ECOLE POLYTECHNIQUE FEDERALE DE LAUSANNE

SEMESTER PROJECT :

SYSTEM IDENTIFICATION & CONTROL DESIGN OF A 2DOF HOVER

ALY ELBINDARY (300247)

DDMAC LAB

Supervising Professor : Alireza Karimi

Supervising TA : Mert Eyuboglu

January 8, 2024

Table of Contents

1 Introduction

In this project, we will be working with a 2DOF hover called Aero2, manufactured by Quanser. The aim of the project is to put into practice different system identification methods in order to be able to determine a model that can describe the hover. This is an important process, since before developing controllers in order to perform a specific task, we need to have a model on which we can design this controller.

We will start off by describing the working principle of the hover and its specifics, in order to properly describe the project goals. Then we will dive into the theoretical concepts that will be employed in order to achieve these goals. Finally, we will go through the different results that we have obtained in order to identify the different models.

2 Project Description

2.1 Description of Quanser Aero2 Hover

Figure 2.1.1: Quanser Aero2 and its Features [2]

As we can see in figure 2.1.1 above, the Quanser Aero2 hover is equipped with two different rotors, one facing upwards and another positioned sideways. Each rotor is actuated by a separate motor,

thus meaning that we have two different inputs : a voltage input V_0 for the vertical motor, and another voltage input V_1 for the sideways motor.

Additionally, the hover is equipped with a multitude of different sensors, such as an IMU, an accelerometer and a gyroscope which can all be seen in figure 2.1.1, as well as some additional ones, such as the tachometer. But more importantly, we can see that there are two encoders : one for the pitch angle and one for the yaw angle. The encoders give us the motor counts for each motor, which means that we can obtain the pitch and yaw angles at any given moment by converting the motor count values through the usage of the encoder resolutions. This is where we'll need the following specifications for the Aero2 hover :

Quanser Aero2 Specifications	
Device Dimensions (D x W x H)	18 cm x 52 cm x 40 cm
Operating Space (D x W x H)	52 cm x 52 cm x 62 cm
Mass	4.7 kgm
Pitch Angle Range	90° (± 45° from horizontal)
Yaw Angle Range	360º
Pitch Encoder Resolution	2880 counts/revolution
Yaw Encoder Resolution	4096 counts/revolution
Prop Thrust Constant	5 Compact 6-Axis MEMS Device
Inertial Thrust Contant	0.042 Nm/A
Inertial Measurement Unit (IMU)	IIM-42652 Compact 6-Axis MEMS Device
Tri-axis gyroscope range	+- 500 dps
Tri-axis accelerometer range	+- 2g

Figure 2.1.2: Quanser Aero2 Specifications [2]

Now looking at the table within figure 2.1.2 above, the two important specifications are the pitch and yaw encoder resolutions : these constants will allow us to convert the encoder readings into angle readings.

Now that we have an overview of the machine itself, we will now move onto the software that is used to communicate and interact with it.

2.2 Quanser Simulink Template

To be able to interact with the Aero2 hover, Quanser provide a built-in template within Simulink, thus we can use MatLab in order to apply custom inputs, and collect the corresponding output data. In figure 2.2.1 below, we can see an example of a simulink block diagram that allows this interaction.

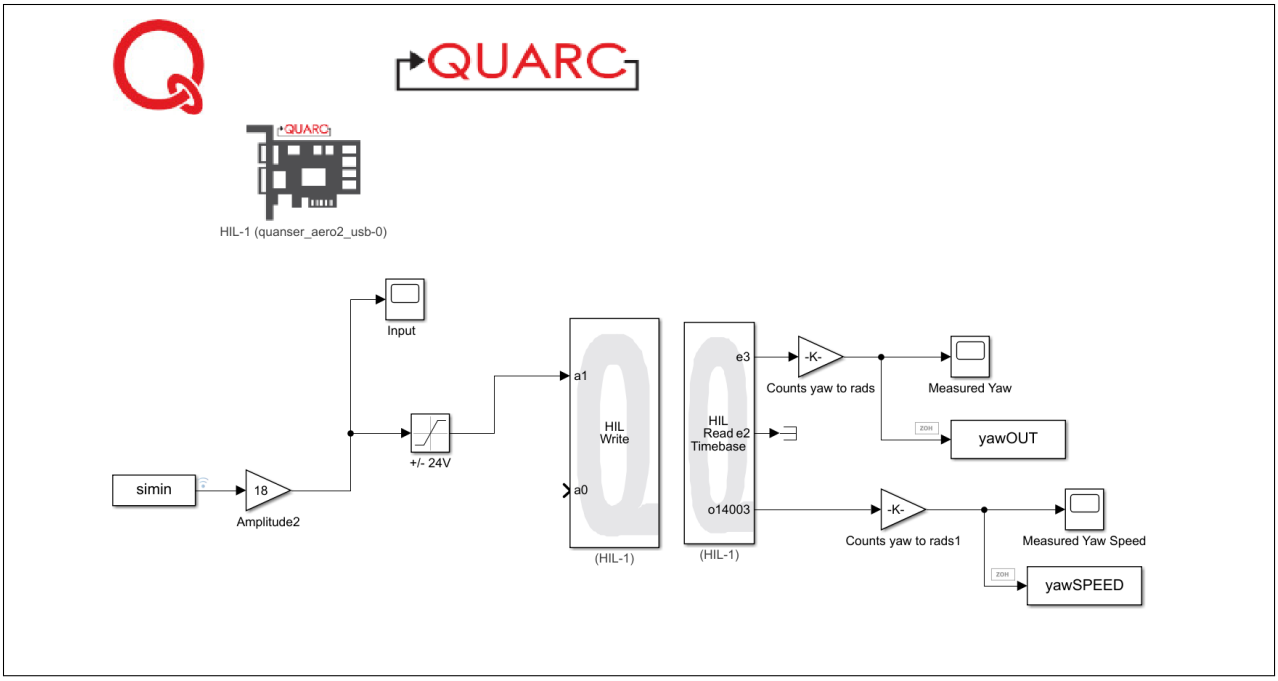


Figure 2.2.1: Example of Quanser Simulink Template

In figure 2.2.1 above, there are three main blocks (provided by Quanser) that allow for this communication :

- **HIL Initialize (top left)** : sets up the communication between the simulink file and the Aero2 hover ;
- **HIL Write (middle)** : two inputs can be seen, a0 (voltage input V_0) and a1 (voltage input V_1) ;
- **HIL Read Timebase** : two main inputs, e2 (motor counts given by pitch encoder) and e3 (motor counts given by yaw encoder). There are outputs that can be configured into the block, such as the IMU readings, however these are the important ones to mention for the time being.

Thus, any kind of custom signal (transferred through the ”simin” block which can be seen on the left-hand side in figure 2.2.1) can be given into the voltage inputs, in order to conduct different studies.

Finally, we can see in the right-hand side figure 2.2.1 that all of the outputs can be recorded and measured through ”to workspace” and scope blocks respectively. Note that gain blocks (”Counts yaw to rads”) correspond to the encoder resolutions that we had previously seen in figure 2.1.2.

Now that the Aero2 hover has been described and we also know how to interact and collect data from it, we can now move onto the objectives that we wish to achieve within this project.

2.3 Objectives of the Project

In subsections 2.1 and 2.2, we have seen that when the hover is given certain inputs (V_0 and V_1), this will the translate into certain rotations, which will thus give us certain pitch and yaw angles (θ

and ψ respectively). This means that we can represent this system as a black-box model, which can be seen in figure 2.3.1 below :

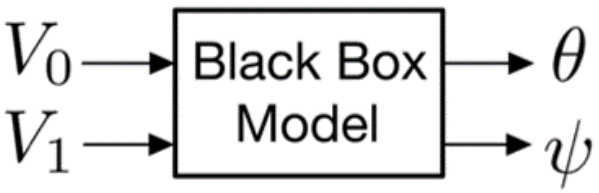


Figure 2.3.1: Aero2 Black Box Model

A good mathematical representation of a black-box model that has multiple inputs and multiple outputs is a MIMO (multi-input-multi-output) system. We will assume that each relationship between an input and an output is represented by a SISO (single-input-single-output) system. This can be summarized through the following equation :

$$\begin{pmatrix} \theta \\ \psi \end{pmatrix} = \begin{pmatrix} \boldsymbol{G_{11}} & \boldsymbol{G_{12}} \\ \boldsymbol{G_{21}} & \boldsymbol{G_{22}} \end{pmatrix} \begin{pmatrix} V_0 \\ V_1 \end{pmatrix} \Rightarrow \begin{cases} \theta = \boldsymbol{G_{11}} V_0 + \boldsymbol{G_{12}} V_1 \\ \psi = \boldsymbol{G_{21}} V_0 + \boldsymbol{G_{22}} V_1 \end{cases} \tag{2.3.1}$$

In equation 2.3.1 above, the two SISO models $\boldsymbol{G_{11}}$ and $\boldsymbol{G_{12}}$ correspond to the pitch models and the other two SISO models, $\boldsymbol{G_{21}}$ and $\boldsymbol{G_{22}}$, correspond to the yaw models.

Thus, the goal of this project is to identify these different SISO models through input and output measurements, so that a clearer relationship is established between the applied voltages and the subsequent angles. This would provide a more solid baseline for more in depth control designs of this hover for future projects.

Before going into the different experiments that were conducted and their corresponding results, we will first have a brief overview of the system identification theory that will serve as the foundation for these experiments.

3 System Identification Theory

In general, when we have any kind of system, we initially have access to only two sets of information : the input data that we feed the system, and the resulting output data. But there is nothing that relates these two sets together, to our knowledge. Thus, when it comes to control design, an important first is to be able to know what model we are trying to control. System Identification consists in identifying a model for the system that we are working with (frequency response, time-domain model, model order...), by using only the input and output datasets. There are many different methods to identifying different kinds of models, but for this project we will be focusing only on the handful of methods that were employed throughout the experiments. We will be basing ourselves off of the EPFL System Identification lecture notes [1].

3.1 Frequency Response Identification

When it comes to frequency response identification, what we aim to find is the Fourier transform of the model, ($G(e^{j\omega})$), knowing that we only have access to the input and output data $u(t)$ and $y(t)$.

There are mainly two different methods to doing so :

- Fourier Analysis ;
- Spectral Analysis.

In Fourier Analysis, the frequency response is given by the following equations :

$$y(t) = g(t) * u(t) \tag{3.1.1}$$

$$\Rightarrow Y(j\omega) = G(e^{j\omega})U(j\omega) \Rightarrow G(e^{j\omega}) = \frac{Y(j\omega)}{U(j\omega)} \tag{3.1.2}$$

From equation 3.1.1 to equation 3.1.2, we apply the Fourier transform on both sides of the equation. Thus, by computing the Fourier transforms of both the input and output signals through the FFT algorithm, we can directly obtain the frequency response of our model. The Fourier Analysis method is mainly used when the input and output data is periodic, and thus averaging the different periods together would help reduce noise.

In Spectral Analysis, the frequency response is given by the following equation :

$$G(e^{j\omega}) = \frac{\phi_{yu}(\omega)}{\phi_{uu}(\omega)} \tag{3.1.3}$$

where $\phi_{yu}(\omega)$ is Fourier transform of the intercorrelation between the input and output signals, and $\phi_{uu}(\omega)$ is the autocorrelation of the input signal. This method is usually employed when the input and output signals are not periodic, however when this is the case it is important to note that equation 3.1.3 becomes equivalent to equation 3.1.2 :

$$G(e^{j\omega}) = \frac{\phi_{yu}(\omega)}{\phi_{uu}(\omega)} = \frac{Y(e^{j\omega})U(e^{-j\omega})}{U(e^{j\omega})U(e^{-j\omega})} = \frac{Y(e^{j\omega})}{U(e^{j\omega})} \tag{3.1.4}$$

Although we will be mainly using PRBS signals as our input data, and therefore we would only need the Fourier analysis method, we will also be using spectral analysis in order to compare the two.

Additional methods, such as windowing and averaging, will also be used in order to reduce noise.

3.2 Time-domain Model Identification (Parametric Models)

The first challenge when it comes to identifying a parametric time-domain model for our system is to first find out what its structure is, i.e. the order of the system and the value of the delay. In this project, multiple techniques will be used in order to estimate the structure :

- Subspace Method (for order estimation) ;
- Loss Function (for order estimation) ;
- Pole Zero maps (for order estimation) ;

- FIR and ARX structure (for delay estimation).

The reason why multiple methods are used for the same objective is to be able to make sure that the results that we get are the most consistent and accurate. We will not be going into detail when it comes into the theory behind each of these methods, as they can all be viewed within the lecture notes [1], however it is important to mention them since these are the methods that will be used within our workflow to get the structure of our model, and thus create different parametric models fitting the found structure using the built-in MatLab functions. The different parametric models that we will be creating are the following :

- ARX ;
- IV4 ;
- ARMAX ;
- Output Error (OE) ;
- Box Jenkins (BJ) ;
- N4SID.

Again, the theory behind each of these structures can be found within the lecture notes [1], however it is important to note that in order to validate each of the identified models through these methods, not only will we simply compare them to the measurement data (for both the frequency response and the time-domain measurements), but we can also use their corresponding noise models (except for OE, since it does not have a noise model) for statistical validation (i.e. whiteness test). The principle of a noise model is to look at the intercorrelation between the input data and the noise . If they are not correlated, we then observe the autocorrelation of the noise. If it's equal to 0 (within a 95% confidence interval for this project) for all values except at the origin.

3.3 Closed-Loop System Identification

In subsections 3.1 and 3.2, the input data and output data is assumed to be measured in open-loop, meaning that input data that we generate is applied directly to the system, and thus meaning that equation 3.1.1 is applicable and direct relationships between the model and the measurement data can be established.

However, these system identification techniques are applicable only if the system that we are working with is stable. But we will see later on in the experiments that the yaw models are not stable, but in fact have an integrator. Therefore, different approaches would have to be used in order to identify the model, including the closed-loop system identification method that we will be viewing in this subsection.

The idea of closed-loop system identification is that we would like to stabilize our system before attempting to identify it. To do this, we would need to apply any stabilizing controller (a stabilizing proportional controller for example would suffice), so that now the system is attempting to track a certain reference signal, and thus will not diverge. Here's a block diagram illustrating this new scenario :

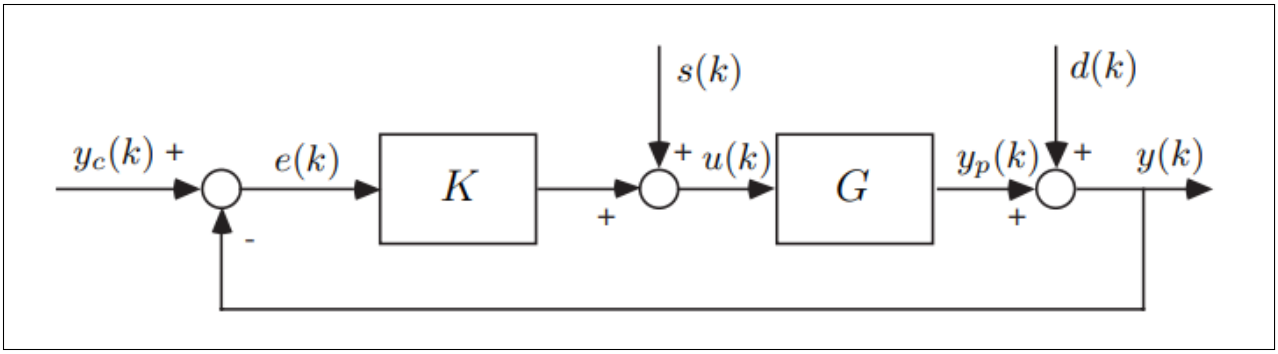


Figure 3.3.1: Closed-loop system with the external excitations $s(k)$ and $y_c(k)$ and output disturbance $d(k)$ [1]

We can see in figure 3.3.1 that the system is no longer directly excited by the input signal $u(t)$, but by other external signals. therefore a different method would need be used. The one that we will use for this project is the indirect method [1], where the entire closed-loop system is represented instead by a black-box model in open-loop, where the input signal is the reference signal $y_c(k)$ and the output signal is still $y(k)$. The transfer function of this new black-box model $\mathcal{T}(q^{-1})$ is given by :

$$\mathcal{T}(q^{-1}) = \frac{y(k)}{y_c(k)} = \frac{K(q^{-1})G(q^{-1})}{1 + K(q^{-1})G(q^{-1})}. \tag{3.3.1}$$

Therefore, using the previously discussed system identification techniques, $\mathcal{T}(q^{-1})$ can be identified, and by isolating the original plant model $G(q^{-1})$ within equation 3.3.1, we get :

$$G(q^{-1}) = \frac{\mathcal{T}(q^{-1})}{K(q^{-1})[1 - T(q^{-1})]}. \tag{3.3.2}$$

Now that all of the different theoretical techniques have been discussed, we will now move onto the experimental results.

4 Identification of Different SISO Models

For all the different experiments that we will be conducting, we will be mostly using a PRBS (with different characteristics each time) signal as our input. In system identification, PRBS signals are crucial : when looking at the spectral density of their autocorrelation signal [1] is very similar to a sum of sinusoids, meaning that multiple frequencies are being excited when applying a PRBS signal to a system. The greater the shift register (n) of a PRBS signal is, then the greater the degree of excitation (M) :

$$M = 2^n - 1. \tag{4.1}$$

A general rule of thumb to have a high enough degree of excitation is to apply a shift register of at least 7.

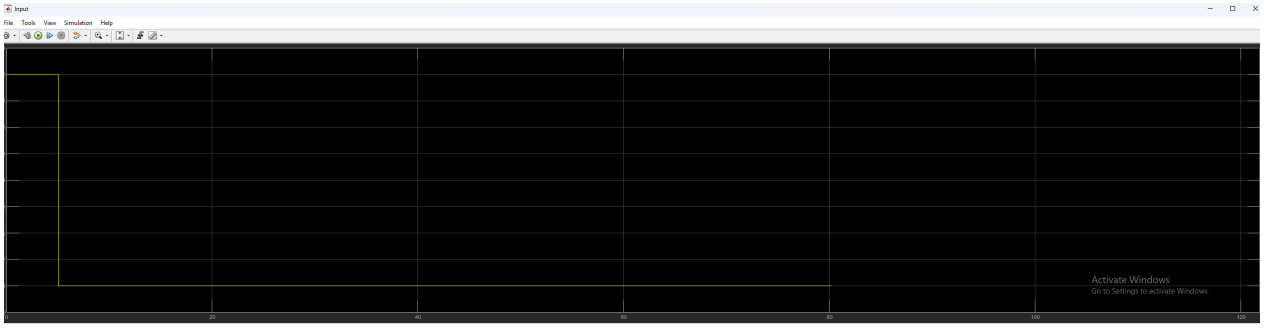
Furthermore, throughout this section, we will be referencing the different listings (MatLab Code), located within the appendix in section 7, that have been used in order to generate the subsequent results.

