UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

FELIPE RIBAS SILVA DE AZEVEDO

Complete system for quadcopter control

Final report presented in partial fulfillment of the requirements for the degree in Computer Engineernig.

Advisor: Prof. Dr. Paulo Martins Engel
Co-advisor: Prof. Dr. Renato Perez Ribas

Porto Alegre
2014

ACKNOWLEDGEMENTS

I would like to thank my professors, especially my advisor Prof. Dr. Paulo Martins Engel for all the technical support throughout the development of this work and my co-advisor Prof. Dr. Renato Perez Ribas for the personal support and good advices since the beginning of the course.

Also, a very special thanks to Prof. Dr. Diego Eckhard who gave me great advices and support which helped me a lot. Thanks to his recommendations I was able to develop a successful method and achieve nice results as well as organizing my goals in this work. I am very thankful to his willingness and patience.

Finally, I would like to thank my family, friends and other members of my university who contributed directly and indirectly to the person I have become and also to my studies over all these past years.

RESUMO

Este trabalho apresenta algumas técnicas para construir um sistema completo capaz de controlar um quadricóptero durante todo seu vôo. São apresentadas algumas questões práticas como a escolha de componentes adequados e também questões teóricas como o desenvolvimento do sistema de controle responsável pela estabilidade do vôo. Apesar da diversidade do sistema, o foco deste trabalho é apresentar algumas técnicas diferentes para o sistema de controle tanto com uma abordagem mais voltada para a área de inteligência artificial (utilizando redes neurais artificiais) como também uma abordagem mais voltada para teoria de controle clássica (utilizando funções de transferência e resposta em frequência). Serão apresentados os pros e contras de cada método.

Palavras-chave: Sistema de controle, redes neurais artificiais, quadricóptero, Virtual Reference Feedback Tuning.

ABSTRACT

This work presents some different techniques to build a complete system to control a quadcopter throughout its flight. It is presented some practical issues like the choice of suitable components and also theoretical issues like the development of a control system responsible for the flight stability. Despite the diversity of the system the focus of this work is to present some different techniques for the control system both with an artificial intelligence approach (using artificial neural networks) as with a classical control theory approach (using transfer functions and frequency domain). It will be shown the pros and cons of each technique.

Keywords: Control system, artificial neural network, quadcopter, Virtual Reference Feedback Tuning.

LIST OF FIGURES

Figure 2.1 - Quadcopter example 13
Figure 3.1 - Block diagram of the complete system 17
Figure 4.1 - Common closed-loop configuration 21
Figure 4.2 - PID controller internal blocks 22
Figure 4.3 - Artificial neural network neuron 24
Figure 4.4 - Artificial neural network complete structure 25
Figure 5.1 - Controller design flow diagram 29
Figure 5.2 - Control system autonomy chart 30
Figure 5.3 - Root locus comparison 31
Figure 5.4 - Recurrent neural network configurations 33
Figure 5.5 - Step response of example system 35
Figure 5.6 - (a) Neurons positions and (b) regression composition by hyperbolic tangent functions 36
Figure 5.7 - Random vs Nguyen parameters initialization 37
Figure 5.8 - Best training result 41
Figure 5.9 - Bad training result 42
Figure 5.10 - Test signal used in the example system 43
Figure 5.11 - Error evolution during a single trial 44
Figure 5.12 - Sum squared error in each trial 45
Figure 5.13 - Single-pole recursive low-pass filter 50
Figure 5.14 - Single-pole recursive low-pass filter bode plot 51
Figure 5.15 - Median filter 52
Figure 5.16 - Complete system proposal 56
Figure 5.17 - Software interface 58
Figure 5.18 - Quadcopter 3D Simulator 60
Figure 5.19 - Acquired data sample 60
Figure 6.1 - P controller with gyroscope data 61
Figure 6.2 - Step response of closed-loop system with P controller 62
Figure 6.3 - Step response with higher gain 62
Figure 6.4 - Step response with slower motors 63
Figure 6.5 - Step response with slower motors and increased gain 63
Figure 6.6 - Verification signal 1 64
Figure 6.7 - Verification signal 2 64
Figure 6.8 - Recurrent MLP 1-step ahead prediction of training signal 65
Figure 6.9 - Recurrent MLP 1-step ahead prediction of verification signal 1 65
Figure 6.10 - Recurrent MLP 1-step ahead prediction of verification signal 2 66
Figure 6.11 - Recurrent IGMN 1-step ahead prediction of training signal 66
Figure 6.12 - Recurrent IGMN 1-step ahead prediction of verification signal 1 67
Figure 6.13 - Recurrent IGMN 1-step ahead prediction of verification signal 2 67
Figure 6.14 - IGMN recursive prediction 68
Figure 6.15 - IGMN 10 steps prediction of the training signal 69
Figure 6.16 - IGMN 10 steps prediction of verification signal 1 69
Figure 6.17 - IGMN 10 steps prediction of verification signal 2 70
Figure 6.18 - Real and estimated plant comparison 71
Figure 6.19 - Step response of the plant without a controller (open-loop) , desired closed-loop system and the plant with the tuned controller (closed-loop) 71
Figure 6.20 - PID vs PI controller tuned by VRFT algorithm 72

Figure 6.21 - PI and PID controllers tuned by VRFT using fast motors and M(z) pole at 0.96 73
Figure 6.22 - PI and PID controllers tuned by VRFT using slow motors and M(z) pole at 0.96 73
Figure 6.23 - PI and PID controllers tuned by VRFT using fast motors and M(z) pole at 0.9 73

LIST OF ABBREVIATIONS AND ACRONYMS

AC	Alternate Current
ANN	Artificial Neural Network
ART	Auto Retransmission
CRC	Cyclic Redundancy Check
DC	Direct Current
EMF	Electromotive Force
ESC	Electronic Speed Controller
HID	Human Interface Device
I^2C	Inter-Integrated Circuit
IGMN	Incremental Gaussian Mixture Network
IMU	Inertial Measurement Unit
ISM	Industrial, Scientific and Medical
MIPS	Million instructions per second
MLP	Multi-layer perceptron
P	Proportional
PI	Proportional-Integral
PID	Proportional-Integral-Derivative
PID&VID	Product ID & Vendor ID
PWM	Pulse-Width Modulation
RF	Radio frequency
SPI	Serial Peripheral Interface
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
USART	Universal Synchronous Asynchronous Receiver Transmitter
USB	Universal Serial Bus
VRFT	Virtual Reference Feedback Tuning

CONTENTS

ACKNOWLEDGEMENTS **3**
RESUMO **4**
ABSTRACT **5**
LIST OF FIGURES **6**
LIST OF ABBREVIATIONS AND ACRONYMS **8**
CONTENTS **9**
1 INTRODUCTION **10**
2 QUADCOPTER DYNAMICS AND MEASURES **13**
3 QUADCOPTER ELECTRONICS AND SENSORS **17**
4 CONTROL THEORY BACKGROUND **20**
4.1 PID **21**
4.2 NEURAL NETWORKS **23**
5 COMPLETE SYSTEM **28**
5.1 CONTROLLER **28**
5.1.1 PID 30
5.1.2 Artificial Neural Networks 32
5.1.3 VRFT 45
5.2 FILTERS **48**
5.3 COMPONENT CHOICE AND SYSTEM ASSEMBLY **52**
5.4 SIMULATOR **57**
6 RESULTS AND COMPARISONS **61**
7 CONCLUSION AND FUTURE WORKS **75**
REFERENCES **78**
APPENDIX A: MEDIAN FILTER COMPLEXITY COMPARISON **80**
APPENDIX B: INITIAL WORK DESCRIPTION (TG1 PAPER) **82**

1 INTRODUCTION

With the advancement of technology and reduced electronic devices prices, it is becoming more and more common the use of these devices both for hobbyists and for professionals, often replacing the need of human labor. Between these devices, a group called quadcopters (or quadrotors) is increasingly being the subject of studies and being used for many different kinds of tasks. They are a specific type of multirotor helicopters aircrafts with four rotors which generates lift with moving wings as opposed to fixed-wing aircrafts.

In general, hobbyists use quadcopters for leisure purposes (e.g. aeromodelling) or simple experimentss like First-Person-View while flying through real-time video transmission.

As for researchers and industries, quadcopters might be used in many different kinds of tasks like autonomous product delivering or reaching difficult places to perform specific tasks (e.g. photographing external parts of a high building to examine).

To be able to use a device like that, we must have a system capable of handling both flight stability and data exchange between the device and its host which is responsible for giving the device its flight guidelines or retrieving flight data.

For most hobbyists this boils down to a quadcopter with a few sensors and a radio control playing the role of a host which directly controls the flying device in real time. Usually no data is extracted or used in a more elaborate control system outside the quadcopter. These data is only used internally in the embedded system in a simple control system whose reference to be followed is supplied by the radio at each moment. In other words, an embedded control system is already programmed in the device by third parties and the hobbyist main function is to control the radio sticks in order to provide reference signals to the quadcopter to follow in real time. Therefore, user must know how to fly the quadcopter in order to prevent it from falling to the ground.

For more advanced hobbyists or researchers this task becomes more complex. It is interesting to acquire sensors data in almost all cases. Sensor types may vary depending on the objective but usually it is very useful to extract some data that could be used both for real time processing and for analysis purposes.

It keeps rising the number of studies and research in this field which seek finding better techniques of controlling the whole system. Most tasks can be accomplished with simple systems or techniques easily found in the literature, but researches are still more and more being made in this field in order to improve existent methods, leaving the system more accurate or even more autonomous. Much of these studies are focused in the control and stability problems of the quadcopter during its flight.

Some of them will perform some set of tests so one can acquire enough data in order to build a mathematical model of the device to make it possible to design and simulate its control system before using it directly. This usually involves some caution while running the tests and an extra work to prepare them. However, if enough tests are made and a well approximated model is created, then a very robust and optimum control system may be found. Yet, one should note that the designed control system in this case will be specific to that device once the theoretical model was extracted from the set of tests with the device. Thus, a change in the physical properties of the quadcopter may require a complete redesign of this system.

Another approach is to perform some tests and extract sensors data as before but instead of finding the mathematical model of the physical device one can design a controller directly from the extracted data. Thereby, one would not need to devote time in finding the theoretical model of the quadcopter. This type of controller design is called data-driven control and it is an alternative to model-based method mentioned above. It is important to note that since we do not need the mathematical model of the device anymore, it is much simpler to create an algorithm capable of finding a controller based on extracted data in real time. Thus, such algorithm may be implemented in different types of quadcopter with different physical properties since we do not need its mathematical model like before. However, one should pay attention to the premises which must be followed so that the algorithm works as expected (e.g. linear model).

Finally, another widely used method is to use the acquired data in real time so the system is constantly learning and improving its controlling parameters. This is a generic approach that can include both the model-based solution and the data-driven solution typically in an iterative process. It's worth mentioning that regardless of the method chosen one can opt for designing a controller with an artificial intelligence approach (e.g. artificial neural networks) or with a classic control theory approach (e.g. PID).

We shall present here some of the various techniques used to control the system, the situations where each one can be the best choice and a proposal of a buildable system. Even though the focus of this work is to investigate the control techniques, we will discuss the quadcopter behavior, its electronic system and then present some of the main controllers used by hobbyists and researchers. In the end we will show the results of tests made with comparisons the other works and suggestions of future works.

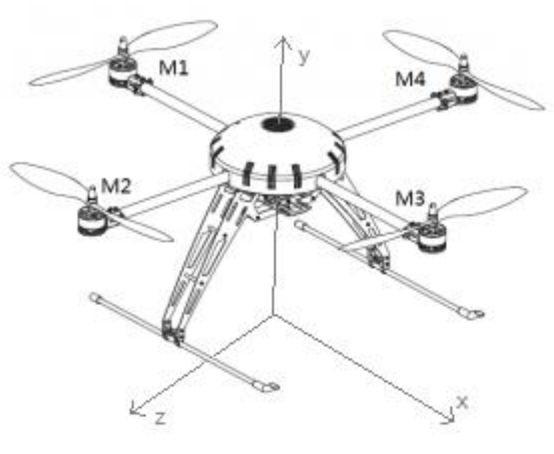
The next chapters of this work are divided as follows: in Chapter 2, the quadcopter dynamics will be shown. Next, in Chapter 3, it will be presented the electronics involved in the whole system. In Chapter 4 it will be shown some background information about control theory and methodology while designing a controller. Each sub-topic will mention a different approach (e.g. artificial neural networks, PID, etc). Chapter 5 will show the complete structure of the buildable system. Each sub-topic will cover a different part of the system (e.g. filters, controller, simulator and component choice) with a greater focus in the controller section. In Chapter 6, results will be presented along with possible comparisons. An important note is that even though a buildable system is presented, the techniques discussed for controlling the system will be tested on a simulator implemented for this purpose. Finally, Chapter 7 shows our conclusion and suggestions for future works.

2 QUADCOPTER DYNAMICS AND MEASURES

To be able to develop a control system one should know how is the physical device dynamics and what data may be read from it. Even in autonomous systems where an adaptive controller is used, one should know at least what data is being read, what kind of actuators the system has and what signals the controller should provide to the system. With that in mind, we will present here the basics of the quadcopter mechanics and its dynamics.

We can see in Figure 2.1 an example of a quadcopter. It is made of three main components: body, embedded electronics system and rotors. We refer to embedded electronics system here as all electronic components in the flying object with exception of the rotors which are the transition from electronics to mechanics. These electronic components will be explained in the next chapter. The body is the main structure which holds everything and it is primarily composed by two perpendicular bars (in a cross shape) united within a central core as shown. This structure is usually symmetric and as lightest as possible to save power while flying and should be enough to hold the remaining components. Some bodies can also be enhanced with protection structures to avoid damage, improve mass distribution or assist in takeoff and landing (as shown in Figure 2.1). At the tip of each bar there is a single electric motor which is the main component for lifting the whole frame. One should note that a complete flyable system may require external components like a radio control which will be dealt with later.

Figure 2.1 - Quadcopter example



Besides the quadcopter structure, Figure 2.1 also shows a coordinate system which is normally used when dealing with the quadcopter orientation. Some may adopt different axes names like switching Y by Z (as some 3D designing tools do). Also, one might consider two different coordinate system references: the inertial frame reference and the body frame reference. The first one is a fixed coordinate system with reference to the earth (or external environment) while the other is fixed with reference to the quadcopter body. In this chapter we will consider the body frame as our reference.

Motors called M1, M2, M3 and M4 are responsible for lifting the aircraft. Each motor is composed by its fixed part which is attached to the main frame and a rotor which is the moving part inside the motor and holds the rotary wings that produce lift. This force will exist only in the Y axis direction and centered in each respective motor. As the whole frame tends to be symmetric when considering the Z-X plane (horizontal), the gravity center point of the frame projected to this plane is usually near the geometric center point projected to the same plane. Therefore, one can easily realize that each individual rotor lift force may generate a torque with respect to the gravity center. This might be a problem when comparing with a single main rotor aircraft (e.g. helicopter) where the main rotor lift force will not always produce such torque and thus the body frame will not tend to rotate in neither X nor Z axes. Helicopters can change the main rotor direction in order to produce this torque or not, and so it can hover or change its flight direction. It also uses an auxiliary rotary wing to control the Y rotation. Quadcopters only have four fixed main rotors (e.g. cannot change its direction as the helicopters do) and no auxiliary wings. Therefore, the torque produced by each individual rotor cannot be eliminated and will be responsible for the aircraft rotation around X and Z axes which by turn will allow translations in all directions. So, to prevent this rotation one may produce lift forces in diametrically opposed motors (e.g. M1 and M3) such as the net torque produced by them is zero. The rotation around the Y axis has the same principle of a helicopter which is caused also by its main wings rotation. In other words, if a helicopter did not have the auxiliary wings, it will keep rotating around Y axis while lifting to preserve its angular momentum (as the wings rotate in one direction, the body rotates in the opposite direction with lower angular velocity due to its larger mass in comparison to the wings mass). As quadcopters do not have auxiliary wings, one might prevent this rotation by making rotors rotating in opposite directions. In our example, M1 and M3 would rotate in the same direction while M2 and M4 would also rotate in the same direction between them but in the opposite direction with respect to the first pair. Thus, each rotor will rotate in the opposite direction when compared to its neighbors. One should note that since we want to generate lift force (upwards) we must also reverse the blade geometry in the rotors that are spinning in the opposite direction.

In a higher level of abstraction on might consider directions of flight beyond the coordinate system axes. So, instead of using axes names like X, Y and Z, one can name directions like front, back, right, left, up and down. One should also note that these directions are freely chosen by the user. Upward direction is commonly defined by Y positive direction. As for the forward direction, commonly some users choose to be where a single motor points to (e.g. X axis positive direction or Z axis positive direction) while others choose to be between motors (e.g. in the direction of the bisector of the angle between X and Z axes). Other uses of forward direction can be used but are not common. While dealing with these directions, one can define specific names for these axes as well as the rotations around them. For aircrafts in general, the rotation around the forward direction axis is called roll rotation while the rotation around the lateral direction axis is called pitch rotation and the rotation around the upward direction is called yaw rotation. It is very common to refer to these only by pitch, roll and yaw although some might use these names to refer to the axis of rotation itself (e.g. pitch axis). Also, one could refer to pitch angle, roll angle and yaw angle as the angles in which the aircraft is rotated with respect to those axes.

