Time-delay estimation on identified model of a quadrotor UAV

Tesi di Laurea Magistrale in Aeronautical Engineering - Ingegneria Aeronautica

Author: **Marco Guzzon**

Student ID: 953034
Advisor: Prof. Marco Lovera
Academic Year: 2021-22

Abstract

This study proposes a selection of discrete-time time delay identification methods implemented in MATLAB® environment with a flexible choice of sample time and adapted to work with MIMO systems, such as the UAV model adopted by my department and described in the first chapter of this thesis.

The aforementioned UAV model is illustrated with its underlying hypotheses, as is the identification process used to find the values of the model parameters, then the results and the model are adapted to the discrete-time MATLAB® environment, they form the system from which the data necessary to estimate the time delays are extracted.

Four methods are then presented, specifically two related to the maximization of the cross-correlation between input and output quantities: the thresholding method, as described by S. Björklund, and the criterion functions proposed in an article by J. Kekalainen; one founded on the least-squares minimization of a cost function built using the state dynamics of the system, by R. Waschburger and R. K. H. Galvão; the last, also described by S. Björklund and called `oestruc`, being built on the estimation of an output-error model of the analysed system. Their theoretical bases are explored and then they are adapted to the discrete-time MATLAB® environment.

Each method has been tested for 90 times, 30 for each chosen input, on each dynamics (lateral, longitudinal, directional and vertical) of the UAV model. The results have been compared and conclusions regarding the performance of each method have been drawn, outlining which inputs and use cases each method is most suitable fir, how reliable their estimates are and how long they take to yield said estimates. Special attention has been paid to compare the difference in behaviour of each delay estimation method with a stable or with an unstable system.

Keywords: time delay estimation, discrete time, MATLAB, PBSID, UAV

Abstract in lingua italiana

Questo studio propone una selezione di metodi di identificazione, in tempo discreto, di ritardi di tempo implementati in ambiente MATLAB® con flessibilità sulla scelta del tempo di campionamento e adattati per l'uso con sistemi MIMO, come il modello di UAV adottato dal mio dipartimento e descritto nel primo capitolo di questa tesi.

Il modello di UAV menzionato è illustrato con le ipotesi su cui si fonda, così come il processo di identificatione usato per trovare i valori dei parametri del modello, questi risultati e il modello sono poi adattati all'ambiente MATLAB® in tempo discreto e formano il sistema da cui sono estrapolati i dati necessari alla stima dei ritardi di tempo.

Sono poi presentati quattro metodi, in particolare due legati alla massimizzazione della correlazione incrociata delle quantità in ingresso e in uscita: il metodo di thresholding, descritto da S. Björklund, e le funzioni criterio proposte in un articolo di J. Kekalainen; un altro metodo, di R. Waschburger e R. K. H. Galvão, è fondato sull'ottimizzazione ai minimi quadrati di una funzione di costo costruita a partire dalla dinamica degli stati del sistema; l'ultimo metodo, anche questo descritto da S. Björklund e chiamato `oestruc`, è costruito sulla stima di un modello output-error del sistema analizzato. Le loro basi teoriche sono spiegate e, in seguito, i metodi sono stati adattati all'ambiente MATLAB®.

Ogni metodo è stato provato 90 volte, 30 per ciascun tipo di ingresso scelto, e per ciascuna dinamica (laterale, longitudinale, direzionale e verticale) del modello di UAV. I risultati sono stati paragonati tra loro e sono state tratte conclusioni sulle prestazioni di ogni metodo, illustrando per quali ingressi e casi d'uso ogni metodo è più adatto, quanto sono affidabili le loro stime e quanto tempo è stato impiegato per ricavare queste stime. Si è posta attenzione a confrontare le differenze tra l'applicazione di ogni metodo a un sistema stabile e a un sistema instabile.

Parole chiave: stima ritardo di tempo, tempo discreto, MATLAB, PBSID, UAV

Contents

Introduction

Among the many aspects of dynamic models, whether linear or non-linear, of physical systems, there is delay associated to the dynamics, an aspect that is sometimes overlooked during identification procedures, with the estimation of parameters usually being given precedence. These two problems are different from each other, as such they are studied and described in literature separately.

There are innumerable methods to identify the parameters of a model, both in continuous and in discrete time and, parallel to them, run the methods to estimate the time-delays affecting the model under study.

There are works that describe model identification methods capable of yielding a model that has physical meaning, such as the PBSID$_{opt}$-H$_\infty$ method illustrated in Giovanni Fumai's master degree thesis ([4]), a method that initially performs *black-box identification* of a state space model through the Predictor-Based Subspace Identification method, then forces the parameters to fit into a known model structure through model matching by using the minimization of the H$_\infty$ of a cost function.

The model obtained this way constitutes the basis of this study: different discrete-time time-delay identification methods will be applied to it in MATLAB® environment, in order to extract results needed to compare the methods to each other and understand the strength and weaknesses of each estimation method.

The objective of this study is to select at least one time-delay estimation method that is compatible with the aforementioned parameters estimation technique, and to apply it to the identification of the dynamics of a quadrotor UAV.

The chosen methods have been elaborated in a time span of about two decades, some are from the early 2000s, others have been published less than a decade ago, the underlying route during the choice of methods was choosing methods that were not similar to each other, in terms of necessary data, of reliability, of scenarios they are suited to, of computational time, and according to these characteristics they will be judged.

The structure of this thesis is as follows:

- Chapter 1 is a brief overview of the relevant sections of G. Fumai's thesis ([4]), illustrating the main hypotheses, the structure of the model, how the parameter identification was performed, its results and how they have been manipulated to fit into the MATLAB® environment.
- Chapter 2 describes the theoretical foundations of the four chosen time-delay estimation methods, the categories they belong to and then explains why precisely those four methods have been chosen.
- Chapter 3 illustrates how the chosen methods have been adapted to work with MIMO systems, flexble choice of sample time and in the MATLAB® environment.
- Chapter 4 presents how the tests were carried out, the chosen inputs, how each method was set up. Then, the results are shown, divided by input that generated them, and commented, comparing the methods with each other according to the different situations they were used in.
- The last chapter, chapter 5, is a summary of the observations and conclusions drawn in the preceding chapter, and finally it suggests how this work can be expanded on in the future.

1 | UAV model

This chapter outlines the physical model, in state space form, of a simple quadrotor UAV, briefly explaining the major hypotheses upon which the model is built, and what the components of the model indicate. Next, a method, namely a Predictor-based Subspace Identification method augmented by model matching with H_∞ norm optimization, to identify the state space model will be presented, along with all its steps and the results stemming from experimental activities carried out on a quadrotor drone. These results make up the sets providing information vital to work with the chosen discrete time delay estimation methods, in order to compare them to each other in a setting relevant to real applications.

1.1. Description of the system and major hypotheses

1.1.1. Configuration of the UAV

The system that has been taken as reference is an X or + shaped UAV with four rotors, one for each arm of the X.

A generic quadrotor UAV, with all four rotors pointing in the same direction, is an under-actuated system: the number of independent control inputs is lower than the six degrees of freedom allowed.

The rotors are divided in two couples, one spinning in clockwise motion, while the other counterclockwise, each couple is made up of two diametrally opposed rotors. Pitch and roll controls are obtained by varying the rotation speed of the rotors on each side of the aircraft, so that two rotors on one side spin faster than those on the other side. Altitude control is simply given by collectively varying the speed of all rotors.

1.1.2. Main hypotheses affecting the model

The main assumptions adopted in [4] to create the aforementioned physical model are the following:

- **Rigid frame**. Vibrational modes are not taken into account as they are assumed to be faster than rotor-induced actions and it is also assumed that deformations of the UAV structure do not affect the rotor aerodynamics.
- **Aeroelasticity of propellers not modeled**. Drag induced by translational velocity is included in steady-state aerodynamic damping. This approximation is, according to [4], valid only for low operating velocities in an indoor environment, such an environment is considered to have no wind of any kind.
- **Ground effect is neglected**.
- **Gyroscopic effect is neglected**. It can be safely neglected only on small scale UAVs that are not piloted in ways that do not combine yawing with pitching or with rolling.
- **Fixed configuration**. The rotors cannot tilt, there are no lifting surfaces nor flight control surfaces on the drone.
- **Electrodynamics of servomotors are neglected**. These dynamics are much faster than rotor dynamics, thus they can be assumed to not influence the latter ones.

1.2. LTI model of the UAV

The procedure that leads to the linear time-invariant structured model can be found in [4], SEZ 1.2, here the major steps that pave the way to the model will be briefly outlined and the model itself will then be presented.

1.2.1. Framework of the model

The model is based on the equation linear and angular motion and aerodynamic forces and moments, the latter two depend on the following variables:

- Reynolds number, Re
- Mach number, Ma
- Velocity of the aircraft, V_G
- Angular speeds vector of the aircraft, $\boldsymbol{\omega}$
- Control inputs vector, $\boldsymbol{\delta}$

$\boldsymbol{\omega} = \begin{bmatrix} r & q & r \end{bmatrix}$ is the vector of angular velocities in body-axes, respectively around x, y and z.

The last variable from the list is a vector of four components: $\boldsymbol{\delta} = \begin{bmatrix} \delta_{ver} & \delta_{dir} & \delta_{lat} & \delta_{lon} \end{bmatrix}$, they are dimensionless and express the intensity of the manoeuvres described in Section 1.1.1:

- δ_{ver} For collective thrust
- δ_{dir} For changes in direction
- δ_{lat} For rolling and lateral motions
- δ_{lon} For pitching and longitudinal motions

The equations are linearized around a reference steady-flight condition. In [4] the choice of reference condition falls upon the trim condition, that is when the flight trajectory such that "the linear and angular velocity (sic) remain constant for a constant input: linear and angular accelerations must be zero", leading to equations (1.1) and (1.2):

$$0 = \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} + F_{aero}(0, 0, \boldsymbol{\delta}_0, \mathrm{Re}, \mathrm{Ma}) \tag{1.1}$$

$$0 = M_{aero}(0, 0, \boldsymbol{\delta}_0, \mathrm{Re}, \mathrm{Ma}) \tag{1.2}$$

where $\boldsymbol{\delta}_0$ is the value of $\boldsymbol{\delta}$ that makes (1.1) and (1.2) true: $\boldsymbol{\delta}_0 = \begin{bmatrix} 0 & 0 & 0 & \delta_{hold} \end{bmatrix}$, the last component being the intensity of vertical command needed to keep a constant altitude. The solution of the perturbed equations is approximated by first order terms of the Taylor expansion. The results are the equations from Equation 1.3 to Equation 1.8 :

$$m\Delta\dot{u} = -mg\Delta\theta + \frac{\partial X}{\partial \delta_{lon}}\Delta\delta_{lon} + \frac{\partial X}{\partial u}\Delta u + \frac{\partial X}{\partial q}\Delta q \tag{1.3}$$

$$m\Delta\dot{v} = mg\Delta\phi + \frac{\partial Y}{\partial \delta_{lat}}\Delta\delta_{lat} + \frac{\partial Y}{\partial v}\Delta v + \frac{\partial Y}{\partial p}\Delta p \tag{1.4}$$

$$m\Delta\dot{w} = \frac{\partial Z}{\partial \delta_{ver}}\Delta\delta_{ver} + \frac{\partial Z}{\partial w}\Delta w \tag{1.5}$$

$$J_{xx}\Delta\dot{p} = \frac{\partial \mathcal{L}}{\partial \delta_{lat}}\Delta\delta_{lat} + \frac{\partial \mathcal{L}}{\partial v}\Delta v + \frac{\partial \mathcal{L}}{\partial p}\Delta p \tag{1.6}$$

$$J_{yy}\Delta\dot{q} = \frac{\partial \mathcal{M}}{\partial \delta_{lon}}\Delta\delta_{lon} + \frac{\partial \mathcal{M}}{\partial u}\Delta u + \frac{\partial \mathcal{M}}{\partial q}\Delta q \tag{1.7}$$

$$J_{zz}\Delta\dot{r} = \frac{\partial\mathcal{L}}{\partial\delta_{dir}}\Delta\delta_{dir} + \frac{\partial\mathcal{N}}{\partial r}\Delta r \tag{1.8}$$