4.1 Identification of Pitch Models

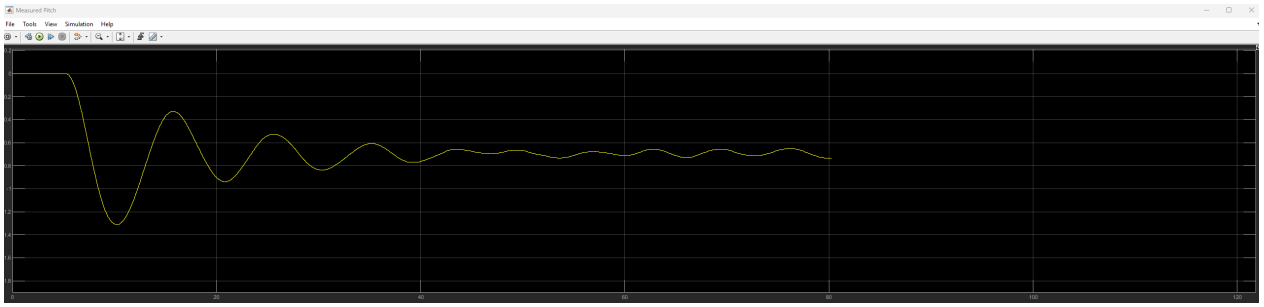
4.1.1 Identification of G_{11}

We will start off our experiments with the G_{11} system, meaning that we will be applying a certain voltage signal for V_0 and we will be recording the corresponding pitch angle θ.

To have an initial idea about the G_{11} system, we decided to apply a step input in order to view the step response.



(a) Step Input (Step Time : 5 s ; Step Value : -10 V)



(b) Step Response (Pitch Angle (rad) vs Time (s))

Figure 4.1.1.1: Step Response of G_{11} System

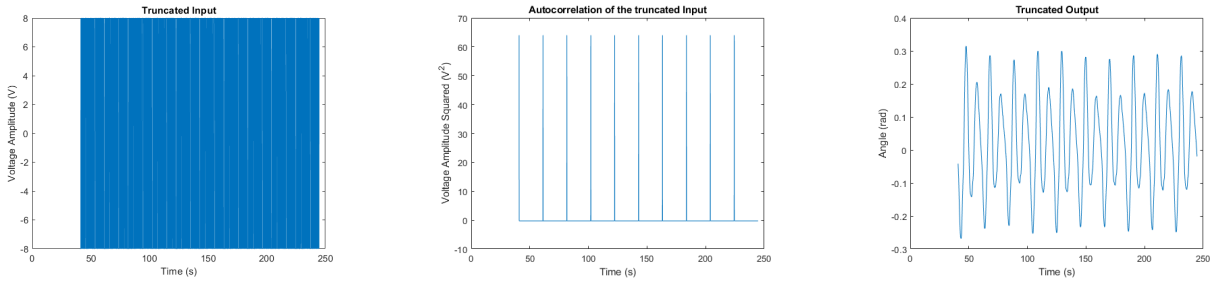
Looking at figure 4.1.1.1b above, we can see that the step response of the system appears to be that of a model that has an order of at least 2. We will be keeping this in consideration when moving onto to the identification.

We will now show the full identification process for a set of parameters :

- PRBS with a shift register $n = 8$;
- PRBS Voltage amplitude of 8V ;
- PRBS signal has 12 periods ;
- Sampling period $T_s = 0.08s$.

The applied PRBS signal has 12 periods : this is done since we will need to filter out the first two periods which are generally noisier than the rest of the signal and would diminish our final results.

We first use the MatLab listing 11 in order to apply the desired PRBS signal to the Aero2 hover, and then with the 12 listing, we get the following plots below :

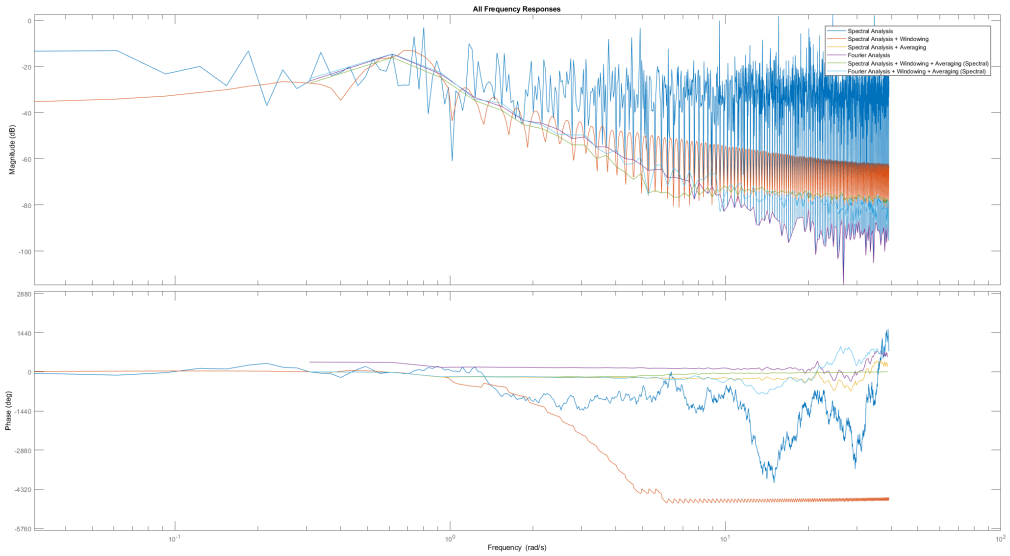


(a) PRBS Input (b) Autocorrelation of PRBS (c) Pitch Measurement

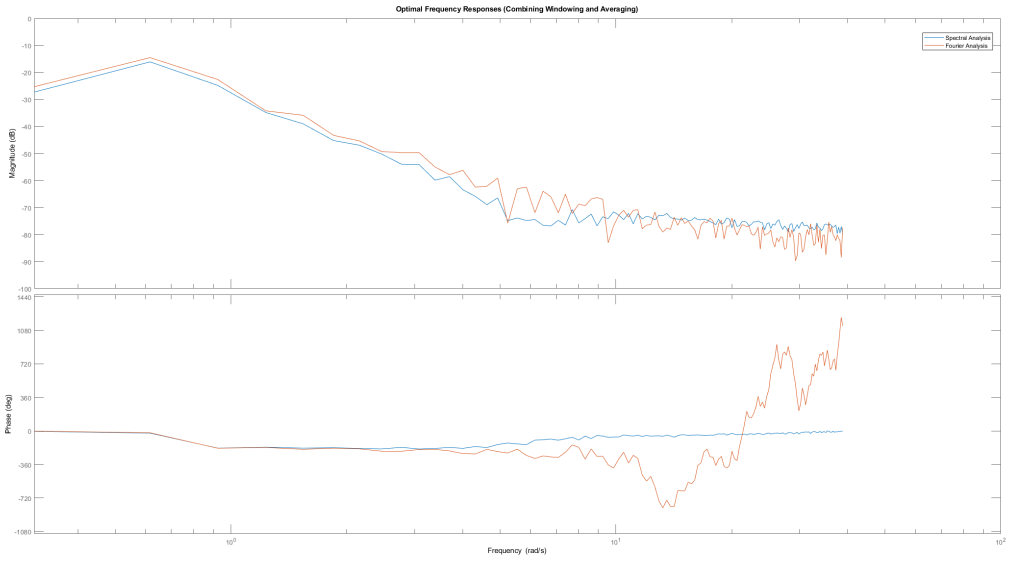
Figure 4.1.1.2: Truncated Input and Output Measurements for G_{11}

Looking at the autocorrelation of the PRBS signal in figure 4.1.1.2b, we can see that we are working with 10 different periods, after removing the first two.

Now that the measurements are done, we will move onto to the frequency response identification, using listing 13 :



(a) Frequency Response Identification with All Methods



(b) Frequency Response Identification with Two Best Methods

Figure 4.1.1.3: Frequency Response Identification for G_{11}

Looking at the two plots in figure 4.1.1.3 above, we can see that the frequency responses seem to follow our assumptions, where the order of the model appears to be at least 2, with a general peak at around 0.6 rad/s, and then a general trend downwards of the magnitude plots at the higher frequencies.

We will now move onto the order and structure estimation of the model, using the script within

listing 14.

The first thing we need to determine is the model order, thus firstly we will look at the singular values of the Q matrix from the subspace method, to get an initial idea of what the order might be :

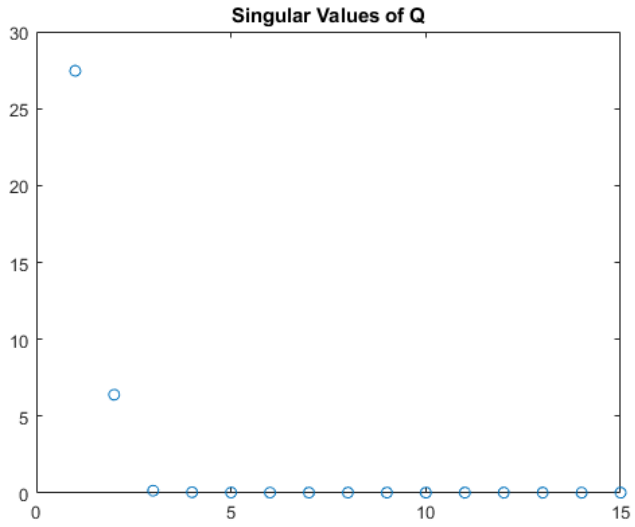


Figure 4.1.1.4: Singular Values of Q matrix for G_{11}

In figure 4.1.1.4 above, we can see that the number of nonzero singular values is 2, suggesting that this would be the model order δ. However, when we then determine a state-space model using this order, using the helper function within listing 10, we don't get great results, as we can see in figure 4.1.1.5 below :

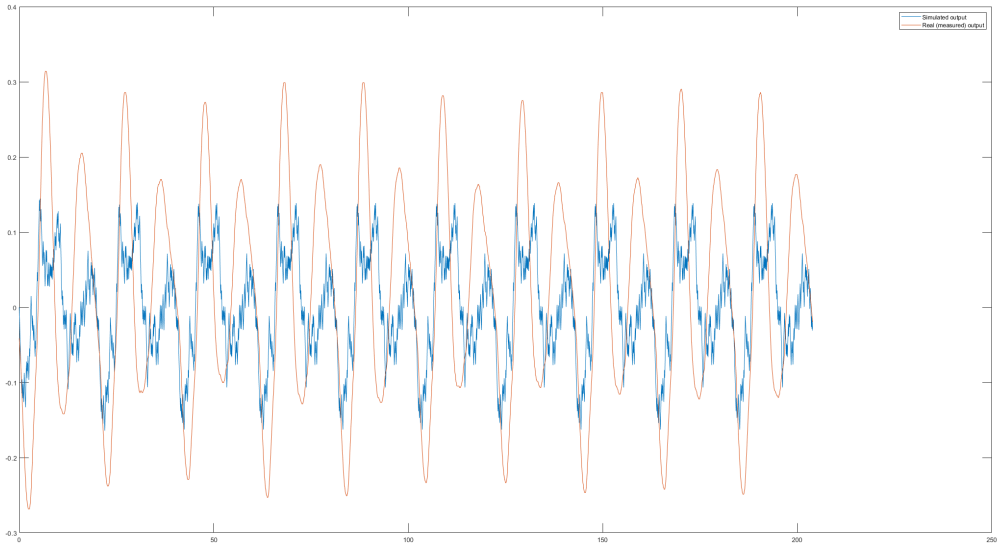


Figure 4.1.1.5: Comparison Between Identified State-Space Model and the Real Measured Output for G_{11}

Now let's evaluate the loss function and the pole/zero maps, to see if we can get different results with respect to the model order :

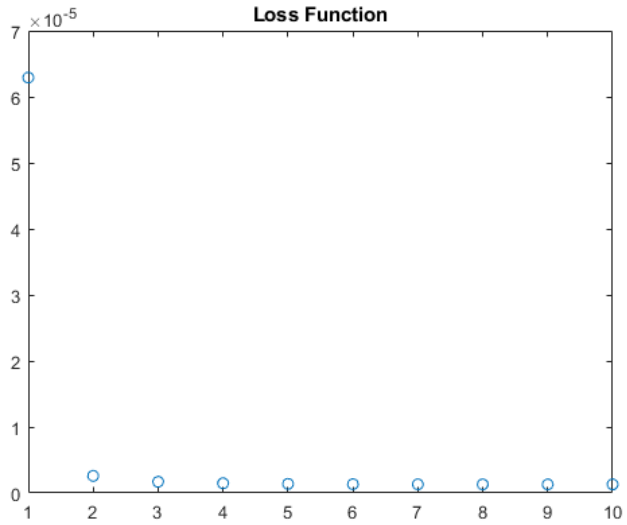


Figure 4.1.1.6: Loss Function for G_{11}

Again, from the loss function in figure 4.1.1.6, it seems that the model order is $\delta = 2$, since the first order at which the loss function significantly drops is $\delta = 2$.

However, the most accurate method is the pole/zero maps, so let's see what information they provide :

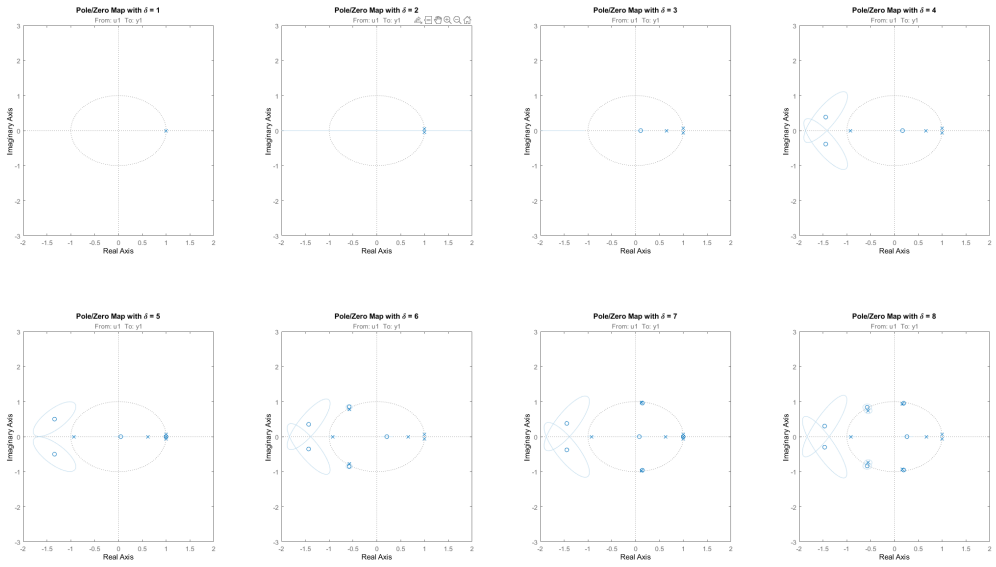
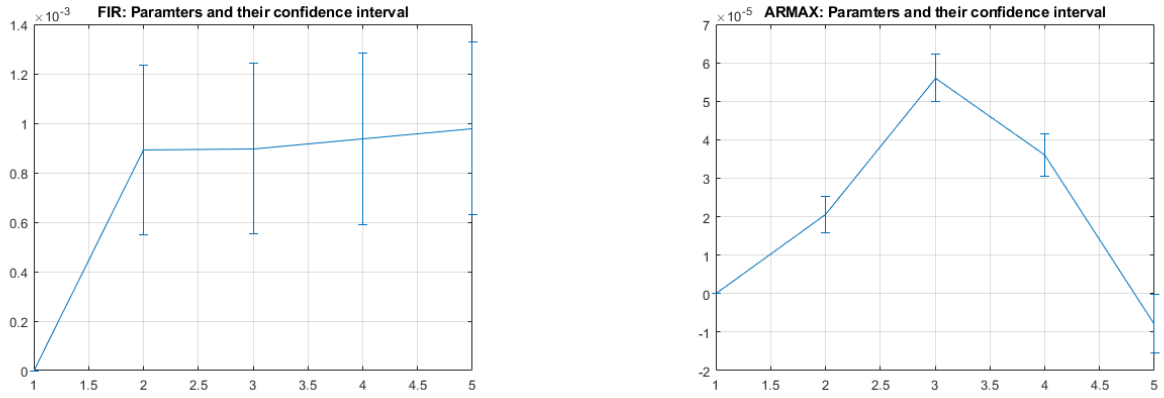


Figure 4.1.1.7: Pole/Zero Maps for Different Orders for G_{11}

Looking at the pole/zero maps in figure 4.1.1.7 above, we can see that the first map at which we stop seeing pole/zero cancellations is the one for a model order $\delta = 4$. Therefore setting the model order equal to 4 later on when it comes to the parametric model identification will yield better results.

Finally, we'll now need to determine the value of the delay n_k. Let 's look at the FIR and ARX parameters :



(a) FIR Parameters

(b) ARX Parameters

Figure 4.1.1.8: FIR and ARX Numerator Parameters and their Confidence Intervals G_{11}

We can see that, for both the FIR and ARX parameters in figure 4.1.1.8 above, that only the initial parameter of the numerator is equal to 0, meaning that the delay is most likely $n_k = 1$.

We have now estimated both the model order and structure, thus we can now move onto the final parametric model identification using structure that we have determined and the script within

listing 15. The structure that we will be using is $n_a = n_b = \delta = 4$ (number of parameters of the numerator and denominator polynomials, $A(q^{-1})$ and $B(q^{-1})$, being equal to the order) and $n_k = 1$. We will start off with the time domain comparisons :

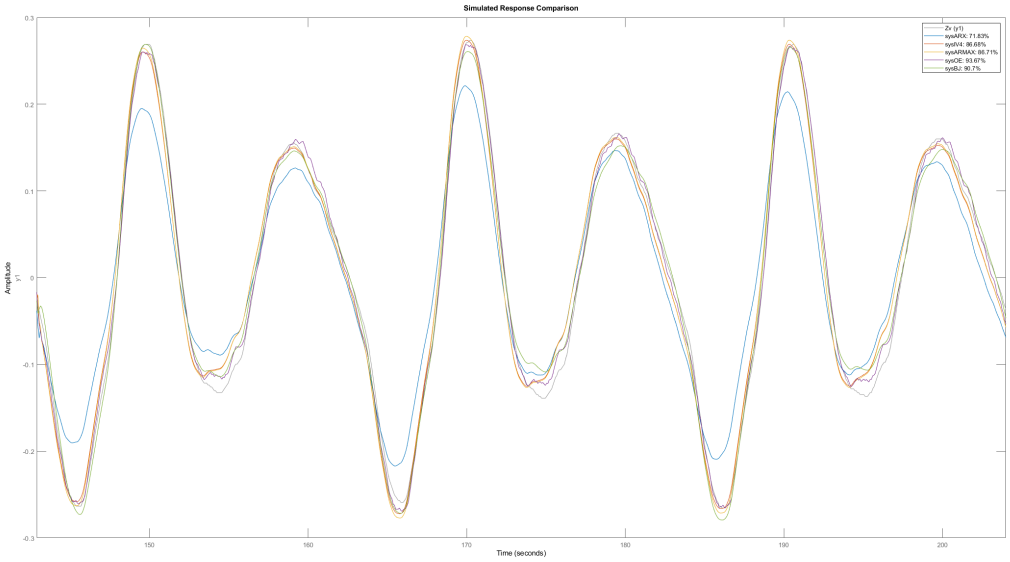
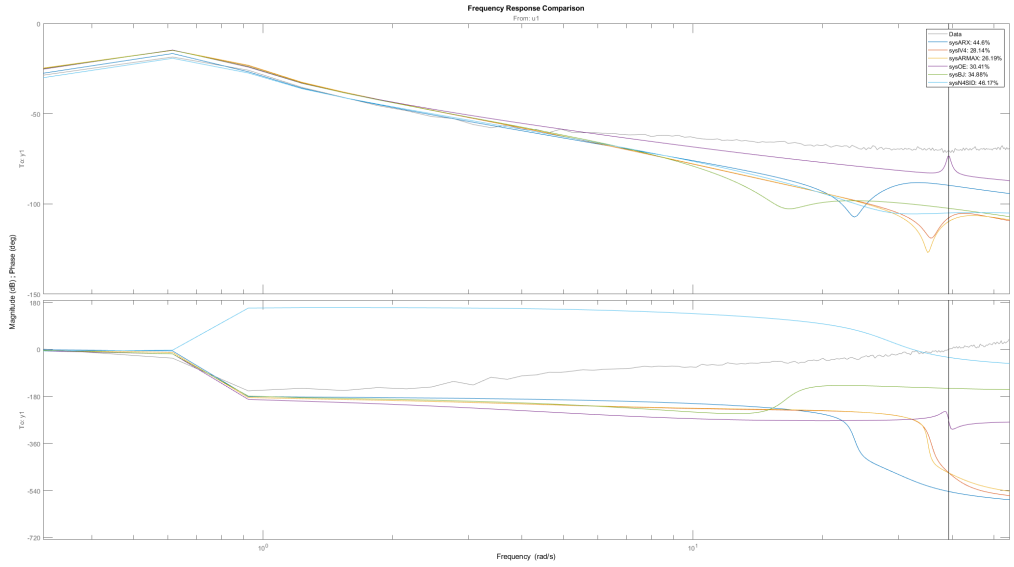


