Y}_{(n)}$. The training and the validation of all the models has been done using the Pytorch framwork for python.
The performances of this models have been validated using cross validation, splitting each of these datasets into subsets for training, test and validation sets. There are mainly three categories of architectures that have been developed and trained in this work:

- A FFNN[12] models with 4 layers and 3 ReLU[12] activation functions. The input layer is of size 19 (ϕ, θ, ψ are encoded into sines and cosines) meanwhile the hidden layers are of size 50, the output layer is size 16.
- A Deep FFNN model , with 5 layers of size 19, 50 and 16 for input, hidden and output layers respectively.
- An LSTM[13] model made by 4 LSTM cells whose hidden state's size is 4 and hyperbolic tangent activation function. The output of the final cell is then fed

into another linear layer of size 16 with a ReLU activatoin function; the output layer is of size 4. This problem is a "many to one" problem, so, only the output of the last cell is evaluated in the loss function.

All these architectures have been trained both in a data-driven and hybrid fashion using RMSE[12] loss for the first and the physics based loss function described in section 4.4. The choice for the optimizer to be used is discussed in chapter 5.

4.4 Physics Based Loss function

Using the equations obtained in chapter 2, it's possible to extend the architectures discussed in section 4.3 for the development hybrid models. This task can be achieved using physics based loss functions during the training phase of each models.
The objective of the architectures proposed previously is to develop a model able to produce reference values for the forces applied on each vertex on the quadrotor. From section 2.4 it is possible to introduce a penalty term in the loss function in order to introduce physical information in the optimization process. The physical constraints put in relation the forces $\boldsymbol{u}_{(t)}$ with $\boldsymbol{a}_{b(t)}, \boldsymbol{v}_{b(t)}, \boldsymbol{\Phi}_{(t)}, \boldsymbol{\omega}_{(t)}, \dot{\boldsymbol{\omega}}_{(t)}$. During each step of the control loop the network must provide the forces applied during the next one, but the measurement for $\dot{\boldsymbol{\omega}}_{(t+\Delta t)}$ are not available in the Airsim environment, so, among the equations discussed in section 2.4, only the equation relating the thrust(F_1, F_2, F_3, F_4) and the acceleration is suitable for the implementation of the loss function.

$$\boldsymbol{a}_b = \frac{d\boldsymbol{v}_b}{dt} = \frac{1}{M}\left(\sum_{i=1}^{4} u_i \boldsymbol{z}_b + R_b^e \boldsymbol{F_g}\right) + \begin{pmatrix} rv - qw \\ pw - ru \\ qu - pv \end{pmatrix} = f_c(\boldsymbol{u}, \boldsymbol{\Phi}, \boldsymbol{\omega}, \boldsymbol{v}_b)$$

The architecture proposed in section 4.3 can be trained and extended providing both as output $\boldsymbol{u}_{t+\Delta t}$ and $\boldsymbol{X}_{t+\Delta t}$ in order to evaluate f_c. Using this method, the physics based loss function becomes:
given $\boldsymbol{u}^*$ and $\boldsymbol{X}^*$ as the predicted command and state and $\boldsymbol{u}$ as expetcted command,

$$L(\boldsymbol{u}^*, \boldsymbol{X}^*) = RMSE(\boldsymbol{u}^*, \boldsymbol{u}) + \lambda MSE(\boldsymbol{a}_{b_{(t+\Delta t)}}, f_{c(\boldsymbol{u}^*, \boldsymbol{X}^*)}) \tag{4.13}$$

Chapter 5

Training and Validation Results

5.1 training framework

The models proposed in section 4.1 have been trained using the cross validation technique. Each dataset has been split in three different parts:

- The training set is about 70% of the full set, it is used in order to optimize the networks paramters during each epoch. The loss function evaluated in each sample is the physics based loss function described in section 4.4.
- The validation is about 15% of the full set, it is used to compute the validation loss after each epoch. Since our objective is to achieve the best approximation for the expected forces, the evaluated loss function is the RMSE loss with respect to the expected forces.
- The test set about 15% of the full set, it is new, unseen data which is used to evaluate the performances of each model after the training and validation procedure. The loss function evaluated is the same used for the validation set.