1.2.2. LTI state space model

The linearized equations presented at the end of Section 1.2.1 are recast into a continuous time, linear time-invariant state space form such as:

$$\begin{cases} \dot{x}(t) = Ax(t) + Bu(t) \\ y(t) = Cx(t) + Du(t) \end{cases}$$

If the perturbed terms of the previous equations have a null reference value in the hovering condition, they are cast out in favour of analogous actual quantities, by doing so the input and state vectors are obtained:

$$\boldsymbol{u} := \begin{bmatrix} \delta_{lat} & \delta_{lon} & \delta_{dir} & \Delta\delta_{vert} \end{bmatrix}^T$$

$$\boldsymbol{x} := \begin{bmatrix} v & p & \phi & u & q & \theta & r & \Delta\psi & w \end{bmatrix}^T$$

The outputs have been chosen by the author of [4] to be variables measured by on-board sensors (namely Inertial Measurement Units):

$$\boldsymbol{y} := \begin{bmatrix} p & a_y & q & a_x & r & a_z + g \end{bmatrix}^T$$

The first equation of the state space form, is derived from the linearized equations, from Equation 1.3 to Equation 1.8, by dividing each by m or by their inertia term J_{ij} (with appropriate values for i and j) and the matrices A and B are built as follows:

$$A = \begin{bmatrix} Y_v & Y_p & g & 0 & 0 & 0 & 0 & 0 & 0 \\ L_v & L_p & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & X_u & X_q & -g & 0 & 0 & 0 \\ 0 & 0 & 0 & M_u & M_q & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & N_r & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & Z_w \end{bmatrix} \tag{1.9}$$

$$B = \begin{bmatrix} Y_\delta & 0 & 0 & 0 \\ L_\delta & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & X_\delta & 0 & 0 \\ 0 & M_\delta & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & N_\delta & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & Z_\delta \end{bmatrix} \tag{1.10}$$

with, for example, $Y_v = \frac{\partial Y}{m \partial v}$.

Due to A having a clear block diagonal structure, it can be easily understood that the four dynamics (respectively lateral, longitudinal, directional and vertical) are decoupled from each other.

Moreover, some initial observations regarding the open-loop stability of these four dynamics are also given in [4]:

- **Lateral and longitudinal dynamics are unstable**: the UAV departs the hovering condition if a lateral or longitudinal input is given, even if the command is restored to zero and if any acceleration and angular motions are assumed to have been dissipated by aerodynamic damping.
- **Directional and vertical dynamics are stable**: aerodynamic damping counteracts any yawing motion and, as soon as the command is released, it starts to reduce the influence of that motion, slowing it down while maintaining the UAV in hovering condition; the vertical command is brought back to the hovering condition value from a different value, the UAV is brought back in hovering condition.

The output equation is based on the equations that model how the measurements are taken:

$$a_x := \dot{u} + g\sin(\theta_0 + \Delta\theta) \approx \dot{u} + g\Delta\theta = X_u u + X_q q + X_\delta \delta_{lon} \tag{1.11a}$$

$$a_y := \dot{v} - g\cos(\theta_0 + \Delta\theta)\sin(\phi_0 + \Delta\phi) \approx \dot{v} - g\Delta\phi = Y_v v + Y_p p + Y_\delta \delta_{lat} \tag{1.11b}$$

$$a_z := \dot{w} - g\cos(\theta_0 + \Delta\theta)\cos(\phi_0 + \Delta\phi) \approx \dot{w} - g = Z_w w - g + Z_\delta \Delta\delta_{ver} \tag{1.11c}$$

The previous equations (Equations 1.11) lead to these structures for matrices C and D:

$$C = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ Y_v & Y_p & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & X_u & X_q & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & Z_w \end{bmatrix} \tag{1.12}$$

$$D = \begin{bmatrix} 0 & 0 & 0 & 0 \\ Y_\delta & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & X_\delta & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & Z_\delta \end{bmatrix} \tag{1.13}$$

1.3. Identification of the parameters

The state space identification of the model performed in [4] is performed in two main phases:

1. A predictor-based subspace identification (PBSID$_{\text{opt}}$), driven using discrete-time sequences of inputs and outputs. The result is a discrete-time state space model that does not necessarily have physical meaning, it is not the structured state space model presented in Section 1.2.2, hence why the second phase exists.
2. The identified black-box model is forced to match the structured model through:

$$\min_{\theta} ||G_{identified,unstructured}(s) - G_{strcutured}(s,\theta)||_\infty$$

   which is the H$_\infty$ norm of the difference of the two transfer functions, one of the identified model, the other of the structured model; θ is the symbol indicating the parameters of the structured model (the non-null components of the state space matrices) and it is the variable over which the minimization is carried out.

1.3.1. PBSID$_{\text{opt}}$ optimization

The first phase has been carried out through the *PBSID Toolbox*, developed by Technische Universiteit Delft, in works such as [5], [6], [11] and other works found at [7] with the

Toolbox itself, which allows to apply PBSID estimation in the MATLAB® environment. The theoretical elaboration of the method is explained in [4], just the two flows followed in that thesis are presented in this document, beginning with Figure 1.1.

The upper flow only uses function from the aforementioned *PBSID Toolbox*, obtaining the non-structured state space matrices from a reduced order state sequence, then proceeds to build the asymptotic variance of frequency response magnitude. The lower flow exploits other toolboxes for the last steps, instead of going through all the necessary steps with *PBSID Toolbox*:

- `mvnrnd` comes from *Statistics and Machine Learning Toolbox* of MATLAB®. Using the Monte Carlo method, it generates random matrices A, B, C, D, K for the unstructured state space model according to the asymptotic covariance matrix of the elements of the state space matrices.
- `frd` comes from *Control System Toolbox*. It produces the frequency response data model over a set of frequencies represented, in fig. 1.1, by the green "discrete frequencies" input. Its inputs are the set of frequencies and the sets of inputs and outputs at those frequencies, these latter two sets are obtained by simulating the many models produced by the Monte Carlo method.

1.3.2. H_∞ norm model matching

The generic framework for this second phase is based around the usage of invariants of the dynamic systems, specifically, in this case, the transfer functions of the identified unstructured model and of the structured model. A cost function having a lower cost the closer the two transfer functions are is thus created: it is the one presented at the beginning of section 1.3. This choice of cost function carries some advantages:

- There is no need for an explicit similarity transformation, so the only optimization variables are the state space parameters and their number is independent from the dimension of the model class.
- The optimization variables, that is to say the state space parameters, are physical parameters, some of their characteristics might already be available.
- This method can be applied to all model classes for which the H_∞ can be computed, even time-varying models.

The non-smooth optimization of H_∞ norm, the type of optimization this method belongs to, is implemented in the MATLAB® *Robust Control Toolbox* as the function `hinfstruct`.

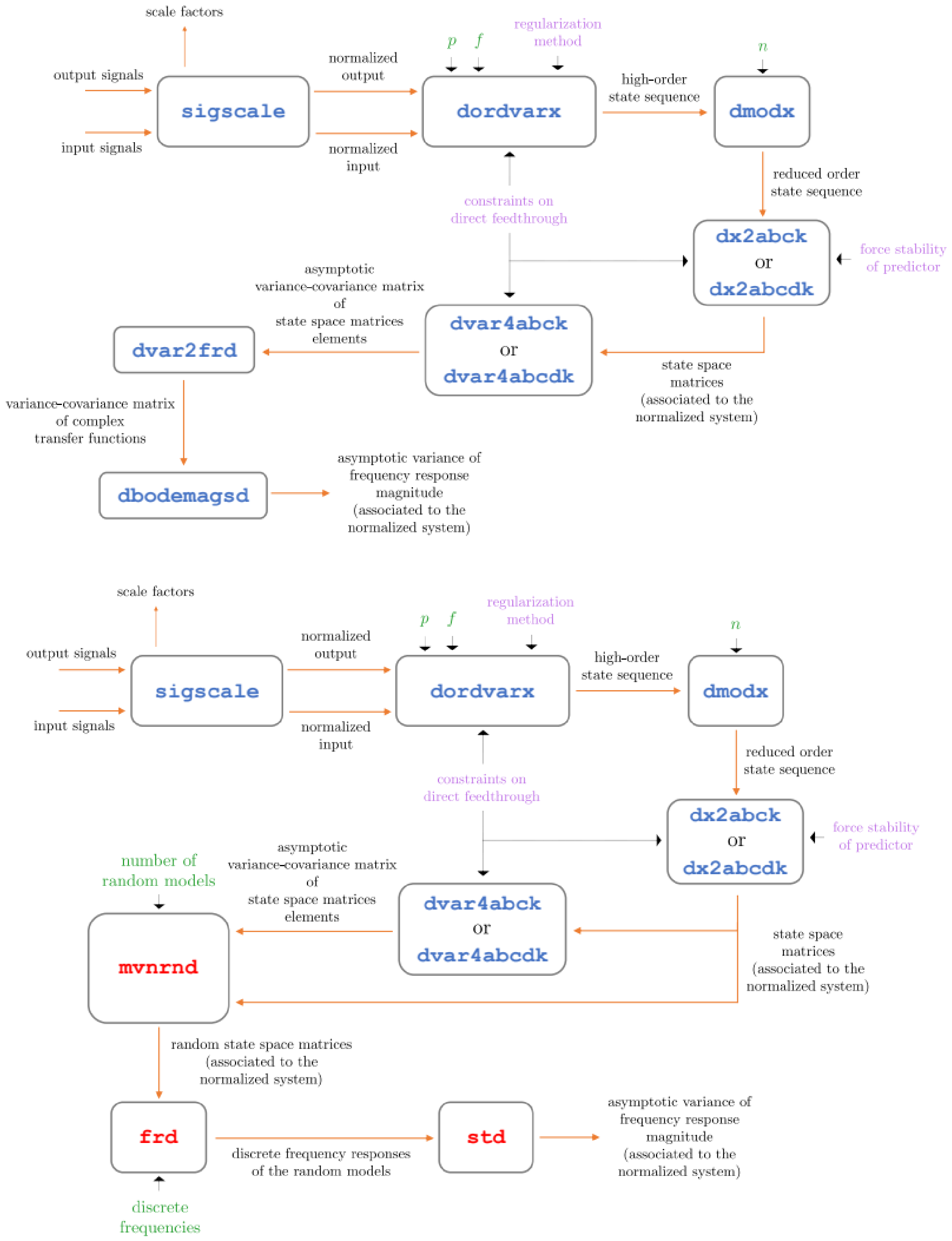
scale factors
output signals
input signals
sigscale
normalized output
normalized input
p
f
regularization method
dordvarx
high-order state sequence
n
dmodx
reduced order state sequence
constraints on direct feedthrough
dx2abck or dx2abcdk
force stability of predictor
asymptotic variance-covariance matrix of state space matrices elements
dvar2frd
dvar4abck or dvar4abcdk
state space matrices (associated to the normalized system)
variance-covariance matrix of complex transfer functions
dbodemagsd
asymptotic variance of frequency response magnitude (associated to the normalized system)
scale factors
output signals
input signals
sigscale
normalized output
normalized input
p
f
regularization method
dordvarx
high-order state sequence
n
dmodx
reduced order state sequence
constraints on direct feedthrough
dx2abck or dx2abcdk
force stability of predictor
number of random models
asymptotic variance-covariance matrix of state space matrices elements
mvnrnd
dvar4abck or dvar4abcdk
state space matrices (associated to the normalized system)
random state space matrices (associated to the normalized system)
frd
discrete frequency responses of the random models
std
asymptotic variance of frequency response magnitude (associated to the normalized system)
discrete frequencies

Figure 1.1: Two possible routes to implement PBSID in MATLAB, as adopted in [4]

The algorithm contained in the aforementioned function guarantees convergence to a local minimum, even if it does not converge to a global minimum. The preprocessor for `hinfstruct`, which is the function `systune`, allows to:

- Remove the stability constraint on the optimization scheme, which is useless during model matching,
- Focus the optimization on a given bandwidth.