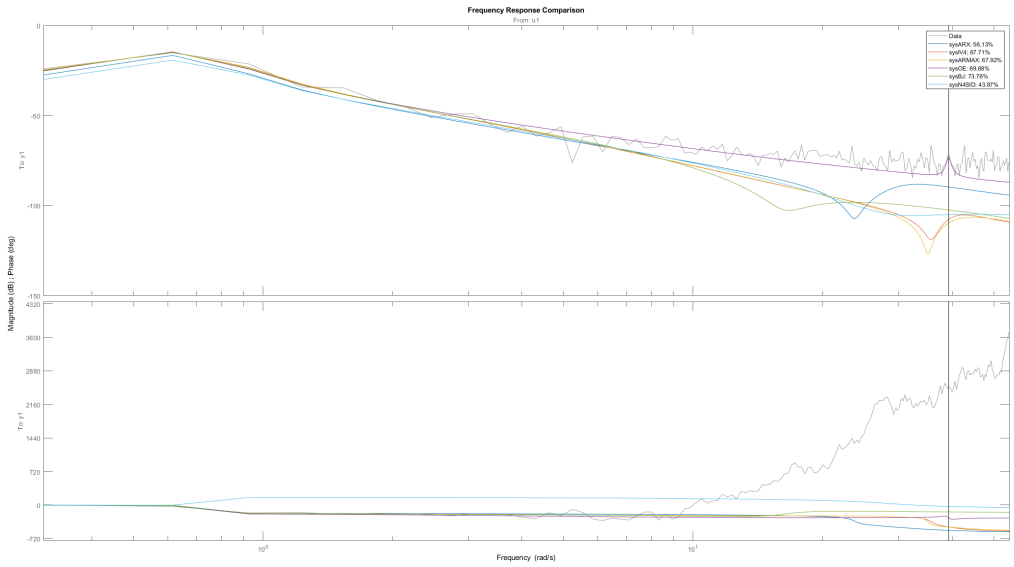
Figure 4.1.1.9: Time Domain Validation for Different Parametric Models

Firstly, we will note that in figure 4.1.1.9 above, there are only the last 3 periods of the output measurements. This is because there has been a 70/30% split of the measured data, for identification of the parametric models and and then to validate respectively. We will also note that we did not include the N4SID parametric model, since its results is significantly poorer than the other models in this case. We can see that multiple models have achieved a comparison percentage with the validation data, with the highest one being the Output Error model (93.67%). But this doesn't mean that this is necessarily the optimal model, we would still need to evaluate the time domain comparisons and evaluate the noise models.

When looking at the frequency response comparisons, we will firstly need to select one of the previously identified frequency responses. More specifically, one of the two from figure 4.1.1.3b. Thus, we decided to perform the comparisons relative to both frequency responses, and see which will yield better comparison results :



(a) Comparison with Spectral Analysis (with Windowing and Averaging)



(b) Comparison with Fourier Analysis (with Windowing)

Figure 4.1.1.10: Comparison between Frequency Responses of Parametric Models and identified Frequency Responses G_{11}

We can see that the comparisons with the frequency response identified through Fourier analysis along with windowing in figure 4.1.1.10b yields better results than the one with the Spectral Analysis along with Windowing and Averaging in figure 4.1.1.10a. Meaning that the Fourier frequency response is likely a better representation of the frequency response of the G_{11} system. In

this case, we get good comparison results, with the Box Jenkins model having best result (73.76%).

Finally, let's move onto the noise models :

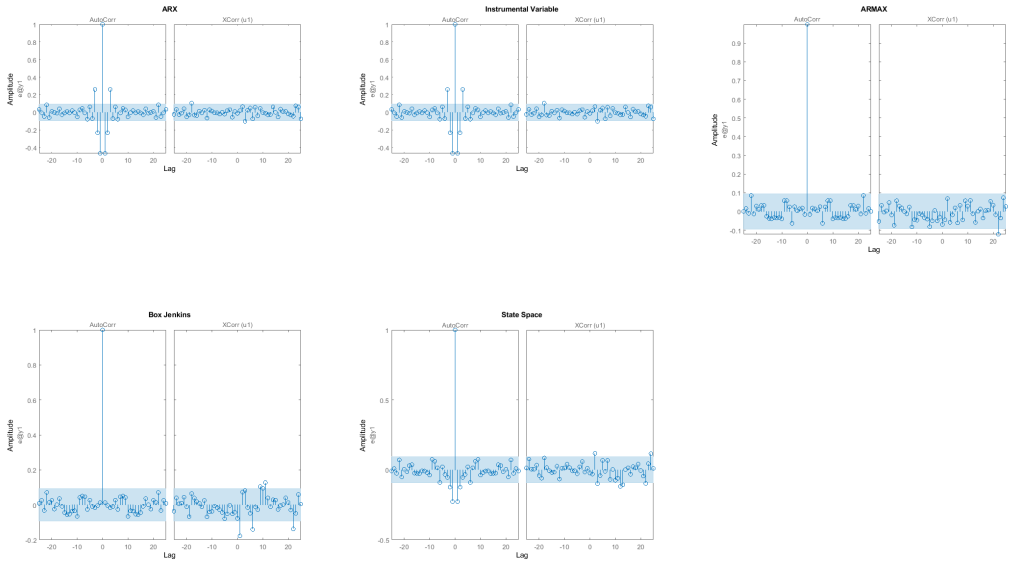


Figure 4.1.1.11: Noise Models for G_{11} parametric models

Note that in figure 4.1.1.11 above, there's isn't a noise model for Output Error, since this parametric method doesn't have one. Now, using the whiteness test principle that we described in subsection 3.2, we can see from figure 4.1.1.11 that the structures that pass the whiteness test are mainly the ARX and Box Jenkins structures, since they are the only ones to have their autocorrelation signals being equal to 0 for all values (except at the origin) within a 95% confidence interval.

Finally, for the chosen input PRBS signal, we can conclude that the optimal parametric model is either the Box Jenkins (BJ) structure or the Output Error (OE) structure. Although the OE model yields slightly better results in the time domain-comparisons (93.67% vs 90.7% when looking at figure 4.1.1.9), since the BJ structure has better results in the frequency domain (73.76% vs 69.68% when looking at figure 4.1.1.10b) and it passes the whiteness test, it might be more suitable to select the BJ model, however both options in this case are acceptable.

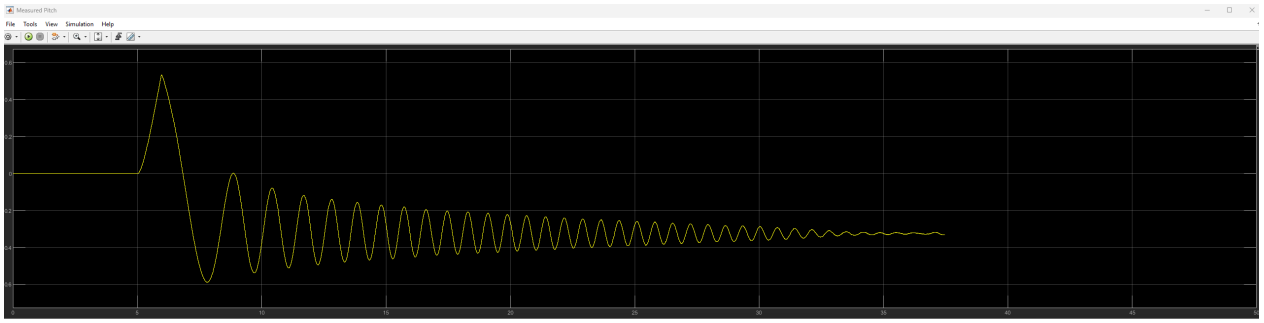
This process will serve as a pipeline for other system identifications. However, before moving onto the identification of the G_{12} model, it is important to note that for the G_{11} system, once a voltage input of more than around 12V (as an absolute value) is applied to the hover, then there is a very high likelihood that the system will start bottling, and therefore the identified models might not be too applicable in these scenarios. Therefore, the vertical facing motor should only be operated within a particular voltage range (between +/- 12V) in order to have proper control design later on using the identified parametric models.

4.1.2 Identification of G_{12}

For this system, we are trying to identify the impact of of the horizontal motor on the pitch angle, which is the secondary output in this case, in the sense that the primary objective of the horizontal motor is to modify the yaw angle. The voltage range at which the horizontal motor starts affecting the yaw angle of the hover is for very high voltage values (higher than about 14V in absolute value). However, for these very high values, we get intense bottling of the hover, which would then give very inconsistent readings in terms of pitch. In order to properly analyze, we will identify the model for lower voltage values (around 5V) : even if yaw variations are minimal at this range, there are still significant variations in terms of pitch, which can be identified, and would give us a model that would best represent the effect of the horizontal motor on the pitch angle.

Again, in order to have an idea about the system that we are working with, we will view the step response of the system when applying a step signal :

(a) Step Input (Step Time : 5 s ; Step Value : 20 V)



(b) Step Response (Pitch Angle (rad) vs Time (s))

Figure 4.1.2.1: Step Response of G_{12} System

Looking at the step response in figure 4.1.2.1b above, we can see that the model again seems to be at least a second order. We can also that that the first peak within this figure is much ”sharper” than the other oscillations, this is because of the bottling that the hover had initially done at the beginning of the experiment.

We will now be using the same pipeline that we have used in the previous sub-section 4.1.1 for system identification. The input signal that we will be applying is a PRBS signal of order 8, with 12 periods, and a sampling period of 0.2s. The results from the process can be seen within the appendix in sub-section 7.3.

To summarize, using the pole/zero maps and FIR parameters within figures 7.3.5 and 7.3.6 respectively, we can deduce that the model order is $\delta = 3$ (thus we have $n_a = n_b = \delta = 3$) and that the delay is $n_k = 1$ (even if the FIR parameters would seem to suggest that the delay is equal to 2, through trial and error during the parametric model identification, we can deduce that a delay of 1 is more adequate).

The time-domain comparisons in figure 7.3.7 give great results, with almost all models achieving more than 90% similarity with the measurement data.

Now, for the frequency-response comparisons in figure 7.3.8, comparing with the frequency-response identified through spectral analysis along with windowing and averaging yields better results, with all models achieving more than 60% similarity.

Finally, when looking at the whiteness tests in figure 7.3.9, we can see that the ARMAX, BJ and state-space structures validate the whiteness test.

Combining all these results together, the identified BJ structure seems to be the best fit for the G_{12} system.

4.2 Identification of Yaw Models

For the G_{21} model, as stated in section 4.1.1, we should only operate the vertical motor at voltage values for which the bottling in terms of pitch is minimal, in order to allow for the most optimal control of pitch later on. However, within this voltage range, the Aero2 hover has very insignificant yawing motion. Therefore, we will be neglecting this system and will be focusing our attention on identifying the G_{22} system.

When it comes to the G_{22} model, there is one particular challenge that comes with it, and it is the fact that it is an unstable system. When applying a step input, we can see that the system has an integrator, since the yaw angle diverges towards infinity, following a linear trajectory, as we can see in figure 4.2.1 below :

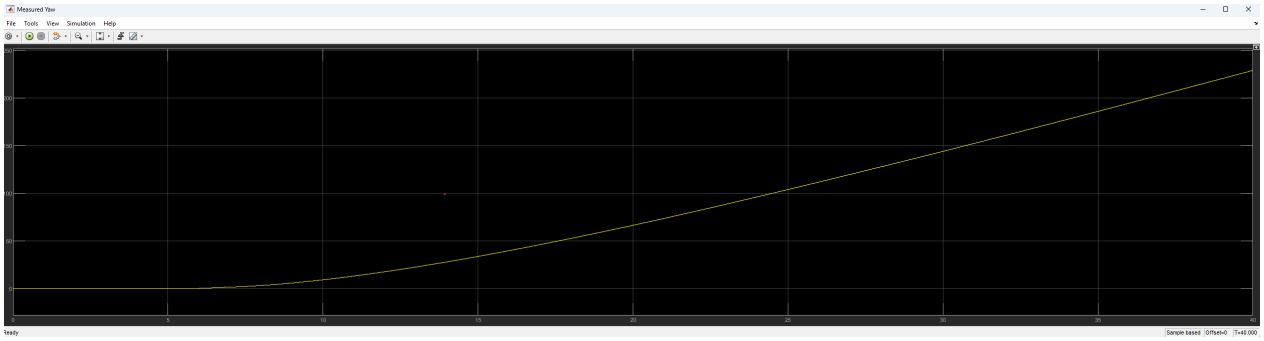


Figure 4.2.1: Step Response of G_{22} system (Step Time : 5s ; Step Value : 18V) : Yaw Angle (rad) as a function of Time (s)

Classical system identification methods, such as the ones we used in sections 4.1.1 and 4.1.2, would not be as efficient : when applying a PRBS input to our model, we get output measurements that has a general trend upwards, because of the integrator.

There are multiple ideas that we will go through in an attempt to identify the model :

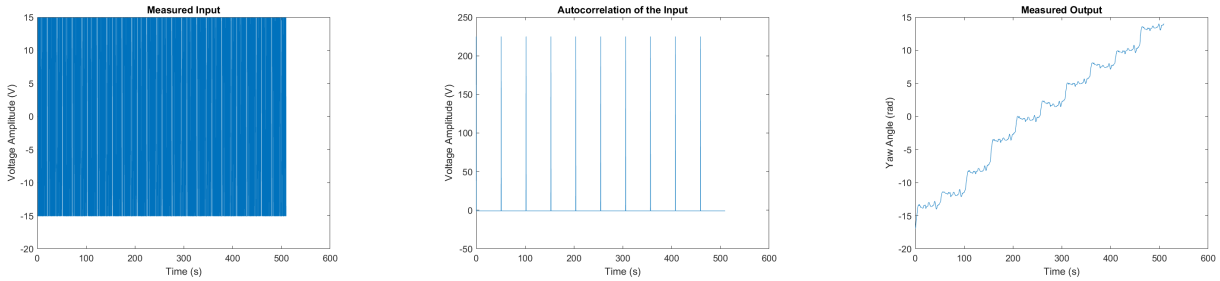
- Using the same techniques and attempt to fit a model directly to the data with the drift ;

- Identifying the yaw speed model then integrating it ;
- Closed-loop System Identification (refer to section 3.3).

We will go through each method individually and then compare them all together at the end.

4.2.1 Directly Fitting on Data with Drift

For this part we will be using the same methodology pipeline that we have been using in sections 4.1.1 and 4.1.2. We will be applying a PRBS signal of order 8, a voltage amplitude of 15V, a sampling period of 0.2s and 10 periods.



(a) PRBS Input (b) Autocorrelation of PRBS (c) Yaw Measurement

Figure 4.2.1.1: Input and Output Measurements for G_{22} (output measurement has been detrended)

When looking at figure 4.2.1.1c above, we can see that the output signal has this linear drift upwards, with each period being slightly elevated than the previous one. Let's move onto to the frequency response identification. Since the output signal signal is no longer periodic, we can no longer average out the periods together to reduce noise. Therefore, we can only use spectral analysis alongside windowing to achieve the best possible results :

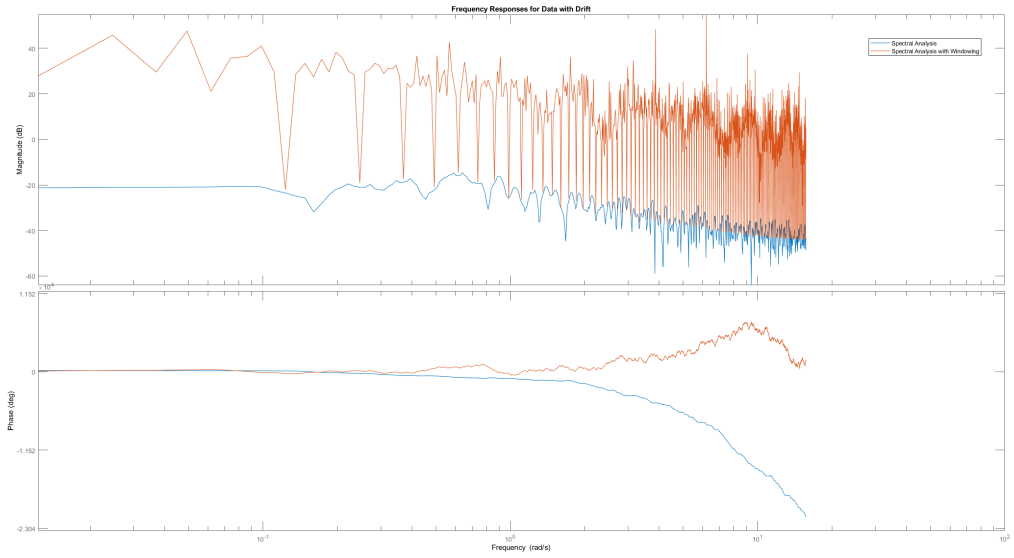


Figure 4.2.1.2: Frequency Response Identification for G_{22} Model

Looking at figure 4.2.1.2 above, we can see that although windowing does decrease the noise, especially at the higher frequencies, the windowed frequency response is still considerably noisy at the higher frequencies.

Now let's move onto the model order and structure estimation. When looking at the loss function, the pole/zero maps and the FIR parameters in figures 7.4.1, 7.4.2 and 7.4.3 respectively, we can deduce that the model order is either $\delta = 2$ or $\delta = 3$, which we will decide by trying both out in the parametric model identification and see which will yield better results. However what's problematic is the delay estimation using the FIR parameters : we can see that according to this method, the delay should be equal to the model order, since the confidence intervals of all the numerator parameters intersect with the horizontal axis. However, after trying out multiple configurations for the structure of the model, setting the delay equal to the model order doesn't lead to the best results.