Each subset is also shuffled randomly before each training, but, in order to remove the stochastic component in the evaluation of each model, each training has been performed with the same random seed. All the models have been trained for a maximum 1000 epochs using early stopping in case of overfitting or if the model's validation loss stops to improve. The update of the model's parameters is based on adaptive moments estimation through the adam optimizer module provided in PyTorch. The effectiveness of this algorithm has been proven in several articles and also in this work, it performed better with respect to other algorithms, such as SGD with Nesterov Momentum. The choiche of the batch size for all models is 1024 samples.
This value has been chosen in order to stabilize the the training and validation errors, that for small batch sizes are extremely volatile and drastically increase the time needed to optimize the model through each epoch. The learning rate hyper-parameter λ has been tuned manually and is the same for both the data-driven version and the hybrid version, in order to make the performances of both models comparable for the evaluation of the improvements achieved through physics based loss function with respect to the standard one. Of course this models are not fully optimized yet, fine tuning wasn't done(due to time issues) and learning rate schedulers can be used to further improve the performances.

Algorithm 1 Adam optimizer algorithm. All operations are element-wise, even powers. Good values for the constants are $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$. ϵ is needed to guarantee numerical stability.

procedure ADAM($\alpha, \beta_1, \beta_2, f, \theta_0$)
  ▷ α is the stepsize
  ▷ $\beta_1, \beta_2 \in [0, 1)$ are the exponential decay rates for the moment estimates
  ▷ $f(\theta)$ is the objective function to optimize
  ▷ θ_0 is the initial vector of parameters which will be optimized
  ▷ Initialization
  $m_0 \leftarrow 0$ ▷ First moment estimate vector set to 0
  $v_0 \leftarrow 0$ ▷ Second moment estimate vector set to 0
  $t \leftarrow 0$ ▷ Timestep set to 0
  ▷ Execution
  while θ_t not converged **do**
    $t \leftarrow t + 1$ ▷ Update timestep
    ▷ Gradients are computed w.r.t the parameters to optimize
    ▷ using the value of the objective function
    ▷ at the previous timestep
    $g_t \leftarrow \nabla_\theta f(\theta_{t-1})$
    ▷ Update of first-moment and second-moment estimates using
    ▷ previous value and new gradients, biased
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$
    ▷ Bias-correction of estimates
    $\hat{m}_t \leftarrow \dfrac{m_t}{1 - \beta_1^t}$
    $\hat{v}_t \leftarrow \dfrac{v_t}{1 - \beta_2^t}$
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \dfrac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$ ▷ Update parameters
  end while
  return θ_t ▷ Optimized parameters are returned
end procedure

5.2 Low velocity flight Dataset results

It's well evident that for this dataset, all the models are capable of reaching a good level of approximation with an RMSE value between 0.1N-0.2N. The best results are the ones obtained with the LSTM architecture. It's also worth noticing that the physics based loss function doesn't improve the performances of any model. The forces applied during this flight are almost constant and the total acceleration and jerk are close to 0, so the data driven models reach very good level of approximation,so the hybrid models perform with no noticeable improvements.

Low Velocity flight				
Model	λ_{phy}	Validation	Test	Δ L()%
LSTM	0	0.132	0.151	-
LSTM	1	0.129	0.151	0%
DFFNN	0	0.134	0.161	-
DFFNN	0.5	0.132	0.163	0%
FFNN	0	0.121	0.133	-
FFNN	0.01	0.131	0.147	0%