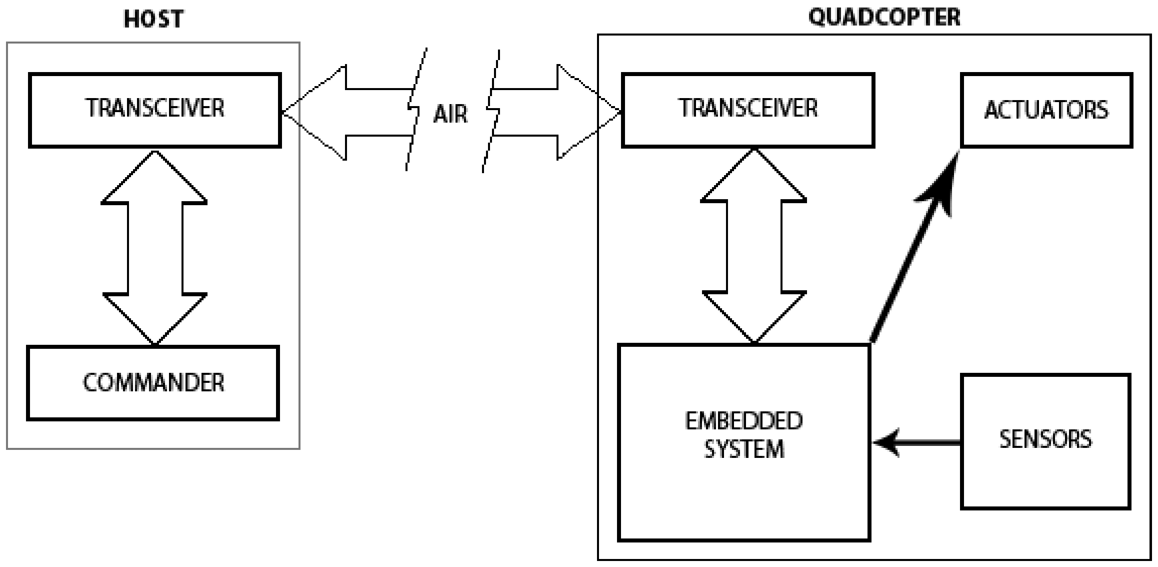
Finally, different kinds of movements can be achieved by different combinations of forces produced by each rotor. If all rotors are producing the same lift force and each rotor is spinning in the opposite direction of its neighbors (as mentioned above) so no yaw rotation will occur, so the quadcopter will hover (angles will not vary) if the net force produced by all motors is equal (in absolute value) to the quadcopter weight. If we want to translate upwards, we must increase all rotors speed equally, producing a greater net force (which will now be greater than the absolute value of total weight). Therefore, if we want to go downwards, we must decrease all rotors speed equally. Since we cannot produce pure lateral force then the lateral translation must be carried out by a rotation around the axis which is orthogonal to the axis one wants to translate over and to the vertical axis simultaneously. In other words, if we want to translate forward, we must realize a pitch rotation so the quadcopter will then naturally drift throughout the roll axis as it is tilted with respect to the horizontal plane. This occurs because whenever pitch or roll angle is different from zero then even though the sum of absolute values of rotors forces can be equal to absolute value of weight force, the net force will not be zero anymore because the weight force points in –Y direction of the inertial frame and the lift force points in the +Y direction of the body frame and they are not aligned anymore (because pitch angle or roll angle is not zero). This will result in a lateral force which makes a lateral translation possible. A simple rotation is made only by decreasing a motor speed by some value and increasing the diametrically opposed motor speed by the same value. For example, if we are hovering and we increase M4 speed by some value and decrease

M2 speed by the same value, quadcopter will tend to rotate around the X axis. And if we want a yaw rotation, one must increase a diametrically opposed pair of motor speed by some value and decrease the remaining two motors speed by the same value. For example, if we are hovering and we increase M1 and M3 motors speed by some value and we decrease M2 and M4 motors speed by the same value, a rotation around the Y axis will be generated.

3 QUADCOPTER ELECTRONICS AND SENSORS

All movements mentioned above are realized by providing correct controlling signals to the actuators (motors). Therefore, we will present here a general structure of the embedded system in a quadcopter, which is responsible for receiving commands from outside and controlling the actuators. A block diagram of this general structure can be seen in Figure 3.1. We refer to host as being any external device which controls the quadcopter remotely (e.g. radio control).

Figure 3.1 - Block diagram of the complete system



For most hobbyists the host is a simple radio control which is capable of sending various different signals modulated into different channels. The electronics in the quadcopter then receives the radio signal and demodulate it so each signal can be used as a reference for a different task. For example, one of these signals can be driven by the throttle analog stick in the radio. The more the user pushes the stick greater values of throttle are being modulated and sent to the quadcopter. When it receives, all values will be demodulated and the channel associated with throttle will be used to increase all motors speed equally. The embedded system box in Figure 3.1 is generally a microcontroller responsible for these data exchanges and controlling the actuators with the data received. Sensors are essential to make a closed-loop control with those actuators. Although the control theory will be dealt with in the subsequent chapters, one can intuitively think that the system will only be able to make sure that its control signals are producing the desired effects when it can read those values. Thus, sensors output are fed to the embedded system so it can provide a correct control signal to actuators in order to try following the reference value received from the radio. A simple and flyable quadcopter usually contains only a gyroscope sensor which measures angular velocity with 3 orthogonal axes and an accelerometer which measures proper acceleration also in 3 orthogonal axes.

Since this work aims the elaboration of a complete system, most of above components will be replaced by others that best suit our needs. It is essential that our host is able to send and receive specific commands and data for debug and tests purposes besides being able to control the quadcopter directly like a radio control would. Therefore, instead of a radio control, one might use a computer host running one or more software capable of doing that. This configuration is widely used by researches and professionals that seek the development of an incremented control system, usually with detailed data analysis. One should also choose extra components in order to communicate with the aircraft since computers usually will not have any wireless peripheral which uses the native quadcopter modulation.

There are many possibilities when choosing a transceiver to establish the wireless communication. Besides the differences between brands and technologies, one must decide what main features the whole system needs or prioritize. For example, for a given modulating process usually there is a trade-off between physical range and data rate. Higher ranges normally come with lower data rates while shorter ranges can achieve higher data rates. High volumes of data also present a commitment with reliability. Whenever data packets are guaranteed to deliver, part of the data exchange must be sacrificed to ensure that. This implies in a reduction of the payload transmission rate (e.g. data that we are actually transmitting). Also, retransmitting mechanisms may take place when transmission fails, which will increase the time taken to exchange that information (which reduces the data rate). These and other decisions are up to the user and should be made taking the project needs or specifications into consideration.

The transceiver in the aircraft side must be connected to some central processor that will read the received data and process them. In virtually all cases some microprocessor is needed since a flyable quadcopter will require more than just direct communication between the transceiver and the actuators. Once again the user must decide what type of microcontroller will best fit the system goals. Since this is the central element of the whole embedded system, it must provide means of interacting with the sensors, with the transceiver and with the actuators. Therefore, usually features like native communication peripherals and clock speed are of high importance when choosing the microcontroller. Also, one must look carefully for other features like power consumption which may imply a change in other components like the battery. Some more subjective characteristics (but not always less important) may also be observed like available documentation about that microcontroller and its availability in the market (as well as its price). Sometimes this may not be critical but it can also negatively impact on some aspects like development time or cost.

Once the embedded system is able to receive and process data from the host, it must use this data (along with sensors data which will be discussed next) in order to actually control the aircraft. This is done through the actuators which are a set of four motors with blades attached to them. These motors are usually DC brushless motors which are powered by a DC voltage from the main battery through an electronic circuit which is responsible for switching the signal according to necessary conditions. Normally an electronic speed control (ESC) is used, which is an electronic circuit that receives a reference value in the input (usually PWM signal) and produces the necessary signals in the output to the motor (usually a 3-phase AC signal, where AC is used here in a broader sense than a pure sinusoidal wave). Most ESC’s found in the market work on a standard of PWM input frequency (50 Hz and sometimes up to 300 Hz) and PWM duty cycle (1ms represents idle state and 2ms represents full speed). So a simple configuration in a PWM peripheral output of a microcontroller could fully control a DC motor.

As mentioned above, the embedded system should not only look for received data from the host but also read the sensors in the aircraft. The received commands usually have the role of being a reference to be followed by the aircraft while the sensors data will help the system to know if it is going in the right direction or not. Most quadcopters use only a pair of sensors composed by an accelerometer and a gyroscope. This is a simple configuration that can achieve great results. The gyroscope reads the angular velocity of each axis in a local frame reference. One should note that it is useful to keep the sensor axes aligned with the body frame axes so the sensor data will represent directly the angular velocity of its orthogonal axes mentioned above (pitch, roll and yaw). This is equally useful when placing the accelerometer, which also has its local frame reference. Therefore, when placed like the gyroscope, its data will represent the proper acceleration in the direction of each axis. Finally it is important to note that one might use other sensors to realize specific tasks or improve the whole system performance like an atmospheric pressure sensor (helps in altitude control), magnetometer sensor (helps in yaw rotation control), rangefinder (helps avoiding collisions or landing) among others.

4 CONTROL THEORY BACKGROUND

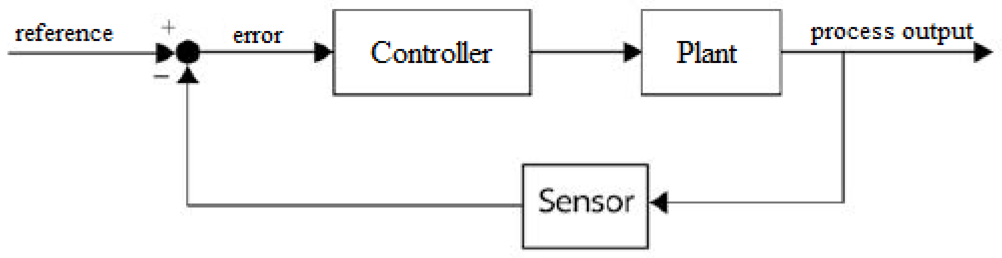
In order to execute tasks with a quadcopter, one must have some kind of certainty that the aircraft behavior is predictable and controllable. The greater the certainty, the more accurate the system can be.

Studies and researches in this field seek to find new techniques and improvements to existent methods in order to elevate the system predictability and control skills. Most times if we have a theoretical model of our plant, it can be relatively easy to find an optimum solution to our control requirements also theoretically. However, mathematical models usually come with some degree of uncertainty in its parameters and the practical behavior of the system also may be followed by noisy environment (which is unpredictable by definition). Besides that, the whole system should be capable of rejecting disturbances which sometimes are predictable and sometimes are not. It is also interesting to note that the more autonomous the system, the more robust it should be since autonomous systems tend to have less human interference which sometimes is good (avoid human labor) but sometimes can be a problem (to correct unpredicted problems). Without anyone to watch the system and help in correcting errors, the system should be capable of doing that in itself. This usually requires most complex techniques of control and sometimes even involves artificial intelligence.

Controlling the quadcopter dynamics is not a trivial task as its physical model has some high degree of complexity. Therefore, as this field has attracted much attention, many works are being published showing different techniques to improve a quadcopter control. Some of them find the controller parameters directly from the mathematical model of the aircraft (ARGENTIM, 2013). Others have proposed adaptive controllers like Achtelik (2011) using classical control theory approach or Nicol (2008) using neural networks although both of them also rely on the quadcopter mathematical model. Neural networks are known to be a very powerful tool although sometimes it can become very difficult to use it correctly or achieve desired results (HAYKIN, 2009). From this point of view, some attempts of using the neural networks power to create an adaptive controller without the need of the mathematical model of the quadcopter have been made (BURKA, 2012). The idea is that the quadcopter dynamics can be intrinsically learned by the neural network controller. Our objective here is to investigate control methods that also do not require any mathematical model of the quadcopter so it can be used in a large set of different plants. In the next subsections we will discuss some different approaches for this type of controller design and the following chapters will cover the rest of the system.

The most used structure with a generic controller is depicted in Figure 4.1. The main idea is to read the outputs of the plant through the sensors and use them as the feedback signal. The difference between this feedback signal and a given reference signal (which is called error signal) is used to feed the controller which will generate the next input to the plant. Usually one tries to reduce the error signal as much as possible at any given moment. Thus, one must determine what the parameters of highest priority to be optimized in the controller are. One might choose a fast response controller while others might prioritize minimum steady state error. Other parameters may also be taken into consideration like stability of the closed-loop system and maximum overshoot.

Figure 4.1 - Common closed-loop configuration



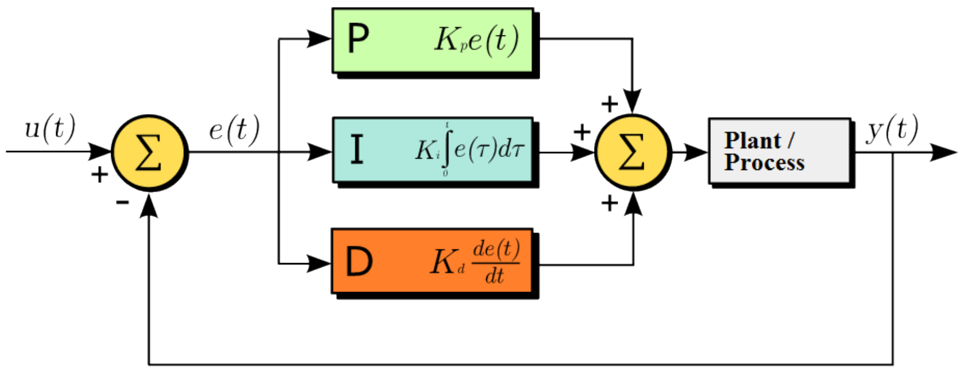
A brief description of the control techniques we are about to investigate will be discussed in the following subsections.

4.1 PID

The PID controllers (Proportional-Integral-Derivative) are widely used due to its simplicity along with its high efficiency in many cases. Most industry problems can be solved with a well tuned PID (BAZANELLA, 2005). The PID controller can be seen as the union of three independent controllers (Figure 4.2): proportional controller (P), integral controller (I) and derivative controller (D). The intuitive idea behind each of these three components is pretty simple. The proportional (P) controller generates a control signal which is proportional to the present error. That is, as the error arises, higher control signals will be produced by the P block to try to minimize this error. The integral (I) term will generate a signal which is proportional to the sum of past error values. This block is very useful in cases where a single P controller cannot reach zero error in steady state. When that occurs, a constant error will remain in the input of the controller and a simple I block can integrate that constant producing an increasing control signal to lead the error to zero. Finally, the derivative (D) term will generate a control signal which is proportional to the rate of change of the error signal. Thus, it has a predictive effect. Usually this term contains a built-in low-pass filter to avoid reacting to noisy data. This block has no effect on steady state.

In many cases a simple P controller is enough to run a stable system. Although sometimes it would be better to use a complete PID structure or even a more complex controller, one might use the simplest structure possible just to run the first tests. In our case, all investigated methods rely on a running system (online methods). Therefore, we will use a simple P controller as a starting point.

Figure 4.2 - PID controller internal blocks



One widely used strategy to control a quadcopter with a PID (we refer to PID as the complete structure although any combination of its internal blocks can be considered) is to create an individual closed-loop system for each independent axis of each sensor. That is, we would have the same structure depicted in Figure 4.2 for each sensor axis where $y(t)$ would be the sensor output (angular velocity or acceleration of each axis) and $u(t)$ the desired value to that axis. This way we would have three different PID controllers for the gyroscope (one for each axis) and the input to each control loop would be an angular velocity reference. That is, making all references equal to zero would make the quadcopter preserve all its rotation angles. Other three distinct PIDs would then be used to control the accelerometer axes. Therefore, setting zero to X and Z axes would make the quadcopter hover (all weight force would be concentrated in Y axis).

Another widely used strategy is to mix the accelerometer and gyroscope signals to find a single angle value for each axis. That is, combining the gyroscope information with the accelerometer information one can find a unique angle value for each axis that can represent an absolute angle of rotation or a variation with respect to the previous time step. Therefore, one can use a total of only three closed loop systems instead of six. One should also be aware that there are multiple ways of combining those signals and the control system efficiency will directly depend on which way is chosen. Since the sensors can have errors (e.g. noise and drift), one should choose a method that minimizes the main problems of each sensor and combine the best features of each. Details about those filters will be discussed in prospective chapters.

It is worth noticing that the Plant/Process box in Figure 4.2 represents the quadcopter dynamics including the actuators (motors). Therefore, the control signals generated by each closed-loop structure must be combined before sending to the motors. We will use a common adopted way of combining those signals:

$$m_1 = T + u_p + u_y \tag{4.1}$$
$$m_2 = T + u_r - u_y \tag{4.2}$$
$$m_3 = T - u_p + u_y \tag{4.3}$$
$$m_4 = T - u_r - u_y \tag{4.4}$$

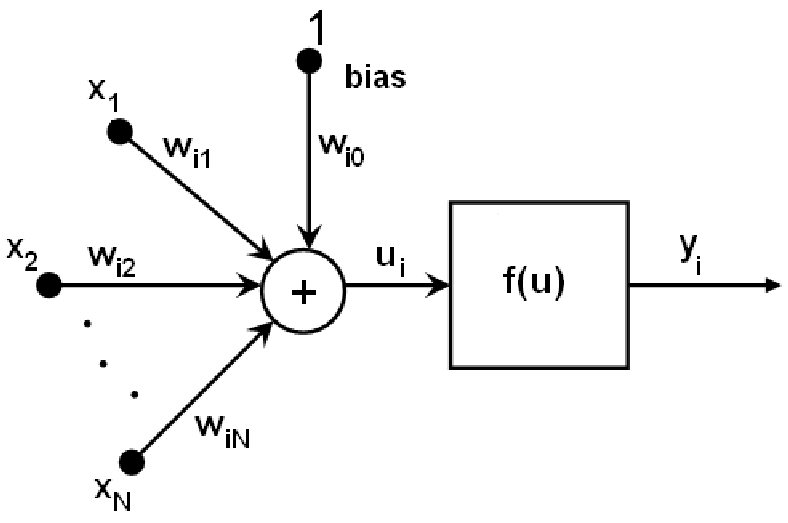
where m_i is the signal sent to the i^{th} motor, T is the throttle signal which is common to all motors and used in an open-loop configuration and u_p, u_r and u_y are the control signals related to the $pitch$, $roll$ and yaw axes respectively. That is, if one of the PID closed-loops is controlling the pitch axis angular velocity then u_p will be the output of this PID. It can be seen in Equations (4.1) through (4.4) that those signals are used in a differential mode due to the plant symmetry. Thus, another important detail is that the signals of each term inside each equation depends on each axis reference (e.g. whether positive pitch rotation is clockwise or counter-clockwise) as well as where each motor is positioned. So, one must watch carefully motors configuration before writing their equations. Finally, each motor will be individually controlled in an open-loop configuration which can already lead us to a flyable quadcopter.