The optimal model order structure that we have found, after multiple comparisons have been done, are $n_a = n_b = \delta = 2$ and $n_k = 1$. This gives us the following time-domain comparisons :

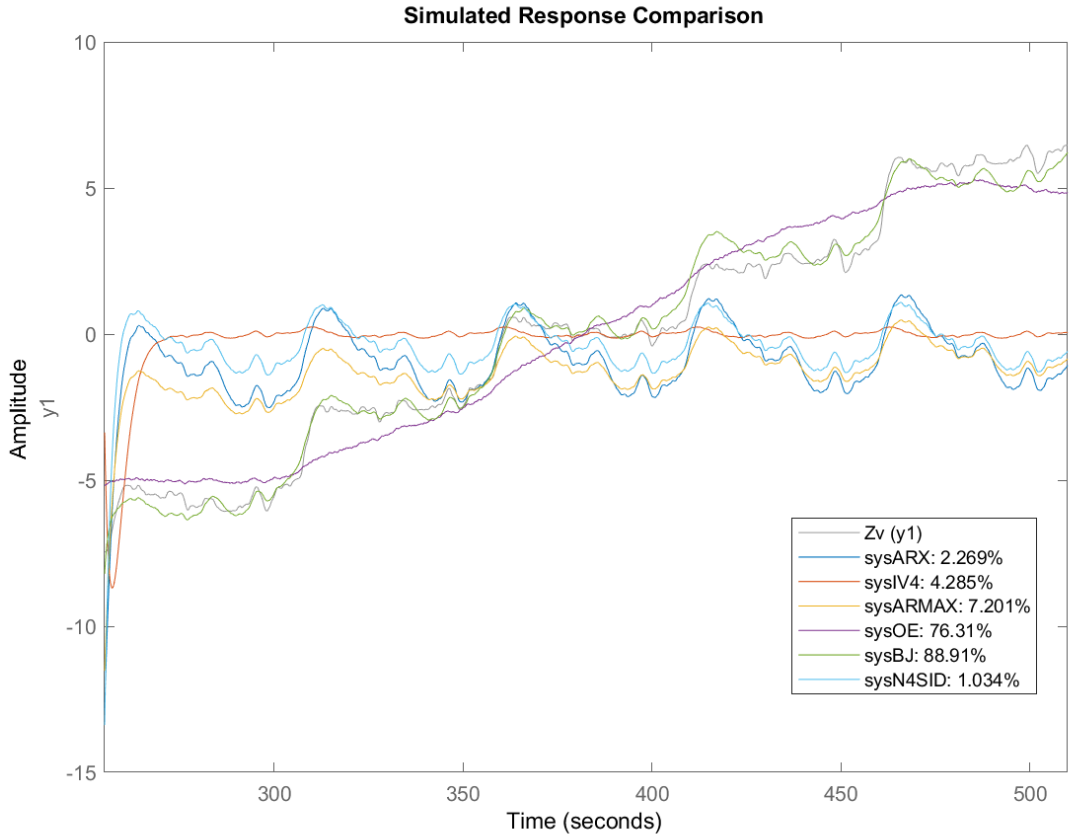


Figure 4.2.1.3: Time Domain Comparisons with all Parametric models for G_{22}

Within figure 4.2.1.3 above, most of the identified models yield extremely poor results, however both the Output Error and the Box Jenkins structures possess strong similarities with the measured data. We can see this in more detail within figure 4.2.1.4 below :

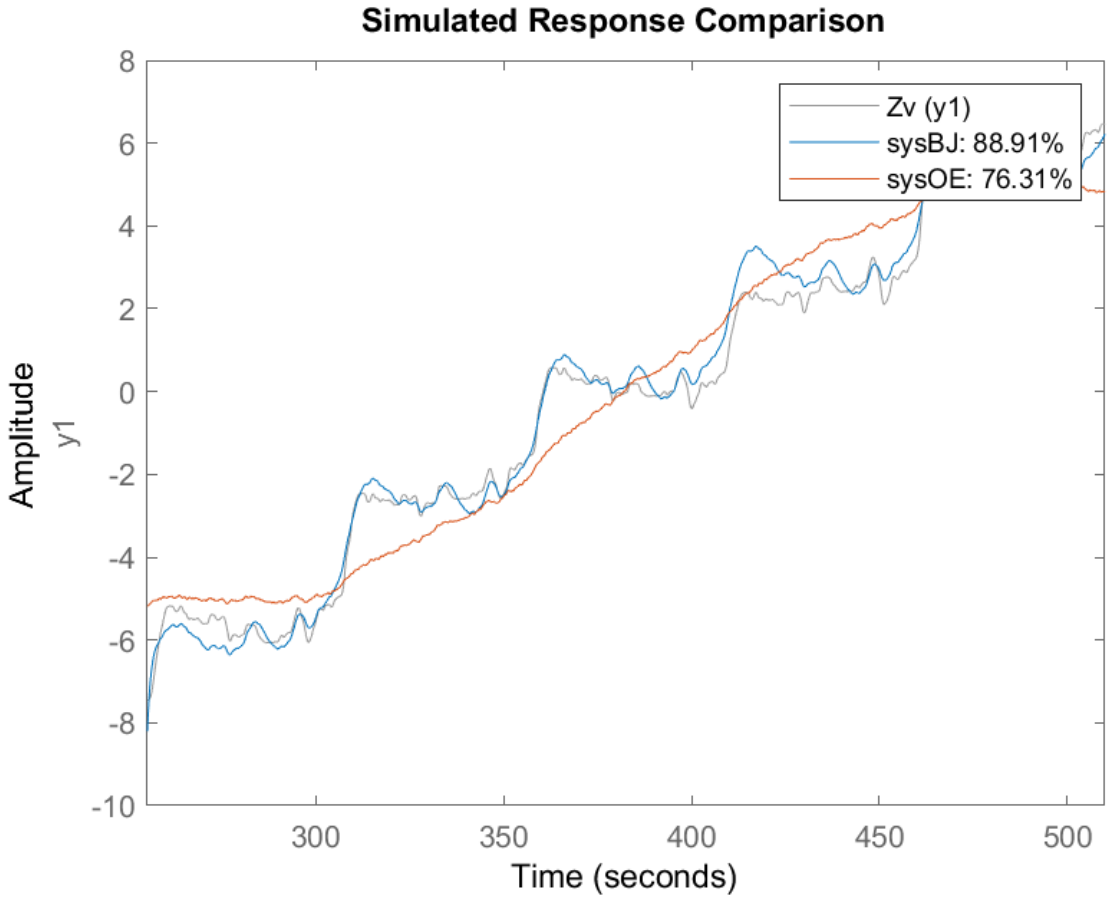


Figure 4.2.1.4: Time Domain Comparisons with OE and BJ models for G_{22}

We can see that the OE model, although displaying relatively good comparison results with 76.31%, isn't actually tracking the entire dynamics of the model, but rather only the linear drift within the data. However, when it comes to the BJ model not only tracks the drift, but it also tracks the model dynamics for each individual period, which explains the high similarity results of 88.91%.

We will not be looking at the frequency response comparisons, since the identified frequency response using spectral analysis and windowing is too noisy, which would directly lead to extremely poor comparison results. So we will move onto the noise models :

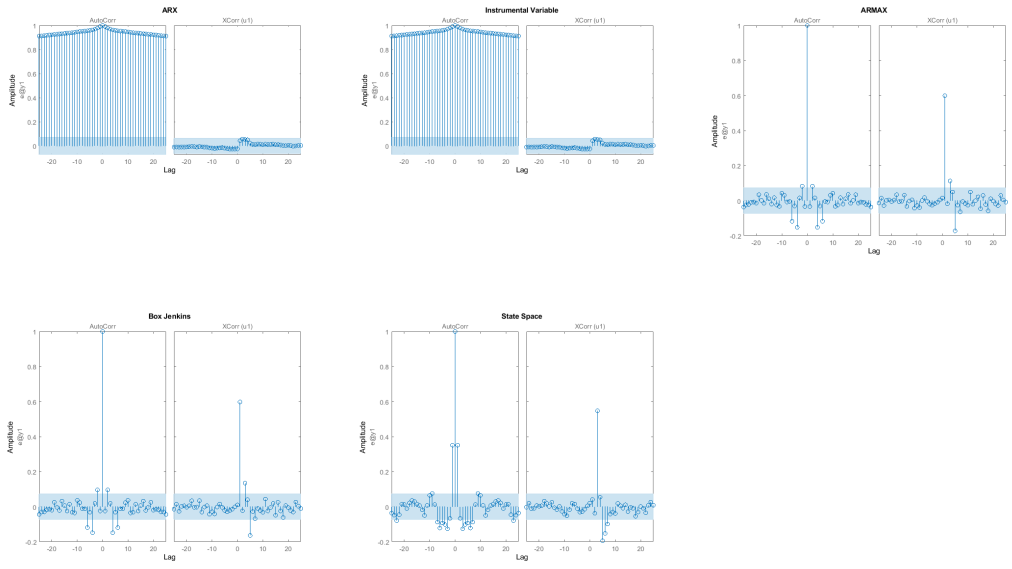


Figure 4.2.1.5: Noise Models for Different Parametric G_{22} Structures

Although a few of the models fail the whiteness test in figure 4.2.1.5, the one we are solely interested in is the BJ noise model, which does validate the whiteness test.

Therefore, by attempting to directly fit a model onto the data containing the drift, we do manage to obtain a good model for the time-domain comparisons (BJ Structure), however the identified frequency response is poor and all of the other models are very inaccurate, with the order and structure estimation being slightly misleading.

We will now move onto identifying the yaw speed model.

4.2.2 Identifying Yaw Speed

In order the to get the speed measurements, we will be using the Aero2's tachometer, which gives the number of encoder counts per second, and thus with the proper conversion gain, the angle per second. The tachometer output can be seen as the "o14003" output port within the "HIL Read Timebase" quanser block in figure 2.2.1.

We have seen that the system has an integrator in figure 4.2.1, this means that the yaw speed model itself is stable and directly connected to the yaw angle model through that integrator. This connection is highlighted in figure 4.2.2.1 below :

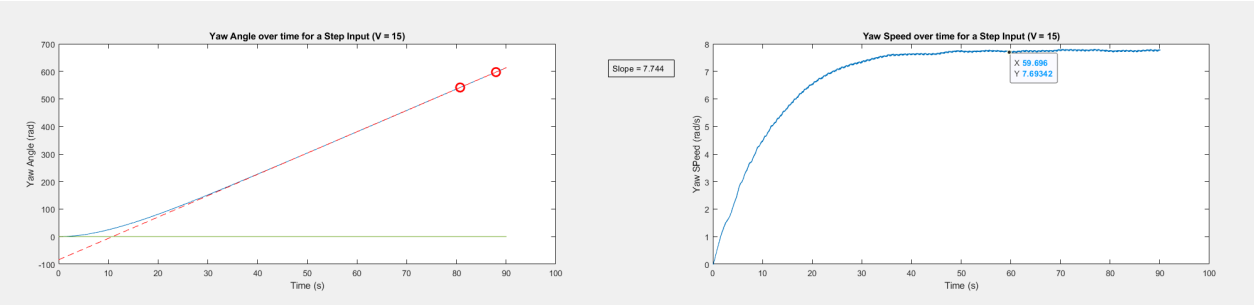


Figure 4.2.2.1: Yaw Angle (rad) and Yaw Speed (rad/s) Measurements for the same Step input

We can see that the settling amplitude of the yaw speed is nearly identical to the slope of the linear trend (for high time values) of the yaw angle, clearly indicating the relationship through an integrator of the two models.

This means that, in theory, identifying a parametric model for the speed system, and then integrating it should give us a good fit for the yaw angle model. This will be the objective of this part : we will be using the same methodology pipeline that we have used for both pitch models in section 4.1. The plots can be seen in section 7.5 within the appendix.

Looking at the plots within this section, we can see that the most suitable model for the yaw speed is the OE structure, fitted to the following parameters :

- $n_a = 4, n_b = 2\delta = 4$;
- $n_k = 1$.

Despite the poor frequency response comparison results, the time-domain results remain good. We now need to integrate the yaw speed and compare it with the yaw measurement for the same input.

Now that the speed model has been identified, we will use two different methods to integrate :

- Directly numerically integrating the yaw speed data and compare it to the measured yaw output ;
- Adding an integrator to one of the identified parametric speed models and then comparing it to the measured output.

Numerical Integration :

To integrate numerically, we can either use the already implemented MatLab function `cumtrapz`, or manually integrate the yaw speed data. The following listing 1 implements these two different methods :

Listing 1: Custom Intercorrelation Function

```
%% Load and Detrend Data

u = inputData.signals.values;
yaw = yawOUT.signals.values;
yawdot = yawSPEED.signals.values;
ts = inputData.time(2)-inputData.time(1);

[len_period, ~] = seqperiod(u); % Periodoc signal, retrieve
   number/length of periods
D = detrend(iddata(yawdot(5*len_period+1:end), u(5*len_period+1:
   end), ts));

%% Numerical Integration

% Manual Integration
pos = 0;
for ii=1:1:length(yawdot)
    pos(ii+1) = pos(ii)+yawdot(ii)*ts;
end

% Integration Using "cumtrapz"
cdistance = cumtrapz(ts, yawdot);

% Plot Results
figure(1);
plot(yaw);
hold on
plot(pos);
hold on;
plot(cdistance);

xlabel('Time (s)')
ylabel('Angle (rad)')
title ('Measured Yaw Compared with Integrated Data')
legend('Measured Yaw', 'Yaw by Manual Integration', 'Yaw by
   Cumtrapz');
```

From this code, we get the following results :

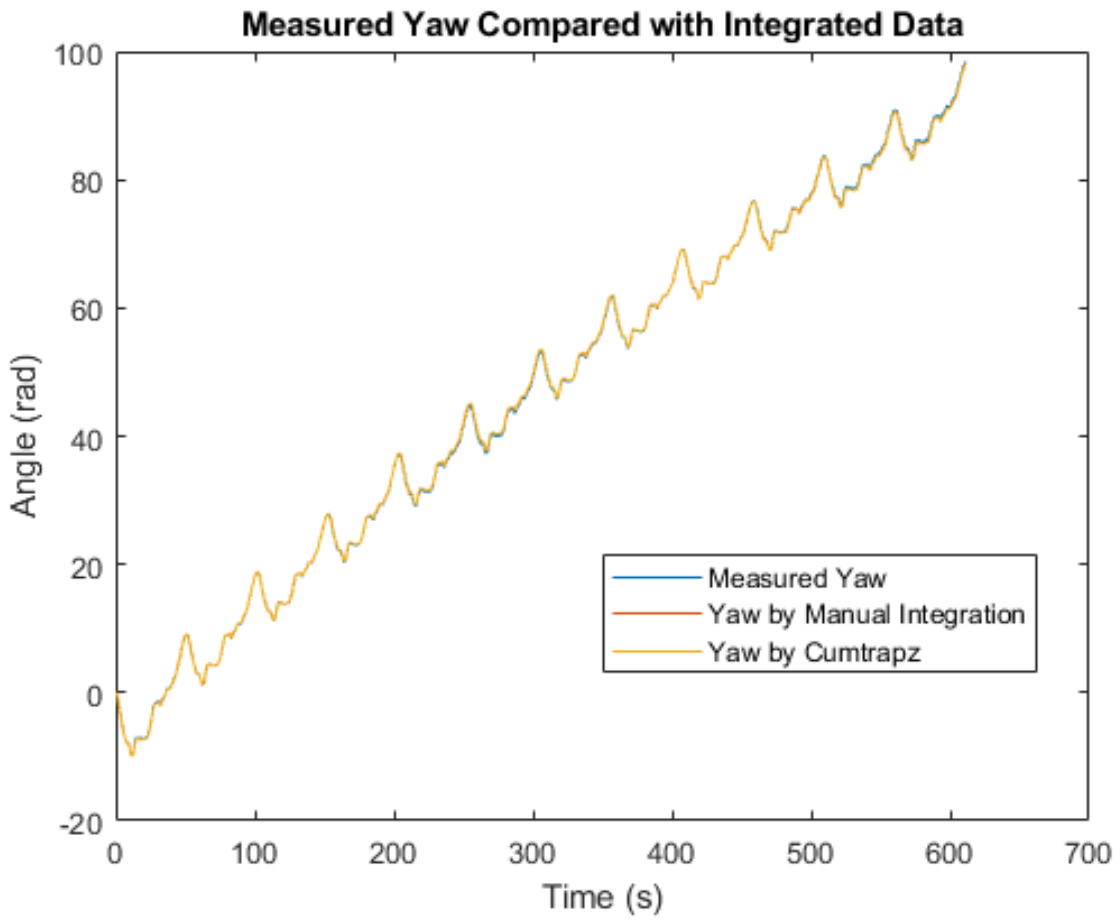


Figure 4.2.2.2: Measured Yaw Data vs Numerically Integrated Speed Data for G_{22}

We can see in figure 4.2.2.2 above that the integrated speed data coincide almost directly with the measured yaw data, justifying the direct correlation between the two measurement data. We will now move onto the integration of a selected parametric model.

Parametric Model Integration :

When looking at the time-domain comparisons within figure 7.5.7, we can see that the optimal model to select corresponds to that of the output error method. Thus, we will be using the listing 2 below in order to add an integrator to the model :

Listing 2: Custom Intercorrelation Function

```
function [R, h] = intcor(u, y)
%% Parametric Model Integration

% Select Model to Integrate
model = sysOE;

% Get Num and Denom and create TF (code depends on selected model
   ):
num = model.B;
denom = model.F;

G_speed = tf(num, denom, Ts);

% Create Integrator
K = 1;
z = tf('z',Ts);
integrator = K/(1-z);

% Integrate Speed Model;
G = integrator*G_speed;

% Test New Plant Model;
u = frest.PRBS('Order', 8, 'NumPeriods', 12, 'Amplitude', 2,...
    'Ts', Ts);
u = u.generateTimeseries;
u = u.data;
u = 15*u;

data = iddata(yaw, u, Ts);

figure;
compare(data, G);
title('Measurement Data vs Integrated Model Response');
```

From this code, we get the following results :

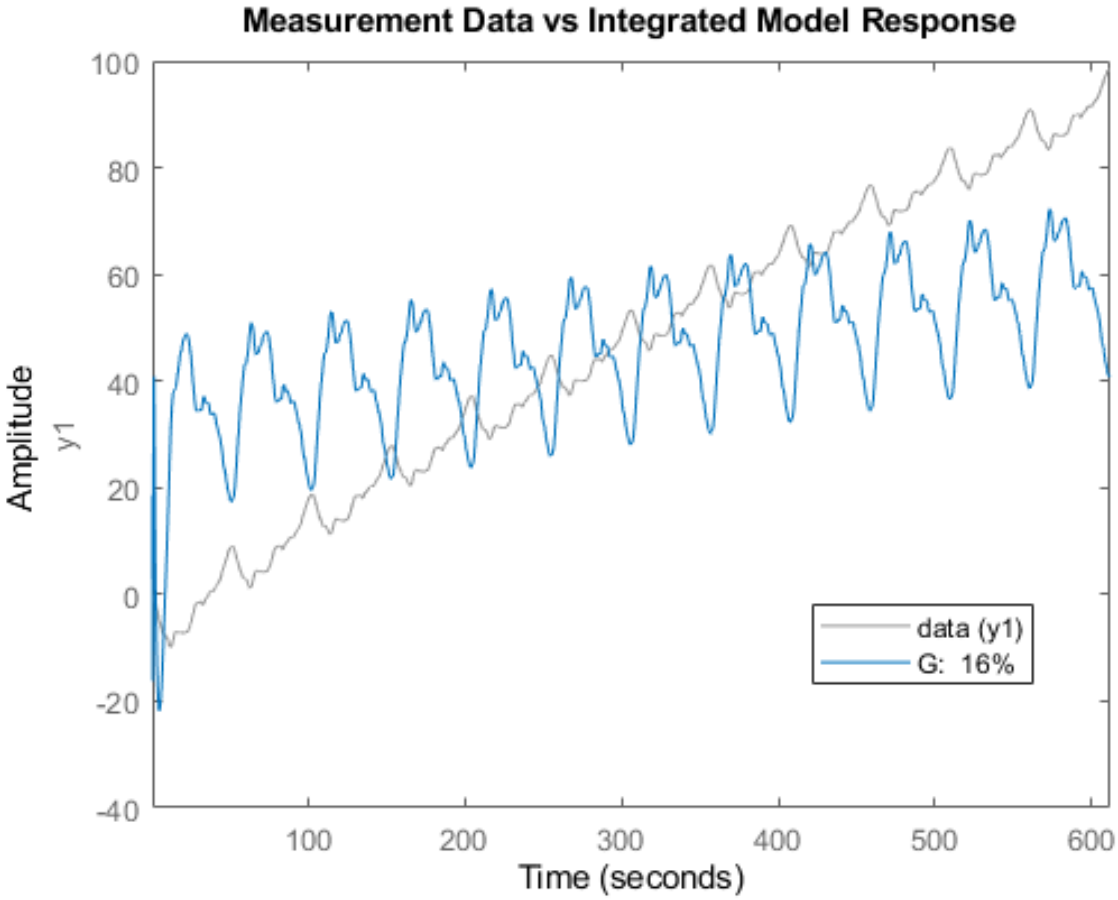


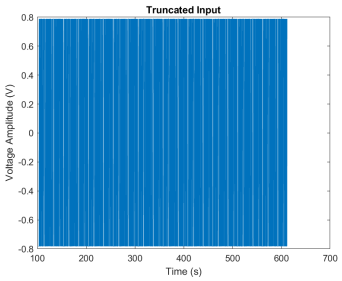
Figure 4.2.2.3: Measured Yaw Data vs Integrated OE Parametric Model Response for G_{22}

In figure 4.2.2.3 above, we can see that simply multiplying the model by an integrator and then comparing its response to the measured data does not yield great results, despite the correlation between the measured speed and yaw data.