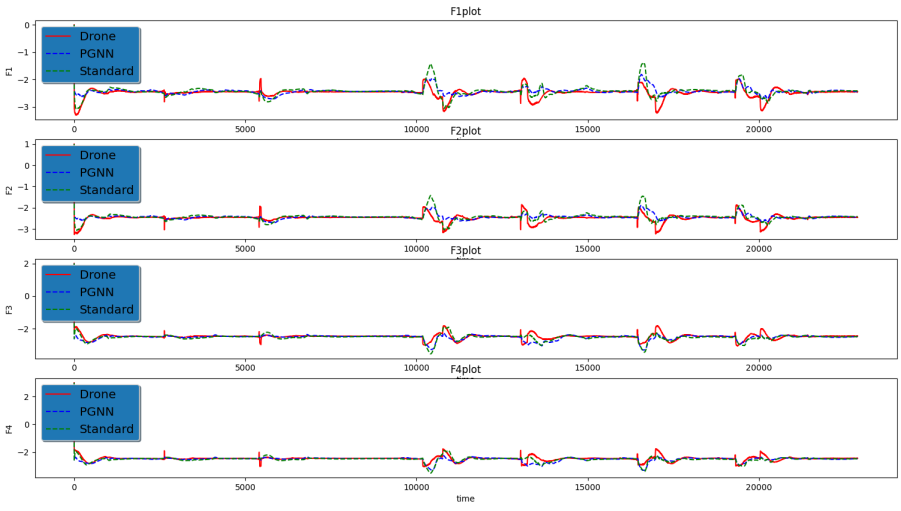

Figure 5.1: Low velocity LSTM prediction comparison

As figure 5.1 shows, the value of the expected output is almost constant, and the

lstm network seems to follow predictions.

5.3 Mixed dataset results

The results obtained with this dataset are quite different from the ones analyzed in section 5.2 and 5.4. Although the RMSE is worse than what achieved in section 5.2 the improvements achieved using hybrid models are significant, for both the LSTM model, and FFNN with residual block model, with a riduction of the RMSE of 10% and 15.7% respectively, compared with their data-driven counterparts. The dynamics related to this dataset is harder to approximate and that is reflected by the increase of RMSE value to 0.3N-0.6N. It is also worth of mentioning that for all the models, the test loss is lower than the validation loss. This is due to an higher presence of high speed maneuvers in the validation set. The lstm hybrid model is the best model again, with an RMSE of 0.317N.

Mixed flight				
Model	λ_{phy}	Validation	Test	Δ L()%
LSTM	0	0.367	0.359	-
LSTM	0.3	0.355	0.317	10%
DFFNN	0	0.501	0.500	-
DFFNN	0.2	0.467	0.421	15.7%
FFNN	0	0.470	0.43	-
FFNN	0.01	0.450	0.480	0%

5.4 High velocity flight Dataset results

The loss of performance related to high velocity maneuvers is more evident using this dataset. Even if the hybrid based model perform slightly better than the data-driven ones, the influence of the information driven approach is minor(2%-5% of RMSE reduction). This is related to an higher influence of the non-linear forces which are not considered in the kyno-dynamic equations described in chapter 2. It is also noticeable that that the LSTM model's performances are much better than the FFNN models with an error of 0.492N.

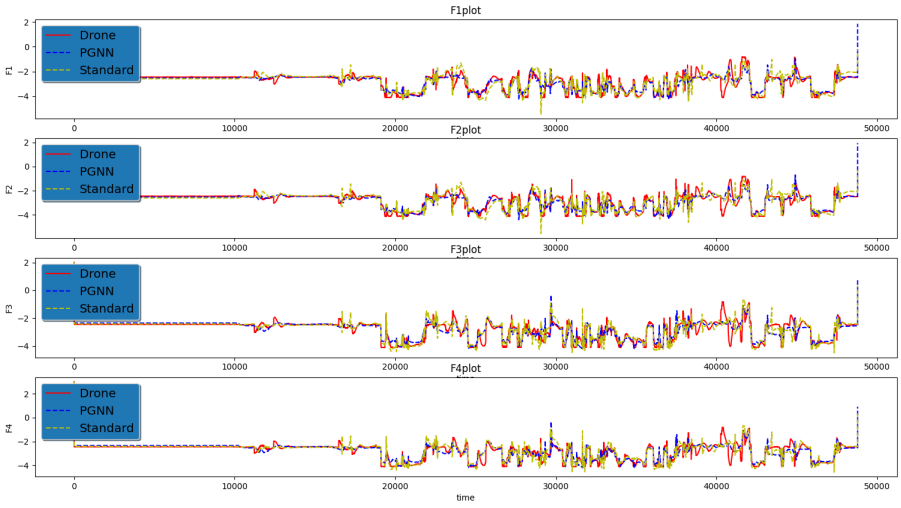

Figure 5.2: Mixed flight LSTM prediction comparison

High velocity flight				
Model	λ_{phy}	Validation	Test	Δ L()%
LSTM	0	0.427	0.500	-
LSTM	0.001	0.419	0.492	2%
DFFNN	0	0.480	0.732	-
DFFNN	0.1	0.463	0.692	5.4 %
FFNN	0	0.514	1.169	-
FFNN	0.1	0.496	0.675	0%