To be able to investigate the control methods discussed in the next sections we will then use a simple P controller to each axis of the gyroscope. Therefore, a well tuned controller will be able to control the angular velocity in the three axes of rotation, which most times is enough for a human with some experience can control the aircraft. The idea behind the control techniques can be easily applied to other signals like the accelerometer signals or the combination between both sensors.

4.2 Neural Networks

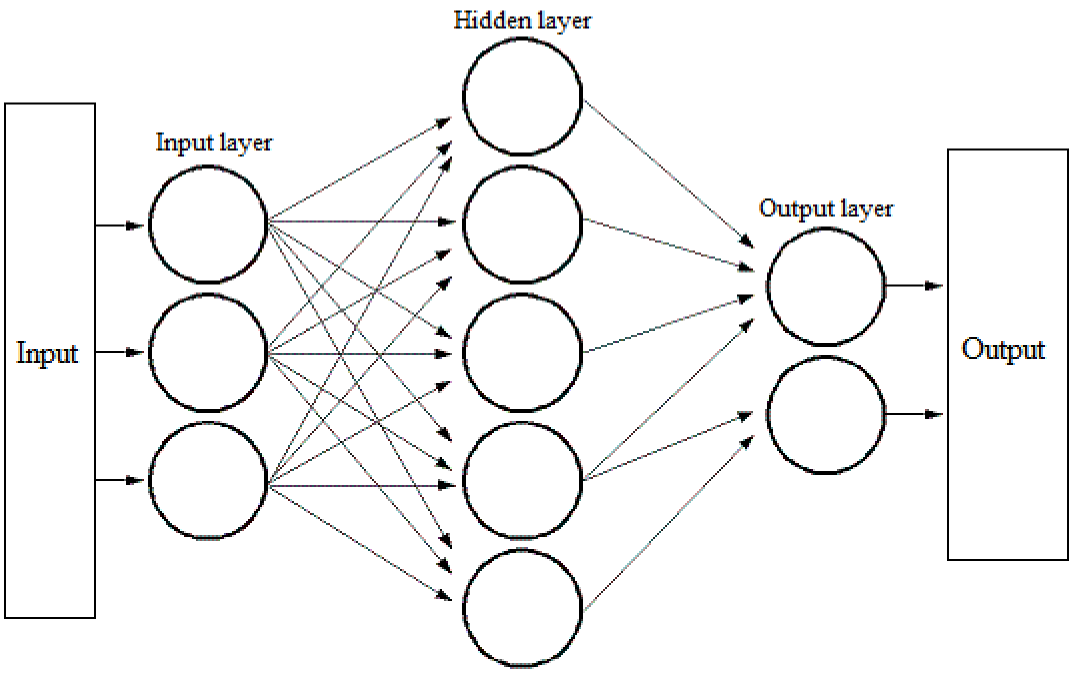
Artificial Neural Networks (ANN) or simply Neural Networks are an extremely powerful tool used in many fields like statistics, artificial intelligence, among others. It was inspired in real nervous system and it is an attempt of creating a mathematical model with learning capabilities like a real organism. Many different types of neural networks have been proposed and these learning capabilities can often be seen as a nonlinear regression since the network mathematical structure in this case usually is represented by a nonlinear function with one or more inputs and outputs. Although different types of ANN models have been proposed, we will focus in the Multi-Layer Perceptron (MLP) which is one of the most used. The MLP structure comprises smaller structures called neurons (Figure 4.3). Each neuron receives a sum of inputs coming from the previous neurons outputs (called x_N) and use this value as an input to a function (called function of activation, $f(u)$) to produce an output. Each connection between a neuron output and the next neuron also has an associated weight value. Optionally, there can be a fixed input which will be called bias.

Figure 4.3 - Artificial neural network neuron



These neurons are then connected to other neurons and arranged in layers. The first layer is called input layer and its neurons inputs represent the network input. The last layer is called the output layer and its outputs represent the network output. Between the input and the output layer there can be one or more hidden layers. We are not considering neural network structures with no hidden layers here since its regression capabilities are restricted to linear problems only (HAYKIN, 2009). In the other hand, although many hidden layers can be used, a single hidden layer is enough to achieve satisfying results in most known problems (HAYKIN, 2009). Each individual output from a given layer will be connected to all inputs of the next layer, as depicted in Figure 4.4.

Figure 4.4 - Artificial neural network complete structure



Usually the neural network nonlinearity is concentrated in the hidden layer whose neurons have a nonlinear activation function while the input and output layers have a simple linear activation function. Also, input neurons usually do not have a bias factor. Thus, the role of the input layer in the network is only to distribute the inputs to the hidden layer. A widely used nonlinear function in the hidden layer is a sigmoid function:

$$f(x) = tanh(x) \tag{4.5}$$

Thus, considering that all neurons have a bias input (except for the input neurons), a single hidden layer perceptron can be described mathematically by the following equation:

$$y_o(t) = \sum_{m=1}^{n_h} f\left(\sum_{i=1}^{n_i} w_{i,m}^h x_i + b_m^h\right) w_{m,o} + \mathrm{b_o} \tag{4.6}$$

where x_i is the output from the i^{th} neuron from the input layer which is the same value of its input (no bias in the input layer), b_m^h is the bias value of the m^{th} neuron of the hidden layer and b_o is the bias value of the output o. The number of neurons in the input layer (or the number of inputs of the network) is represented by n_i while the number of neurons in the hidden layer is represented by n_h. The connections weights are named $w_{i,m}^h$ (weights from the i^{th} neuron of the input layer to the m^{th} neuron of the hidden layer) and $w_{m,o}$ (weights from the m^{th} neuron of the hidden layer to output o). One should note that the above example represents a single output of a network with o outputs.

The weights and bias are adjustable parameters used to map the input set to a desired output set. Furthermore, the network should be able to generalize this behavior to new inputs. Therefore, many different algorithms have been proposed in order to find the correct set of values to these weights and bias in order to find an optimum mapping function. This process is called learning and can be classified as supervised learning, unsupervised learning or reinforcement learning. This work will be restricted to supervised learning where a set of inputs with respective target outputs is given and some algorithm is performed to find the best values of weights and bias that make the neural network reproduce the set of given target outputs with the given inputs. One must be careful to the overfitting problem. That is, usually one uses acquired data (from a sensor) to train the network and this data often has noise. If the network is trained at some point where this noise is well reproduced in its output, then this network probably will have a poor performance in generalizing unseen data. Better results will be obtained when the noise coming from the data is not reproduced by the network. Instead, the network function will always have some associated error when compared with original data (due to noisy data) but may produce satisfying generalization of new incoming data.

The training process can occur in different ways, depending on the chosen algorithm. Among the most used ones, we can divide them into two groups: iterative and batch. An iterative training process is when a new step of training can be done at each new training data. In contrast, a batch training process is when the whole training process occurs only when all training data is already available. Depending on the network architecture and in the selected training algorithm, there can also be a training process which is neither pure batch-mode nor iterative-mode. Instead, one can realize each training step at every group of samples. It is important mentioning that we use the term iterative here to designate training processes that do not require multiple samples to be executed although a batch training process can run iteratively. Within this work it will be used the backpropagation of errors, which is a widely used method of training a neural network. This method is used in conjunction with an optimization technique which can be chosen among several available. Here we are going to use the gradient-descent algorithm which is widely used (and also one of the simplest) and the Levenberg-Marquardt (LM) which is more complex than the first one but also more powerful.

Besides the network topology, one must choose previously what kind of controller is being implemented. That is, how the neural network will influence in the control loop. An often used strategy is to put the neural network as the controller block directly. The error signal of the control loop will be fed directly to the neural network input and its output will be used as a control signal to the plant. Thus, training the network is the equivalent of tuning the controller. Another widely used structure is to use the neural network to learn the plant dynamics. Once the network has learned how the plant behaves, one can choose between many techniques of controlling the system based on the plant model. Details about the controller design methods will be discussed in the next chapter. Finally, one can opt for a control structure where neither the neural network will learn the model nor it will control the plant directly. Instead, one can choose any known controller with tunable parameters and use the neural network to tune these parameters. This can be seen as an adaptive controller and has been proposed by many authors like Chan (1995), Suzuki (2004) and Song (2013).

As the training process of the neural network and the configurations of its parameters (number of neurons, layers, etc) require extra attention and dedication, we will focus in only one of the above mentioned strategies of controlling the system with ANNs. The method chosen here is to use the neural network to learn the model dynamics in contrast to the other algorithm investigated in prospective sections (Virtual Reference Feedback Tuning algorithm) which already tune a known controller without knowledge of the plant model.

Finally, the Incremental Gaussian Mixture Network (IGMN) is a type of neural network proposed by Heinen (2010) and will be also tested here as well as the MLP for comparison purposes. Its structure is different from the traditional multi-layer perceptron as it works based on Gaussian Mixtures. The intuitive idea is to observe data samples and group them into Gaussian Mixture components. Hence, the data modeling is now treated from a statistical point of view. At every new sample, the network will verify if it can be grouped into any existent mixture or if it must create a new mixture component. This also means that the training process can be iterative and usually each sample can be used only once to train the network (in contrast to the MLP training algorithms that commonly need many iterations).

The IGMN has many interesting advantages over the MLP. Among other features, the main attractive characteristics to our comparison are: it requires single step training process for each sample; it does not require the whole training data set previously (training process can be done iteratively); it does not suffer from the plasticity problem, which can be briefly explained as the inability of learning new patterns once the training process is over and it does not require a careful parameter initialization. Besides all these differences, the IGMN is supposed to have the same capabilities of a traditional MLP in terms of data representation (HEINEN, 2010). Thus, tests made with neural networks in this work can be done with both network types.

5 COMPLETE SYSTEM

The goal of this work is to propose a complete system infrastructure and investigate some of these control techniques that could be tested within this system. In order to propose a complete system infrastructure, different topics will be discussed in this chapter like filter design and the physical components choice. The focus is to investigate the control techniques so this will be the most detailed topic. However, other topics not only may help in the overall understanding of many problems that go beyond control theory, as well as provide a suggestion of a framework that can be used to test other techniques not presented here.

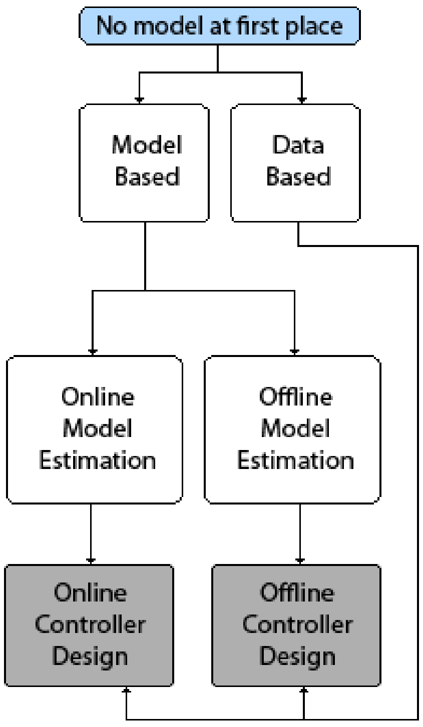
5.1 Controller

As mentioned above, many different types of controllers can be used to achieve our goals. Besides the works that show the use of control techniques directly applied to quadcopters, it is also interesting to investigate the use of other techniques that were tested with different kinds of plants but might be useful here.

We will start by understanding the main approaches when designing a controller. Many paths can lead to a good controller and this choice usually depends on what kind of data is available. The simplest case is when the plant of the processes already has its mathematical model available so one can directly design the controller based on that plant. In most cases this model is not available though. In simple processes with simple plants normally is easy to make a trial and extract the necessary parameters or estimation model so the controller design can be done with that estimated model. With other plants it may not be so easy to realize a trial which can hinder the model estimation. In our case, we cannot simply realize open loop trials with the quadcopter because it can damage the structure with a collision. Therefore, one must prepare specific environment test and test strategies to avoid this kind of problem. These strategies should be prepared carefully so it can truly represent a real life situation as we will see in the next sections. The use of simulators is often recommended, especially in those cases where trials may become expensive or too complex.

As in most cases the plant model is not available, we will begin our analysis from this point. Thus, we can organize the controller design strategies in a flow diagram as depicted in Figure 5.1. Controllers can be first divided in two groups: model-based and data-based. The first one may be the most common which the controller is developed based on the model of the plant. Since we do not have this model available at first, then some estimation must be done which can also be divided into more groups and will be discussed next. The other group of controllers represents those which are obtained based on acquired data directly. One might say that model estimation also depends on acquired data, but in that case this data is used to first elaborate the model and then find a suitable controller. In this case, data is used directly to find the controller and the plant model will remain unknown.

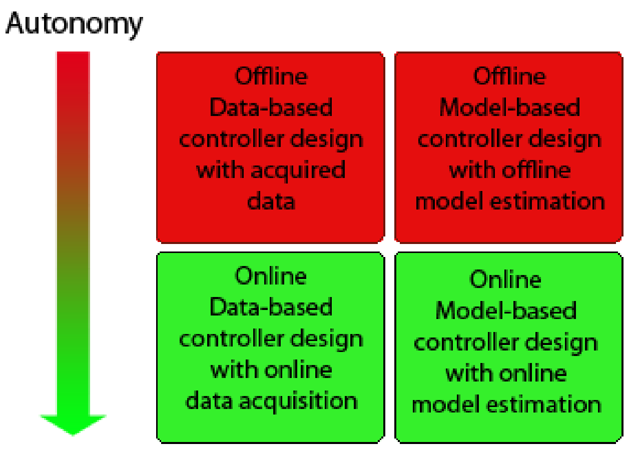
Figure 5.1 - Controller design flow diagram



Whereas autonomy is a major factor in this work, methods of model estimation and calculation of controllers can be divided also in two groups that are intrinsically related to adaptability: offline methods and online methods. In the offline methods, one must perform the calculations before starting the process while in the online methods the algorithms and calculations are realized while the process is running without the need to stop it. Find the desired results while the process is running can be extremely useful but usually requires extra computational capabilities which sometimes is impracticable.

The controller design can be online or offline regardless if it is model-based or data-based. Clearly the data acquisition is an online event by definition, but from the moment the data is collected one can opt for an online or offline technique. One should also note that an online controller design can be chosen when an offline model estimation process is used but this case is not considered here because once the model estimation is made offline there is no great advantage in using an online method for the controller design. An autonomy chart can be seen in Figure 5.2 where the green boxes indicate our priorities in this work.

Figure 5.2 - Control system autonomy chart


Autonomy
Offline Data-based controller design with acquired data
Offline Model-based controller design with offline model estimation
Online Data-based controller design with online data acquisition
Online Model-based controller design with online model estimation

We are going to focus on the online methods as the autonomy is the main factor of this work. Neural networks will be used in an attempt of estimating the plant model in real time while the Virtual Reference Feedback Tuning (VRFT) algorithm will be used to find a suitable controller directly from the acquired data. Even though this last technique can be used offline, we will give preference for an online implementation to prioritize the autonomy of the system. Since we are going to investigate online methods, we must have at least a poor performance working system so the data can be acquired. Therefore, a brief section will be dedicated to discuss a classical PID controller implementation.

A special class of controllers will also be discussed here which is the predictive controller class. There are many types of predictive controllers but the main idea is to find the optimum control signal based on a prediction of the system behavior. To be able to preview this behavior one must have some knowledge of the process dynamics which usually is done through system identification. Our attempt to do so will be using a neural network to learn the system dynamics and so it can predict system next steps through time.

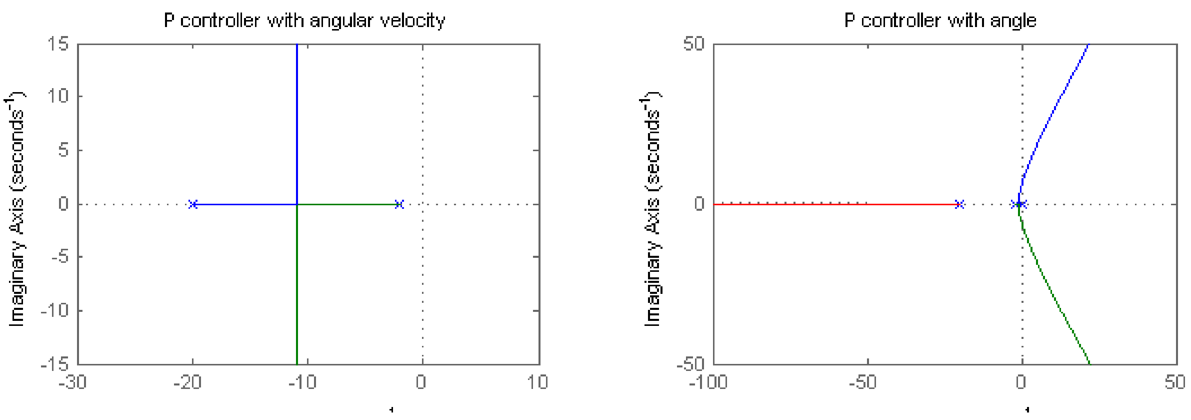
5.1.1 PID

As discussed in Chapter 4, a classical PID controller is very useful in many situations and so it will be used as a starting point in our experiments. We do not want to keep our efforts focused on finding the best PID controller for the quadcopter manually. Otherwise we would not need an adaptive controller anymore (or at least it would not be so useful). Hence, we will use a simple P controller which is enough to run the closed-loop system.