1.3.3. Identification results

The identified parameters are collected in the following Tables, 1.1 to 1.4, respecting the order of the dynamics in the state space matrices.

Parameter	Value and unit
Y_v	-0.211 1/s
Y_p	0 m/(s rad)
L_v	-3.613 (s rad)/m
L_p	0 s
Y_δ	9.379 m/s^2
L_δ	851.287 rad/m^2

Table 1.1: Identified parameters for lateral dynamics

Parameter	Value and unit
X_u	-0.282 1/s
X_q	0 m/(s rad)
M_u	6.297 (s rad)/m
M_q	0 s
X_δ	-8.758 m/s^2
M_δ	473.061 rad/s^2

Table 1.2: Identified parameters for longitudinal dynamics

Parameter	Value and unit
N_r	-8.178 s
N_δ	255.590 m/s^2

Table 1.3: Identified parameters for directional dynamics

Tables 1.1 and 1.2 have the derivatives in p and in q, respectively, set to 0 according to how the results were presented in [4]. The reason behind the imposition of a null value in those positions is that the results were not deemed accurate enough for further analyses,

Parameter	Value and unit
Z_w	-0.731 1/s
Z_δ	-34.35 m/s^2

Table 1.4: Identified parameters for vertical dynamics

especially X_q and M_q, as their standard deviations from the nominal values are reported to be 96.3% and 111.6%, respectively.

The newly introduced constraint allowed the author of [4] to search for more accurate solutions during optimization, on the other hand, for the purposes of this thesis, it does not allow to work with a complete set of identification results. This shortcoming should be addressed, should further study on this subject be started.

1.4. Manipulation of the results

The objective of the present study is to estimate the time delay (given that the UAV is a MIMO system, delays is the more accurate form) affecting the discrete-time model of the UAV previously described in Section 1.1.

The first major issue encountered, even more pressing than choosing the methods that shall be exploited to identify the time delays, is that the final result of the PBSID-H$_\infty$ identification is expressed in continuous time.

Working around this issue is fairly straight-forward using MATLAB®, specifically its *Control System Toolbox* and *System Identification Toolbox*.

The former toolbox provides the function `ss`, which yields a state space model by receiving the matrices A, B, C, D as the first four inputs. The function can yield either continuous- or discrete-time models, depending on whether a sample time is given as fifth input: if only four inputs are given or the fifth is null, then the resulting model is a continuous-time one.

The results presented in Section 1.3.3 are the non-null components of the identified A, B, C, D matrices, and they are set into the matrices according to their description given in Section 1.2.2. As previously explained the identified matrices have been forced, through model matching, to have the same shape as those of the model described in Section 1.2.2.

The function `ss` is used to produce a continuous-time state space model by using such matrices.

The *System Identification Toolbox* provides the function `c2d`, thouroughly described in [9], which is used to convert a continuous-time model into a discrete-time one with specified sample time.

Six methods to perform the conversion are available:

- Zero-Order Hold (ZOH). It provides an exact match between the continuous- and discrete-time systems in the time domain for staircase inputs. The relation between continuous- and discrete-time input signals is as follows:

$$u(t) = u[k] \qquad kT_s \leq t(k+1) \leq T_s$$

- First-Order Hold (FOH). This method matches exactly the continuous- and discrete-time systems in the time domain for piecewise linear inputs. For this method, the relation between continuous- and discrete-time input signals is presented just below:

$$u(t) = u[k] + \frac{t - kT_s}{T_s}(u[k+1] - u[k]), \qquad kT_s \leq t(k+1) \leq T_s$$

  The MATLAB® documentation states that this second method is more accurate than ZOH for smooth inputs.

- Impulse-Invariant Mapping produces a discrete-time model with the same impulse response as that of the continuous-time system.

- Tustin Approximation. It operates in the frequency domain, as opposed to ZOH and FOH. What follows is the discretization $H_d(z)$ of the continuous-time transfer function $H(s)$ in `c2d`:

$$H_d(z) = H(s'), \qquad s' = \frac{2(z-1)}{T_s(z+1)} \tag{1.14}$$

  It is very important to understand that the states are not preserved during this conversion, they depend on state-space matrices and on the presence of time delay.

  Estimation of time delay on inputs is still possible.

- Zero-Pole Matching Equivalents. Unsuitable for MIMO systems, thus it cannot be used on the model at hand, which is indeed MIMO.

- Least Squares. As the one above, the least squares method is unsuitable for MIMO systems.

To keep the highest level of generality and not be forced to switch from one conversion method to the other according to the inputs at hand, Tustin's approximation has been preliminary chosen to convert the identified continuous-time state space model into its discrete-time counterpart.

The manipulated results of the identification process will be used in the subsequent chapters.

The manipulated identified model, henceforth simply called "the model", will be split into its different dynamics (the longitudinal, the later, the directional and the vertical dynamics), then input sequences suitable for each of the methods chosen in the next chapter will be applied to the model to generate output and state time sequences. Finally, the time sequences will be used to study the chosen methods.

Knowledge of the model matrices (or of the transfer functions) might be necessary, depending on the method chosen, but this is the subject explored in the next chapter, dedicated to describing some time-delay estimation methods and choosing them.

2 | Time-delay estimation methods

The time delay estimation can be performed by a plethora of methods, greatly varying in domain, data needed, range of systems to which the methods can be applied.

Svante Björklund compiled a list of time delay estimation methods in 2003, see [1], which was used as groundwork for the study presented in this thesis.

In [1] the methods are divided in four major categories:

- Time-delay approximation methods, comprising those methods where the time-delay is not an explicit parameter of the model; this category is further divided into three subcategories, according to domain of the respective methods:
  - Time domain approximation methods, usually based around finding when the impulse response becomes non-zero. This subcategory contains all those methods related to finding the maximum cross-correlation between input and output.
  - Frequency domain approximation methods, in the frequency domain a time-delay becomes a phase shift, thus these methods revolve around estimating phase shifts like $e^{-i\omega T_d}$.
  - Laguerre domain approximation methods, these methods estimate the time-delay from a relation, expressed in Laguerre functions, between input and output.
- Explicit time-delay parameter methods, this category is subdivided in three parts:
  - One-step explicit methods, which estimate time delay and all the other parameters of the model simultaneously, these methods are divided in those that work in continuous time and those that are used in discrete time.
  - Two-step explicit methods, which alternate between estimating the time delay and estimating all the other parameters.
  - Sampling methods

- Area and moment methods, relations between areas above (or below) the step response or moments of the impulse response form the core of these methods, they do require knowledge or, at least, an estime of these responses to be applied.
- Higher order statistics methods, these methods allow to eliminate noises with symmetric probability density function. Their main issue is that, if the signal has a symmetric probability density as welll, it might disappear.

Deeper analyses of these methods are presented in other documents by the same author: in [2] the time domain approximation methods are further explained and examined, and in [3] a more thorough comparison of the three one-step explicit methods is carried out.

Another method, related to the time domain approximation methods from [1], is presented in [10]. This method, suitable for discrete time models, compares the time differences of the time sequences of two different channels collecting data, it is different from the traditional cross-correlation methods in that the latter use as data the magnitudes values of time functions, while the method from [10] uses the time differences.

A third method, quite different from all those already presented so far is illustrated in [12], it is applied to discrete time state space models and it can be applied as is to MIMO systems such as the one this thesis aims to examine.

This method is based on the minimization of a quadratic cost function related to the prediction error of the state variables over a given time frame. It is needed that the system inputs are known and that noisy measurements of the states are available. No measurements of the outputs are needed, thus actually only a part of a state space model must be known, the part that revolves around the matrices A and B. The following sections, from Section 2.1 to Section 2.4, will explor in greater detail the theoretical foundations of these four methods, while Section 2.5 will explain the reasons behind the choice of the aforementioned quartet of methods.

2.1. Criterion functions

According to [10], this method is useful to determine the time delay between the detection of a measurable quantity by one sensor and the detection of the same quantity by another sensor.

"Any criterion function that is able to utilise that statistical similarity between the channel signals and to fix the time elements corresponding each other in sequences t and T" (quoted from [10]) is a valid criterion.

The two sequences are, respectively, the input and the output, or the first and the second measurements.

The two measurements must have a significant statistical similarity, that is, correlation, and each criterion function yields the best estimate for the time delay only "within the meaning of that criterion" (again, quoting [10]).

All criteria have a handful of common characteristics as foundational elements:

- two time sequences:
    - t, the input time sequence,
    - T, the output time sequence
- the two sequences compose the $\Delta(i+k,\, i) = T(i+k) - t(i)$ difference,
- the brackets $\langle\ \rangle$ indicate taking the sample mean of whatever quantity is inside,
- the difference $\Delta T(i+k) = T(i+k) - T(i+k-1)$ and its counterpart for t, $\Delta t(i) = t(i) - t(i-1)$,
- $v\hat{a}r\{\Delta T\} = \langle \Delta T^2(i+k)\rangle - \langle \Delta T(i+k)\rangle^2$

Specifically, in [10] four different criterion functions are illustrated, in order of appearance:

- minimum value of the sample variance delay function, Equation 2.1,
- minimum value of the sample average delay difference function, Equation 2.2,
- minimum value of the sample variance delay difference function, Equation 2.3,
- the maximum value of the correlation coefficient, Equation 2.4. This last criterion is equivalent to the maximisation of the cross-correlation function and, in stationary sequences, to the minimization of the quadratic delay difference function, shown in Equation 2.5.

The time index k that minimizes (or maximizes, in the case of the fourth function) the criterion functions is the estimate for the delay value.

$$\min_k\{\hat{e}_{vd}(k)\} = \min_k\{\langle \Delta^2(i+k,\, i)\rangle - \langle \Delta(i+k,\, i)\rangle^2\} \tag{2.1}$$

$$\min_k\{\hat{e}_{add}(k)\} = \min_k\{\langle |\Delta(i+k,\, i) - \Delta(i+k-1,\, i-1)|\rangle\} \tag{2.2}$$

$$\min_k\{\hat{e}_{add}(k)\} = \min_k\{E - F\} \tag{2.3}$$

with:

$$E = \langle|\Delta(i+k,\, i) - \Delta(i+k-1,\, i-1)|^2\rangle$$

$$F = \langle|\Delta(i+k,\, i) - \Delta(i+k-1,\, i-1)|\rangle^2$$

$$\max_k\{\hat{e}_{cor}(k)\} = \max_k\left\{\frac{\langle\Delta T(i+k)\Delta t(i)\rangle - \langle\Delta T^2(i+k)\rangle\langle\Delta t(i)\rangle}{\sqrt{v\hat{a}r\{\Delta T\}v\hat{a}r\{\Delta t\}}}\right\} \tag{2.4}$$

$$min_k\{\hat{e}_{qdd}(k)\} = min_k\{\langle|\Delta T(i+k) - \Delta t(i)|^2\rangle\} \tag{2.5}$$

2.2. Thresholding methods

This class of methods, presented in [1] and elaborated on in [2], is a type of cross-correlation methods. It consists in estimating the impulse or step response of a system (with or without noise) and comparing it to a threshold, so that the time from t_0, the time when sampling begins, to the moment when the chosen response oversteps the threshold is the estimate for the time delay.

The adopted method uses the impulse response, estimated using the MATLAB® *System Identification Toolbox* `arx` function, from [8], and the coefficients suggested in [2].