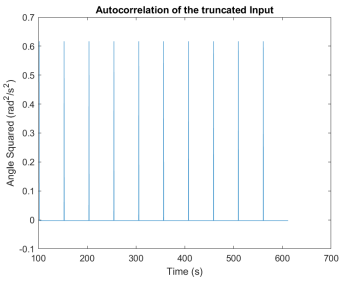
Finally, we will now move onto the closed-loop system identification.

4.2.3 Closed-Loop System Identification of G_{22}

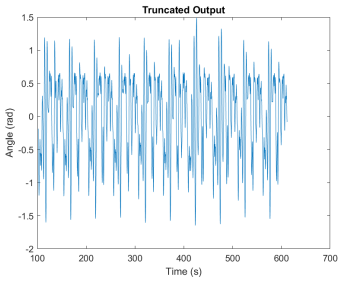
As mentioned previously in section 3.3, the point of closed-loop system identifying is to applying any stabilizing controller that will stabilize the output that corresponds to the model that we would like to identify. For our case, we decided to apply a simple proportional controller that allows this stabilization. Thus, in this case, the ”input” that we will be using for system identification is no longer the voltage input to the motors (since that is computed via the controller), but rather the reference signal. We will still be applying a PRBS input, however its amplitude would have to correspond to an angle that the model will have to track. We decide to apply an amplitude of $\frac{\pi}{4}$, and the measurements that we have obtained can be found in figure 4.2.3.1 below :



(a) PRBS Input



(b) Autocorrelation of PRBS



(c) Yaw Measurement

Figure 4.2.3.1: Input and Output Measurements for Closed-Loop System Identification of G_{22}

We can see that in figure 4.2.3.1c above that the yaw angle measurement does not diverge anymore towards infinity as time goes on thanks to the stabilizing controller. Using the structure-estimation plots in section 7.6, we will be fitting the parametric models to the following structure :

- $n_a = n_b = \delta = 4$;
- $n_k = 2$.

With these parameters, we obtain the following time domain comparisons in figure 4.2.3.2 below :

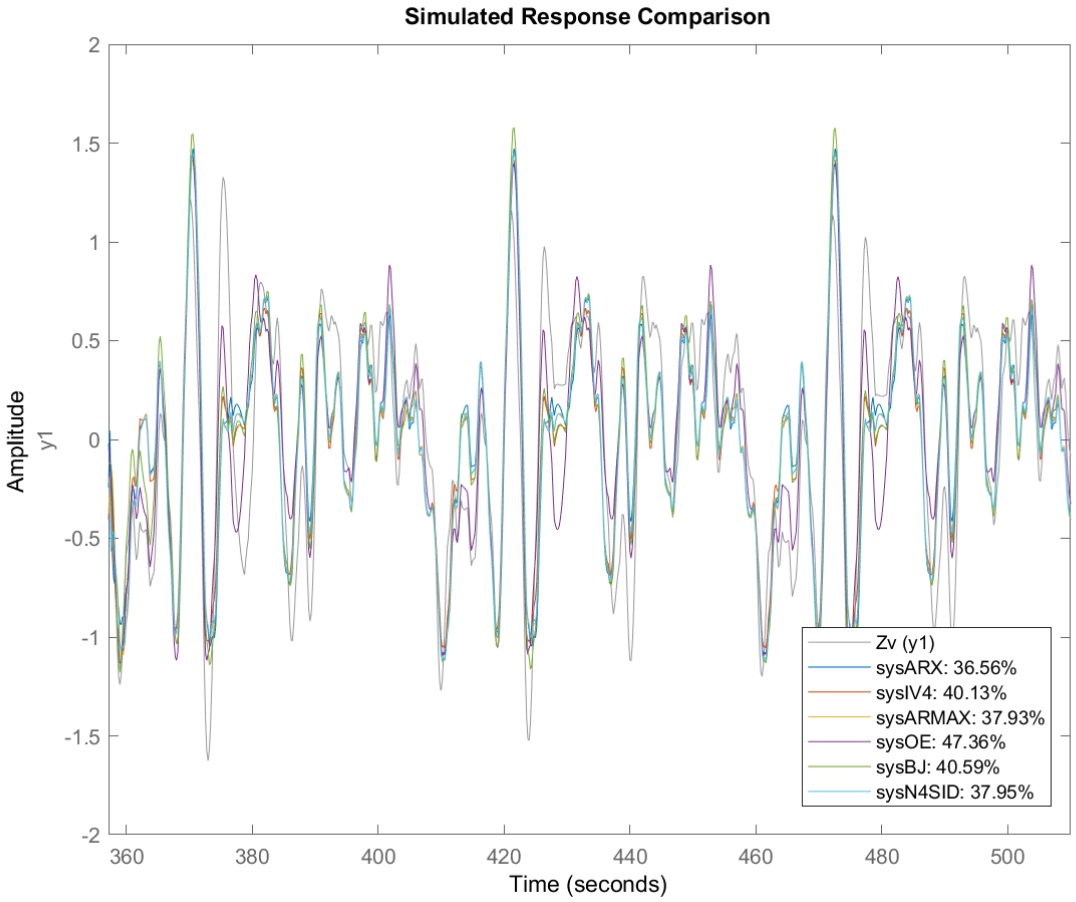


Figure 4.2.3.2: Time Domain Validation for Different Parametric Models for Closed-Loop Identification of G_{22}

In figure 4.2.3.2 above, we can see that the time-domain comparisons are not very good, despite stabilizing the output.

From all of the different methods that we have used, it seems like directly fitting a model to the data with drift might be the most optimal solution.

5 Summary

In this section we will be giving the final parametric models that we have obtained for G_{11}, G_{12} and G_{22}.

$\boldsymbol{G_{11}}$:

The discrete-time OE structure is given by :

$$y(t) = \frac{B(z)}{F(z)}u(t) + e(t), \tag{5.1}$$

with

$$\begin{aligned} B(z) &= -0.0002001z^{-1} + 0.000158z^{-2} + 0.0002335z^{-3} - 6.605 \cdot 10^{-5}z^{-4} \\ F(z) &= 1 - 1.569z^{-1} - 0.3776z^{-2} + 1.473z^{-3} - 0.5243z^{-4}. \end{aligned} \tag{5.2}$$

$\boldsymbol{G_{12}}$:

The discrete-time BJ structure is given by :

$$y(t) = \frac{B(z)}{F(z)}u(t) + \frac{C(z)}{D(z)}e(t), \tag{5.3}$$

with

$$\begin{aligned} B(z) &= 0.002249z^{-1} - 0.0002751z^{-2} - 0.001752z^{-3} \\ F(z) &= 1 - 2.346z^{-1} + 1.73z^{-2} - 0.3727z^{-3} \\ C(z) &= 1 - 0.3677z^{-1} - 0.337z^{-2} + 0.3458z^{-3} \\ D(z) &= 1 - 1.257z^{-1} - 0.4035z^{-2} + 0.6916z^{-3}. \end{aligned} \tag{5.4}$$

$\boldsymbol{G_{22}}$:

The discrete-time BJ structure is given by :

$$y(t) = \frac{B(z)}{F(z)}u(t) + \frac{C(z)}{D(z)}e(t), \tag{5.5}$$

with

$$\begin{aligned} B(z) &= 0.0002922z^{-1} + 0.000583z^{-2} \\ F(z) &= 1 - 1.9331.805z^{-1} + 0.80521.805z^{-2} \\ C(z) &= 1 + 0.8606z^{-1} + 0.2901z^{-2} \\ D(z) &= 1 - 1.805z^{-1} + 0.80521.805z^{-2}. \end{aligned} \tag{5.6}$$

6 Conclusion

This project has been an opportunity to explore different theoretical system identification techniques, and apply onto a real world Twin-Rotor-MIMO-System (TRMS). The objective was to be

able to identify a model for each of the aforementioned SISO systems in section 2, which we were able to obtain through the different MatLab scripts mentioned within the appendix in section 7.

However, this project has also shown some of the difficulties that come with these experiments. For example, data collection can take a lot of time, especially for different PRBS orders and sampling periods, making it complicated to be able to test out a high variety of combinations of these different parameters. Furthermore, the non-linearities of the Aero2 hover add an extra layer of difficulty to the identification process.

This report allows for a solid baseline for future projects on the Aero2 hover, providing not only identified SISO systems, but also a pipeline of the system identification process, which can allow for further system identifications if needed (to have a multi-model structure for each system, or to account for different regimes, such as fixing the horizontal axis of the hover at different angles).

7 Appendix

7.1 Custom MatLab Helper Functions

Listing 3: Custom Intercorrelation Function

```
function [R, h] = intcor(u, y)
% Calculates R_uy (h) for periodic signals

%%
M = size(u, 1); %length of sequence R
R = zeros(M, 1);

h = [0:1:M-1];

%half = ceil(M/2);
%h = [-half:1:half-1]'; %compute intcor on [-M/2, ..., M/2]

for i=1:M
    R(i, 1) = sum(u .* circshift(y, -h(i))); %R(i) = sum_k u(k)y(
       k-i)
end

R = 1/M * R;

end
```

Listing 4: Determine Frequency Response Using Fourier Analysis

```
function [G, w, newN] = freqRespFourier(y, u, Ts, per)
%freqRespFourier
%   FOURIER ANALYSIS, nonparametric frequency response
   identification
%   Inputs: y, u, Ts, per (period of the signal)
%   Outputs: G, frequency response of averaged data
%            w, vector of frequencies
%            newN, new size of cut data

%% Compute fourier transform for each period
% Separate u into 'per' periods and get FFT
N = size(u, 1);
M = floor(N/per);
newN = M;

up = zeros(M, per);
for i=1:per
    up(:, i) = u( ((i-1)*M + 1):(i*M), :);
end
Up = fft(up);

% Get fft of output (long input applied)
yp = zeros(M, per);
for i=1:per
    yp(:, i) = y( ((i-1)*M + 1):(i*M), :);
end
Yp = fft(yp);

% Average Y and U and get transfer function
Y = mean(Yp,2);
U = mean(Up,2);
G = Y./U;

stepsize = (2*pi/Ts)/M;
w = 0:stepsize:(M-1)*stepsize;

end
```

Listing 5: Determine Frequency Response Using Fourier Analysis + Windowing

```
function [G, w, newN] = freqRespWindowFourier(y, u, Ts, per,
   window)
%   Fourier ANALYSIS, nonparametric frequency response
   identification
%   using avraging followed by windowing
%   Inputs: y, u, Ts, per (number of periods to average on)
%           window (string, either 'hann' or 'hamming')
%   Outputs: G, frequency response of averaged+windowed data
%            w, vector of frequencies
%            newN, new size of averaged data

%% Modify hann and hamming windows
% To better fit on our Ruu and Ryu functions
N = size(u, 1);

hannN = hann(N);
hannBetter = zeros(size(hannN));
n=N/2; %shift
hannBetter(1:end-n) = hannN(n+1:end);

% Better hamming window
hammN = hamming(N);
hammBetter = zeros(size(hammN));
n=N/2; %shift
hammBetter(1:end-n) = hammN(n+1:end);

%% Apply windowing

if strcmp(window, 'hann')
    uWindow = u .* hannBetter;
    yWindow = y .* hannBetter;
end

if strcmp(window, 'hamming')
    uWindow = u .* hammBetter;
    yWindow = y .* hammBetter;
end

%% Compute Fourier Transform
% Separate u into 'per' periods and get FFT
N = size(u, 1);
M = floor(N/per);
newN = M;
```

```
42
43 upWindow = zeros(M, per);
44 for i=1:per
45     upWindow(:, i) = uWindow( ((i-1)*M + 1):(i*M), :);
46 end
47
48 % Get fft of output (long input applied)
49 ypWindow = zeros(M, per);
50 for i=1:per
51     ypWindow(:, i) = yWindow( ((i-1)*M + 1):(i*M), :);
52 end
53
54 %% Apply Averaging
55
56 % Get FFTs of windowed functions
57 UpWindow = fft(upWindow);
58 YpWindow = fft(ypWindow);
59
60 % Average Y and U and get transfer function
61 Y = mean(YpWindow,2);
62 U = mean(UpWindow,2);
63 G = Y./U;
64
65 stepsize = (2*pi/Ts)/M;
66 w = 0:stepsize:(M-1)*stepsize;
67
68 end
```

Listing 6: Determine Frequency Response Using Spectral Analysis

```
function [G, w] = freqRespSpectralAnalysis(y, u, Ts)
%freqRespAveraging
%   SPECTRAL ANALYSIS, nonparametric frequency response
   identification
%   Inputs: y, u, Ts,
%   Outputs: G, frequency response of averaged data
%            w, vector of frequencies
%% Compute freq response using spectral analysis
N = size(u,1);
stepsize = (2*pi/Ts)/N;
w = 0:stepsize:(N-1)*stepsize;

[Ryu, ~] = intcor(u, y);
[Ruu, ~] = intcor(u, u);

Phiyu = fft(Ryu);
Phiuu = fft(Ruu);

G = Phiyu./Phiuu;

end
```

Listing 7: Determine Frequency Response Using Spectral Analysis + Windowing

```
function [G, w] = freqRespWindow(y, u, Ts, window)
%freqRespWindow
%   SPECTRAL ANALYSIS, nonparametric frequency response
   identification
%   using windowing
%   Inputs: y, u, Ts, window (string, either 'hann' or 'hamming')
%   Outputs: G, frequency response of windowed data
%            w, vector of frequencies

%% Modify hann and hamming windows
% To better fit on our Ruu and Ryu functions
N = size(u, 1);

hannN = hann(N);
hannBetter = zeros(size(hannN));
n=N/2; %shift
hannBetter(1:end-n) = hannN(n+1:end);

% Better hamming window
hammN = hamming(N);
hammBetter = zeros(size(hammN));
n=N/2; %shift
hammBetter(1:end-n) = hammN(n+1:end);

%% Compute correlations
stepsize = (2*pi/Ts)/N;
w = 0:stepsize:(N-1)*stepsize;

[Ryu, ~] = intcor(u, y);
[Ruu, ~] = intcor(u, u);

%% Apply windowing
if strcmp(window, 'hann')
    RyuHann = Ryu .* hannBetter;
    RuuHann = Ruu .* hannBetter;
    G = fft(RyuHann)./fft(RuuHann);
end

if strcmp(window, 'hamming')
    RyuHamm = Ryu .* hammBetter;
    RuuHamm = Ruu .* hammBetter;
    G = fft(RyuHamm)./fft(RuuHamm);
end
```

```
43
44 end
```

Listing 8: Determine Frequency Response Using Spectral Analysis + Windowing + Averaging

```
function [G, w, newN] = freqRespAveraging(y, u, Ts, m)
%freqRespAveraging
%   SPECTRAL ANALYSIS, nonparametric frequency response
   identification
%   using averaging
%   Inputs: y, u, Ts, m (number of groups to average on)
%   Outputs: G, frequency response of averaged data
%            w, vector of frequencies
%            newN, new size of averaged data

%% Cut the data into m groups and average
% Separate u into m groups
N = size(u, 1);
newSize = floor(N/m);

allRyu = zeros(newSize, m);
allRuu = zeros(newSize, m);
for i = 1:m
    uCut = u( ((i-1)*newSize + 1):(i*newSize), :);
    yCut = y( ((i-1)*newSize + 1):(i*newSize), :);

    [currentRyu, ~] = intcor(uCut, yCut);
    [currentRuu, ~] = intcor(uCut, uCut);

    allRyu(:, i) = currentRyu;
    allRuu(:, i) = currentRuu;
end

allPhiyu = fft(allRyu);
allPhiuu = fft(allRuu);

% Average Phiyu and Phiuu and get transfer function
meanPhiyu = mean(allPhiyu,2);
meanPhiuu = mean(allPhiuu,2);
gAvg = meanPhiyu./meanPhiuu;

% Plot
newStepsize = (2*pi/Ts)/newSize;
newW = 0:newStepsize:(newSize-1)*newStepsize;
%Gavg = frd(gAvg(1:newSize/2), newW(1:newSize/2));

% Set outputs
newN = newSize;
```

```
43 G = gAvg;
44 w = newW;
45
46 end
```

Listing 9: Determine Q Matrix For Subspace Method

```
function [Q, Y, Uperp] = stateSpaceComputeQ(y, u, r)
%   Computes Q Matrix for Subspace Method
%   stateSpaceComputeQ
%   Compute Q=Y*Uperp
%   Inputs: y, u, r (upper bound on system order)
%   Outputs: Q,
%            Y,
%            Uperp,
%% Construct Y and U, then compute Q
N = size(u,1)-r+1; %available data points, slide 23

Y = zeros(r, N);
U = zeros(r, N);
for i=1:size(Y,2)
    Y(:, i) = y(i:(i+r-1));
    U(:, i) = u(i:(i+r-1));
end

Uperp = eye(N) - U' * inv(U*U') * U;
Q = Y*Uperp;

end
```

Listing 10: Identify State-Space Matrices Using Subspace Method

```
function [A, B, C, D] = stateSpaceGetMatrices(Q, n, r, u, y)
%stateSpaceGetMatrices
%   Compute A, B, C and D=0
%   Inputs: Q, n (estimated order), r (estimated order),  u (
   input), y (output)
%   Outputs: A,
%            B,
%            C,
%            D = 0

%% Compute A and C
Or = Q(:, 1:n); %extended observability matrix
ny = size(Or,1)/r;

C = Or(1:ny, :);

G = Or(1:((r-1)*ny), :); %first (r-1)ny rows of Or
F = Or(2*ny:end, :); %last (r-1)ny rows of Or

A = pinv(G'*G) * G' * F;

%% Estimate B, compute D
% Construct the signals u_f
q = tf('z');
F = C*inv(q*eye(n) - A);
uf = zeros(size(u,1), n);
for i=1:n
    uf(:,i) = lsim(F(i), u);
end

% Compute estimates for B and D
B = pinv(uf) * y;
D = 0;

end
```