In order to allow a more flexible initial controller we made the control loop only with the angular velocity of the quadcopter through the gyroscope data. This will not invalidate our experiment since this simple closed-loop control configuration can make the quadcopter flyable so we can acquire data in order to run a controller tuning algorithm. In Figure 5.3 we can see the root locus of the position of the poles of a system that approximates the quadcopter dynamics in a closed-loop configuration with a P controller. If we are controlling only the angular velocity, we have only two dominant poles which represent the frame dynamics and the DC motors mechanical dynamics. If high gains are used to control this system, both poles are going off the real axis but their real parts will not change, so the system will not become unstable. As for the angle control we can see that now there is a pole at the origin (since we are integrating the angular velocity). We can see that if we want to use the same P controller for this system, high gains can easily destabilize the system beyond the fact that in the best case the system will probably be slow since the first non-zero pole is too close to the origin.

It is worth noticing that this second order system used for representation purposes is an approximation of the real system. Thus, other poles might exist in the real system so one must be careful with some actions such as increasing the controller gain too much in angular velocity control so these poles that were not considered can start influencing significantly in the result. Also, the real system is not linear. However, the key point here is that the angular velocity control will be much more flexible in terms of setting an initial high gain that allow us to fly the quadcopter without destabilizing the system.

Figure 5.3 - Root locus comparison



5.1.2 Artificial Neural Networks

In this section we will investigate the use of neural networks to learn a system dynamics. In previous sections we have seen that having the system model will let us design a controller through different methods. When the plant dynamics can be described by a linear differential equation, one can find its transfer function and so design the controller based on classical control theory. The transfer function can be approximated through different methods, but usually involves some experiments with the plant. Here, two major factors will make our strategy to go in a different way. First, our plant is not linear due to all nonlinearities of the fluid dynamics present in the air which are directly related to the quadcopter dynamics since its forces are exclusively made by its propellers. And the second reason is that we do not necessarily need the physical parameters (e.g. mass, momentum, etc) or the transfer function of the plant. Instead, we just need any structure that can copy the plant behavior. With that structure in hand, one can design a predictive controller or even use it to make experiments that would be unfeasible with the real plant. Therefore, we will create a neural network structure to feed with the same input signal that we provide to the real plant, and use the output from the plant as the target output to the network.

A mathematical structure that is often used for this is called Nonlinear Autoregressor Exogenous model (NARX) and is described as:

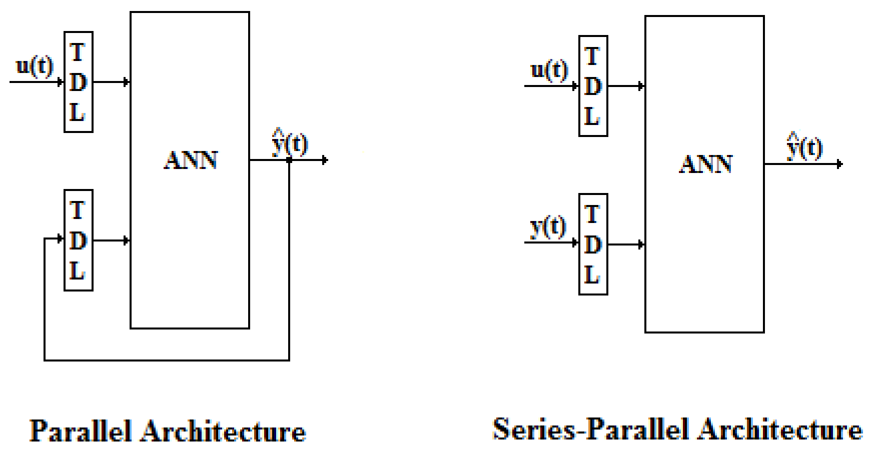
$$y_t = F\left(y_{t-1}, y_{t-2}, \dots, y_{t-n_y}, u_t, u_{t-1}, u_{t-2}, \dots, u_{t-n_u}\right) + \varepsilon_t \qquad (5.1)$$

where y_t can be the values read from the system (output of the plant), u_t the input signal and ε_t an error function. The F function is a nonlinear relation that takes as inputs the past values of the input, the past values of the output and the current value of the input and should estimate the current system output. As we are dealing with regression, ε_t represents the error between the target values and the estimated function output.

NARX is commonly used to predict future values of times series. When used to predict a physical system output, one can imagine that the F function must have the system dynamics incorporated internally. Since F is a nonlinear function, we can use a neural network to fulfill this role in the equation. Therefore, we provide the current input, past inputs and past outputs to the network and train it to learn the next output.

One should note that if Equation (5.1) is shifted one time step ahead, then y_{t+1} will appear and it will depend on y_t, which is the value that the network just predicted. Thus, the neural network output can be fed back to one of its own inputs creating a recursive neural network. This may let one make successive predictions of the dynamics of the plant.

Figure 5.4 - Recurrent neural network configurations



Recursive networks can have two ways of training and using it which is depicted in Figure 5.4. The first one is when the network is in a closed loop as mentioned above, and it is called Parallel Architecture. The other is called Series-Parallel Architecture and it is when the network is used in an open loop configuration. That is, the past values from the output are provided by the user since they are known. The notation $\hat{y}(t)$ denotes the approximation for $y(t)$ provided by the network and TDL is the abbreviation for Tapped Delay Line, which represents the past values of that signal. In the input signal case, one can use the present value $u(t)$ or not. Despite the great advantage of the network being its recursive configuration, better results are achieved using the Series-Parallel Architecture during the training process. This way, as the network is in an open loop configuration, one can use regular feedforward training algorithms like a simple regression with target values.

We must define the network architecture itself. That is, number of neurons of each layer, number of layers and activation functions of each neuron. Even though there is not any consistent way of defining the best configuration for the network, one should know that one hidden layer is usually enough for most problems to be solved. Moreover, too few neurons in the hidden layer will restrain the network capacity as well as too many neurons may lead us to a more difficult training process because of the excess of parameters to optimize. Hence, we are going to start with one single hidden layer with five neurons and a single output. This single output will represent one control variable (e.g. angular velocity of a single axis). All layers with exception of the hidden layer will have a linear activation function in its neurons. The hidden layer neurons will have a hyperbolic tangent function as their activation function.

We are going to use a generic transfer function with a damped oscillatory step response to represent a poor tuned system as reference to the network regression. As the quadcopter can be approximated by a second order system whose poles represent the mechanic pole of the DC motor and the frame dynamics itself, we are going to use the following transfer function as the closed loop transfer function of the system:

$$T(s) = \frac{Y(s)}{R(s)} = \frac{10}{s^2 + 2s + 10} \tag{5.2}$$

which has complex poles and unity DC gain. Its step response can be seen in Figure 5.5. $Y(s)$ could represent the angular velocity of one axis and $R(s)$ the desired angular velocity to that axis although it does not really matter in this section since we only want an oscillatory response curve to test our neural network regressions.

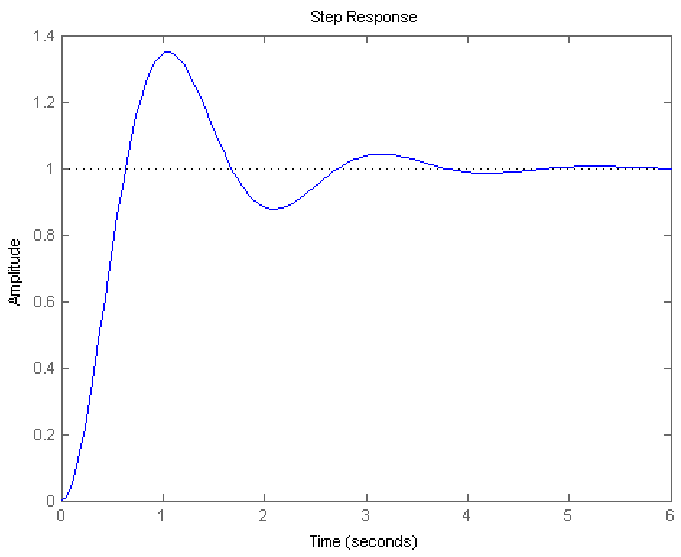
We must train the network so it can learn the dynamics of this system. Series-Parallel architecture will be adopted so the training process is like a common error backpropagation training process and traditional algorithms can be used. One must prepare the input data so it can be used like a simple feedforward network training process. Usually the training data is a set of arrays (patterns) where the first elements in the array are the inputs to the network and the last ones are the target outputs used to calculate the error relative to its current output. In our case, only the last element will be the target output $y_t(t)$:

$$[u(t)\ \ u(t-1)\ \dots\ u(t-n_u)\ \ y(t-1)\ \ y(t-n_y)\ \ y(t)] \tag{5.3}$$

One should note that the number of delayed samples of the input and output can be closely related to the system order. Although we are using a second order transfer function, other poles are present in the system and may manifest themselves depending on the situation. Therefore, we are going to use the last three samples of the input and the last three samples of the output (i.e. $n_y = n_u = 3$). So the network should have a total of 7 inputs and one output. Each next training pattern will be shifted version of the previous one with updated values for $u(t)$, $y(t-1)$ and $y(t)$.

With all the training patterns, we should now choose an algorithm to realize the training process. We are going to investigate two well known algorithms based on backpropagation of errors. The first one is the classical Backpropagation Through Time (BPTT) which uses gradient descent technique and the second one is the backpropagation of errors with Levenberg-Marquardt (LM) optimization. But before the training process begins, one must initialize the network weights and bias values. Normally these values are random selected although this selection has an important role in the training efficiency as we will see below.

Figure 5.5 - Step response of example system

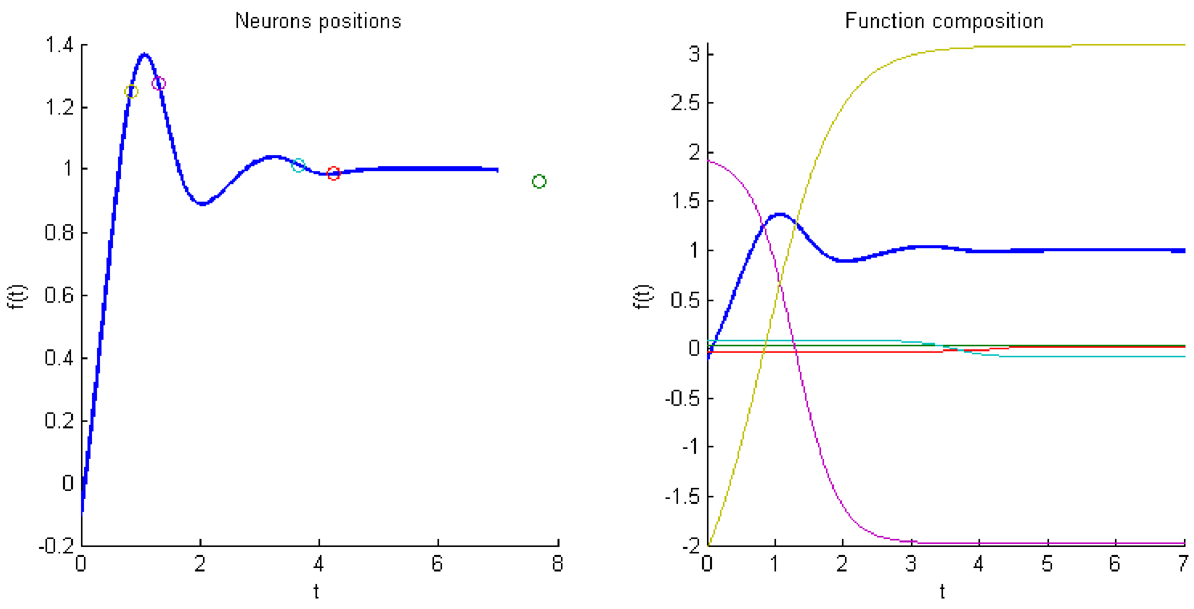


We will use now a simple single input and single output neural network example to illustrate the weights and bias initialization importance. As the input and output layers have linear activation function in its neurons then the nonlinearities of the function we are trying to learn will be incorporated in the hidden layer. Each neuron in the hidden layer can be described by a hyperbolic tangent function:

$$j_h = \tanh(w_h x + b_h) \tag{5.4}$$

where j_h is the output of the h^{th} neuron, w_h is the weight between the input x and the h^{th} neuron and b_h is its bias. Therefore, w_h can be seen as a scale factor of the hyperbolic tangent curve and b_h can be seen as its lateral translation. Thus, as the nonlinearity of a hyperbolic tangent is concentrated mostly between -2 and 2, one can imagine that the training process will try to arrange the neurons in such a way that their nonlinearities will cover different parts of the original function. So, Nguyen (1990) proposed that the bias and weights from the hidden layer should be adjusted so each neuron is initially responsible for a single piece of the original function. That is, the scale factor w_h is set depending on the number of neurons and the original function input range (so the input range is divided between the hidden neurons) and the translation factor b_h is set so each neuron is equally spaced throughout the original input range. A graphical result can be seen in Figure 5.6. A simple regression of the step response showed in Figure 5.5 was made with a single-input and single-output neural network with five hidden neurons. After the training process we can see the neurons influence in the final result. Figure 5.6 (a) shows which point of the curve is most affected by each neuron while Figure 5.6 (b) shows the composition of the curve by each hyperbolic tangent function. Some interesting points should be mentioned here. First, even though the initialization parameters of the hidden layer are set to some specific values, they also readjust themselves during the training process as before, but most times these adjustments will be much smaller when compared to random initialization. Second interesting point is that one can note that one hidden neuron is centered outside the input range on the right. This means it has little influence on the composition and may be removed. Thus, probably similar results can be obtained with only four neurons in the hidden layer.

Figure 5.6 - (a) Neurons positions and (b) regression composition by hyperbolic tangent functions



Besides the formal demonstration made by Nguyen (1990), a performance example was made here using the regression made for test purposes (showed in Figure 5.6). A comparison between a random parameters initialization and Nguyen (1990) initialization is depicted in Figure 5.7 where the blue line represents the approximated function while the green line represents the original function. It is clear that the initialization process has significant influence on training results. The error criterion established here is:

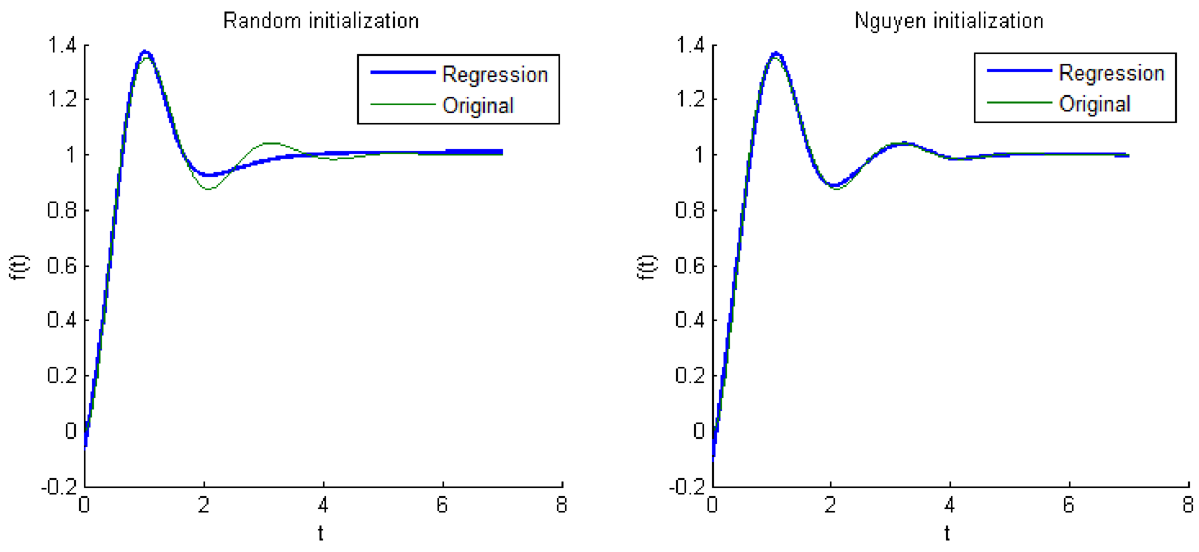
$$Error = \sum_{t=0}^{n_s-1} \left(o_t(t) - o_{nn}(t)\right)^2 \tag{5.5}$$

where n_s is the number of samples of the original curve, $o_t(t)$ is the target output at a given t and $o_{nn}(t)$ is the neural network output for the same t. Following this error criterion, the error found with random initialization was 0.1687 while the error using the initialization method proposed by Nguyen (1990) was 0.0184. Besides the reduced error, only 14 iterations were necessary to reach the training stop criterion while 26 iterations were necessary with random initialization with the same criterion. Even though the stop criterion should take into consideration the overfitting problem and verify the neural network generalization capabilities, here we used the variance of the last $N + 1$ errors as the stop criterion:

$$var(error(t), error(t-1), error(t-2), \ldots, error(t-N)) < maxVariance \qquad (5.6)$$

where var denotes the variance between its parameters, $maxVariance$ is the limit value and $error$ is the error vector which keeps the error of each training iteration. Therefore, when the rate of change of the error becomes smaller than a threshold the training process ends. Thus, this stop criterion does not favor the best neural network results in terms of generalization. Instead, it aims in the training algorithm convergence efficacy.