The chosen threshold, as suggested in [2] and presented in Equation 2.6, is $h(t)$:

$$h(t) = h_{std}\hat{y}_{std}(t) \tag{2.6}$$

where $\hat{y}_{std}(t)$ "is the vector of the estimated standard deviations of the estimated impulse response coefficients and h_{std} is a constant chosen by the user", to quote [2].

Given that the MATLAB® functions involved in the methods only accept SISO-derived input and output time sequences, a MIMO transfer function must be examined component by component, yielding a matrix of estimated time delay values, this concept will be further explored in Chapter 3

The MIMO version of this method can be structured to work by using either a polynomial transfer function representation or a state space representation, with no need to switch from one to the other, ensuring greater flexibility in its input arguments.

2.3. One-step explicit methods

[1] and [3] describe three methods in this class: `arxstruc`, `oestruc` and `met1struc`. A common characteristic is casting the system, of which the delay must be estimated, into a polynomial form, simulating this form for a set of delay values and choosing the value associated with the polynomial form with the lowest loss function.

`arxstruc` estimates many Autoregressive models with Exogenous variable (ARX), this variable being the input. All these models have the structure presented in the following Equation 2.7:

$$A(q)y(t) = B(q)u(t - n_k) + e(t) \tag{2.7}$$

and they differ from one to the other only in the value of the time delay n_k.

The MATLAB® function `arx` produces a structure containing a large amount of data, including loss function and fit percentage for the estimated ARX model, which are used to identify the best estimate out of the set of produced estimates and how fitting is the estimate to the original system.

In [3] an analysis is carried out to identify the best combination of degrees for the A and B polynomials, the results reported in the fourth chapter state that $n_a = 10$ as degree for $A(q)$, $n_b = 3$ as degree for $B(q)$ is the best combination, as it is the combination with the lowest root mean square (RMS) error, although the author subsequently adopted $n_a = 10$, $n_b = 5$ as values because it led to better results in two tests.

The same framework is the basis for `oestruc`, the striking difference being that the model(s) have a different form, an output-error (OE) model is estimated instead of and ARX one.

A generic OE model shape is illustrated below, in Equation 2.8:

$$y(t) = \frac{B(q)}{F(q)}u(t) + e(t) \tag{2.8}$$

All OE models created for `oestruc` have the same values for the two degrees while the third value, the one related to time delay, is different from model to model.

The values for the degrees of the polynomials $F(q)$ and $B(q)$, n_f and n_b, respectively, suggested in [3] have been chosen following the same logic as those for `arxstruc`: those that yield the lowest RMS error. These are $n_f = 2$ and $n_b = 1$.

In both `arxstruc` and `oestruc` there has been no prewhitening, because the RMS error has been observed to increase with prewhitening compared to without, while keeping the optimal orders in both cases.

2.4. Least-squares minimization of a cost function

This method is presented in [12], it was chosen because it did not rely on input and output sequences, but on input and states sequences to compute a cost function. The method also requires the system to be set in state space representation, it cannot work by directly operating with a polynomial transfer function representation of the system.

To create the cost function, the input and state time sequences are needed. The output sequence, although usually available (especially as it is needed to perform black-box estimation, such as with the PBSID method adopted in [4]) is not needed to estimate time delay using this method.

Admissible values of time-delay τ_j on the j-th input are collected in a set $\{\tau_{j,max}, ..., \tau_{j,min}\}$ ordered in sequence going from the greatest admissible delay value to the lowest admissible delay value.

The data making up the input sequence must be collected starting from $t \leq -\tau_{max}$, so that the earliest necessary element of the input sequence is $u(t = -\tau_{max})$, differently from the other methods that employ $u(t = 0)$ as earliest element of the input time sequence.

Thus, by collecting input data at every sampling moment, the input time sequence $\boldsymbol{u}(t)$ with $t = -\tau_{max}, -(\tau_{max} - 1), ..., 0, 1, 2, ..., N$ is created. $u_j(t)$ indicates the j-th input variable at time t.

This method relies on the states being measurable. If the states are measurable, then they must be measured at the same moments and rates as the inputs and the state sequence is created using $x(t = 0)$ as its first element and then the other elements are $\boldsymbol{x}(t)$ with $t = 1, 2, ..., N$

Other data needed are the matrices A and B of the state space representation.

Given p inputs, k as the discrete time variable, $k = N$ as the last sampling moment, the following equation is the cost function. The value of $\hat{\tau}_j$ that minimizes it is the optimal estimate of time delay for the j-th input.

$$J(\hat{\tau}_1, \hat{\tau}_2, ..., \hat{\tau}_p) = \sum_{k=0}^{N-1} [x(k+1) - \hat{x}(k+1)]^T [x(k+1) - \hat{x}(k+1)] \tag{2.9}$$

Equation 2.9 is subject to:

$$\hat{x}(k+1) = A\hat{x}(k) + \sum_{j=1}^{p} B_j u_j(k - \hat{\tau}_j) \qquad (k = 1, ..., N-1) \tag{2.10}$$

with B_j being the column of state space matrix B related to the j-th input.

This last equation, Equation 2.10, is the estimated form of the following Equation 2.11, which describes a discrete time state space system with input delays.

$$x(k+1) = Ax(k) + \sum_{j=1}^{p} (B_j u_j(k - \tau_j)) \qquad (k = 1, 2, ..., N-1) \tag{2.11}$$

In order to reduce the workload, a new quantity, γ, is introduced, and it is the set of $\gamma_{i,j}$ presented in Equation 2.12:

$$\gamma_{i,j} := \begin{cases} 1, & t_{i,j} = \tau_j \\ 0 & \text{otherwise} \end{cases} \tag{2.12}$$

In Equation 2.12, $t_{i,j}$ is the i-th value of admissible time delay on the j-th input. All t_{ij} with different index i but same j compose the set $\{t_{i,j},\ i = 1, 2, ..., M_j\}$, which is another form of the admissible delay values set, with M_j representing the quantity of admissible delay values for the j-th input.

The quantity $t_{i,j}$ respects the relationship outlined by Equation 2.13:

$$\tau_j = \prod_{i=1}^{M_j} \gamma_{i,j} t_{i,j} \tag{2.13}$$

So the state variables propagation is rewritten as Equation 2.14:

$$x(k+1) = Ax(k) + \sum_{j=1}^{p} \sum_{i=1}^{M_j} \gamma_{i,j} B_j u_j(k - t_{i,j}) \qquad (k = 1, ..., N-1) \tag{2.14}$$

The cost function variable becomes the estimate of γ and is rewritten using the new quantities of Equation 2.15 and Equation 2.16:

$$z(k) := x(k+1) - Ax(k) \tag{2.15}$$

$$\begin{aligned}\Phi(k) = [\ & B_1 u_1(k - t_{1,1}) \quad B_1 u_1(k - t_{2,1}) \quad \dots \quad B_1 u_1(k - t_{M_1,1}) \quad \dots \\ & B_2 u_2(k - t_{1,2}) \quad B_2 u_2(k - t_{2,2}) \quad \dots \quad B_p u_p(k - t_{M_p,p})\ ]\end{aligned} \tag{2.16}$$

Yielding Equation 2.17

$$J(\hat{\gamma}) = \sum_{k=0}^{N-1} [z(k) - \Phi(k)\hat{\gamma}]^T [z(k) - \Phi(k)\hat{\gamma}] \tag{2.17}$$

The minimum of $J(\hat{\gamma})$ can finally be recovered from the imposition of $\nabla J(\hat{\gamma}) = 0$, which is related to Equation 2.18 for what concerns finding the optimal estimate of γ:

$$\hat{\gamma}^* = \left(\sum_{k=0}^{N-1} \Phi^T(k)\Phi(k)\right)^{-1} \left(\sum_{k=0}^{N-1} \Phi^T(k) z(k)\right) \tag{2.18}$$

And thus the estimated tau can be computed as illustrated by Equation 2.19:

$$\hat{\tau}_j^* = \prod_{i=1}^{M_j} \hat{\gamma}_{i,j}^* t_{i,j} \qquad (j = 1, 2, \dots, p) \tag{2.19}$$

One important condition for this method to work is that the input sequence must be “sufficiently rich so that the matrix inverse exists”, implying that a step or, even worse, an impulse input are usually not suitable for this method.

The value of γ does not depend on N, as long as N allows the matrix inverted in Equation 2.18 to actually be invertible.

2.5. Choice of methods

The methods to test and compare at our disposal have initially been reduced to four methods, the rationale being having representatives of different categories of methods to compare to each other.

Chosen methods are those presented in [10] and [12], plus two shown in [1]: `oestruc`, a one-step explicit method, and simple thresholding on both impulse and step response, two variants of a time domain approximation method.

The last method has been chosen because it is, conceptually, the simplest of all, thus testing would reveal if the difference in theoretical concepts underlying each method translated into a performance difference upon application on various models, both SISO and MIMO, chief among them the model of a four-rotor drone that has been described previously. The second to last model has been chosen because it comes from the second best described category of methods in [1] and, given that the four-rotor drone model has already been identified, there is no need to have a more complex identification section of the model, given that the focus of this thesis is time-delay estimation.

Of the three methods composing the one-step explicit time-delay parameter category, `oestruc` is the slowest, from a computational time point of view, but the most reliable, this is the reason why it has been chosen over `arxstruc` and `met1struc`.

The method illustrated in [12] and Section 2.4 has been chosen due to its quirks, chiefly needing knowledge about the states time sequence instead of the output time sequences and the altogether very different approach to the problem, compared to the other methods chosen.

The criteria proposed in [10] and shown in Section 2.1 cannot actually be applied to the estimation of time delay of a system such as that illustrated in Chapter 1 because the input and output vectors have different dimensions and units of measure, which means they are incompatible and cannot be compared to each other, which is instead what should happen between two time sequences with in the criteria from Section 2.1.

3 | Adaptation of methods to MIMO models in MATLAB

This chapter is dedicated to integrating the methods chosen at the end of Chapter 2 into the MATLAB® environment, starting from what is suggested in the sources and then developing the MATLAB® algorithm into versions suited for discrete-time MIMO systems and with a sample time that might be different from 1 second.

The real system was not available to measure the necessary data, such as outputs or states, thus the identified parameters have been used to build a state space system, its shape being the same as that described in Chapter 1. This system has been used either to directly obtain outputs and states through the MATLAB® *System Identification Toolbox* function `lsim`, or to be converted into the corresponding transfer function $G(s)$.

$G(s)$ has never been used as-is, it has always been used split into the sub-transfer functions relating a single input signal to a single output signal.

3.1. Thresholding algorithm

This method, as introduced in Chapter 2, cannot directly operate with MIMO systems: if a MIMO system is directly used as-is in the estimation the only delays that can be estimated are the shortest for each output, all other delays contributing to each output might not be detectable. Thus, whether a state space or a polynomial transfer function is used, each input-output combination must be analyzed on its own, so that the time delay specific to each combination can be estimated.

The method adapted for MIMO systems still follows, for each input-output couple, the original method presented in [1] and [2], meaning that the impulse response of each input-output combination is estimated using the `arx` command from the MATLAB® `System Identification Toolbox`, as computed by Algorithm 3.1.

A `y_est` is the set of output time sequences collected as a matrix, its first index is the

Algorithm 3.1 Impulse response estimation using `arx`

```
data = iddata(y_est(:,idx_y),u_est(:,idx_u),ts);
arxsys = arx(data,[0 71 0]);
imp_resp(:,idx_y,idx_u) = sim(arxsys,[1;zeros(length(t)-1,1)]);
```

time index, indicating when the value was sampled.

The same is true for the input counterpart of `y_est`: `u_est`. The `idx_u` and `idx_y` indices are integers, respectively indicating the `idx_u`-th input and the `idx_y`-th output time sequences used to estimate the impulse response.

The function `iddata` combines an input sequence, an output sequence and a value of time delay to create a dataset that will be fed to the function `arx`.

The value `ts` is the sample time used to estimate the discrete-time impulse response, it must be the same as the sample time used to define the space state system or polynomial transfer function (either can be input in the position indicated by sys) from the `idx_u`-th input to the `idx_y`-th output.