7.2 MatLab Files For System Identification Experiments (Applying Signals to Aero2 + Analyzing Measurements)

Listing 11: G-inputOutput.m : Code to Apply a Custom PRBS Signal to Aero2 using Simulink

```
1
2  clear all; close all; clc
3
4  %% Constants
5
6  Ts = 0.1; %[s]
7  N_20000 = 20000/Ts; %length of signal array (=20'000 seconds)
8  t_20000 = [0:Ts:(N_20000*Ts)]';
9
10 %% Applying prbs signal to system
11
12 u = frest.PRBS('Order', 10, 'NumPeriods', 12, 'Amplitude', 2,...
13                          'Ts', Ts);
14 u = u.generateTimeseries;
15 u = u.data;
16 len = size(u, 1);
17
18 t_final = t_20000(1:len, :);
19 simin.time = t_final;
20
21 simin.signals.values = u;
```

Listing 12: a1-inputOutputIdentification.m : Code to Analyze Input and Output Measurements + Filtering Out First Couple of Periods

```
1
2  clear all; close all; clc
3
4  %% Get experimental data
5
6  load('G22_open_n10\inputDATA_V18_T01_n10.mat');
7  load('G22_open_n10\yawSPEED_V18_T01_n10');
8
9  Ts = 0.1;
10 u = inputData.signals.values;
11
12 y = yawSPEED.signals.values;
13
14 time = yawSPEED.time;
15
16 data = iddata(y,u,Ts);
17 data = detrend(data, 0);
18
19 u = data.u;
20 y = data.y;
21  %% Input signal is PRBS, compute its period
22
23 [Ruu, ~] = intcor(u, u);
24
25 figure;
26 plot(time, Ruu);
27 xlabel('Time (s)')
28 ylabel('Voltage Amplitude (V)')
29 title('Autocorrelation of the Input')
30
31 figure;
32 plot(time, y);
33 xlabel('Time (s)')
34 ylabel('Yaw Angle (rad)')
35 title('Measured Output')
36
37 %% Filter Out (Determine How to truncate first periods)
38
39 figure;
40 plot(Ruu);
41 xlabel('Time (s)')
42 ylabel('Voltage Amplitude (V)')
43 title('Autocorrelation of the Input')
```

```
44
45 u_trunc = u(2047:end, 1);
46 [Ruu_trunc, ~] = intcor(u_trunc, u_trunc);
47
48 figure;
49 plot(Ruu_trunc);
50 xlabel('Time (s)')
51 ylabel('Voltage Amplitude (V)')
52 title('Autocorrelation of the truncated Input')
```

Listing 13: a2-freqRespIdentification.m : Code to Compute Frequency Responses Using Different Methods

```
1
2  clear all; close all; clc
3
4  %% Get experimental data
5
6  load('G22_open_n10\inputDATA_V18_T01_n10.mat');
7  load('G22_open_n10\yawSPEED_V18_T01_n10');
8
9  Ts = 0.1;
10
11 % Correctly filter out initial periods from measurements and time
       vector
12 u = inputData.signals.values;
13 u = u(2047:end, 1);
14 y = yawSPEED.signals.values;
15 y = y(2047:end, 1);
16 time = yawSPEED.time;
17 time = time(2047:end, 1);
18
19 data = iddata(y,u,Ts);
20 data = detrend(data, 0);
21
22 u = data.u;
23 y = data.y;
24 N = size(u,1);
25
26 %% Plot Output
27 figure;
28 plot(time, y);
29 xlabel('Time (s)')
30 ylabel('Yaw Angle (rad)')
31 title('Selected Output')
32
33 %% Compute freq response using Spectral Analysis
34 [Gspec, wspec] = freqRespSpectralAnalysis(y, u, Ts);
35 freqRespSpectralAnalysis = frd(Gspec(1:N/2), wspec(1:N/2));
36
37 %% Compute freq response using windowing (Spectral Analysis)
38 [Ghann, whann] = freqRespWindow(y, u, Ts, 'hamming');
39 freqResponseWindow = frd(Ghann(1:N/2), whann(1:N/2));
40
41 %% Compute freq response using averaging (Spectral Analysis)
42 [Gavg, wavg, newN] = freqRespAveraging(y, u, Ts, 10); %10=nbr of
```

```
   groups
freqRespAvg = frd(Gavg(1:newN/2), wavg(1:newN/2));

%% Compute freq response using Fourier analysis
per = 10; %cf part 1
[Gfourier, wfourier, Nfourier] = freqRespFourier(y, u, Ts, per);
freqRespFourier = frd(Gfourier(1:Nfourier/2), wfourier(1:Nfourier
   /2));

%% Compute freq response using mix of windowing then averaging
% (With Spectral Analysis)
[Gmix, wmix, newNmix] = freqRespWindowAndAvg(y, u, Ts, 10, 'hann'
   );
freqRespMix = frd(Gmix(1:floor(newNmix/2)), wmix(1:floor(newNmix
   /2)));

%% Compute freq response using mix of Fourier Analysis and
   Windowing
[GmixF, wmixF, newNmixF] = freqRespWindowAndAvgFourier(y, u, Ts,
   10, 'hann');
freqRespMixF = frd(GmixF(1:floor(newNmixF/2)), wmixF(1:floor(
   newNmixF/2)));

%% Compare all Frequency Responses
% right click -> options -> wrap phase -360
figure;
bode(freqRespSpectralAnalysis);
hold on
bode(freqResponseWindow);
hold on
bode(freqRespAvg);
hold on
bode(freqRespFourier);
hold on
bode(freqRespMix);
hold on
bode(freqRespMixF);
legend('Spectral Analysis', 'Spectral Analysis + Windowing',...
    'Spectral Analysis + Averaging', 'Fourier Analysis',...
    'Windowing + Averaging + Spectral Analysis)',...
    'Windowing + Fourier Analysis');
title('All Frequency Responses');

%% Compare Two Optimal Frequency Responses

```

```
figure;
bode(freqRespMix);
hold on
bode(freqRespMixF);
legend('Spectral Analysis', 'Fourier Analysis');
title('Optimal Frequency Responses (Combining Windowing and
   Averaging)');
```

Listing 14: a3-orderAndStructureEstimation.m : Code to Determine Order and Structure of the System using Subspace method + Loss Function + Pole-Zero Maps + FIR and ARX Structures

```
1
2  close all; clear all; clc
3
4  %% Get experimental data
5
6  load('G22_open_n10\inputDATA_V18_T01_n10.mat');
7  load('G22_open_n10\yawSPEED_V18_T01_n10');
8
9  Ts = 0.1; %20ms, cf moodle 2552
10
11 % Correctly filter out initial periods from measurements and time
       vector
12 u = inputData.signals.values;
13 u = u(2047:end, 1);
14
15 y = yawSPEED.signals.values;
16 y = y(2047:end, 1);
17
18 time = yawSPEED.time;
19 time = time(2047:end, 1);
20
21 data = iddata(y,u,Ts);
22 detrend(data);
23
24 u = data.u;
25 y = data.y;
26 N = size(u,1);
27
28 figure;
29 plot(time, y);
30 xlabel('Time (s)')
31 ylabel('Yaw Angle (rad)')
32 title('Measured Output')
33
34
35 %% Identify state space model (to estimate order)
36 % Compute Q
37 r = 15; %upper bound on system order
38 [Q, ~, ~] = stateSpaceComputeQ(y, u, r);
39
40 % Estimate order using rank of Q
41 S = svd(Q); %singular values
42
```

```
figure;
plot(S, 'o');
title('Singular Values of Q')

%% Test state space model
% Get state space model;
n = 4; %estimated order, consider the first 8 as nonzero
[A, B, C, D] = stateSpaceGetMatrices(Q, n, r, u, y);

% Define system using identified state space model
sys = ss(A,B,C,D,Ts);
[y_sys, t_sys, x_sys] = lsim(sys, u);

% Plot and compute error
figure
plot(t_sys, y_sys)
hold on
plot(t_sys, y)
legend('Simulated output', 'Real (measured) output')

error = sum( (y - y_sys).^2 );
fprintf('The value of the error is %s\n', error);


%% Loss function
na = 0;
nb = 0;
nk = 3;
deltaMax = 10;
lossFunctions = zeros(1, deltaMax);


for i = 1:deltaMax
    na = i;
    nb = i;
    sys = arx(data,[na nb nk]);
    loss = sys.EstimationInfo.LossFcn;
    lossFunctions(1, i) = loss;
end
figure;
plot(lossFunctions, 'o')
title('Loss Function');
%deduce order is 3-9

%% Order validation using pole/zero diagrams
```

```
88  %watch out for pole/zero cancellations using the confidence
       interval
89  %increase order and look at which poles stay constant, they are
       the true
90  %poles of the system
91  for i = 1:7
92      delta = i;
93
94      nk = 1; %d
95      sysArmax = armax(data, [delta delta delta nk]); %[na=n nb=m
           nc nk]
96
97      figure;
98      h=iopzplot(sysArmax);
99      axis([-2 2 -3 3])
100     showConfidence(h,2)
101     title("Pole/Zero Map with \delta = " + delta)
102 end
103
104
105 %% Estimate delay nk using OE
106 % Use FIR model, cf slide 58
107 delta = 4; %deduce that model order is 8
108 m = 50; %=nb, needs to reach steady state
109 nk = 1;
110 sysFIR = oe(data, [m 0 nk]); %oe with 0 degree F is FIR
111
112 figure;
113 stairs(0:m, sysFIR.b); grid;
114 title('FIR model with m = 50');
115
116 figure;
117 errorbar(sysFIR.b(1:5), 2*sysFIR.db(1:5)); %only need to check
       first few coeffs
118 title('FIR: Paramters and their confidence interval');
119 grid;
120
121 %% Estimate delay nk using ARMAX (better because oscillatory
       system)
122 % Use ARMAX model, cf slide 58
123 nk = 1;
124 sysARMAX = armax(data, [delta delta delta nk]); %oe with 0 degree
       F is FIR
125
126 figure;
```

```
127 stairs(1:size(sysARMAX.b, 2), sysARMAX.b); grid;
128 title('ARMAX model');
129
130 figure;
131 errorbar(sysARMAX.b(1:5), 2*sysARMAX.db(1:5)); %only need to
      check first few coeffs
132 title('ARMAX: Paramters and their confidence interval');
133 grid;
134
135 d = 1; %deduce that delay d=1 because only 1 coefficient of B is
      close to 0
136 %number of parameters in B is =delta, because delay d=1 (si d=2
      alors
137 %nbParamInB=delta-1) (maybe wrong)
138 %sinon vu que y a 3 zeros, nb=3
139 %^ ALL OF THE COMMENTS ABOVE IS FOR THE CE, NOT THE PROJECT
140 %CF SLIDE 52
141
142 %% Comparison with struc, arxstruc, selstruc
143 nn = struc(1:15, 1:15, 1:15);
144 V = arxstruc(data, data, nn);
145 selstruc(V);
```

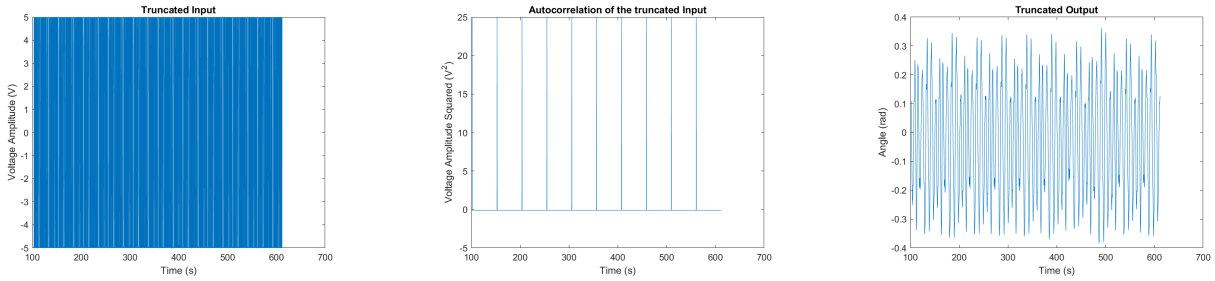
Listing 15: a4-severalModelsIdentification.m : Code to Identify Parametric Models and Compare with Measurement Data

```
1
2  close all; clear all; clc
3
4  %% Get experimental data
5
6  load('G22_open_n10\inputDATA_V18_T01_n10.mat');
7  load('G22_open_n10\yawSPEED_V18_T01_n10');
8
9  Ts = 0.1; %20ms, cf moodle 2552
10
11 % Correctly filter out initial periods from measurements and time
       vector
12 u = inputData.signals.values;
13 u = u(2047:end, 1);
14
15 y = yawSPEED.signals.values;
16 y = y(2047:end, 1);
17
18 time = yawSPEED.time;
19 time = time(2047:end, 1);
20
21 data = iddata(y,u,Ts);
22 data = detrend(data);
23 N = size(u,1);
24
25 u = data.u;
26 y = data.y;
27
28 figure;
29 plot(time, y);
30 xlabel('Time (s)')
31 ylabel('Yaw Angle (rad)')
32 title('Measured Output')
33
34 %% Divide data into train and validation (for identification and
      validation)
35 ratio = 0.7;
36 N = size(u,1);
37
38 Zi = data(1:ceil(ratio*N));    %identification
39 Zv = data(ceil(ratio*N)+1:N); %validation
40
41 Zi = detrend(Zi);
```

```
Zv = detrend(Zv);
%% Estimated parameters from previous analysis

na = 4;
nb = 4;
nk = 1;

%% Parametric identification

sysARX = arx(Zi, [na nb nk]);
sysIV4 = iv4(Zi, [na nb nk]);
sysARMAX = armax(Zi, [na nb na nk]);
sysOE = oe(Zi, [nb na nk]);
sysBJ = bj(Zi, [nb na na na nk]);
sysN4SID = n4sid(Zi, na);

%% Compare with measured data

figure;
compare(Zv, sysARX, sysIV4, sysARMAX, sysOE, sysBJ, sysN4SID);

%% Compute frequency response using nonparam methods of
   validation data
% Select Appropriate Frequency Response from a2
N = size(Zv.u, 1);
[Gavg, wavg, newN] = freqRespWindowAndAvg(Zv.y, Zv.u, Ts, 10*
   ratio, 'hann');
validationSystem = frd(Gavg(1:ratio*newN), wavg(1:ratio*newN));

%% Freq response of parametric methods
figure;
compare(validationSystem, sysARX, sysIV4, sysARMAX, sysOE, sysBJ,
    sysN4SID);

%% Resid (Noise Models)

figure;

subplot(3,3,1)
resid(Zv,sysIV4); title('ARX');

subplot(3, 3, 2)
resid(Zv,sysIV4); title('Instrumental Variable');

subplot(2,3,3)
```

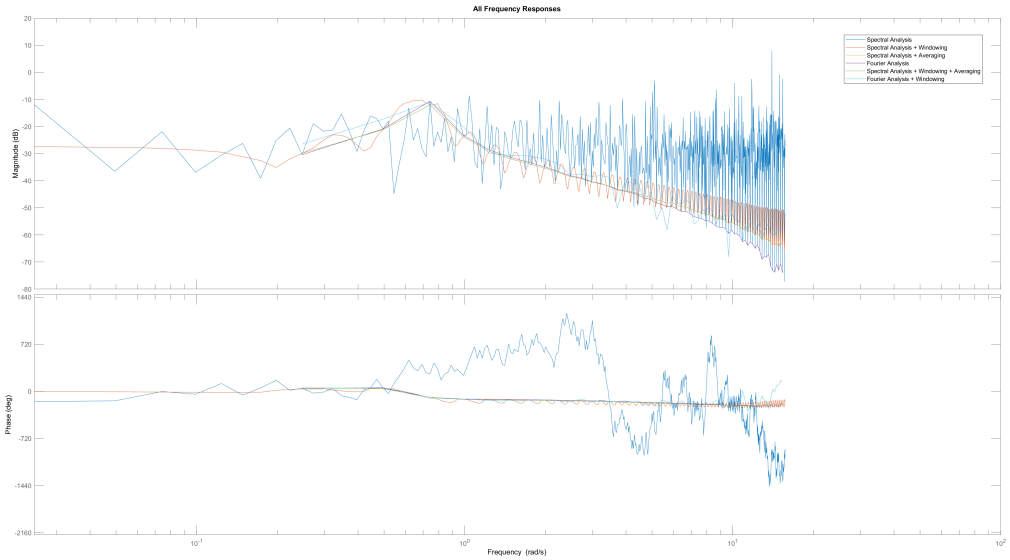
```
resid(Zv,sysARMAX); title('ARMAX');

subplot(2,3,4)
resid(Zv,sysOE); title('Output Error');

subplot(2,3,5)
resid(Zv,sysBJ); title('Box Jenkins');

subplot(2,3,6)
resid(Zv,sysN4SID); title('State Space');
```