Figure 5.7 - Random vs Nguyen parameters initialization



Now that the initialization criterion is established we will focus on the training algorithm. Again, for simplicity we are going to use the same example as above with a single input neural network. The purpose of following tests is to compare the backpropagation algorithm using gradient descent method against the LM method. The same network structure will be used. That is, 5 hidden neurons, no bias to the input neuron, linear activation function in the output and hyperbolic tangent function in the hidden layer neurons.

Starting with the gradient-descent based backpropagation, we will define our error criterion to be similar to the one already used in Equation (5.5), using bold letters for vectors:

$$E(\boldsymbol{x}, \boldsymbol{v}) = \frac{1}{2}\sum_{p=1}^{P}\left[(e_p)^2\right] \qquad (5.7)$$

where p is the pattern index which varies from 1 to P, $\boldsymbol{x}$ is the input vector, $\boldsymbol{v}$ is the parameters vector (composed by adjustable bias and weights) and e_p is the error generated by each pattern, defined as:

$$e_p = t_p - o_p \tag{5.8}$$

where t_p is the target output when pattern p is used and o_p is the actual network output with that same pattern:

$$o_p = bo + \sum_{n=1}^{5} wo_n \; tanh(x_p wh_n + bh_n) \tag{5.9}$$

where bo is the output bias, n is the index that varies from 1 to the number of hidden neurons (5 neurons in our example), wo_n is the weight from the n^{th} hidden neuron to the output, x_p is the p^{th} term of the input vector $\boldsymbol{x}$, wh_n is the weight from the input to the n^{th} hidden neuron and bh_n is the bias value of the n^{th} hidden neuron.

One should note that this error criterion is very similar to the one presented by Equation (5.5) with the exception of the division by a factor of 2. This term was introduced to simplify derivations. Also, now the error is presented like a function that depends on the input vector and parameters vector since both vectors have influence on the error value. Although weights and bias have different meanings in the network, they can be grouped into a single vector of parameters where the objective is to find the best values to this vector for a given criterion. In our case, the criterion is the error criterion and the objective is to minimize this error. Therefore, one can imagine that, given an input vector, the error function is a scalar field in the 16th dimension (5 weights plus 5 bias from the hidden layer and 5 weights plus 1 bias from the output layer) so the gradient of that field will point towards the next local maximum or local minimum (depending on the sign). Thus, the parameter vector is updated towards this gradient vector, scaled by a step size called λ:

$$\Delta\boldsymbol{v} = -\lambda\boldsymbol{g} \tag{5.10}$$

where

$$\boldsymbol{g} = \begin{bmatrix} \frac{\partial E}{\partial v_1} & \frac{\partial E}{\partial v_2} & \frac{\partial E}{\partial v_3} & \dots & \frac{\partial E}{\partial v_{16}} \end{bmatrix} \tag{5.11}$$

$$\boldsymbol{v} = [v_1 \;\; v_2 \;\; v_3 \;\; \dots \;\; v_{16}] = [wh_1 \;\; \dots \;\; wh_5 \;\; bh_1 \;\; \dots \;\; bh_5 \;\; wo_1 \;\; \dots \;\; wo_5 \;\; bo] \tag{5.12}$$

This intuitively yields the main problem of the gradient method: the step size. The traditional algorithm uses fixed step size (named λ here) and the problem is that large step sizes may cause the solution not to converge (at every step the parameters update more than enough so it passes through the solution) while small steps may cause the training process to be very slow. Gauss-Newton method assumes the error surface at any point in its space can be approximated by a quadratic curvature towards the nearest local minimum. With this assumption, one can derivate this approximation and set equal to zero which should give a unique solution. The main problem now is that the efficacy of this method relies on the assumption of a linear dependency between o_p and $\boldsymbol{v}$ (which implies in a quadratic dependency between E and $\boldsymbol{v}$). Whenever these relations are not valid, Gauss-Newton method can be even worse than the gradient descent method. Levenberg then adds his contribution to the Gauss-Newton method which combines both previous mentioned methods. The step size is now variable and depends on the result of the previous iteration. When the calculated error increases, the step size is incremented in such a way that the update rule goes towards the gradient descent method. In contrast, when the error decreases the step size is decremented and the update rule becomes more like Gauss-Newton method. Marquardt then realizes that since Levenberg method is a combination of gradient-descent and Gauss-Newton method (which uses second derivative approximations), even though sometimes the step size is large at some point that the update rule is almost the same as the gradient-descent method one always has to calculate second derivatives approximations. Therefore, Marquardt insight was to use this information even with the gradient-descent part of the update rule. Proofs and demonstrations will be omitted here since they were already shown by Levenberg (1944), Marquardt (1963), Ranganathan (2004) and Roweis (1996) and it is not the main objective of this work. Thus, LM update rule becomes:

$$\Delta \boldsymbol{v} = -\left(H + \lambda diag(H)\right)^{-1} \boldsymbol{g} \tag{5.13}$$

where

$$J = \begin{bmatrix} \frac{\partial e_1}{\partial v_1} & \cdots & \frac{\partial e_1}{\partial v_{16}} \\ \vdots & \ddots & \vdots \\ \frac{\partial e_P}{\partial v_1} & \cdots & \frac{\partial e_P}{\partial v_{16}} \end{bmatrix} \tag{5.14}$$

and

$$H = J^T J \tag{5.15}$$

which is an approximation of the Hessian matrix and $diag(M)$ is a matrix with the same diagonal and dimensions of M but with 0 on the other elements. One should notice that the closer the relation between o_p and $\boldsymbol{v}$ is to a linear relation, the better the approximation H of the Hessian will be.

Now we must derive the gradient vector $\boldsymbol{g}$ and calculate the Jacobian matrix (J) elements for our neural network example so we can implement both methods for comparison purposes. Clearly as the LM algorithm is an improvement of the gradient-descent method one should think that there is no reason to use the gradient-descent method anymore. However, because of the Jacobian matrix, the LM algorithm requires error from all patterns to be calculated before running the algorithm. Thus, it must be executed in a batch-mode. In contrast, the gradient-descent method can run iteratively at each pattern. So, in order to run our online neural network model estimator, we must choose between continuously run the gradient-descent method or gather a set of information and so run the LM algorithm once. This will be discussed after comparison is made.

To find the gradient vector, one must find the error derivatives with respect to each parameter of vector $\boldsymbol{v}$. So, applying the chain rule we have:

$$\frac{\partial E}{\partial v_k} = \frac{\partial E}{\partial e_p}\frac{\partial e_p}{\partial o_p}\frac{\partial o_p}{\partial v_k} \tag{5.16}$$

$$\frac{\partial E}{\partial e_p} = \sum_{p=1}^{P} e_p \tag{5.17}$$

$$\frac{\partial e_p}{\partial o_p} = -1 \tag{5.18}$$

Let

$$i_{j,k} = \tanh(x_j wh_k + bh_k) \tag{5.19}$$

so

$$\frac{\partial o_p}{\partial wh_k} = wo_k\left(1 - i_{p,k}^2\right)x_p \tag{5.20}$$

$$\frac{\partial o_p}{\partial bh_k} = wo_k\left(1 - i_{p,k}^2\right) \tag{5.21}$$

$$\frac{\partial o_p}{\partial wo_k} = i_{p,k} \tag{5.22}$$

$$\frac{\partial o_p}{\partial bo} = 1 \tag{5.23}$$

Therefore,

$$\boldsymbol{g} = \left[-\sum_{p=1}^{P} e_p wo_k\left(1 - i_{p,k}^2\right)x_p\,, \quad -\sum_{p=1}^{P} e_p wo_k\left(1 - i_{p,k}^2\right), \quad -\sum_{p=1}^{P} e_p i_{p,k}\,, \quad -\sum_{p=1}^{P} e_p\right] \tag{5.24}$$

with k varying from 1 to 5.

One should remember that the gradient-descent method can be used iteratively (online). Thus, the gradient vector does not depend on all pattern errors like above. Therefore, the error function should be redefined to depend only on the current error:

$$E(p, \boldsymbol{v}) = \frac{1}{2}\left(e_p\right)^2 \tag{5.25}$$

so repeating above steps will give us

$$\boldsymbol{g} = \left[-e_p wo_k\left(1 - i_{p,k}^2\right)x_p\,, \quad - e_p wo_k\left(1 - i_{p,k}^2\right), \quad - e_p i_{p,k}\,, \quad - e_p\right] \tag{5.26}$$

Now we calculate the derivatives inside the Jacobian matrix:

$$\frac{\partial e_p}{\partial wh_k} = -wo_k\left(1 - i_{p,k}^2\right)x_p \tag{5.27}$$

$$\frac{\partial e_p}{\partial bh_k} = -wo_k\left(1 - i_{p,k}^2\right) \tag{5.28}$$

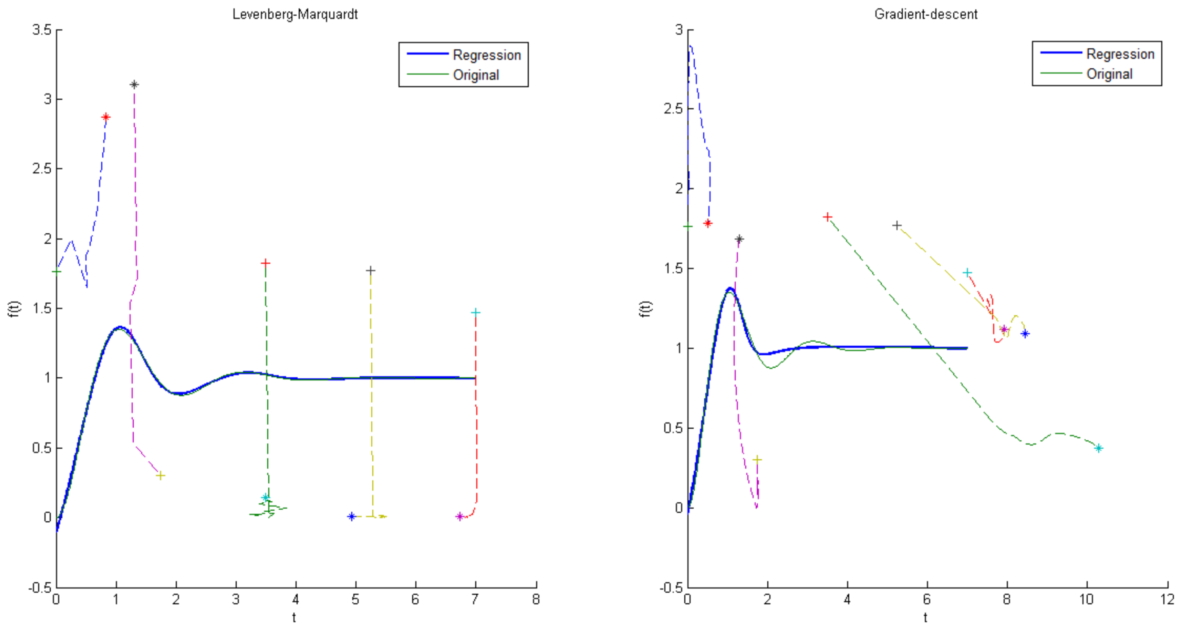
$$\frac{\partial e_p}{\partial wo_k} = -i_{p,k} \tag{5.29}$$

$$\frac{\partial e_p}{\partial bo} = 1 \tag{5.30}$$

which are similar to the gradient terms except for the error term. Thus, all essential terms have been calculated so we can now use both update rules and compare both methods.

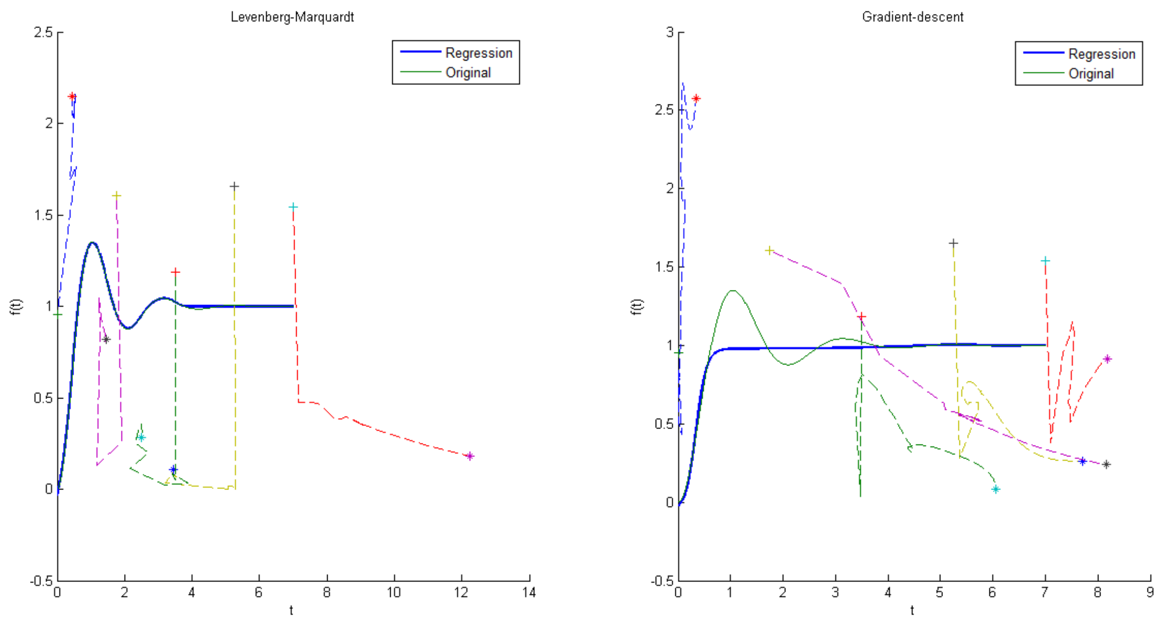
After several trials, the best result is showed in Figure 5.8. It is worth mentioning that in almost all tests the LM algorithm showed similar results while the gradient-descent algorithm did not approximate well the original function. The comparison depicted in Figure 5.8 shows the best result obtained with the gradient-descent which still is not too good. Both of them had been initialized with same parameters (which used the Nguyen (1990 initialization). Also, Figure 5.8 shows the path taken by each neuron during the training process where the cross symbol means the starting point and the asterisk is the ending point. One should notice that as they both start with same parameters, the cross points are equal in both graphs. With LM algorithm the hidden neurons basically change their amplitude while their positions do not alter much which means that the parameters initialization had an important role in the regression.

Figure 5.8 - Best training result



Another test is depicted in Figure 5.9 which shows a bad result from the gradient-descent method. Now, with the LM algorithm the neurons move left so the first four neurons can cover the entire function and the last neuron moves away. This means that this result could be obtained with only four neurons. In contrast, the gradient-descent with the same initialization shows a bad result where most neurons were accumulated in the right part. Thus, most nonlinearities are now concentrated in the end of the graph where there is little or no variations. Too few neurons were left in the critical part where the signal has great variations, which resulted in a poor approximation. Finally, most trials with LM algorithm took only 15 iterations in average while all gradient-descent trials were fixed to 1000 iterations.

Figure 5.9 - Bad training result



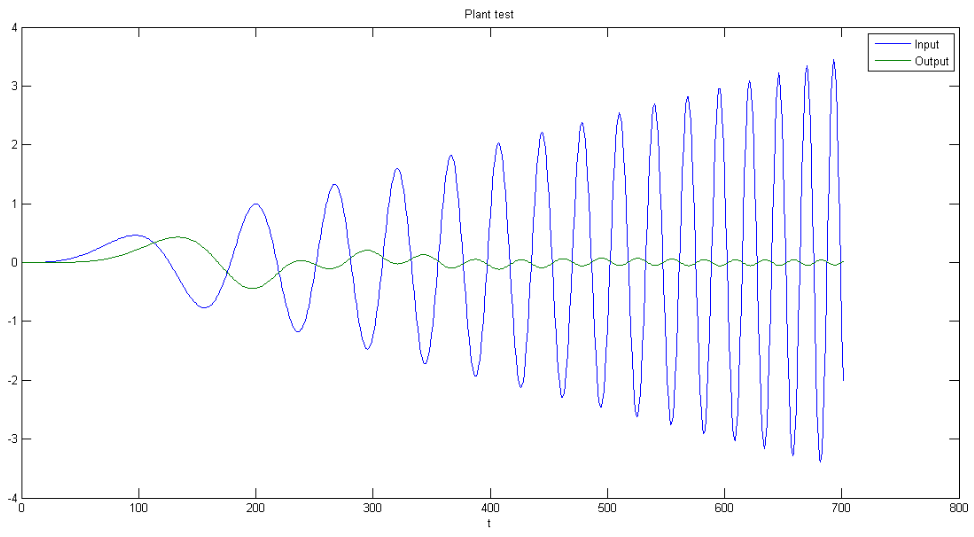
Based on those results, we choose to use LM algorithm to train our recurrent network even though it is to be used in batch-mode. One should carefully choose the training data so the network can learn and generalize the model behavior without the need of huge amount of data because the calculation of the Jacobian matrix can become too costly. As the recurrent network only differs from the above example in the number of inputs, the derivations will not be shown here although steps followed to do so are basically the same. The selection of the training data should be done taking the signal frequencies into consideration. That is, linear dynamical systems can be described by a transfer function which relates the gain of the system with respect to each frequency. Even though our system is not linear, in most times its region of operation is very close to a linear system. Therefore, in order to learn the system dynamics one must know how the system behavior to each frequency input is. Thus, one should consider using signals which are composed by a wide range of frequencies, also called rich signals. Theoretically an impulse signal must contain all frequencies with same amplitude, but as it is a theoretical signal it cannot be used in practice. Even if we use a real approximation to the theoretical signal usually this kind of signal will not produce readable outputs due to its short duration and also most actuators are not capable of producing such high signals in order to make the system react. So, for our tests we will use a sinusoidal signal composed by a range of frequencies from 0 to 14 rad/s whose amplitude starts also in zero and is increased linearly until 7, depicted in Figure 5.10. These values were arbitrary chosen although we want this range to include the oscillating frequency of our transfer function example (Equation (5.2)). That is, the plant oscillating frequency w_d is calculated by:

$$w_d = w_n \sqrt{1-\xi^2} = 3\, rad/s. \tag{5.31}$$

where $w_n = \sqrt{10}$ is the natural frequency and $\xi = \frac{1}{\sqrt{10}}$ is the damping factor.