The vector `[0 71 0]` used in the function `arx` has been derived from suggestions present in [2].

`u_est` and `y_est` are an input sequence and an output sequence, respectively, generated for the sole purpose of estimating the impulse response of the system through the `arx` command. The output sequence is generated by exciting, with the `u_est` input sequence, the subsystem related to the the `idx_u`-th input and the `idx_y`-th output combination.

The whole algorithm is then cycled over all values of `idx_y` and of `idx_u`, leading to an impulse response array (indicated as `imp_resp(idx_t,idx_y,idx_u)`) with three indices:

- The first is the time index, indicating the time when the impulse response value has been sampled. The value of this index is related to the corresponding sample time instant throught to the formula $t = (\text{timeindex} - 1)t_{sample}$, where t indicates the sample time instant, in the same units as the sample time.
- The second is the output index, its value indicates which output the corresponding impulse response value is related to.
- The third is the input index, indicating which input generated the impulse response value.

`arxsys` contains information about the variance of the ARX systems coefficients, which is needed to build the threshold, as explained in [2]. The threshold varies over time because at each time instant the standard deviation is different thus, for each input-output couple a 71-components vector of time-varying standard deviations is generated as:

```
y_std = sqrt(diag(arxsys.Report.Parameters.FreeParCovariance));
```

The threshold time sequence for each input-output combination is then created as:

```
h_t(:,idx_y,idx_u) = y_std.*h_std_imp;
```

The last factor is a dimensionless number used to increase or decrease the threshold as needed, it might be time-varying or fixed, [2] states that a value of 3 for this factor is suitable for most estimation procedures.

In some cases, chiefly when the system is simply stable, the algorithm to compute the impulse response might yield unsatisfactory results in either the response itself or in the standard deviation. To solve this issue, for each input-output combination and for the same time interval, various impulse responses are estimated:

Algorithm 3.2 Averaged impulse response and threshold

```
for idx_1=1:idx_1_max

    data= iddata(y_est(:,idx_y,idx_1),u_est(:,idx_u,idx_1),ts);

    arxsys=arx(data,[0 71 0]);

    imp_resp2(:,idx_y,idx_u,idx_1)=...
        sim(arxsys, [1;zeros(length(t)-1,1)]);

    y_std3(:,idx_y,idx_u,idx_1)=...
        sqrt(diag(arxsys.Report.Parameters.FreeParCovariance));

end

y_std(:,idx_y,idx_u) = mean(y_std3(:,idx_y,idx_u,:),4);
imp_resp(:,idx_y,idx_u) = mean(imp_resp2(:,idx_y,idx_u,:),4);
h_t(:,idx_y,idx_u) = (y_std(1:71,idx_y,idx_u)).*h_std_imp;
```

A new index (`idx_1`) had to be added to create the new `for` cycle, in turn needed to estimate many (whatever value for `idx_1_max` is chosen) impulse responses for a set combination of input and output over a defined time interval, thus producing the following lines from Algorithm 3.2:

```
    imp_resp2(:,idx_y,idx_u,idx_1) = sim(arxsys,[1;zeros(length(t)-1,1)]);
    y_std3(:,idx_y,idx_u,idx_1) =...
        sqrt(diag(arxsys.Report.Parameters.FreeParCovariance));
```

that compute `idx_1`-th impulse response and standard deviation related to the `idx_y`-th output and `idx_u`-th input couple. Having computed an `idx_1_max` amount of impulse responses, all the resulting `imp_resp2` are then averaged over the fourth index to obtain `imp_resp(:,idx_y,idx_u)`, this will be used as illustrated in the initial Algorithm 3.1.

The same is performed to obtain the standard deviation over time, an auxiliary vector `y_std3(:,idx_y,idx_u,idx_1)` is built for each estimation of the impulse response, its fourth index the same as that in `imp_resp2`, and again paralleling what happens to `imp_resp2`, once the `for` cycle over the fourth index is over, the results are all averaged over that index. These two procedures are listed below in MATLAB® code and, in fact, are the third and second to last lines of Algorithm 3.2:

```
imp_resp(:,idx_y,idx_u) = mean(imp_resp2(:,idx_y,idx_u,:),4);
y_std(:,idx_y,idx_u) = mean(y_std3(:,idx_y,idx_u,:),4);
```

The vector `y_std` is then used to build the thresholding sequence by inserting it into Algorithm 3.1.

To estimate delay, the estimated impulse responses for each input-output combination are compared to the threshold sequences for the same combinations, the process is straightforward: a `for` cycle is started on the first value of the time index and stopped whenever the impulse response is first detected overstepping the threshold. The time index is then manipulated to obtain the true time instant. The resulting estimate is placed into a matrix `(tau_imp(idx_y,idx_u))` where the row index indicates the output the delay is related to, while the column index indicates the input. The following algorithm shows how the delay for one input-output combination is estimated: Everything presented so

Algorithm 3.3 Identification of time delay for one input-output couple

```
for idx_t = 1:ts:71
    if abs(imp_resp(idx_t,idx_y,idx_u)) > h_t(idx_t,idx_y,idx_u)
        tau_imp(idx_y,idx_u) = (idx_t-1).*ts;
        break
    else
        tau_imp(idx_y,idx_u) = -1;
    end
end
```

far must be cycled over the output (`idx_y`) and input (`idx_u`) indices to fill in the matrix `tau_imp`.

If the threshold is never exceeded then -1 is returned as delay estimate, given that -1 can never be a delay, the user can easily understand that, for the `idx_y`-th output, `idx_u`-th input combination, the impulse response was always lower than the threshold.

A relevant issue of this method, when applied to discrete-time systems, is that the delay estimate `tau_imp(idx_y,idx_u)` includes the difference between the denominator and numerator degrees of the polynomial transfer function between the `idx_y`-th output and the `idx_u`-th input. This polynomial transfer function, usually identified in the scripts as `G(idx_y,idx_u)`, is a component of the polynomial transfer function G, describing the greater MIMO system being analyzed.

An example of this issue is a stable system with degree 1 numerator, degree 3 denominator and input delay of 3 times the sample time: even if the method correctly estimated the input delay, it would yield a result of $\tau = 5t_{sample}$ instead of $\tau = 3t_{sample}$ because $\Delta(\text{degree}) = 2$.

3.2. `oestruc` algorithm

The `oestruc` method shares one important similarity with the chosen thresholding method: both have MATLAB® estimate a model through built-in functions that take as input arguments the input and output time sequences of a simulation of the system. This similarity leads to `oestruc` needing an adaptation for it to be applicable to MIMO systems, such as the one studied in this document. The similarity ends here because `oestruc` needs the built-in `oe` MATLAB function to estimate an output-error model of the original system, whereas the thresholding method uses the built-in `arx` function to estimate the impulse response of the system.

Another similarity is the issue presented at the end of Section 3.1, related to the degrees of the denominator and numerator polynomials.

An initial MATLAB® code for SISO `oestruc` is presented in [1] and is adopted as framework to support the construction of the variable sample time MIMO version of `oestruc`:

`data` is a collection of input and output sequences, `nf`, `nb` and `nkVec` are the coefficients of the output error model, and only one component of the last coefficient is taken at every iteration of the method. As previously illustrated in Section 2.3, the third coefficient

Algorithm 3.4 Basic oestruc

```
function dtEst = oestructd(data,nkVec_max)

nf = 2;
nb = 1;
nkVec = 1:nkVec_max;
for nnn = 1:length(nkVec)
    model = oe(data,[nb nf nkVec(nnn)], 'Covariance','None');
    lossFunc(nnn) = model.NoiseVariance;
end
[minVal, nnnmin] = min(lossFunc);
dtEst = nkVec(nnnmin);
```

(n_k, or `nk` in MATLAB® code) is the one relaying information about the delay: its value corresponds to the number of sample time intervals of input delay affecting the system, e.g. `nk`= 3 means an input delay long 3 times the sample time.

In order to apply `oestruc` to MIMO systems, two `for` cycles, one nested into the other, are needed, one cycles over the number of outputs, the other over the number of inputs, just like it happens in the MATLAB® realization of the thresholding method.

The function will require:

- the input sequence u, it must be fed as column(s) vector,
- the output sequence y, it must be fed as column(s) vector as well,
- the transfer function G,
- the sample time `ts`.

The method must be usable with a flexible choice of sample time, while the original, from [2] forces the user to adopt a sample time of 1 s, to this end a few notable changes and additions have been made:

- The time sequence used to obtain the output sequence now depends on the sample time: `t_oe(1,:)  = 0:ts:(ts.*(length(u_oe)-1))`; it is a vector that starts from zero, is as long as the input sequence is and its i-th value is its $(i-1)$-th value plus the sample time,
- The best time delay estimate is multiplied by the sample time, recall that the delay estimate is not a time but rather an number of sampling intervals, the two only coincide if `ts = 1`.

The final MATLAB® form of the method is presented as Algorithm 3.5:

Algorithm 3.5 Extended `oestruc`

```
function tau_oe = oestructd(u_oe,y_oe,G,ts,nkVec_max)

t_oe(1,:) = 0:ts:(ts.*(length(u_oe)-1));

[n_y,n_u] = size(G);

nf = 2;
nb = 1;
nkVec = (0:nkVec_max);

tau_oe = zeros(n_y,n_u);

for idx_y = 1:n_y

    for idx_u = 1:n_u

        z = [y_oe(:,idx_y),u_oe(:,idx_u)];

        for nnn = 1:length(nkVec)
            nk = nkVec(nnn) ;
            model = oe(z,[nb nf nk], 'Covariance','None');
            lossFunc(nnn,:) = model.NoiseVariance;
        end

       [minVal, nnnmin] = min(lossFunc);
       tau_oe(idx_y,idx_u) = ts.*nkVec(nnnmin);
    end
end
```

In order to avoid manually transposing the input and output sequences, so that either row or column vectors can be used, the following lines of code (listed as Algorithm 3.6 and Algorithm 3.7) can be inserted in the function presented in Algorithm 3.5, just before the main body of the method; it is important to note that these additions only work if the number of inputs, or outputs (in Algorithm 3.7), is lower than the number of sampling moments:

A very similar procedure can be employed to obtain the suitable shape of the output matrix (or vector), as illustrated in Algorithm 3.7.

Algorithm 3.6 Adaptation to accept both row- and column-shaped inputs

```
[d1u, d2u]=size(u_oe);
if d1u<d2u
    u_oe = u_oe.';
else
    u_oe;
end
```

Algorithm 3.7 Adaptation to accept both row- and column-shaped outputs

```
[d1y, d2y]=size(y_oe);
if d1y<d2y
    y_oe = y_oe.';
else
    y_oe;
end
```

3.3. Least-squares time-delay ID algorithm

Transcribing this method in MATLAB® requires little modifications, and once transcribed it is capable of operating with SISO and MIMO systems. As stated in Section 2.4, the method requires:

- the state space form matrices A and B,
- rich (e.g. no impulse nor step) input time sequences $u_j(t)$ ranging from $t = -(\tau_{j,max})$, the maximum admissible delay value for the j-th input, to $t = N-1$, the penultimate sampled instant,
- the state time sequences from $t = 0$ to $t = N$.

The MATLAB® implementation of the method requires the admissible delay values to be set in a matrix T, each of its columns being the set of delay values for the corresponding j-th input.

For ease of composing the script all the columns of T, that is all the sets of admissible delays, must be the same; if the minimum admissible delay value for one input widely differs from the maximum admissible value on another input the columns of T will be quite long.

Given as a two input system defined in state space form through MATLAB®, a simple (two inputs) example of data initialization for this method is presented as Algorithm 3.8.