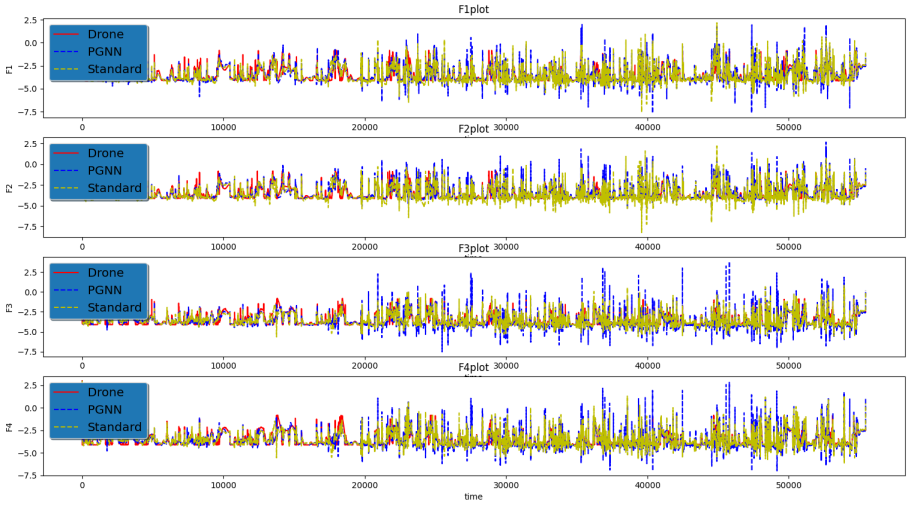

Figure 5.3: High velocity lstm predictoin comparison

Chapter 6

Conclusions and further work

6.1 Conclusions

After the analysis made in chapter 5 it's well evident that the lstm model performs better than the FFNN models, which confirms that this architecture is well suited for time series problems. It is also worth noting the impact of the Physics based loss function is significant, with decrease 10% RMSE and 15.7% decrease for the lsmt and the FFNN with skip connection in the mixed dataset, which is the most representative of the overall quadrotor's dyanmics.
The effect of this hybrid based approach is more evident in the DFFNN with skip connection architecture, where the complexity of the model can easily lead to overfitting. On the other hand, simple model which easily underfit the data, such as the FFNN, don't benefit from this approach, since the loss in performances is related to the low complexity of the architecture itself.
Overall the performances of the best among this models are acceptable, with the lstm hybrid model that achieves 0.151N, 0.317N and 0.492N with the low velocity, mixed, and high speed datasets respectively. This result prove that this architectures are suitable for the approximation of inverse dynamics, and that, PGNN improve the performances of models designed for the approximation of strongly non linear models.

6.2 Further work

The effectiveness of this control architecture must be in the airsim environment through real time simulation. These architectures can be finely tuned to achieve the best performances, and evaluate if the inference time of this models is compatible with the time constant of the quadrotor system. The physical constraints used for the physics based loss functions implemented in this work are limited to the equations related to the acceleration in the body frame, since in the airsim environment the real-time measurement of $\dot{\omega}_b$ is not available. The basic control scheme can be extended using observers to obtain this measurement in real time and further extend these physical constraints. It's also worth mentioning that the output provided by this network is still affected from errors, so an auxiliary PI controller can complement the control action provided by this controller in order to further improve the performances.

Appendix A

PyTorch

labelsec:py PyThorch is a Python based framework aimed for production implementation in machine learning, and its user friendliness with careful performance consideration led to its popularity in the Deep Learning research comunity. Its main purpose is the replacemnt of NumPy for handling complex Matrix computation exploiting the advantages of High performance computation with GPU and other accelerators. Another advantage of PyTorch provides is that it can be used as an automatic Differentiation library, which is crucial for the training of ANN models. These computation are made taking advantage of dynamic computation graphs which allow an efficient computation of the gradients relative to each tensor. The main element of this library is the tensor indeed, a multidimensional object which stores additional information relative to the gradients of each element. This objects can be used for the analysis of numbers, vectors or matrices and allow efficient computations using GPU or CPU.