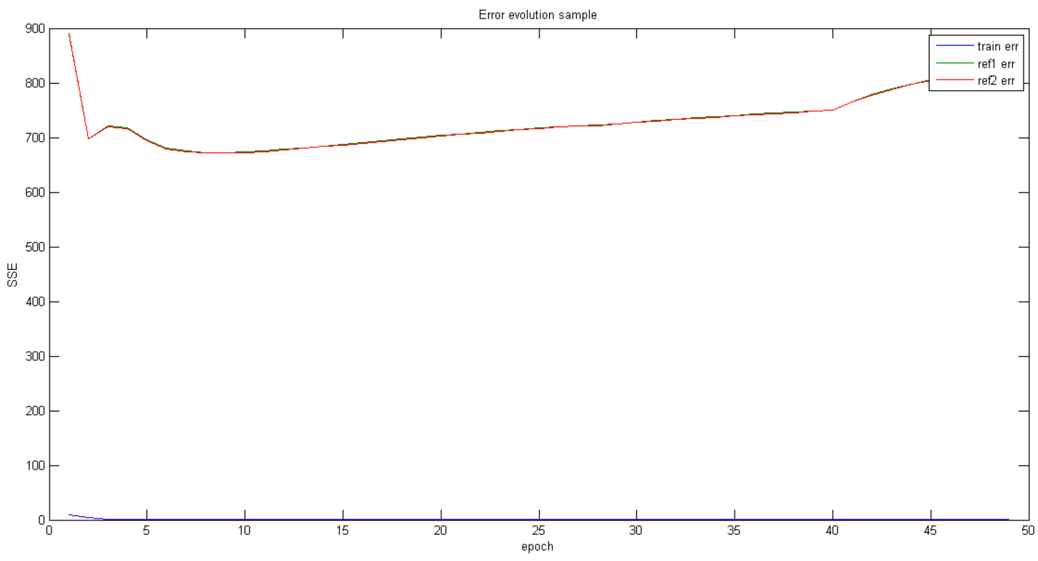
Our algorithm trains 10 neural networks and chooses the best result. Even though network parameters are initialized according to Nguyen (1990), some parameters are still initialized randomly. Therefore, each new training process can give different results, which is shown in Figure 5.12 where each trial represents a new network being trained. The performance comparison will be made according to the error criterion from Equation (5.5) but not only using the training data set. In other words, besides the sinusoidal input signal used for training the network, the total error used for comparisons will be calculated with other two input signals (called verification signals). This will be used to evaluate the network generalization capacity. So, even though one network can achieve low error levels with the training data set, it will be no useful if it produces large error values when applying unseen input signals. Thus, the best among the 10 networks will be selected based on the lowest error levels calculated with the training data set and with other two new data sets.

Figure 5.10 - Test signal used in the example system



We can see in Figure 5.11 a sample of error evolution through epochs of training from one of the ten networks. It is worth mentioning that even though the network error is decreasing when calculated using the training data set, we can see that its generalization capacity starts decreasing from a certain point and at each new training iteration the error calculated using unseen inputs (not used to train the network) increases. Taking this into consideration, our LM algorithm was modified so it returns the best network configuration found during the whole training process (which is approximately in the 8th epoch on the above example).

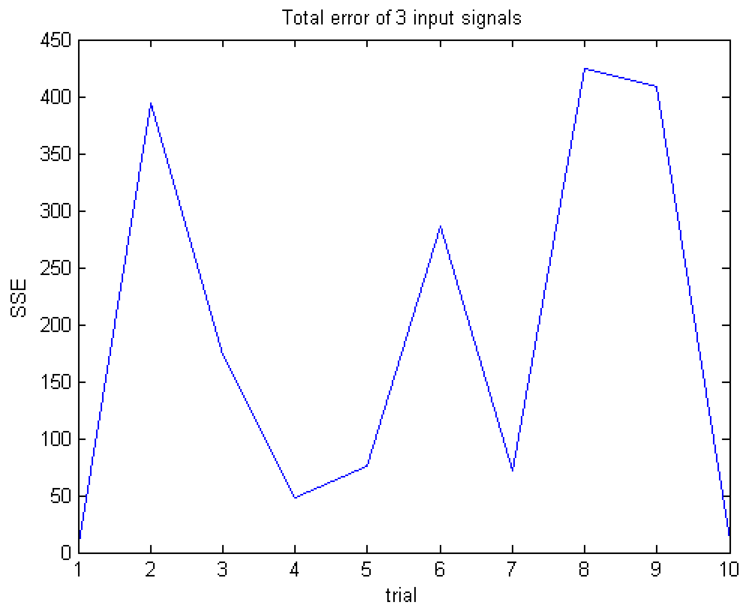
Figure 5.11 - Error evolution during a single trial



In Chapter 4 it was mentioned that the IGMN would be used for comparison purposes. Despite the differences between the networks architectures one can consider they both to be black boxes which receives the same input and should produce similar outputs in this comparison. The traditional feedforward network will do that by a combination of hyperbolic tangent functions as seen above while the IGMN will do the same work by a combination of Gaussian mixtures. Training algorithm to the IGMN is described by Heinen (2010) and will not be investigated here like the backpropagation algorithms since it already produces good results as is and the only proposed algorithm to this network until now.

The first test with the IGMN was using a single output that represents a 1-step ahead prediction like the feedforward tests. Due to its simplicity to operate (i.e. no need to recalculate all derivatives and backpropagation terms in order to train the network with a different number of inputs and outputs) we performed another test with a 10-step ahead prediction. That is, instead of a single output, now the IGMN has 10 outputs which represent the prediction of the next 10 steps of the plant. The basic principle of the IGMN is that one must provide a set of samples so the network will create Gaussian mixtures with as many components as the set has. The network itself does not know what terms are inputs or outputs in this set. So when the user performs a recall operation (i.e. provide a set with missing terms) the network will generate the missing terms. Therefore, one can view this operation as providing inputs and reading outputs.

Figure 5.12 - Sum squared error in each trial



Even though we achieved satisfying results, the neural controller itself was not implemented in the simulator or in the real system. The reasons will be discussed in next chapters along with the presentation of all results obtained here. All comparisons between networks and further comments will be also presented in next chapters.

5.1.3 VRFT

In previous sections we saw the main pros and cons of using the presented methods. On one hand, a classical control theory approach (e.g. simple PID controller) usually requires prior knowledge of the plant and will not be adaptive controllers in simple cases like a standard PID. On the other hand, an artificial intelligence approach (e.g. artificial neural networks) seems to be more powerful but it is also much more complex to deal with. Depending on the network structure one can easily implement an adaptive controller making use of the network learning capabilities. In an attempt to combine these main features of each method, we are going to explore the Virtual Reference Feedback Tuning algorithm proposed by Campi (2002) which is based on the classical control theories but has important features such as no need of the plant model as well as the possibility to create an adaptive controller since it only depends on readable data and can be done in real time.

The basic idea of the VRFT method is to use a virtual reference to calculate the controller. Usually in traditional methods one has the plant model so the controller can be found based on a reference model that represents the whole closed-loop system transfer function. That is, we establish what behavior we want through a transfer function and the only unknown block inside the closed-loop is the controller which is then calculated. Since we do not have the plant model, the VRFT method proposed by Campi (2002) is very suitable as one of its main features is the controller design without the plant model. It begins by providing a known input $u(t)$ to the system and reading its output $y(t)$. Similarly to traditional methods we also must establish a reference model $M(z)$ which has the characteristics that we want for our system in closed-loop as a single block. Therefore, we calculate our virtual reference $\bar{r}(t)$ that satisfies $\bar{r}(t)M(z) = y(t)$ (abstracting the operators since the signals are represented in different domains). Since we know the signal $u(t)$ that must be applied to the plant so it generates $y(t)$, then we can calculate a controller which tries to generate $u(t)$ when some error signal $e(t)$ is presented in its input. One should notice that the error signal is also known since $e(t) = \bar{r}(t) - y(t)$. The controller usually has a predefined format which contains parameters to be optimized so the minimum error between the ideal controller and the achievable controller is found.

Since the VRFT tests will be made using the simulator, the noiseless case described by Campi (2002) will be used here. First we should define our $M(z)$ which is the reference model that we want to approximate and is used by the algorithm to calculate the controller parameters. Even though our plant can be well approximated by a second order system, we can use a first order $M(z)$ since it can have fast response with zero overshoot depending on its pole location. The VRFT algorithm then minimizes a cost function such that the real system behavior is the closest possible to the $M(z)$ behavior according to the cost function parameters. We can choose the $M(z)$ pole location based on the performance that we want the system to have. That is, we can imagine a first-order continuous system that has a settling time of approximately 0.1s, so:

$$0.1 \approx \frac{4}{\sigma} \tag{5.32}$$

where σ is the pole of this continuous system. Therefore,

$$\sigma = 40 \tag{5.33}$$

and we can find the equivalent pole location in the discrete system through the bilinear transform relation:

$$\sigma = \frac{2}{T_s}\frac{\mathrm{z_p} - 1}{z_p + 1} \tag{5.34}$$

where T_s is the sampling period and z_p is the pole location in the discrete system. Using a sampling period of $1ms$ we find that z_p must be approximately 0.96.Thus, our desirable system will be:

$$M(z) = \frac{1 - z_p}{z - z_p} = \frac{0.04}{z - 0.96}$$

which has its pole in 0.96 and unitary DC gain. After we build the $M(z)$ transfer function, we should filter the input signal and the error signal:

$$e_L(t) = L(z)e(t) \tag{5.35}$$

$$u_L(t) = L(z)u(t) \tag{5.36}$$

where $e_L(t)$ and $u_L(t)$ are the filtered signals and $L(z)$ is a filter that one should choose according to some rules stated by Campi (2002). As our case uses simple functions and low (or none) noise, we do not need such filter to obtain good results (BAZANELLA, 2012), so:

$$L(z) = 1 \tag{5.37}$$

And so we define the cost function to be minimized as:

$$J_{VR}(\theta) = \frac{1}{N}\sum_{t=1}^{N}(u_L(t) - C(z;\theta)e_L(t))^2 \tag{5.38}$$

where N is the number of samples, θ represents the parameters to be optimized and $C(z;\theta)$ is the controller defined by:

$$C(z;\theta) = \beta^T(z)\theta \tag{5.39}$$

where $\beta(z) = [\beta_1(z)\ \beta_2(z)\ \ldots\ \beta_n(z)]^T$ is a vector of transfer functions and $\theta = [\vartheta_1\ \ \vartheta_2\ \ \ldots\ \ \vartheta_n]^T$ is the parameter vector and n is the number of parameters to be optimized. We will use two different controllers here: PI and PID. The first one is a bit more sophisticated than a simple P controller as the integral term should lead us to zero steady-state error. Its transfer function can be described by:

$$C_{PI}(z;\theta) = \frac{\vartheta_1 z + \vartheta_2}{z - 1} = \frac{\vartheta_1 z}{z - 1} + \frac{\vartheta_2}{z - 1} = \begin{bmatrix}\frac{z}{z-1} & \frac{1}{z-1}\end{bmatrix}\begin{bmatrix}\vartheta_1 \\ \vartheta_2\end{bmatrix} \tag{5.40}$$

And the second one has an additional term (derivative) to improve the system transient performance. Its transfer function will be:

$$C_{PID}(z;\theta) = \frac{\vartheta_1 z^2 + \vartheta_2 z + \vartheta_3}{z(z-1)} = \begin{bmatrix}\frac{z^2}{z(z-1)} & \frac{z}{z(z-1)} & \frac{1}{z(z-1)}\end{bmatrix}\begin{bmatrix}\vartheta_1 \\ \vartheta_2 \\ \vartheta_3\end{bmatrix} \tag{5.41}$$

Finally, as the cost function $J_{VR}(\theta)$ is quadratic with respect to θ, we can derive Equation (5.38) and find its minimum. Therefore, the parameter vector that minimizes $J_{VR}(\theta)$ can be calculated by:

$$\hat{\theta} = \left[\sum_{t=1}^{N} \varphi_L(t)\varphi_L(t)^T\right]^{-1} \sum_{t=1}^{N} \varphi_L(t)u_L(t) \tag{5.42}$$

where

$$\varphi_L(t) = \beta(z)e_L(t) \tag{5.43}$$

Although more steps are described by Campi (2002), the above steps should be enough to produce good results in our case (BAZANELLA, 2012). The obtained results are showed and discussed in Chapter 6 and Chapter 7.

5.2 Filters

In order to use the discussed controllers in practice, one should design one or more filters to eliminate noise as much as possible from sensor readings and also to combine different sensors information to find most accurate values. Although many filters can be used to these purposes we are going to mention only the most used in quadcopters and will go into details only with two of them. When selecting a filter one must choose between a more complex filter which requires more computation but generate most accurate results or a simpler filter which requires less computation but generates worse results. When using a complex filter, probably the most used with quadcopter is the Kalman filter. Many variants of this filter have been proposed, but in its traditional form it requires the plant dynamical model. Therefore, as we do not have the plant model this filter will not be used here. Two much simpler filters can be used and produce satisfying results to our application as we will see.

The first one is a simple single-pole recursive filter. It can be simply described in the time domain by the following equation:

$$y_f(t) = (1-\alpha)y_s(t) + \alpha y_f(t-1) \tag{5.44}$$

where y_f is the filtered signal, $y_s(t)$ is the sensor signal (unfiltered) and α is the filter parameter which is usually between 0.9 and 1.0. In the Z domain it is described as:

$$y_f(z) = (1-\alpha)y_s(z) + \alpha y_f(z)z^{-1} \tag{5.45}$$

$$y_f(z)(1-\alpha z^{-1}) = y_s(z)(1-\alpha) \tag{5.46}$$

$$f(z) = \frac{y_f(z)}{y_s(z)} = \frac{(1-\alpha)}{(1-\alpha z^{-1})} \tag{5.47}$$

which is a low-pass filter that has a pole at α, a zero at origin and unity DC gain. According to Oppenheim (1989) we can use the bilinear transform to find the relation between discrete and continuous frequencies:

$$\Omega = 2\tan^{-1}\left(\frac{\omega T_s}{2}\right) \tag{5.48}$$

where Ω is the angle within the unity circle in the z-plane which represents the filter frequency response, ω is the frequency in continuous domain in rad/s and T_s is the sampling period in seconds. We are going to use $T_s = 1ms$ which is a reasonable sampling time to use both in the simulator and in the real plant. Thus, our cut-off frequency will be 40 Hz which also seems a reasonable value to attenuate most noise and still not interfere much in the sensor data acquired from the quadcopter movement. Therefore, our discrete angle according to Equation (5.48) will be:

$$\Omega_c = 2\tan^{-1}(40\pi 0.001) \approx 0.25 \tag{5.49}$$

Now we want the filter presented by Equation (5.47) to have its pole in such a way that its cut-off frequency matches above values. So, using the definition of cut-off frequency to be the frequency where the absolute value of the filter output is $\frac{1}{\sqrt{2}}$ and considering $z = e^{j\Omega}$ to make this frequency analysis, we have:

$$\left|\frac{f\left(e^{j\Omega_c}\right)}{f(e^{j0})}\right| = \left|\frac{\frac{(1-\alpha)}{\left(1-\alpha\left(e^{j\Omega_c}\right)^{-1}\right)}}{\frac{1-\alpha}{1-\alpha}}\right| = \left|\frac{e^{j\Omega_c}(1-\alpha)}{e^{j\Omega_c}-\alpha}\right| = \frac{1}{\sqrt{2}} \tag{5.50}$$

$$\left|e^{j\Omega_c}(1-\alpha)\right|\sqrt{2} = \left|e^{j\Omega_c}-\alpha\right| \tag{5.51}$$

$$\sqrt{2}(1-a) = \left|e^{j\Omega_c}-a\right| \tag{5.52}$$

$$\sqrt{2}(1-a) = \left|\cos(\mathbf{\Omega_c}) + j\sin(\mathbf{\Omega_c}) - a\right| \tag{5.53}$$

$$\sqrt{2}(1-a) = \sqrt{(\cos(\mathbf{\Omega_c})-a)^2 + (\sin(\mathbf{\Omega_c}))^2} \tag{5.54}$$

$$2(1-a)^2 = \cos^2(\mathbf{\Omega_c}) - 2.a.\cos(\mathbf{\Omega_c}) + a^2 + \sin^2(\mathbf{\Omega_c}) \tag{5.55}$$