Algorithm 3.8 Example of initialization of data for `LS_noiseless`

```
[n_x, n_u] = size(sys.B);
t = t_start:ts:t_end;

for idx_u = 1:n_u
    T(:,idx_u) = (Tmin:Tmax).*ts;
end

u0 = zeros(n_u,1);
u(1,:) = sin(2.5.*(ts:ts:t_end).*0.5.*pi);
u(2,:) = sin(1.25.*0.5.*(ts:ts:t_end).*pi);
u_neg = zeros(n_u, max(T(:,1))./ts);
x0 = zeros(n_x,1);


u_mod1 =[u0, u];
u_mod =[u_neg, u_mod1];
```

`u_neg` is the matrix containing all input values from `t=-ts` to, going backwards in time, `t = -max(T(:,1))`, `n_u` is the number of inputs, `n_x` is the number of state variables, `ts` is the degree time, `max(T(:,1))./ts` is used to gather the index corresponding to the max(T(:,1)) degree time.

The states vector evolution over time should be that of the system being analysed, but in order to gather data without the actual system, here it is obtained by simulating the system through the MATLAB® function `lsim`, having also specified the input to use (the same that has been defined during the initialization), the time interval and the initial state vector, defined as `x0` during the initialization:

```
[y,t_out,x]=lsim(sys,u_mod1.',t,x0.');
```

Some inputs of `lsim` have been transposed to allow the function to accept the vectors and matrices. The only output of this function needed to the method is the third, the state vector `x` which includes the previously defined `x_0`. The method is fully implemented by the following MATLAB® function, accepting as inputs, in order as they appear on the function syntax:

- the state vector at $t = 0$,
- the state vector time history without the values at $t = 0$,

- the input time history from $t = -(\tau_{j,max})$ to $t = N$,
- the state space form matrices A and B,
- the matrix T containing the sets of the admissible time delay values for all inputs,
- the sample time t_s (written as `ts` in MATLAB® code)

Algorithm 3.9 Core of `LS_noiseless` algorithm

```
function [gamma,tau] = LS_noiseless(x0,x,u_mod,A,B,T,ts)

x_mod = [x0, x];


[n_u, n_t_u] = size(u_mod);
[n_x,n_t_x] = size(x_mod);
[L_T,n_T] = size(T);

for k = 1:n_t_x-1
    z(:,k) = x_mod(:,k+1) - A*x_mod(:,k);
    phi(:,:,k) = phi_k_creator(n_u,u_mod,L_T,T,B,k,ts);
    R(:,k) = (phi(:,:,k).') *z(:,k);
    W(:,:,k) = (phi(:,:,k).') *(phi(:,:,k));
end

R1 = sum(R,2);
W1 = sum(W,3);
gamma = (W1)\R1;

gamma_reshaped = reshape(gamma,L_T,n_u);

for idx = 1:n_u
    tau(:,idx) = max(gamma_reshaped(:,idx).*T(:,idx));
end
```

Here `k` is used as time index, just as in the method presented in [12].

The function `phi_k_creator(n_u,u_mod,L_T,T,B,k,ts)`, illustrated in Algorithm 3.10, yields the quantity $\Phi(k)$, described in 2.4 and in [12], for all inputs and admissible delay values.

The part `round((max(T(:,1)) -T(h,j))./ts))` has been used to have the function work even for sample times different from 1 second, it is analogous to what happens to `max(T(:,1))` during the data initialization: retrieving the time index corresponding to the time instant itself.

The matrices R and W from the script are related to two matrices presented in Section

Algorithm 3.10 `phi_k_creator` algorithm

```
function phi_k = phi_k_creator(n_u,u,dim_1_T,T,B,k,ts)

    phi_k=[];

    for j = 1:n_u % taking inputs one by one
        u_j = u(j,:);
        for h = 1:dim_1_T
            phi_column = B(:,j)*u_j(k+...
                round((max(T(:,1)) -T(h,j))./ts) );
            phi_k = [phi_k, phi_column];
        end
    end

end
```

2.4:

$$W = \left(\sum_{k=0}^{N-1} \Phi^T(k)\Phi(k)\right) \tag{3.1}$$

$$R = \left(\sum_{k=0}^{N-1} \Phi^T(k)z(k)\right) \tag{3.2}$$

The quantity `gamma`, corresponding to γ from 2.4 (where it is obtained as a solution of a system of equations) is initially obtained as a simple vector using the backslash (\) solver in MATLAB® because it is less computationally demanding than solving the system by inverting `W1` over `R1`.

This `gamma` vector has to be reworked into a matrix, with as many rows as admissible time delay values and with columns such that the j-th column is the vector γ_j related to the j-th input. Each of its columns should have all null components except one, this component being equal to 1 and corresponding to a component of the j-th column of T, the relevant set of admissible delay values; the value of that component is the estimated delay for the input.

One key **issue with the least-squares minimization** method is that it **only identifies the shortest time-delay**: if some inputs have a longer delay they cannot be detected and will go unnoticed.

The four identified dynamics, as presented in Chapter 1, are completely separable, meaning they can be analysed one by one. The main effect of the separability is that, for each method, each subsystem is of lower order than the complete, four-dynamics system, which leads to easier estimation by `oestruc` and by thresholding methods, given that the orders of the estimated OE and ARX models, are quite low as one may recall from Chapter 2.

4 | Analysis and comparison of methods

Before comparing results obtained in MATLAB® environment, some considerations have to be made about the expected results and behaviour of the methods:

- All dynamics have state space matrix D with at least one non-zero component, this leads to expecting absence of input delays on every dynamics, whether the analysed dynamics is unstable or stable and regardless of its order.
- Given that the thresholding method, as implemented in Chapter 3, and `oestruc` produce an estimate biased by the difference between the degrees of denominator and numerator of the transfer function under scrutiny, it is expected to see this bias in the delay estimate of, at least, the stable directional and vertical dynamics.
- The thresholding method, as explained in Chapter 3, might fail: the threshold might never be overstepped during the estimation, which would lead to -1 as estimated time delay.

If thresholding were to fail, the mean and standard deviation of its results would not take into account any failed attempt but they would be computed from the rest of the available data, albeit by using a reduced dataset. Furthermore, the times thresholding has failed have been be counted and will be presented as data, in order to show how the dataset size has been affected.

4.1. Simulation settings

4.1.1. Common settings

This section presents values and settings that have used across most tests.

Every simulation lasts 150 times the sample time, which has been set, initially, to $t_{sample} = 0.1$ s. Given the state space models under study, the correct input delay would be null, thus a lower sample time is not expected to carry any benefit, compared to $t_{sample} = 0.1$ s.

Every simulation is sampled from $t_0 = 0$, and that is the first input value that starts affecting the models is $u = u(t_0)$.

The states vector has been set to zero as starting value and then split into four parts, each related to a different dynamics:

$$x(t_0) = [x_{lat}(t_0) \quad x_{lon}(t_0) \quad x_{dir}(t_0) \quad x_{ver}(t_0)]^T$$

$$x_{lat}(t_0) = [0 \quad 0 \quad 0]$$

$$x_{lon}(t_0) = [0 \quad 0 \quad 0]$$

$$x_{dir}(t_0) = [0 \quad 0]$$

$$x_{ver}(t_0) = 0$$

Least-squares minimization and `oestruc` methods maximum admissible delay values have been set to $t = 35t_{sample}$.

Every possible combination of type of input sequence, time conversion method, time-delay estimation method and model dynamics has been run 10 times, thus giving a set of 10 values of identified time-delay for each combination.

All 10-values sets had their mean values and standard deviations computed using the MATLAB® functions `mean` and `std`, these form the core of results presented in Section 4.2.

4.1.2. Inputs

This brief section is dedicated to illustrate what types of inputs were applied to the models in order to study their delays.

Every dynamics is, at most, SIMO (Single Input Multiple Output) thus there has been no reason not to use the same inputs from the model of a dynamics to the model of another.

The first type of input is an increasing and then decreasing succession of two ramps, in total lasting $t = 40t_{sample}$, starting at $t = 0$, created by Algorithm 4.1. The rest of the simulation, from $t = 40t_{sample}$ to $t = 150t_{sample}$, the input is constantly null, which is why Figure 4.1 illustrates the input only up until $t = 5$ s (having set $t_{sample} = 0.1$ s).

The impulse with amplitude 0.01 has also been adopted as input during the tests, it is

Algorithm 4.1 First type of input

```
uaav=[];
for k=0:20
    u_aa=0+(k*0.0005);
    u_aav=[uaav,uaa];
end
u_ab=max(uaav);
for k=1:20
    uaa=uab-(k*0.0005);
    uaav=[uaav,uaa];
end
u=[uaav, zeros(1,length(t)-length(uaav))];
```