7.3 G_{12} Identification Plots

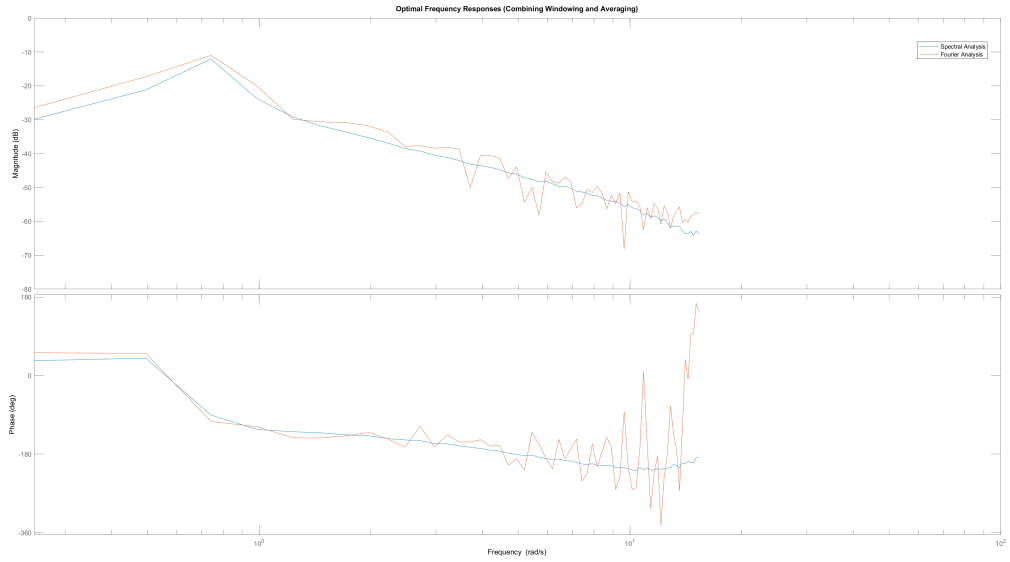


(a) PRBS Input (b) Autocorrelation of PRBS (c) Pitch Measurement

Figure 7.3.1: Truncated Input and Output Measurements for G_{12}



(a) Frequency Response Identification with All Methods



(b) Frequency Response Identification with Two Best Methods

Figure 7.3.2: Frequency Response Identification for G_{12}

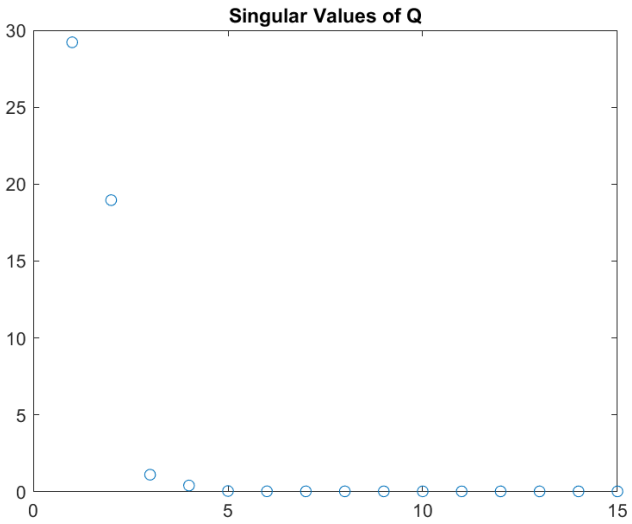


Figure 7.3.3: Singular Values of Q matrix for G_{12}

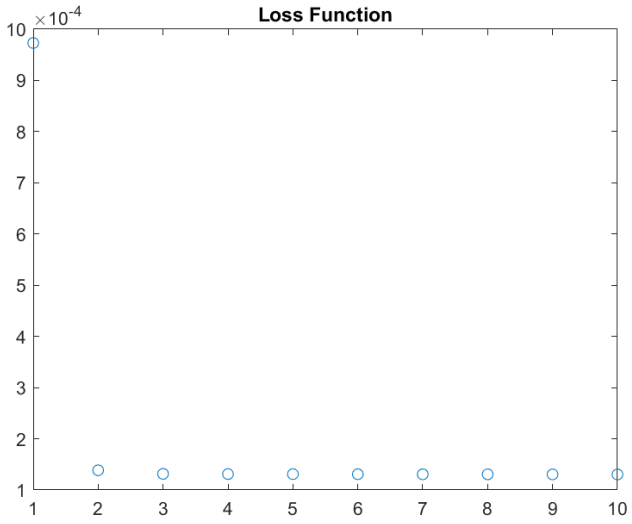


Figure 7.3.4: Loss Function for G_{12}

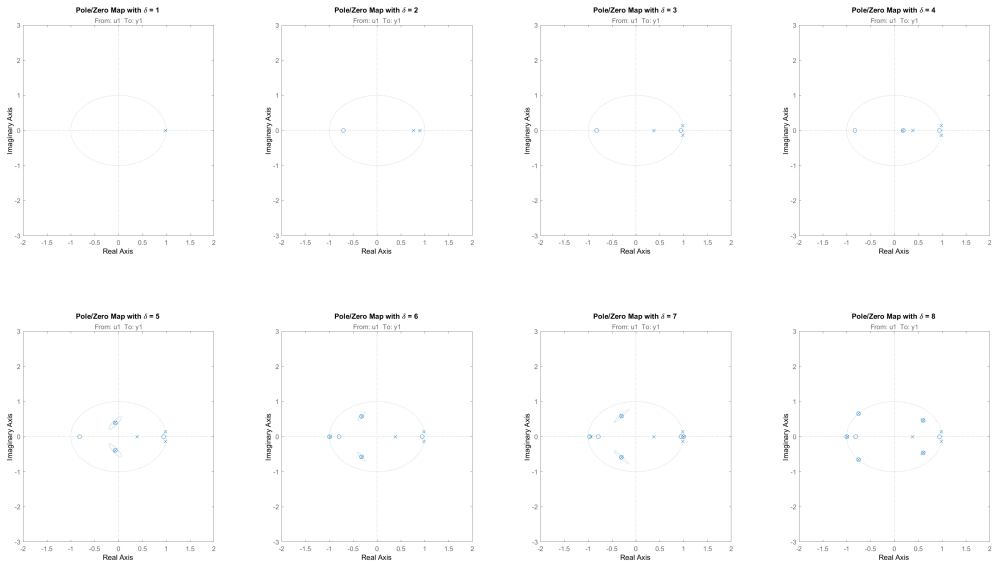


Figure 7.3.5: Pole/Zero Maps for Different Orders for G_{12}

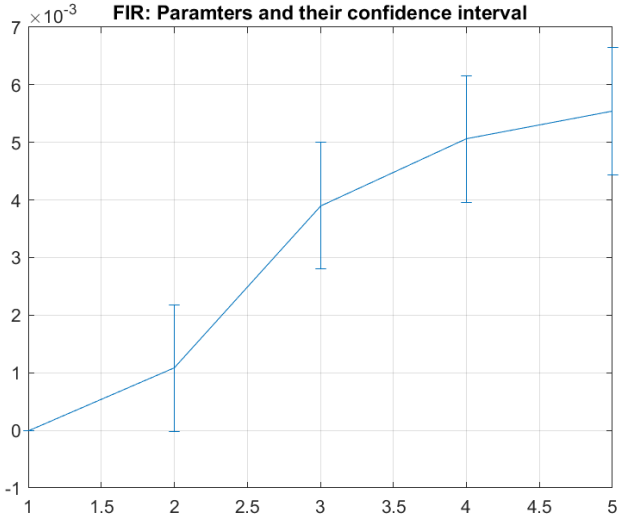


Figure 7.3.6: FIR Parameters for G_{12}

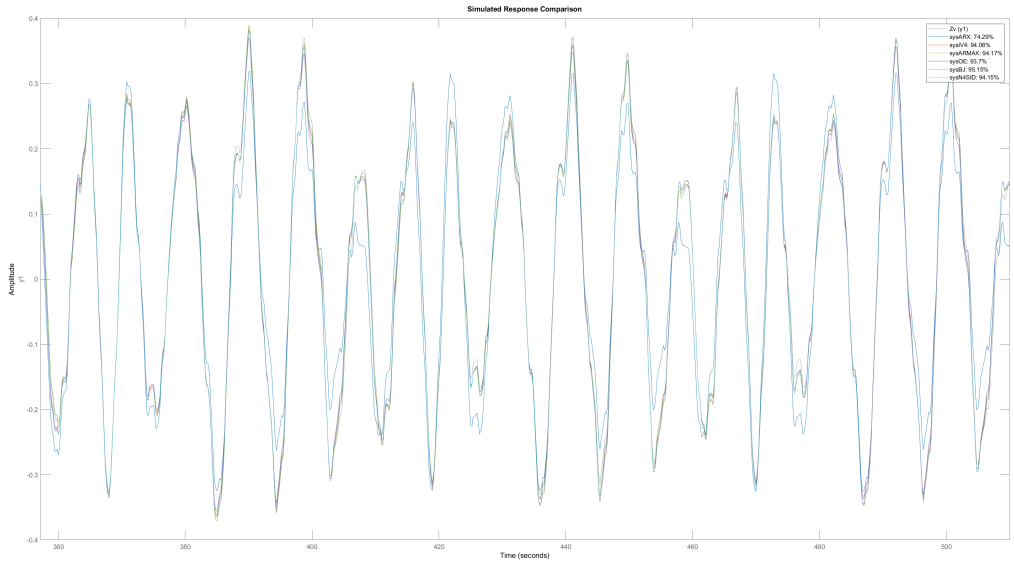
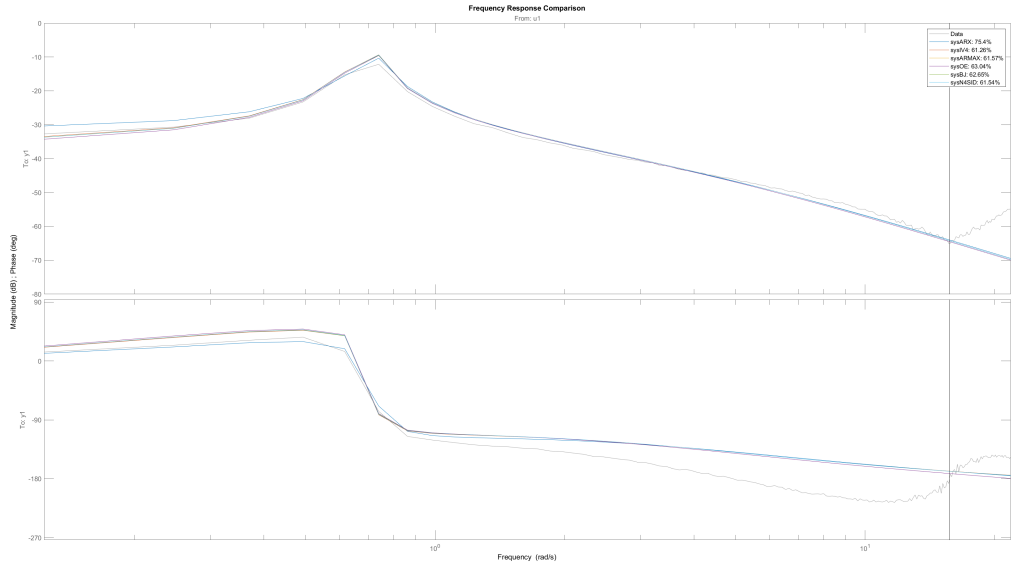
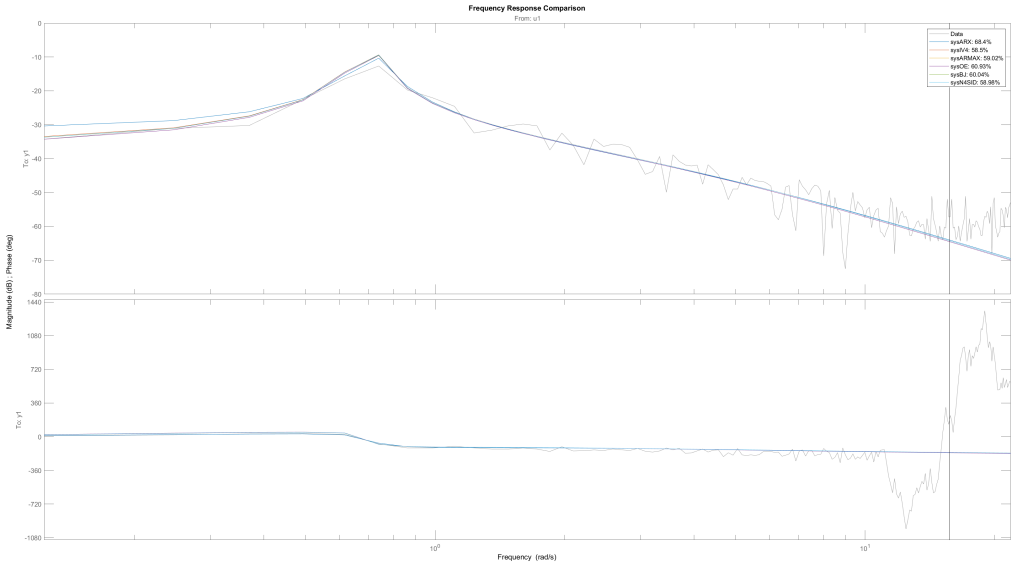


Figure 7.3.7: Time Domain Validation for Different Parametric Models for G_{12}



(a) Comparison with Spectral Analysis (with Windowing and Averaging)



(b) Comparison with Fourier Analysis (with Windowing)

Figure 7.3.8: Comparison between Frequency Responses of Parametric Models and identified Frequency Responses G_{12}

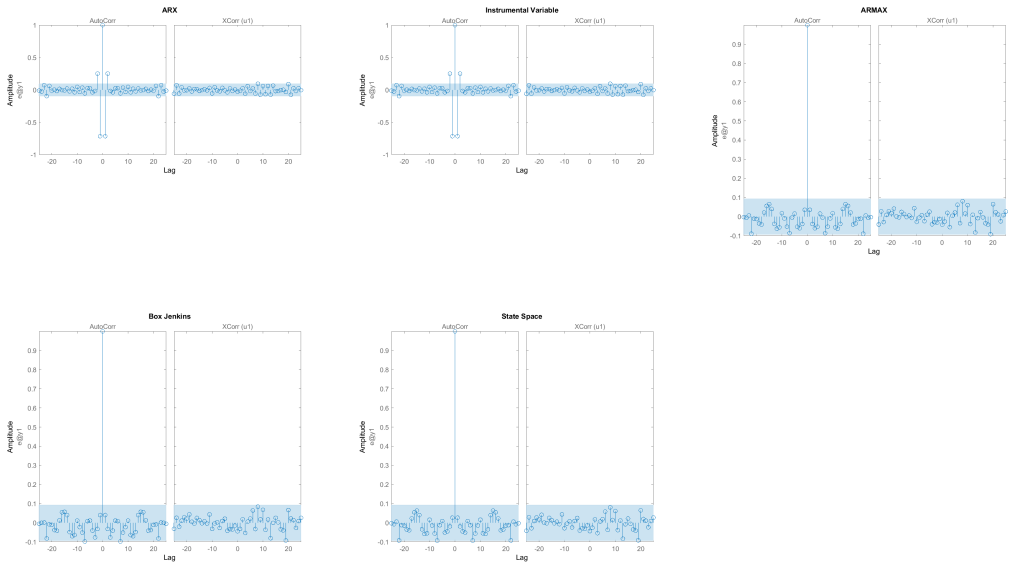


Figure 7.3.9: Noise Models for G_{12} parametric models

7.4 Data with Drift Identification plots

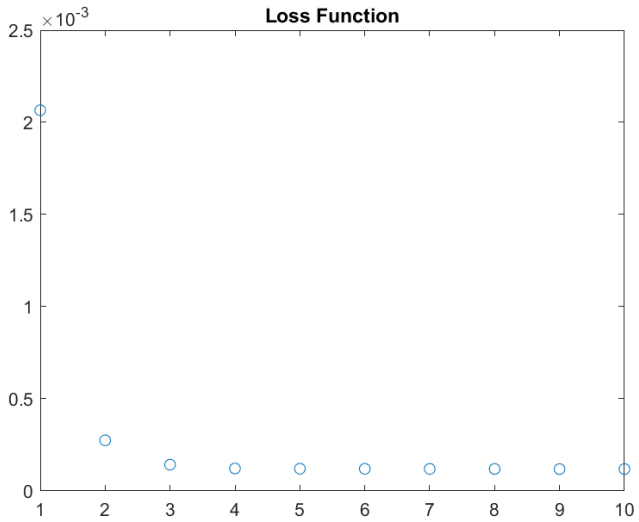


Figure 7.4.1: Loss Function for Data with Drift

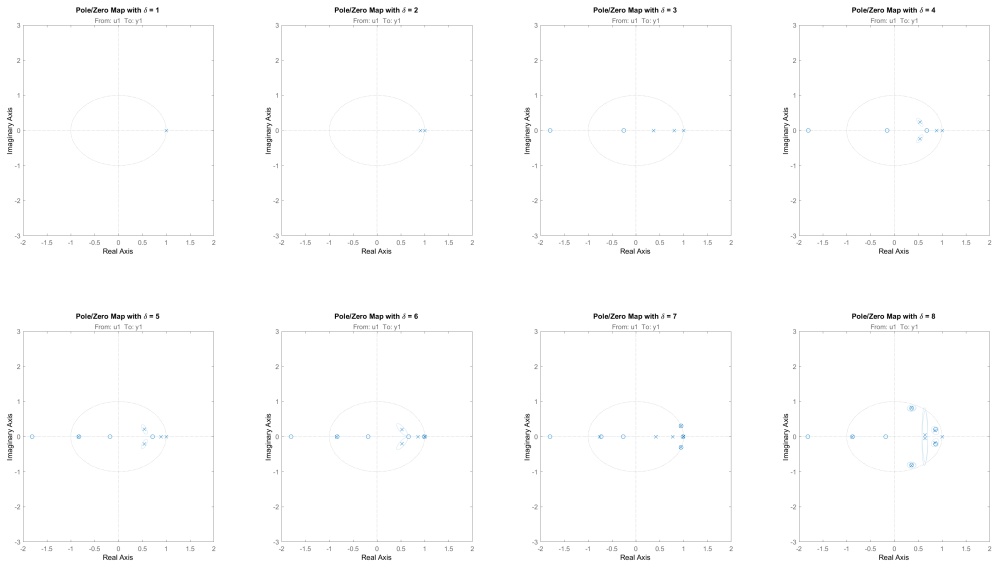


Figure 7.4.2: Pole/Zero Maps for Data with Drift

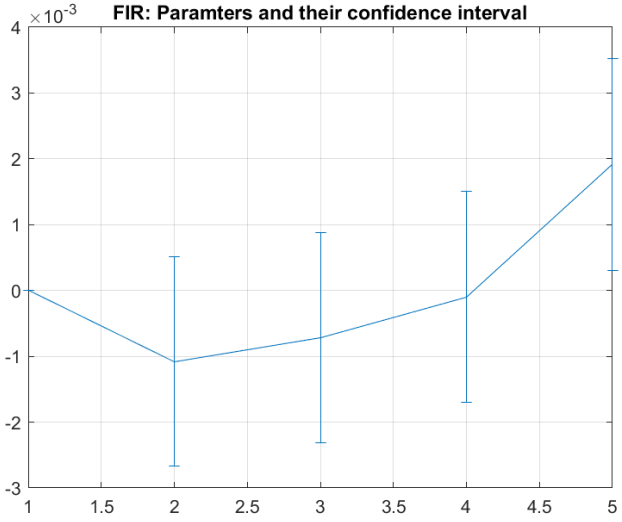
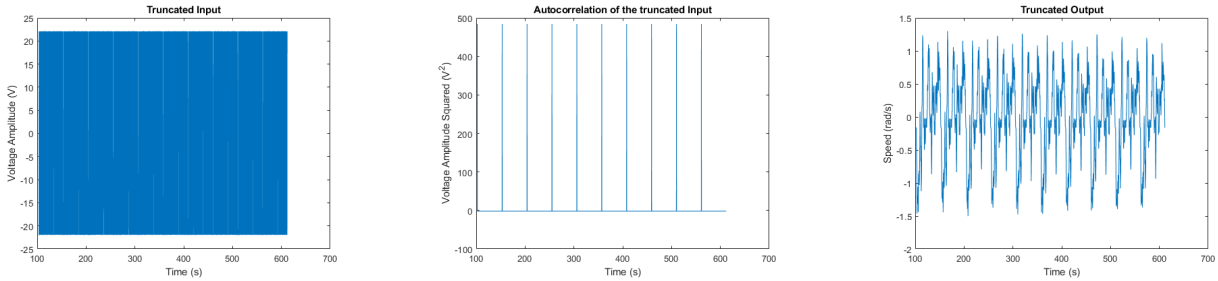


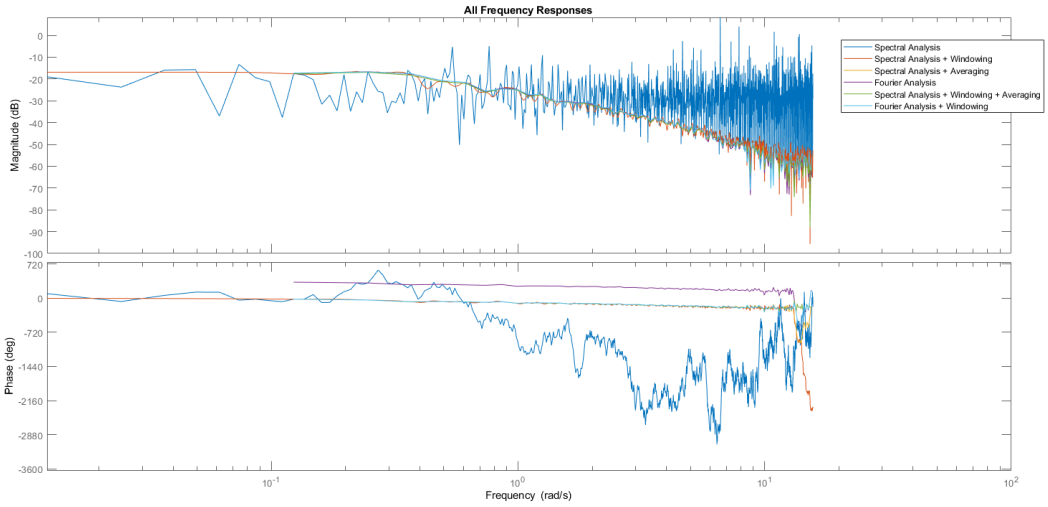
Figure 7.4.3: FIR Parameters and their Confidence Interval for Data with Drift