$$2 - 4a + 2a^2 = a^2 - 2.\mathrm{a}.\cos(\mathbf{\Omega_c}) + 1 \tag{5.56}$$

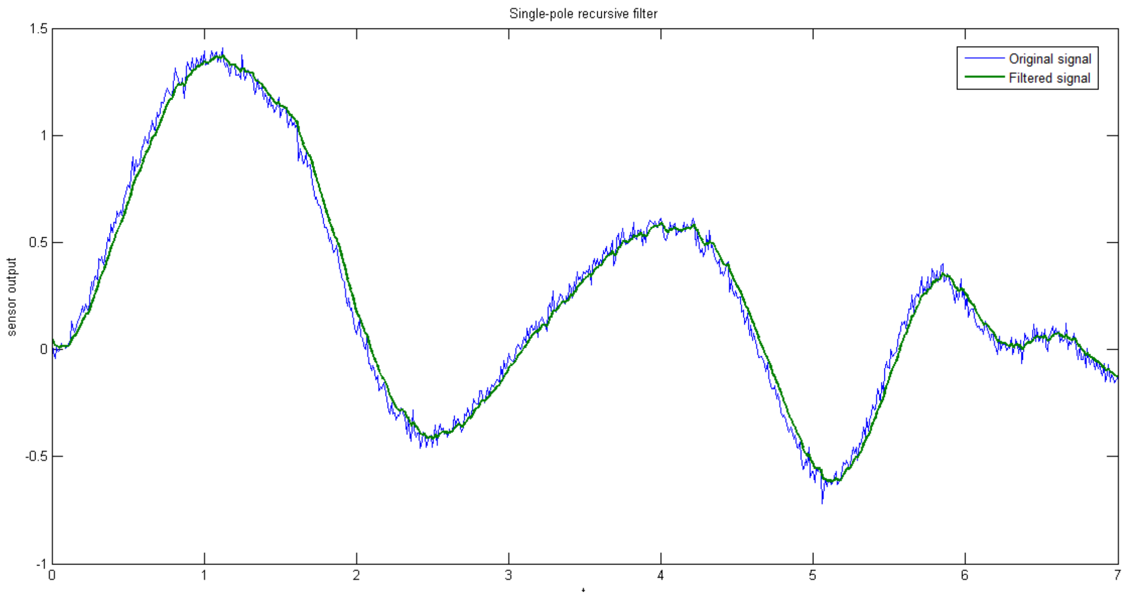
$$a^2 + a(2\cos(\mathbf{\Omega_c}) - 4) + 1 = 0 \tag{5.57}$$

where Ω_c is our cut-off frequency and α is our pole value. As the above equation is quadratic we should find two possible values for α. However, only one should be in our region of interest which is between 0 and 1 (i.e. inside unity circle to have a stable filter). So, with $\Omega_c = 0.25$ we have:

$$\alpha = \begin{cases} 1.2824 \\ 0.7798 \end{cases} \tag{5.58}$$

Therefore, we can round the second value and use $\alpha = 0.78$ which is under common used values because we were more conservative when selecting the cut-off frequency in the continuous domain. Using the same theoretical plant that was used in previous sections, some arbitrary data was generated applying an arbitrary input to the system, reading its output and adding some Gaussian noise to it. In Figure 5.14 we can see a Bode plot of the filter response. One should notice that besides the low decaying of the filter magnitude after the cut-off frequency (due to its low order) we will also have a considerable phase shifting at certain frequencies. The positive side is that the worst scenario in the phase plot is at frequencies above the cut-off frequency. In Figure 5.13 we can see the filter application to our arbitrary signal with noise and conclude that it is not near perfection but still shows a very good result.

Figure 5.13 - Single-pole recursive low-pass filter



According to Smith (1997) a very similar filter in terms of performance and complexity is the median filter. The basic idea of a median filter is to use a sliding window throughout the signal which finds the median value of the window at each iteration and this will be the filter output at that sample time. The main problem of this filter is the need of finding the median value of the window at every iteration, which by the way requires the window ordering. Besides that, large windows will necessarily introduce a phase shift to the filter output since we need to fill the window with many samples before starting to produce the output filtered samples. This problem can be reduced when high data sampling frequency is used so the window with many samples still represents a small time range due to the low time sampling period. The ordination problem will be optimized here with a modification to the standard algorithm. Instead of reordering the window at each new element insertion, we will keep track of elements indices so we only have to reorder the new inserted element at each iteration. Therefore, in the worst case we will have a linear complexity inside the “for loop” when compared to a $nlog(n)$ (where n is the window size) complexity of the original filter (see proof in appendix).

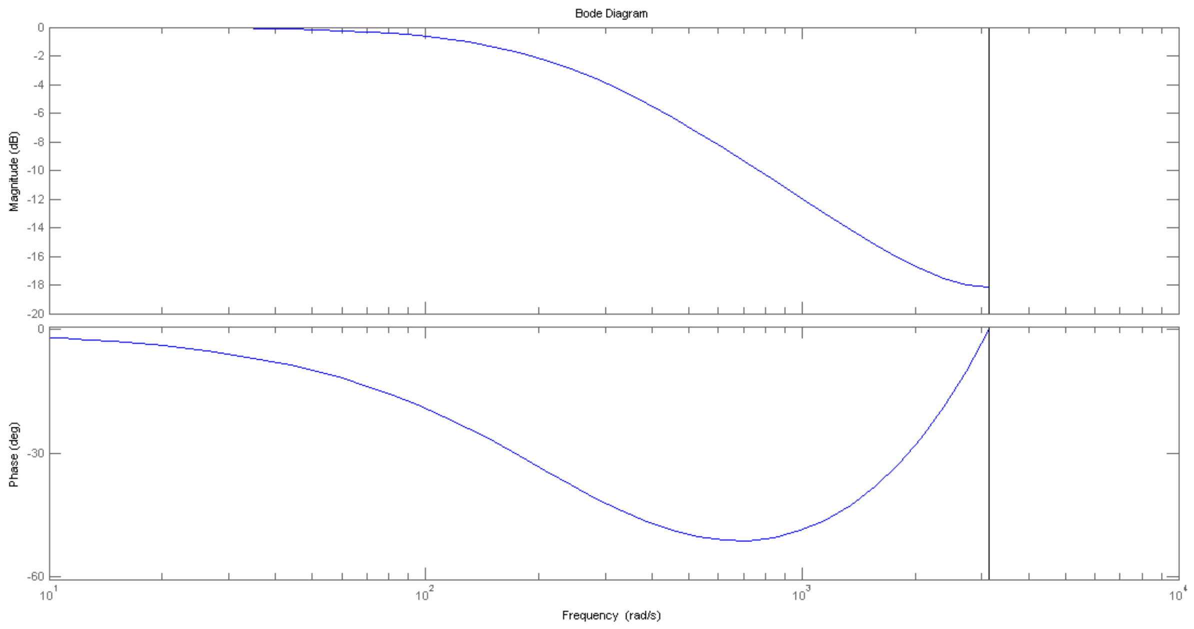
In Figure 5.15 we can see the result of an application of the modified median filter with the same signal used to test the recursive filter with a 15 sample window length. The result is very similar to the single-pole recursive filter but we can already notice a slight phase shift. Thus, we choose to use the single-pole recursive filter.

Finally, we must combine our sensors data in order to improve the overall accuracy like the Kalman filter which uses not only the plant model but the combination of available sensors in order to produce a reliable result. Since our filters are much simpler and will only filter a single signal, we must perform some operation to gather information from the sensors and produce more solid information. This will be done using a structure with the same appearance of the single-pole recursive filter. That is:

$$angle = accel_{angle}\,(1-\alpha) + (\alpha)gyro_{angle} \tag{5.59}$$

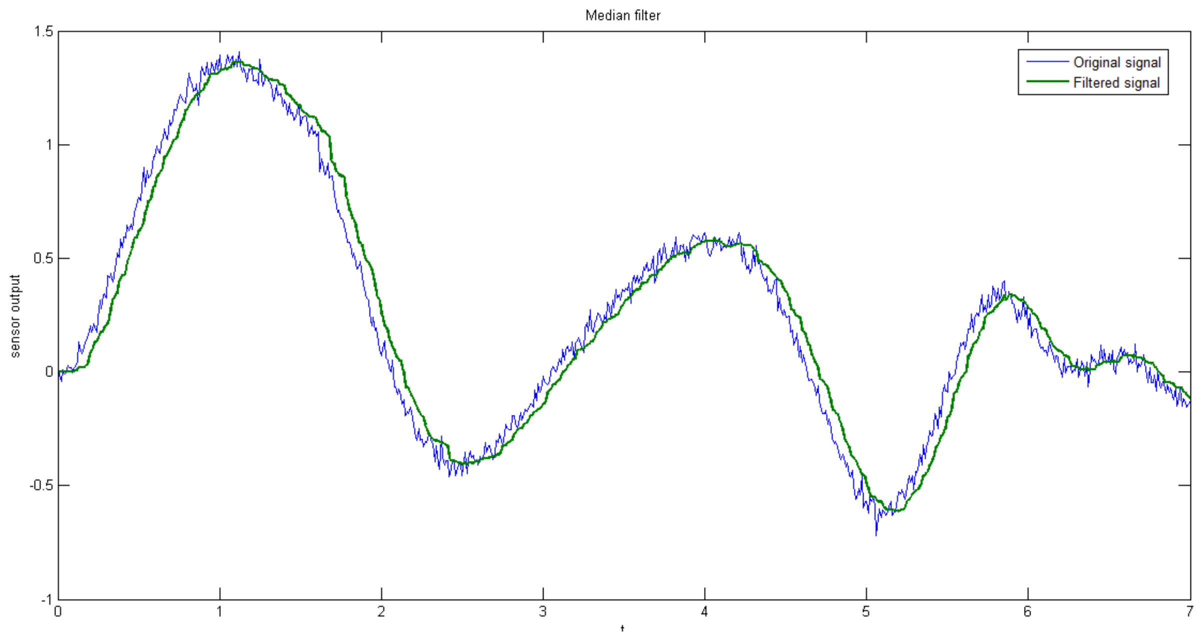
where $accel\_angle$ is the angle inferred from the accelerometer data, $gyro\_accel$ is the angle inferred from the gyroscope, α is again a constant between 0 and 1 and usually used between 0.9 and 1, and $angle$ is the combination result.

Figure 5.14 - Single-pole recursive low-pass filter bode plot



The intuitive idea behind this filter is to read fast variations almost only from the gyroscope which is more reliable for this purpose while slow variations will be mostly acquired by the accelerometer. This will minimize imperfections from the accelerometers for fast variations as well as correct the gyroscope drift in long-term. One should notice that before applying this filter one must calculate the angles from each sensor. For the gyroscope sensor we can easily find the angle by integrating the signal since it reads angular velocity. This integration process is one of the main factors that contribute to the sensor drift in the long run. With the accelerometer data we must find the orientation of the acceleration vector provided by the sensor. One must be careful that despite the sensor name, an accelerometer does not read common acceleration that we are used to work with. Instead, it measures proper acceleration which can be intuitively understood by imagining the acceleration vector of a free-fall object with respect to the object being measured. So, an object at rest will have an proper acceleration vector pointing down and an object accelerating to the right will have an proper acceleration component pointing to the left. Thus, we must find the quadcopter normal vector through the accelerometer data so we can infer our angles or directions. Another important point is that if the quadcopter is in fact accelerating at some direction, we cannot know if the acceleration component that will appear is from an accelerating movement or if the quadcopter is only tilting without actually accelerating towards that direction. However, as the accelerometer data is used mostly for long-term correction, then this problem is not critical.

Figure 5.15 - Median filter



5.3 Component choice and system assembly

In order to build a usable real system one should choose its physical components that suit the system requirements. In the host side, besides the computer playing the role of commander one must have a transceiver connected to this computer in order to exchange data with the quadcopter. Therefore, it must be capable of bi-directional communication in order to send commands to the quadcopter and receive its states (acquired data). As the focus of this work is the development (and test) of control system techniques, then one might prioritize fast communication (high baud rate) and low cost. Thus, high range will not be treated as priority since tests can be realized near the host so the user can observe closely what is going on. It is also desirable that this transceiver component be light and small especially in the quadcopter side. In addition, the possibility that the user can configure the data link parameters (e.g. channel frequency, re-transmission, acknowledgement packet, etc) would also be interesting. Finally, it is helpful using a well known component since it should be easily found in the market and probably should have some documentation and examples of use on the internet.

Building on those criteria, we propose the use of the nRF24L01 from Nordic Semiconductor. It uses RF waves in ISM band with its center frequency going from 2.4 GHz up to 2.525 GHz and channel resolution of 1 MHz. The data rate can be set up to 2Mbps which is one of the highest data rates for small size, cheap and low power devices found in the market these days. Another option could be a Wi-Fi module which uses the same RF band and most common types can achieve up to 11 Mbps or 56Mbps. However, for purposes of this work there is no need of such a high data rate in exchange of more power consumption. One should note that the quadcopter side of the system will run on batteries so the power saving is essential. The nRF24L01 reaches a maximum current of 14mA in TX/RX peaks and can run in the order of µA in power down mode. Yet, some interesting features can be configured like ART (auto re-transmission), auto acknowledgement, auto packet handling (packet assembling), dynamic or static payload and CRC error check.

All these configurations and the data transmission itself must follow a protocol to go in and out. NRF24L01 uses a SPI bus to communicate with external devices. Therefore, one should build an embedded system capable of understanding the SPI protocol. Thus, since computers do not usually have peripherals that natively use SPI protocol, the host side must have an extra component to serve as an interface between the SPI and another protocol commonly used by computers like USB or RS-232. Most microcontrollers these days can do that.

Should be made clear that when it comes to choice of components there are a wide range of types and manufacturers that suits our needs so one can choose different models than those proposed here without much difference in results. Our choice in this work takes as priorities the fundamental requirements shown by the system needs but even so there are many possible options. Therefore, those choices will also take into account user preference which usually do not hold any reasonable proof of being the best possible choice but will suit our needs as desirable. Thus, a microcontroller which should fit well in the host side is a PIC18F2550 from Microchip which is a simple version of the well-known PICF18F4550 (also from Microchip) but still holds the main features that we need like native USB peripheral, SPI port and it also has some other nice features like small size and well explained documentation.

On the quadcopter side we must also choose a microcontroller to exchange data with the transceiver and with sensors and actuators. The starting point of this choice is whether this microcontroller can natively generate 4 different PWM signals (one to control each motor). As our preference is for Microchip devices then the PIC24H family looks like a great option since it is one of the simplest families whose majority of its components meet this requirement. Yet it is not a very simple family since it has a 16-bit architecture (which has greater performance compared to all 8-bit devices). It is worth noticing that devices of this family can achieve up to 40 MIPS in performance which will see later that is more than enough to deal with data sampling from the sensors. Therefore, between the remaining possibilities we will use the PIC24HJ128GP202 which is an intermediate component among the devices in this family.

The last components to be chosen are the sensors and the battery, since the actuators will be considered here to be part of the physical structure of the quadcopter which will not be discussed here (therefore any structure can be considered). The most common batteries used in quadcopters are Li-PO with variable number of cells. Each cell produces around 3.7V which may vary a little depending on its chemistry. We will use a battery that suits our actuators and should last long enough for tests. So, our choice is any 3-cell Li-PO battery (11.1V) with charge capacity of 4000mAh.

As for the sensors, it is quite common to use a single board which contains all the main sensors. Usually the main sensors are composed by an accelerometer (which measures proper acceleration) and a gyroscope (which measures angular velocity). In many cases those sensors are enough to realize stable flights, but one can add extra sensors like barometer (which measures atmospheric pressure and is used in altitude control), magnetometer (which can measure the magnetic field of the earth and is used mainly in the yaw axis orientation), distance sensors and others. Sensors operation, quadcopter orientation and related subjects were already discussed in previous chapters.

There is a wide variety of orientation sensors and when used together in a single board they are usually called an IMU (inertial measurement unit). Our system will use a 6-DOF (six degrees of freedom) IMU composed by an accelerometer (ADXL345 from Analog Devices) and a gyroscope (ITG3200 from InvenSense) which is an widely used IMU with low cost. The differences between sensors found in market are mainly in precision, drift and noise. Even though we choose a cheap IMU, one can filter the incoming data and mix values from both sensors so the resulting angle of orientation can be used in the control system without major problems. Of course one should look for better sensors as the precision sought is higher.

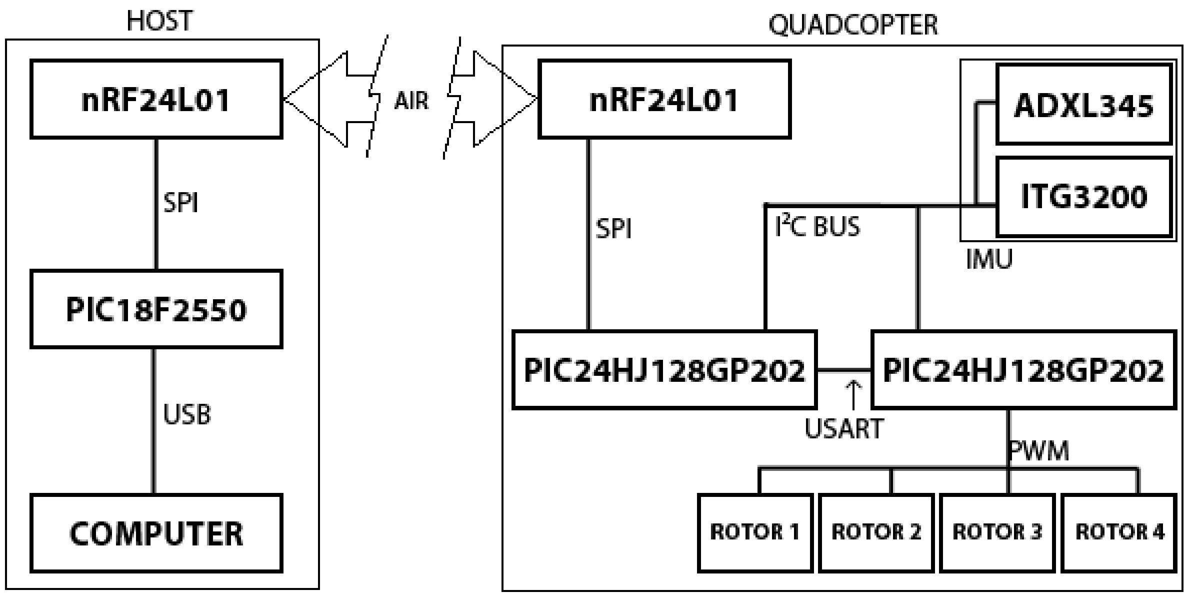
All devices and components mentioned above should be put together in some way. What follows is a proposal of assembling depicted in Figure 5.16. Note that even though the embedded system can run with a single microcontroller, our system uses two units working together. This is because some of the control systems to be tested contain heavy math to be executed.