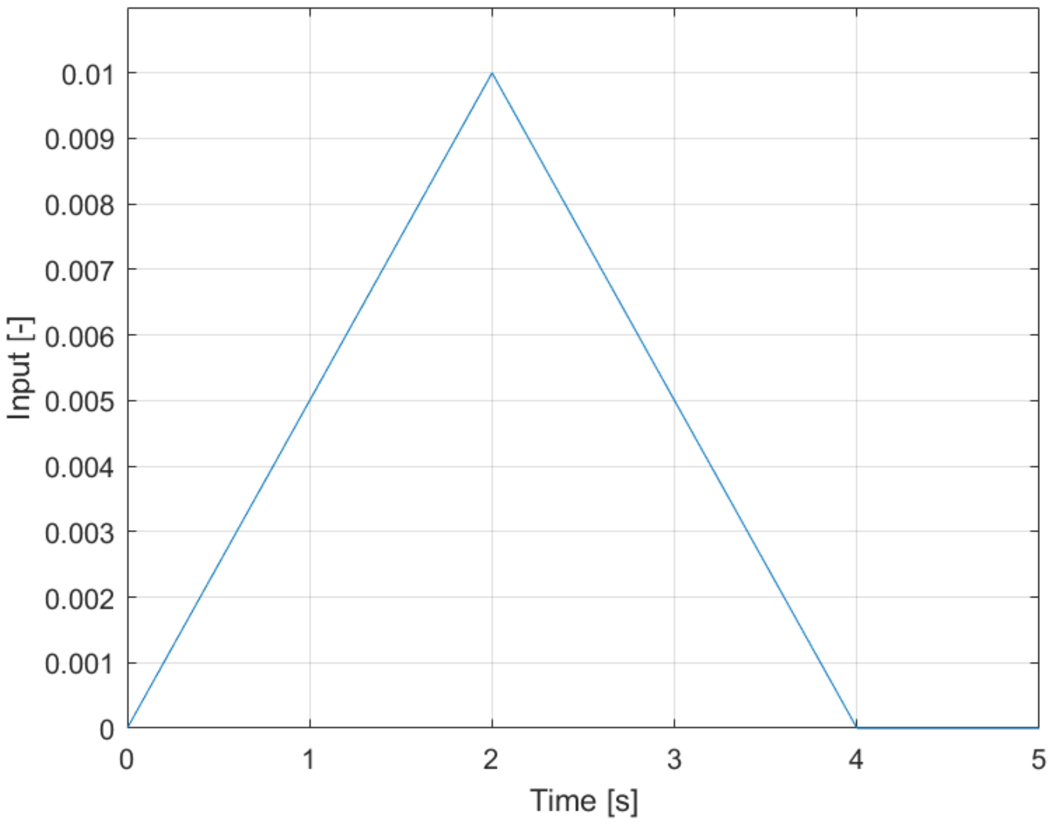

Figure 4.1: Two-ramps input, with $t_{sample} = 0.1$ s

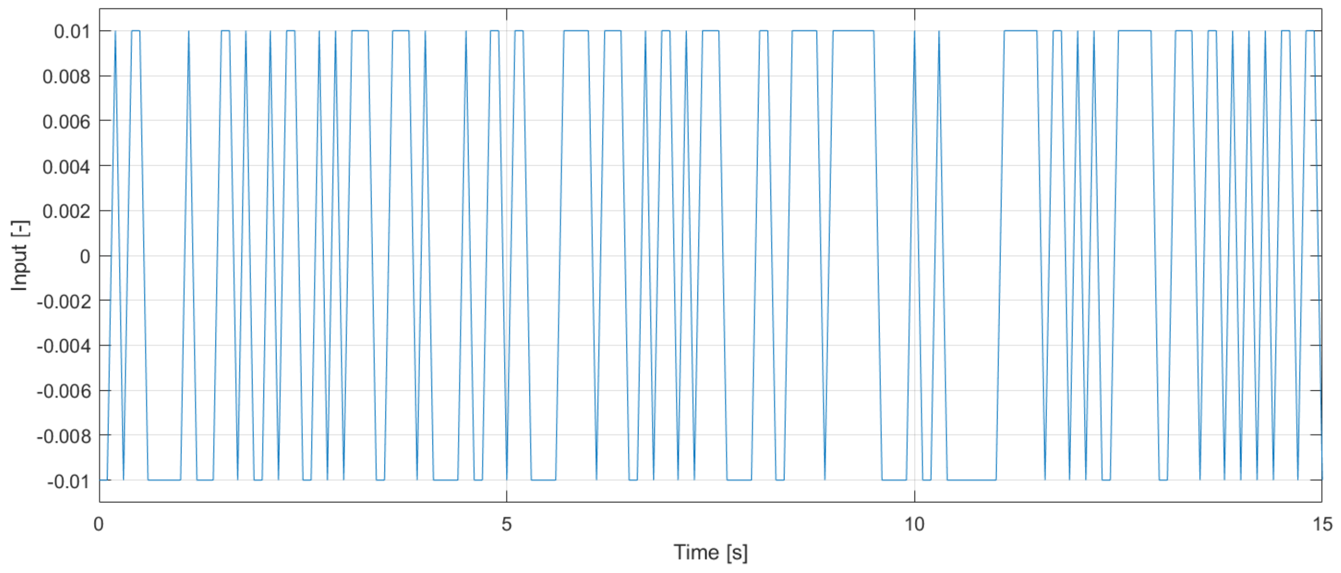

Figure 4.2: RBS input, with $t_{sample} = 0.1$ s

built such that:

$$u(t = 0) = 0.01$$

$$u(t < 0,\ t > 0) = 0$$

The last type of input is a Random Binary Signal (RBS), generated through the MATLAB® *System Identification Toolbox* function `idinput` as follows, one example of such kind of signal is depicted in Figure 4.2:

```
u=0.01.*idinput(length(t),'rbs').';
```

with `t` being the time sequence vector, defined as $t = [0,\ 0.1\ , 0.2,\ ...\ 15]$, and with 0.01 as amplitude, to keep it similar to the other two inputs.

The RBS has been chosen because it yields a "rich" input sequence while still being relatively simple: it is a signal that, at any time can assume one of two possible values and, by default in MATLAB®, these are 1 and -1, which are then scaled down to 0.01 and -0.01. What value appears at what time instant is the randomised part of the signal.

4.1.3. Settings of the methods

The results presented in Section 4.2 have been produced using the three chosen methods, a basic form of thresholding, `oestruc` and the least-squares minimization of a suitable cost function.

Each of the methods implies the conversion of a continuous-time model to discrete-time, the model being that of the dynamics under analysis, as reported in Section 1.4, and for completeness's sake the tests performed not only compared the three time-delay identification methods but, for each of these, also compared the *Tustin*, *First-Order Hold* and *Impulse-Invariant Mapping* methods of continuous- to discrete-time conversion.

The time-delay identification methods have also been used in a comparison between the unstable and stable forms of the lateral and longitudinal dynamics.

Stabilizing the two models has been performed after the conversion from continuous- to discrete-time in order to better create a simulation environment as close as possible to one that does not rely on that conversion: a simulation environment working with discrete-time systems that do not derive from such convsersion.

The stabilization has been achieved by using the MATLAB® *Control System Toolbox* function `dlqr`, which creates the optimal gain matrix K of a discrete-time Linear Quadratic Regulator, given the state space matrices A and B and the cost weighing matrices Q (for the states) and R (for the inputs, do not mistake it for the R matrix of Section 3.3).

The rest of the outputs of `dlqr` have not been used and so has not the option to declare a cross term matrix N as input for the function.

The weighing matrices for `dlqr` have been set with unitary weights because they did not seem to matter for the model at hand.

The thresholding method has been adopted in its simplest form, without the internal averaging.

4.2. Simulation results

In this section, results will be presented and commented, each subsection will deal with one type of input presented in Section 4.1.2, in the order they have been presented there.

All units of measurement in the following tables, unless specified otherwise, are seconds.

`oestruc` and thresholding methods yield a vector of time-delay estimates, each component corresponding to a different output, they will be presented in that form.

4.2.1. Results from two-ramps input

Table 4.1 shows the mean values and standard deviations of the lateral dynamics according to what method has been used to perform the time conversion and to whether the system has been stabilized or not. It is divided into three parts, one for each time-delay identification method.

Time conversion method	**Type**	**Value, stable system**	**Value, unstable system**
Least-squares minimization			
Tustin	mean	2.0689×10^{-12}	7.9119×10^{-12}
	std dev	0	1.7030×10^{-27}
FOH	mean	2.1658×10^{-12}	1.0801×10^{-11}
	std dev	0	1.7030×10^{-27}
Impulse	mean	3.6533×10^{-12}	4.0242×10^{-12}
	std dev	8.5149×10^{-28}	0
oestruc			
Tustin	mean	[0, 0.1]	[0.6, 0.4]
	std dev	[0, 0]	$[1.1703,\ 0.5851] \times 10^{-16}$
FOH	mean	[0, 0.1]	[0.5, 2.4]
	std dev	$[0,\ 1.4628] \times 10^{-17}$	$[0,\ 0.4868] \times 10^{-16}$
Impulse	mean	[0.5, 0.1]	[1.6, 0.3]
	std dev	$[0,\ 1.4628] \times 10^{-17}$	$[2.3406,\ 0.5851] \times 10^{-16}$
thresholding			
Tustin	mean	*failed failed*	
	std dev	*failed*	*failed*
FOH	mean	*failed*	*failed*
	std dev	*failed*	*failed*
Impulse	mean	*failed*	*failed*
	std dev	*failed*	*failed*

Table 4.1: Lateral dynamics results

One very noteworthy phenomenon is that many means are smaller than 10^{-10}, although the sample time is $t_{sample} = 0.1$ s, and thus the time-delay estimates should not be anything but multiples of t_{sample}.

The reason behind this phenomenon is related to unavoidable approximation errors born from the way the identification methods have been adapted to accept values of t_{sample} different from 1 s, such as the function `round` used in Algorithm 3.10.

It is also immediately noticeable that many standard deviations are very small, less than 10^{-15} s, this happens for the same reason some means are lower than 10^{-10} s but are not null, given that the set used to compute the standard deviations is afflicted by the aforementioned approximation issue.

In the next tables such results, both means and standard deviations, will be directly given as 0 s.

It is very important to point out that thresholding failed, meaning it yielded $\tau = -1$ both when applied on the unstable model and on the stable model, the reason for this event is that the function `arx` could not work reliably with the assigned input and the resulting outputs, due to some matrix, computed during usage, being singular. This behaviour is repeated in the next table, Table 4.2.

In Table 4.2 the results have been adjusted, giving a cleaner overview of means and standard deviations.

Time conversion method	**Type**	**Value, stable system**	**Value, unstable system**
Least-squares minimization			
Tustin	mean	0	0
	std dev	0	0
FOH	mean	0	0
	std dev	0	0
Impulse	mean	0	0
	std dev	0	0
oestruc			
Tustin	mean	[0, 0.1]	[0.6, 1.2]
	std dev	[0, 0]	[0, 0]
FOH	mean	[0, 0.1]	[0.52, 0.19]
	std dev	[0, 0]	[0, 0]
Impulse	mean	[3.1, 0.1]	[0.37, 0.14]
	std dev	[0, 0]	[0, 0]
thresholding			
Tustin	mean	*failed failed*	
	std dev	*failed*	*failed*
FOH	mean	*failed*	*failed*
	std dev	*failed*	*failed*
Impulse	mean	*failed*	*failed*
	std dev	*failed*	*failed*

Table 4.2: Longitudinal dynamics results

During the test of the longitudinal dynamics, the thresholding method failed twice on the first input with *Tustin* time conversion and once on the second input with *Impulse* time conversion.

Table 4.3 and Table 4.4 present the results of directional and vertical dynamics.

They are slightly different tables, compared to 4.1 and to 4.2, in that there is not "Unstable system" column, because the directional and vertical dynamics are stable in continuous time, and, upon conversion to discrete time, they are still stable.

The other difference is that each of these two models only has one output, hence the `oestruc` and thresholding parts will not have two-component vectors, instead a single value will be presented.

Time conversion method	**Type**	**Value**
Least-squares minimization		
Tustin	mean	0
	std dev	0
FOH	mean	0
	std dev	0
Impulse	mean	0
	std dev	0
oestruc		
Tustin	mean	0
	std dev	0
FOH	mean	0
	std dev	0
Impulse	mean	0
	std dev	0
thresholding		
Tustin	mean	*failed*
	std dev	*failed*
FOH	mean	*failed*
	std dev	*failed*
Impulse	mean	*failed*
	std dev	*failed*

Table 4.3: Directional dynamics results

By looking at Table 4.3 and Table 4.4 of the two first-order dynamics, no conclusive argument can be drawn in favour of either `oestruc` or the least-squares minimization, furthermore, thresholding is plagued by issues in the `arx` function as much as it happened with the longitudinal and lateral dynamics.

In general, a better comparison between the three delay identification methods is drawn by observing either the lateral or the longitudinal dynamics from Table 4.1 and Table 4.2.

The only method that, with the first input described in Section 4.1.2, consistently achieves a correct or close (1 or $2t_{sample}$ away from the expected result) to correct estimate is the least-squares minimization, with `oestruc` and thresholding being as reliable only for stabilized models.

Time conversion method	**Type**	**Value**
Least-squares minimization		
Tustin	mean	0
	std dev	0
FOH	mean	0
	std dev	0
Impulse	mean	0
	std dev	0
oestruc		
Tustin	mean	0
	std dev	0
FOH	mean	0
	std dev	0
Impulse	mean	0
	std dev	0
thresholding		
Tustin	mean	*failed*
	std dev	*failed*
FOH	mean	*failed*
	std dev	*failed*
Impulse	mean	*failed*
	std dev	*failed*

Table 4.4: Vertical dynamics results

`oestruc` is more consistent than thresholding over the ten attempts on the unstabilized models but, while not as imprecise as thresholding, it still cannot estimate well enough the delays.

It can also be noticed that the *Impulse* method of continuous- to discrete-time conversion seems to yield less accurate results compared to the other two methods, especially when used in conjunction with `oestruc`.

4.2.2. Impulse input results

The results derived from stimulation of the models through the impulse input of Section 4.1.2 are described in this section.

Only the lateral, directional and vertical dynamics will be described, the longitudinal will be skipped due to its similarity with the lateral one.

The underlying aim of this second analysis is to understand which identification method is better suited to identify delays using impulse inputs.

The thresholding method is not expected to perform well because, as described in Chapter 3, it uses input and output data to create an ARX model of the impulse response from which the thresholding is derived, the impulse input and its response should not give enough elements to the method for a reliable ARX modelisation.

`oestruc` is expected to have estimation issues due to the same reason as thresholding: it is expected to have difficulties creating its underlying model and, as a consequence, fail to correctly identify the delay.

It must be noted that `oestruc` does not have a "failure mode" like thresholding does by yielding -1, so, the only ways to understand if the estimation process encountered problems are to check for warnings in the MATLAB® command line, or to check how close the estimated delay is to the maximum admissible value, if it is much closer to that than to the expected result then `oestruc` has encountered issues.