7.5 Yaw Speed Model Identification Plots

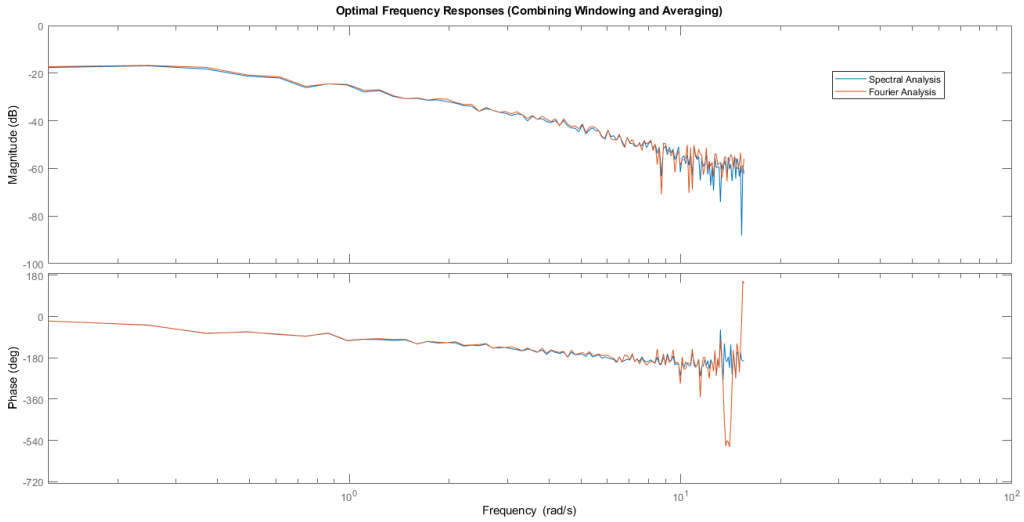


(a) PRBS Input (b) Autocorrelation of PRBS (c) Speed Measurement

Figure 7.5.1: Truncated Input and Output Measurements for G_{22} Speed Model



(a) Frequency Response Identification with All Methods



(b) Frequency Response Identification with Two Best Methods

Figure 7.5.2: Frequency Response Identification for G_{22} Speed Model

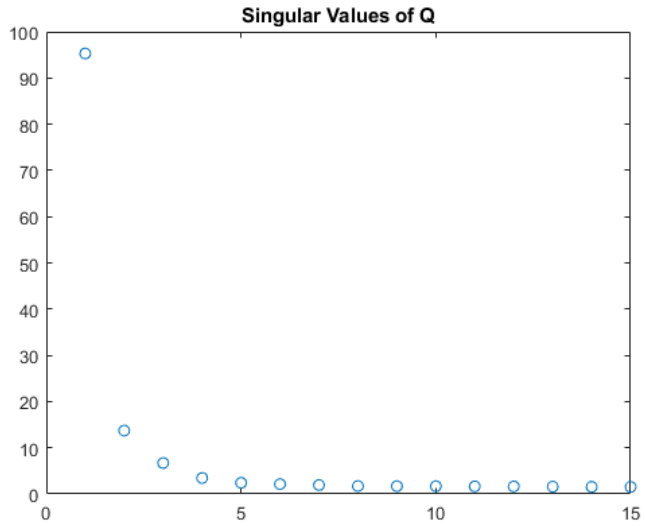


Figure 7.5.3: Singular Values of Q matrix for G_{22} Speed Model

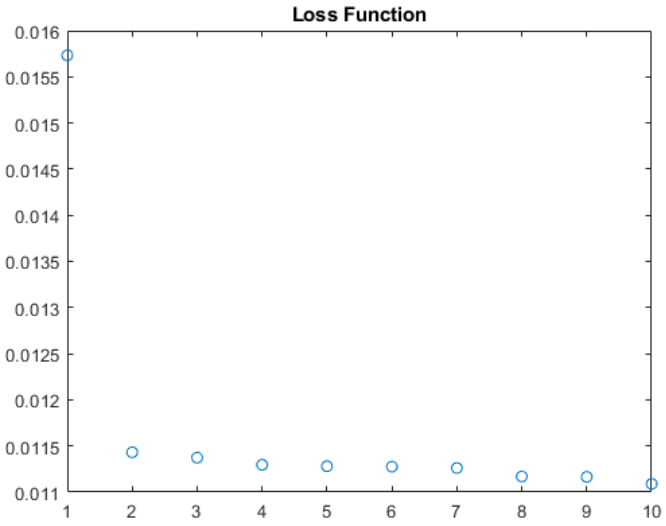


Figure 7.5.4: Loss Function for G_{22} Speed Model

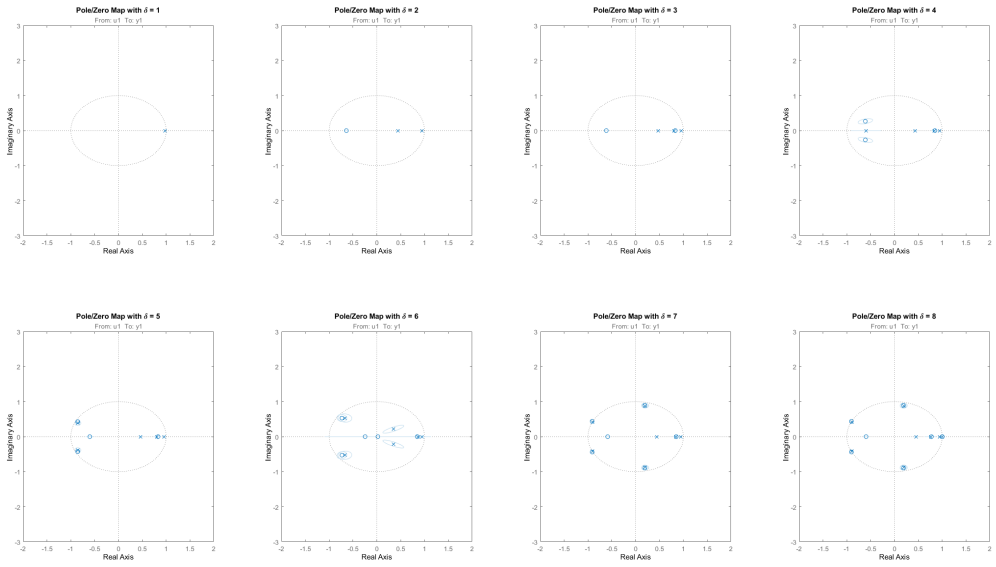


Figure 7.5.5: Pole/Zero Maps for Different Orders for G_{22} Speed Model

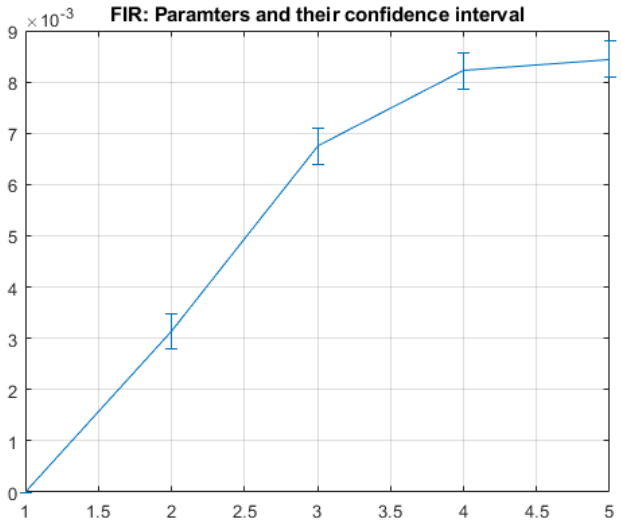


Figure 7.5.6: FIR Parameters for G_{22} Speed Model

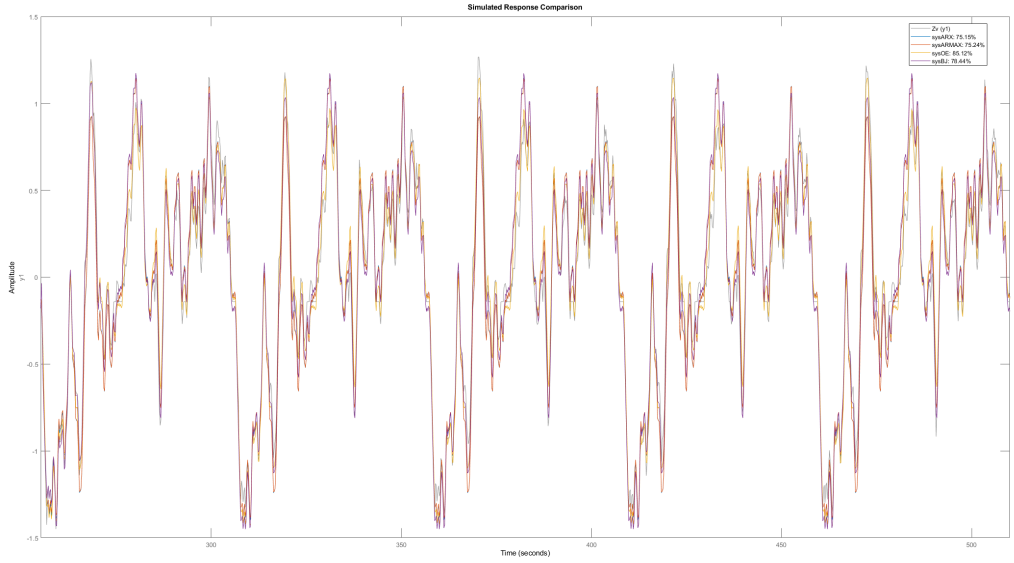


Figure 7.5.7: Time Domain Validation for Different Parametric Models for G_{22} Speed Model

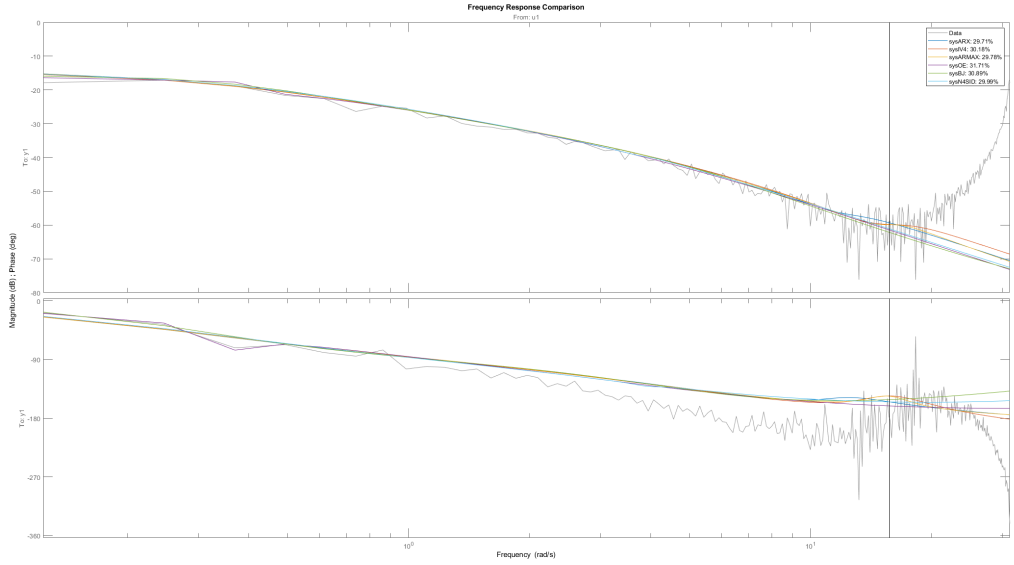


Figure 7.5.8: Fourier Frequency Response Comparison with $G_{22}Speed$ parametric models

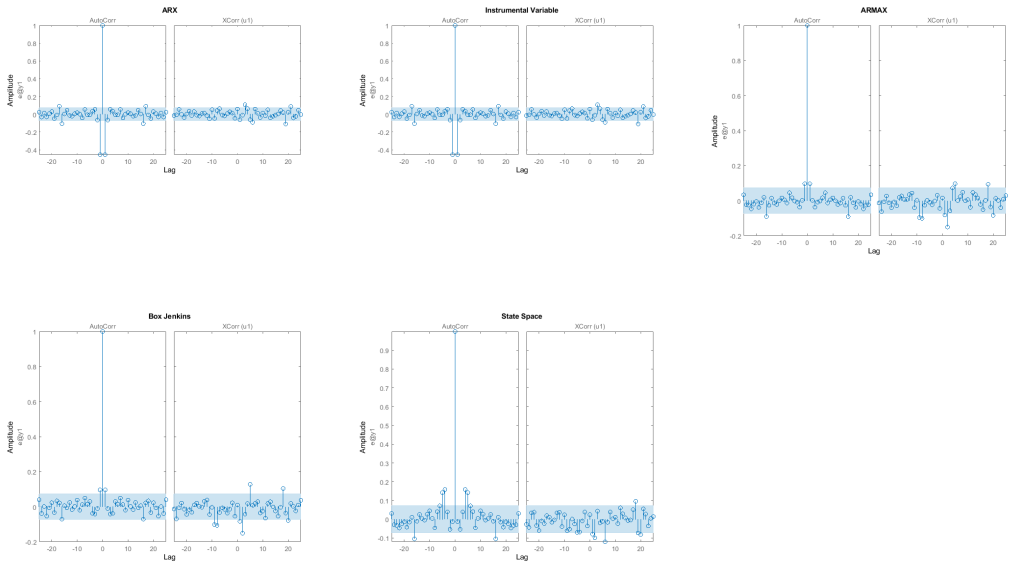


Figure 7.5.9: Noise Models for G_{22} Speed parametric models

7.6 Closed-Loop System Identification Plots

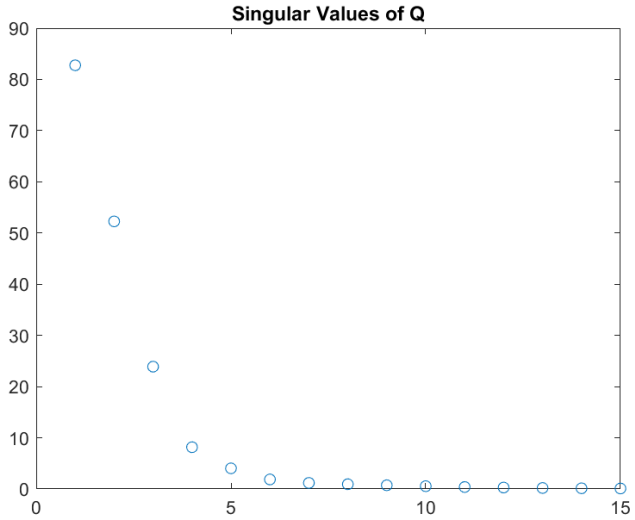


Figure 7.6.1: Singular Values of Q matrix for Closed-Loop Identification of G_{22}

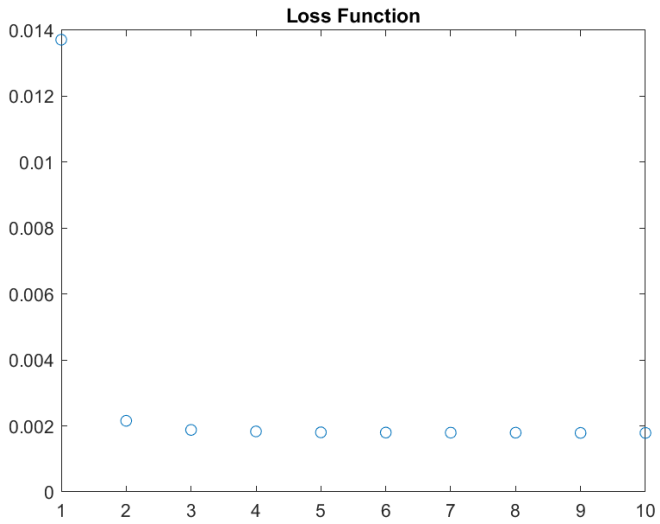


Figure 7.6.2: Loss Function for Closed-Loop Identification of G_{22}

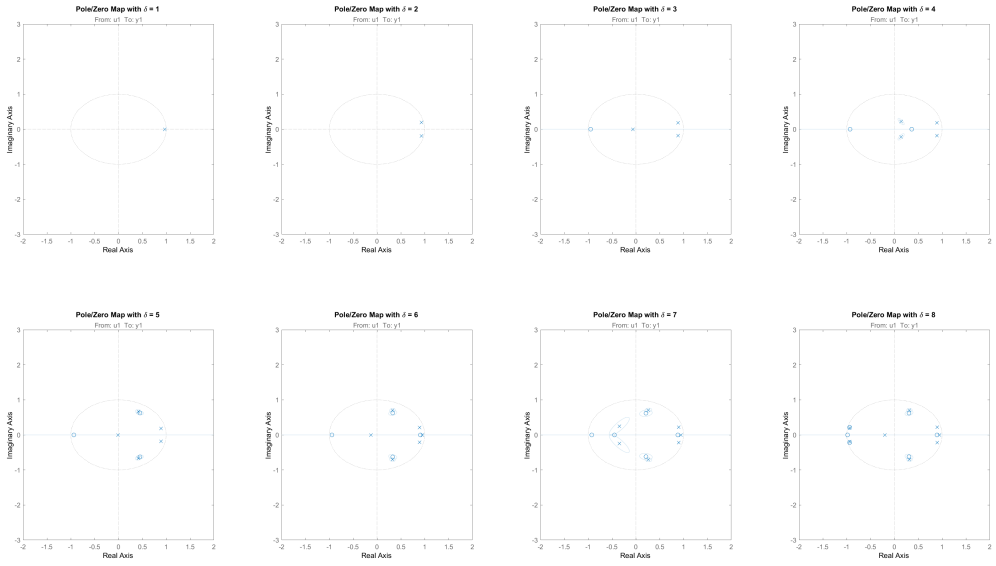


Figure 7.6.3: Pole/Zero Maps for Different Orders for Closed-Loop Identification of G_{22}

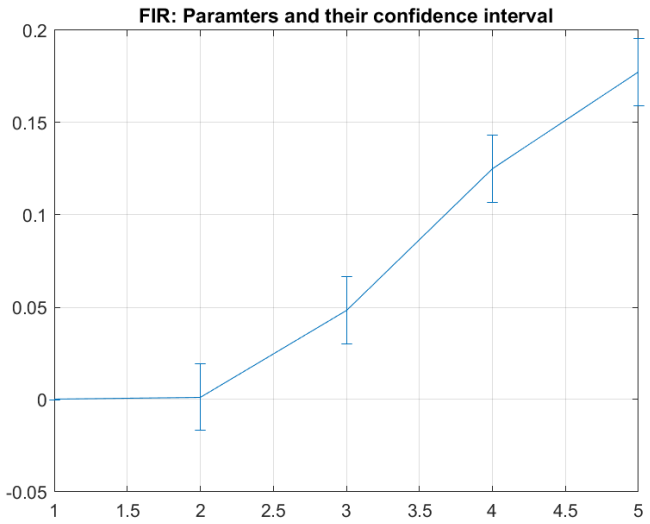


Figure 7.6.4: FIR Parameters for Closed-Loop Identification of G_{22}

References

[1] Alireza Karimi. “System Identification Lecture Notes”. 2023.

[2] Quanser. *Aero 2 - Quanser*. n.d. URL: https://www.quanser.com/products/aero-2/.