A.1 Modules

The main modules used in the development of the ML-models in this work are:

- The nn module, which include the classes necessary for the build of NN models. All the NN modules used in python are subclasses of the nn module.
- Optim is the module that provides common algorithms for the optimizers used to update the NN parameters.
- Autograd is the module that handles automatic differentiation that is exploited in order to compute gradients during the forward pass in real time. It generates a directed acyclic graph where the leaves are the input tensors while the roots are the output tensors.

Bibliography

[1] Weibin Gu, Kimon P Valavanis, Matthew J Rutherford, and Alessandro Rizzo. «UAV model-based flight control with artificial neural networks: A survey». In: *Journal of Intelligent & Robotic Systems* 100.3 (2020), pp. 1469–1491 (cit. on p. 1).

[2] Byoung S Kim and Anthony J Calise. «Nonlinear flight control using neural networks». In: *Journal of Guidance, Control, and Dynamics* 20.1 (1997), pp. 26–33 (cit. on p. 2).

[3] Suresh K Kannan and Eric N Johnson. «Adaptive trajectory based control for autonomous helicopters». In: *Proceedings. The 21st Digital Avionics Systems Conference*. Vol. 2. IEEE. 2002, pp. 8D1–8D1 (cit. on p. 2).

[4] Girish V Chowdhary and Eric N Johnson. «Theory and flight-test validation of a concurrent-learning adaptive controller». In: *Journal of Guidance, Control, and Dynamics* 34.2 (2011), pp. 592–607 (cit. on p. 2).

[5] David Torres Ocaña, Hyo-Sang Shin, and Antonios Tsourdos. «Development of a nonlinear reconfigurable f-16 model and flight control systems using multilayer adaptive neural networks». In: *IFAC-PapersOnLine* 48.9 (2015), pp. 138–143 (cit. on p. 2).

[6] Matthew Garratt and Sreenatha Anavatti. «Non-linear control of heave for an unmanned helicopter using a neural network». In: *Journal of Intelligent & Robotic Systems* 66.4 (2012), pp. 495–504 (cit. on p. 2).

[7] Cheng Peng, Yue Bai, Xun Gong, Qingjia Gao, Changjun Zhao, and Yantao Tian. «Modeling and robust backstepping sliding mode control with Adaptive RBFNN for a novel coaxial eight-rotor UAV». In: *IEEE/CAA Journal of Automatica Sinica* 2.1 (2015), pp. 56–64 (cit. on p. 2).

[8] Shushuai Li, Yaonan Wang, Jianhao Tan, and Yan Zheng. «Adaptive RBFNNs/integral sliding mode control for a quadrotor aircraft». In: *Neurocomputing* 216 (2016), pp. 126–134. ISSN: 0925-2312. DOI: `https://doi.org/10.1016/j.neucom.2016.07.033`. URL: `https:`

//www.sciencedirect.com/science/article/pii/S0925231216307780 (cit. on p. 2).

[9] Tarek Madani and Abdelaziz Benallegue. «Adaptive control via backstepping technique and neural networks of a quadrotor helicopter». In: *IFAC Proceedings Volumes* 41.2 (2008), pp. 6513–6518 (cit. on p. 3).

[10] M. Raissi, P. Perdikaris, and G.E. Karniadakis. «Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations». In: *Journal of Computational Physics* 378 (2019), pp. 686–707. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2018.10.045. URL: https://www.sciencedirect.com/science/article/pii/S0021999118307125 (cit. on pp. 3, 18).

[11] Randal W. Beard. «Quadrotor Dynamics and Control». In: (Feb. 2008) (cit. on p. 5).

[12] Jeff Heaton. *Ian goodfellow, yoshua bengio, and aaron courville: Deep learning*. 2018 (cit. on pp. 11–16, 22, 23).

[13] Yong Yu, Xiaosheng Si, Changhua Hu, and Jianxun Zhang. «A review of recurrent neural networks: LSTM cells and network architectures». In: *Neural computation* 31.7 (2019), pp. 1235–1270 (cit. on pp. 11, 22).