Time conversion method	**Type**	**Value, stable system**	**Value, unstable system**
	Least-squares minimization		
Tustin	mean	0	0
	std dev	0	0
FOH	mean	0	0
	std dev	0	0
Impulse	mean	0	0
	std dev	0	0
	oestruc		
Tustin	mean	[0.2, 0.2]	[3.2, 3.4]
	std dev	[0, 0]	[0, 0]
FOH	mean	[0.2, 0.2]	[3.1, 3.3]
	std dev	[0, 0]	[0, 0]
Impulse	mean	[0.2, 0.2]	[3.1, 3.3]
	std dev	[0, 0]	[0, 0]
	thresholding		
Tustin	mean	*failed*	*failed*
	std dev	*failed*	*failed*
FOH	mean	*failed*	*failed*
	std dev	*failed*	*failed*
Impulse	mean	*failed*	*failed*
	std dev	*failed*	*failed*

Table 4.5: Lateral dynamics results with impulse input

The three Tables, 4.5, 4.6 and 4.7, indeed show results mostly in line with those expected that were stated at the beginning of Section 4.2.2:

- Least squares minimization performance has not been degraded by the lower quality

Time conversion method	**Type**	**Value**
Least-squares minimization		
Tustin	mean	0
	std dev	0
FOH	mean	0
	std dev	0
Impulse	mean	0
	std dev	0
oestruc		
Tustin	mean	3.1
	std dev	0
FOH	mean	3.3
	std dev	0
Impulse	mean	2.9
	std dev	0
thresholding		
Tustin	mean	*failed*
	std dev	*failed*
FOH	mean	*failed*
	std dev	*failed*
Impulse	mean	*failed*
	std dev	*failed*

Table 4.6: Directional dynamics results with impulse input

and less "rich" input sequence.

- Thresholding has, as expected, encountered issues during the construction of the ARX model, thus has failed every estimation problem.

- `oestruc` only performed well on the identification problem of the stabilised lateral dynamics, and even then it didn't perform as well as it did in Section 4.2.1. The other results are mostly $\tau > 3$ s, regardless of the dynamics or the time conversion method, indicating failure to achieve a reasonable estimate, given that $\tau_{max\ admissible} = 3.5$ s and that $\tau_{expected} = 0$ s.

4.2.3. RBS input results

The set of results presented in Table 4.8, in the current subsection, will present results obtained by using an RBS input, which has been described in Section 4.1.2. An important difference compared to Section 4.2.2 and most of Section 4.2.1 is that the means and standard deviations are not approximated to 0 s.

Time conversion method	**Type**	**Value**
Least-squares minimization		
Tustin	mean	0
	std dev	0
FOH	mean	0
	std dev	0
Impulse	mean	0
	std dev	0
oestruc		
Tustin	mean	1.7
	std dev	0
FOH	mean	1.6
	std dev	0
Impulse	mean	3.1
	std dev	0
thresholding		
Tustin	mean	*failed*
	std dev	*failed*
FOH	mean	*failed*
	std dev	*failed*
Impulse	mean	*failed*
	std dev	*failed*

Table 4.7: Vertical dynamics results with impulse input

The reason for abandoning the approximation is that in this case it does not hold anymore: every time the simulation for a given combination of continuous- to discrete-time method and time-delay estimation method is restarted, a new RBS is generated.

The time-delay estimation methods have to work with a slightly different input every time a simulation is started for every combination of methods, so the greatest source of discrepancies is not MATLAB® approximating results, which is of no interest, but is the effect of using a new signal for every single function call, and it is important to see how the methods react over 10 runs for each combination.

The most noticeable change brought by the use of RBS input is that thresholding method has been successful in its estimation process, yielding precise estimates for the stabilized model.

`oestruc` and thresholding still do not work well with unstable systems:

- `oestruc` has yielded estimates in excess of $\tau = 3$ s 5 times out of 30 runs among all continuous- to discrete-time conversion methods,

Time conversion method	Type	Value, stable system	Value, unstable system
Least-squares minimization			
Tustin	mean	0	3.35×10^{-8}
	std dev	0	1.22×10^{-8}
FOH	mean	0	3.88×10^{-8}
	std dev	0	3.57×10^{-8}
Impulse	mean	0	6.59×10^{-8}
	std dev	0	2.21×10^{-8}
oestruc			
Tustin	mean	[0, 0.01]	[0.1889, 0.7778]
	std dev	[0, 0.0316]	[0.0333, 0.7311]
FOH	mean	[0, 0]	[0.4667, 0.64]
	std dev	[0, 0]	[0.35, 0.7975]
Impulse	mean	[0, 0.1]	[0.3222, 0.5222]
	std dev	$[0, 10^{-17}]$	[0.6852, 0.7032]
thresholding			
Tustin	mean	[0, 0]	[0.4250, 1.005]
	std dev	[0, 0]	[0.2053, 0.6790]
FOH	mean	[0, 0]	[0.0857, 0.3]
	std dev	[0, 0]	[0.1464, 0.5333]
Impulse	mean	[0, 0]	[0.5444, 0.36]
	std dev	[0, 0]	[0.8002, 0.4248]

Table 4.8: Lateral dynamics results for RBS input

- thresholding has failed 6 times in total, meaning it either yielded $\tau = -1$ (twice) or yielded $\tau > 3$ s (four times), once, using the *Impulse* time conversion method, it gave a result as high as 6.5 s.

The standard deviations might seem still high, even though values of -1 or greater than 3 s have been discarded, the reason of this is because, while most of the remaining estimates are clumped together, some outliers like $\tau = 2.5$ s have not been discarded.

The undiscarded outliers have been kept because they were not as close to $\tau = 3.5$ s, which is the maximum admissible delay value for `oestruc`. Previously a "region" between 3 s and 3.5 s had been set as the sets of results from `oestruc` that indicated a failure to reach a plausible estimate, this reasoning has been extended to thresholding, so that a better comparison could be carried out.

This set of results from `oestruc` have also another important difference compared to the sets from Section 4.2.1 and Section 4.2.2: their standard deviation is not null or approximable to null, but this is expected given the nature of the RBS input, as stated at the beginning of Section 4.2.3. The other dynamics were not analysed, given their similarity to the one studied in the current subsection or their stability.

5 | Conclusions and future developments

From the results presented and described in Section 4.2.1, Section 4.2.3 and Section 4.2.3, the primary conclusion is that, of the three methods under study, the least squares-minimization is the most reliable for this precise model of UAV, but, in general, it has the great downside of being able to identify just the shortest input delay, and it is not able to detect the relationship between each output and each input as `oestruc` and thresholding do.

Another relevant downside of the least squares-minimization method is that it requires knowledge of the relationship between states and inputs and also of the states themselves, rendering impossible to use this method if these data are unavailable, for instance when the states are not measurable.

`oestruc` is not without downsides either, it is much slower than least-squares minimization: computational time, gathered using the function `cputime`, for `oestruc` is 2 to 3 orders of magnitude (seconds or tens of seconds, compared to hundredths of seconds) greater than the time employed by the least-squares minimization.

The comparison on computational times has been led using the time-delay estimation methods on the lateral dynamics, using both stabilized and unstable models, `oestruc` takes longer if the model is stabilized.

Most importantly, `oestruc` can't reliably compute the time-delay of the analysed models if they are unstable and/or the input sequence is not "rich" enough. The "richness" of the input sequence in general is an issue also for the least-squares minimization, so much so that the source [12] explicitly states it, but it does not seem to affect the particular set of models examined in this study.

Thresholding worked well only with RBS inputs but it achieved good results during that set of simulations nonetheless, being comparable with `oestruc`, in terms of means, of standard deviations and of number of failed attempts (5 for `oestruc`, 6 for thresholding) for what concerns applications on an unstable system.

Good results were also achieved by thresholding when applied to the stabilized system, always yielding the correct estimate.

Thresholding method is thus deemed suitable for stable systems with just certain kinds of inputs, limiting the extent of its applicability. On the other hand, it is faster than `oestruc` by at least an order of magnitude: most measured runs of thresholding on a stabilized system needed between 0.1 s and 0.2 s, only 16.667% of 30 runs needed more than 2 s.

Thresholding seems suited for all the situations where the system under study:

- is stable or has been stabilized,
- can be excited by a suitable input like Random Binary Signals,
- is MIMO and estimating the delay between every input and every output is important,
- has expected time delay(s) already known or hypothesised, in order to skim the outliers out of the results.

Further study should be undertaken to better understand the limits of these methods, regardless of the models they are applied to. The main areas that should receive further attention are:

- whether there are types of unstable models of which the delay can be estimated by `oestruc` without being stabilized,
- where the "richness" of the input starts to matter for the least-squares minimization and for `oestruc`,
- testing more complicated forms of thresholding methods to gather a deeper understanding of their possible application and their limits.

Bibliography

[1] S. Björklund. *A Survey and Comparison of Time-Delay Estimation Methods in Linear Systems*. PhD thesis, Linköpings universitet, Department of Electrical Engineering, 2003. Thesis No. 1061.

[2] S. Björklund. Experimental evaluation of some cross correlation methods for time-delay estimation in linear systems. Technical report, Linköpings universitet, Department of Electrical Engineering, April 2003.

[3] S. Björklund. A study of the choice of model orders in arxstruc-type methods for open-loop time-delay estimation in linear systems. Technical report, Linköpings universitet, Department of Electrical Engineering, August 2003.

[4] G. Fumai. Accuracy analysis of $\mathrm{PBSID}_{opt} - \mathrm{H}_\infty$ for multirotor uavs model identification. Master's thesis, Politecnico di Milano, 2019-2020.

[5] I. Houtzager, J. W. van Wingerden, and M. Verhaegen. Fast-array recursive closed-loop subspace model identification. In *Proceedings of the 15th IFAC symposium on system identification*, Saint Malo, France, July 2009. IFAC.

[6] I. Houtzager, J. W. van Wingerden, and M. Verhaegen. Varmax-based closed-loop subspace model identification. In *Proceedings of the 48th IEEE Conference on Decision and Control*, Shanghai, China, December 2009. IEEE.

[7] I. Houtzager, J. W. van Wingerden, M. Verhaegen, and F. Felici. *PB-SID Toolbox* website. `https://www.dcsc.tudelft.nl/~jwvanwingerden/pbsid/pbsidtoolbox_product_page.html`, 2009.

[8] T. M. Inc. Matlab® *System Identification Toolbox* documentation on `arx` function. `https://it.mathworks.com/help/ident/ref/arx.html?s_tid=doc_ta`, .

[9] T. M. Inc. Matlab® *System Identification Toolbox* documentation on `c2d` function. `https://it.mathworks.com/help/ident/ref/lti.c2d.html?s_tid=doc_ta`, .

[10] J. Kekalainen. A discrete time domain approach on time delay estimation. *Acta IMEKO*, 8(1):77–86, March 2019.

[11] J. W. van Wingerden and M. Verhaegen. Subspace identification of bilinear and lpv systems for open and closed loop data. *Automatica*, 45(2):372–381, 2009.

[12] R. Waschburger and R. K. H. Galvão. Time delay estimation in discrete-time state-space models. *Signal Processing*, (93):904–912, 2013.

List of Figures

List of Tables

List of Symbols

Variable	Description	SI unit
F_{aero}	vector of aerodynamic forces	N
g	gravity acceleration	m/s^2
J_{ij}	inertia moment referred to i, j axes	kg m^2
$\mathcal{L}$	aerodynamic moment around x-body-axis	Nm
M_{aero}	vector of aerodynamic moments	Nm
$\mathcal{M}$	aerodynamic moment around y-body-axis	Nm
$\mathcal{N}$	aerodynamic moment around z-body-axis	Nm
p	angular velocity around the x body-axis	rad/s
q	angular velocity around the y body-axis	rad/s
r	angular velocity around the z body-axis	rad/s
u	velocity along the x body-axis	m/s
v	velocity along the y body-axis	m/s
V_G	Velocity of centre of mass	m/s
w	velocity along the z body-axis	m/s
τ	time-delay	s