Since we want to make the embedded system as independent as possible, those calculations should be made inside the microcontroller. However, some operations may last sufficiently long to produce bad effects on the actuators control. In other words, once established a sampling period, the control system expects that data is acquired and dealt with in that period. To be able to realize complex calculations and still respect the sampling time, one should create more complex algorithms to be able to interrupt calculations whenever needed to process the next sample. Instead, a multi-threading approach seems interesting in this case where one thread can be always dealing with the control system (sampling and updating PWM values) and the other thread can deal with transmission to the host and calculations needed to update the control system parameters. Since these microcontrollers only support single-thread operation, we use two microcontrollers together, each representing one single thread.

From now on we will call one of the microcontrollers the main microcontroller and the other the auxiliary microcontroller. The main microcontroller will be responsible for heavy calculations and exchange data with the host while the auxiliary one will be responsible for data sampling from sensors and updating actuators values. Note that while adopting a data-driven controller design where the embedded system itself will design the controller that suits it, those calculations will take place in the main microcontroller but the results should be sent to the auxiliary one.

The analogous to the cross-thread communication in the above example is a physical connection between the microcontrollers USART port. Through that port the main microcontroller can update the controller which is running inside the auxiliary microcontroller. This architecture is especially useful when using an iterative method of controller tuning because the main microcontroller will be always learning new parameters and updating the auxiliary one. Still a one-shoot method can also be used even though the main microcontroller might be underutilized.

Figure 5.16 - Complete system proposal



Yet in the quadcopter side there is a single bus of communication connecting both sensors and both microcontroller. This is a I²C bus which allows that multiple devices can be connected to it to read the same information. Note that the main microcontroller needs to read sensors information to calculate new controller parameters and the auxiliary microcontroller needs the same information to serve as feedback in its closed-loop control system. The transceiver needs only to be connected to the main microcontroller, so it is connected directly to it through its SPI port.

In the host side we have a PIC28F2550 microcontroller to serve as interface between the computer and the transceiver, as already mentioned. This microcontroller has native USB to communicate with the computer and a native SPI port to exchange data with the transceiver.

The nRF24L01 transceiver allows bi-directional communication even though we must set each one as receiver or transmitter. The communication happens by using acknowledgement packages with payload. In other words, the transmitter can transmit data normally to the receiver but when the receiver wants to transmit back some information it must use the acknowledgement packages of the communication. This means that the receiver can only send back useful information whenever the transmitter had already sent some data. This is not much of a problem since the quadcopter should always be transmitting acquired data to the computer. So, whenever the computer wants to send back some command, it uses one of the acknowledgement packages of the data packets received.

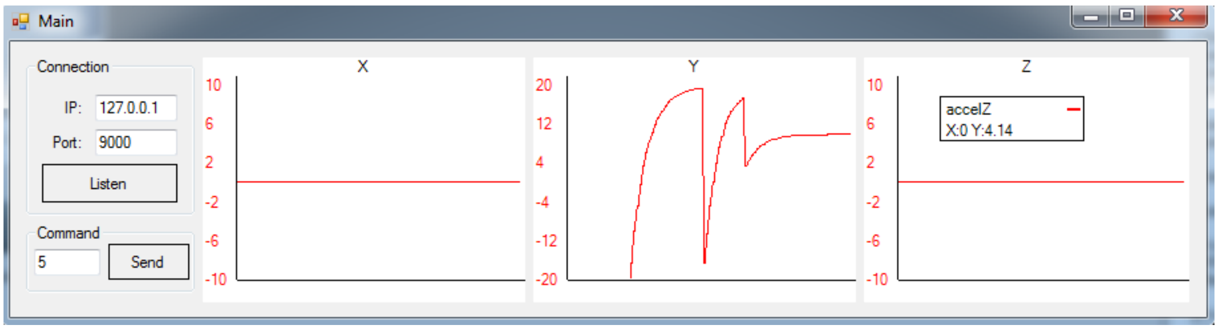
Finally, the sensors maximum output data rate is 800Hz when running I²C in the fast mode with a clock frequency of 400kHz. Therefore, the microcontrollers running at 80MHz which allows an operation of 40M instruction cycles per second will be much faster than the maximum sampling period. Even though the auxiliary microcontroller will not execute heavy calculations, it will have enough time to realize the controller operations for most common controllers like PID besides reading the sensors data.

5.4 Simulator

In order to test the controlling techniques we are going to use a 3D simulator which was created specifically for this purpose since tests directly in the plant may damage the aircraft. Even though there are many flying simulators available, most of them only deals with traditional aircrafts like helicopters and airplanes and do not have a quadcopter model. Some of them allow the user to create his own model with detailed parameters. No ready and realistic quadcopter model was found for testing purposes though. Therefore, a simpler simulator was implemented without detailed physics (nonlinearities) since the objective here is only to test the control system efficacy and not the physics realism. The simulator can be divided in two parts: the host side and the 3D simulator itself.

The host side can also be used with the real plant by switching only the communication interface used (TCP to USB). The software was developed using C# language. The communication with the transceiver is made through the Microsoft HID Class which is an easy-to-use generic USB class that can communicate with devices of this class. In previous section we saw our suggestion of components to use in the real model. Thus, the microcontroller used to communicate with the computer has native USB interface which allows easy configuration as a HID device and other parameters. Similarly, the communication with the 3D simulator is made through TCP classes found in Microsoft .NET Framework. As the software purpose is mainly to monitor received signals from quadcopter, a graph library was developed to plot real-time information as the host receives sensors data. The software interface can be seen in Figure 5.17 where there are 3 instances of the graph component to represent each axis of the accelerometer sensor. In the left side we can see the TCP connection parameters and an extra textbox with a button which allows the user to send a specific command to the quadcopter. A single byte was reserved for this purpose, so we can add up to 255 different commands to the system. The USB parameters are configurable internally only since the device parameters like PID&VID (which are used to identify the device) should not be variable parameters of the hardware. A dedicated thread runs in the host background exclusively to receive incoming data. Then the main thread accesses this data so it can be plotted.

Figure 5.17 - Software interface



The 3D simulator was created using the Unity3D tool which has its own physics engine. We only need to create the 3D model and write the scripts which will describe the interactions between objects. The 3D model of the quadcopter was created using the Blender software and exported to Unity3D in FBX format. The propellers were modeled separately so they can spin freely.

As our simulator should conform to the real system, similar values of physical parameters will be used. Some of them can be extracted directly from the chosen components (e.g. PWM value or operating voltages) and others should be an approximation of most used parts (e.g. propellers geometry, motor speed, etc). Depending on the propellers type, the generated thrust can behave differently with respect to its rotation speed. Therefore, we will approximate this relation by a quadratic equation since most used propellers have a near quadratic relation. Also, a constant of proportionality is introduced so the rotation speed also becomes coherent with generated thrust. This value was also estimated and it was based on the maximum rotation speed of most common rotors that usually is between 15.000rpm(250Hz) or 18.000rpm(300Hz).

To control the DC motors, one must generate PWM signals to the ESCs so they can feed the motors correctly. The PWM parameters of most used ESCs follow the same standard where its frequency is usually around 50Hz (and some ESCs can receive up to 300Hz or 400Hz depending on the model) and the duty cycle varies from 1.0ms (stopped) to 2.0ms(full speed). Therefore, a possible and suitable configuration of our microcontroller is to use a

PWM frequency of 100Hz which will give us a resolution of 200ns. With that configuration, we can vary PWM duty cycle from 5000 (1.0ms) to 10.000(2.0ms).

Now we must establish a relation between the generated PWM signal and the propeller thrust. This relation cannot be direct since motors have angular momentum and it takes some time to achieve the desired speed. From the control theory point of view, there are two main poles in the system and they are the electrical and mechanical poles. Since the mechanical pole is much slower than the other one, we can ignore the electrical pole. Therefore, we must approximate the motor equations in order to model its real behavior. When a motor is powered, it generates back-EMF (electromotive force) which is proportional to its angular velocity:

$$\mathrm{V_{emf}} = \mathrm{k_1}\omega \qquad (5.60)$$

where ω is the angular speed in rad/s and k_1 is a constant of proportionality. Ideally this voltage (EMF) is zero when the motor is at stall speed and it should equal the power source voltage when spinning with no load. The current through the motor is then proportional to the difference between the power source voltage and the back-EMF:

$$\mathrm{I} = \frac{\mathrm{V_s} - \mathrm{V_{emf}}}{\mathrm{R}} \qquad (5.61)$$

where R is the motor resistance and V_s is the power source voltage. By combining both equations we get:

$$\mathrm{I} = \frac{\mathrm{V_s} - \mathrm{k_1}\omega}{\mathrm{R}} \qquad (5.62)$$

And finally, the torque of the motor is proportional to its current:

$$\tau = \mathrm{k_2 I} \qquad (5.63)$$

So,

$$\tau = \mathrm{k_2}\left(\frac{\mathrm{V_s} - \mathrm{k_1}\omega}{\mathrm{R}}\right) \qquad (5.64)$$

where k_2 is another constant. According to these equations there are three parameters to configure the motor behavior (k_1,k_2 and R) and we have a direct relation between the source voltage and torque of the motor. The source voltage can be obtained directly from the PWM value so the torque is calculated and applied to the physics engine in the simulator. The angular velocity is then updated so the torque will be reduced until near zero in steady state. The lift forces can be generated based on the quadratic relation with respect to the motor speed and are always perpendicular to each propeller. The last force to be generated is the force responsible for the yaw rotation. This force can be approximated by a linear or even a quadratic relation with respect to the propeller speed as well as the lifting forces since they are not a critical part of the system and will exist only to test the yaw control. To keep coherence we should set its proportionality constants much lower than the lift forces ones since this force is much smaller. All these equations and values are put into C# scripts that are natively interpreted by the Unity3D. We also kept separate files that represent distinct real components (different microcontrollers will have different source codes). A screenshot of the simulator is shown in Figure 5.18. Also, we can see in Figure 5.19 a simulation of the step response of one motor. The theoretical PWM value is applied and some RPM response is read through TCP and plotted over time. Its behavior is consistent with reality. The simulator is also capable of generating text files that can be easily imported by MATLAB software in order to perform calculations or plot the acquired data.

Figure 5.18 - Quadcopter 3D Simulator

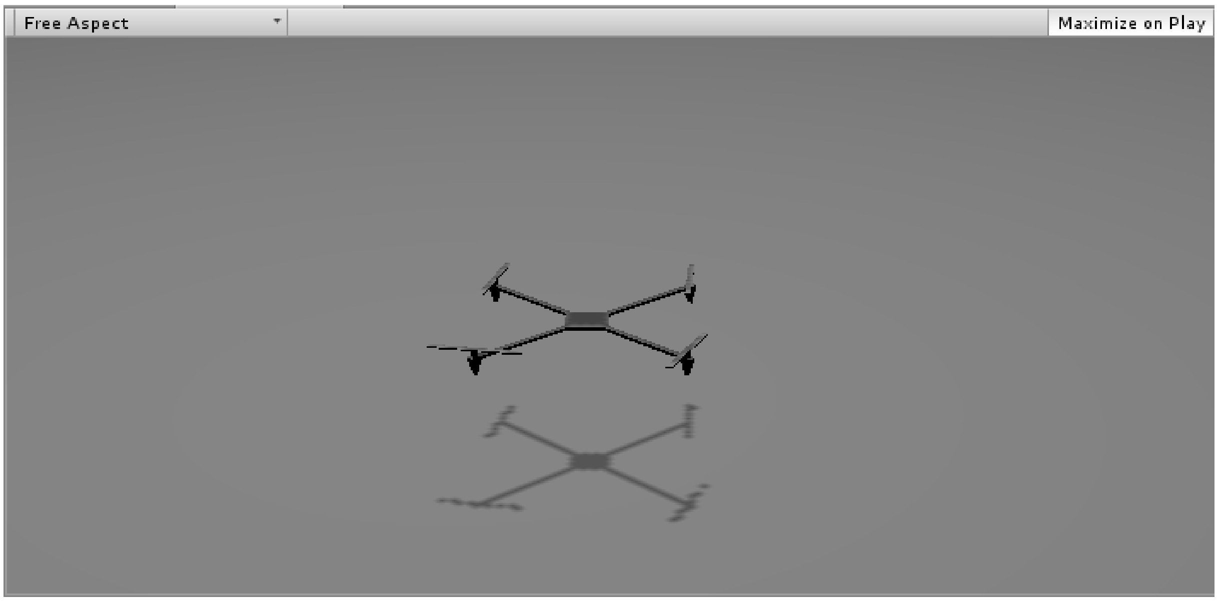
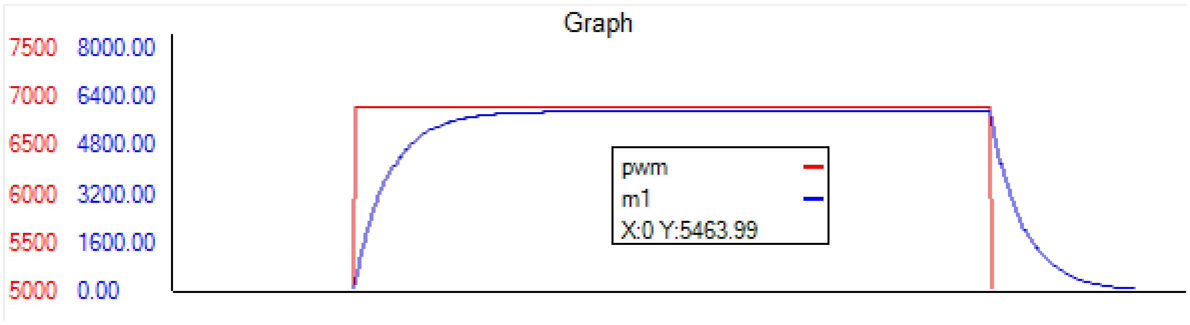


Figure 5.19 - Acquired data sample

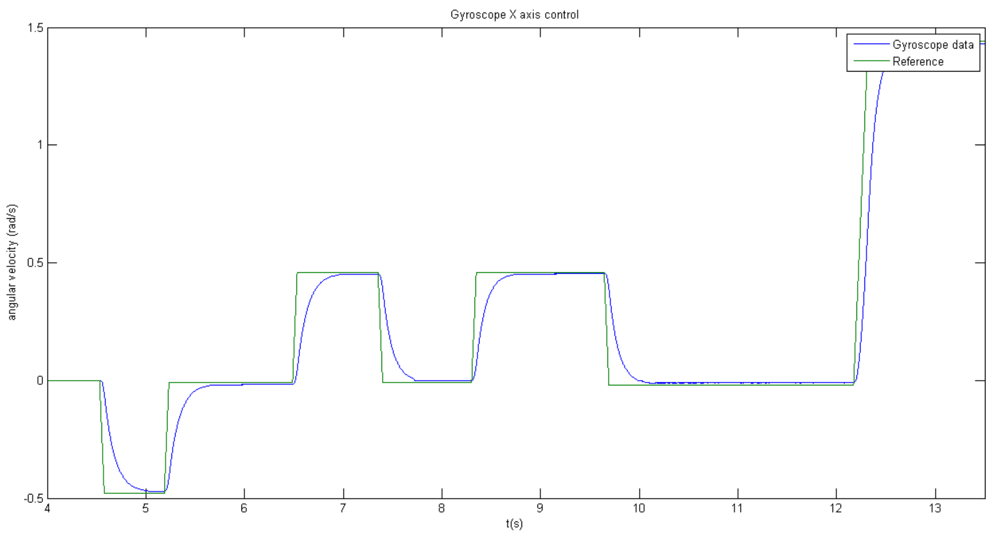


6 RESULTS AND COMPARISONS

The software worked as expected. It was possible to establish a communication channel between the host application and the 3D simulator. One should notice that an UDP connection could also be used since most packets are purely informative (used for plots) and we do not need all of them for any critical operation. However, as we want a system that could be extended to extra operations like using the host software to calculate control parameters, if needed, then it is interesting to establish a reliable communication besides the fact that it is desirable that the commands sent from the host are always received by the quadcopter.

The dynamics in the 3D simulator seems to agree with reality and even though there is a host application to receive acquired data and send flying commands, the 3D simulator shows some essential information in real time and gives the user the possibility of not using the host and controlling directly the quadcopter model through the keyboard. Also, it is possible to export data from the 3D simulator into MATLAB for further analysis (which we will do as follows).

Figure 6.1 - P controller with gyroscope data



The first simulation was made with the simplest controller within our proposals which is a simple P controller. Since we made a closed-loop with the gyroscope data only, we had a wide range of gain values that could be used without major problems. Initial tests with unit gain already worked but higher values were used so the system could respond faster. Figure 6.1 shows an angular velocity reference being followed by the system with the controller gain set to 100 in all axes. It can be seen that the closed-loop system is working as expected. Thus, Figure 6.2 shows the step response of the system. One should notice that the system response is very similar to a first order system and that the P controller is not enough to eliminate the steady state error. If we increase the controller gain, the second pole might begin to have significant influence but the system will still present steady-state error (even though it will be smaller). In Figure 6.3 we can see the same system but with controller gain set to 1000. Now the system response looks like a second order system response. The system is faster but with overshoot. Yet, although some steady state error still exists, we can see that it is very small (almost insignificant).

Figure 6.2 - Step response of closed-loop system with P controller

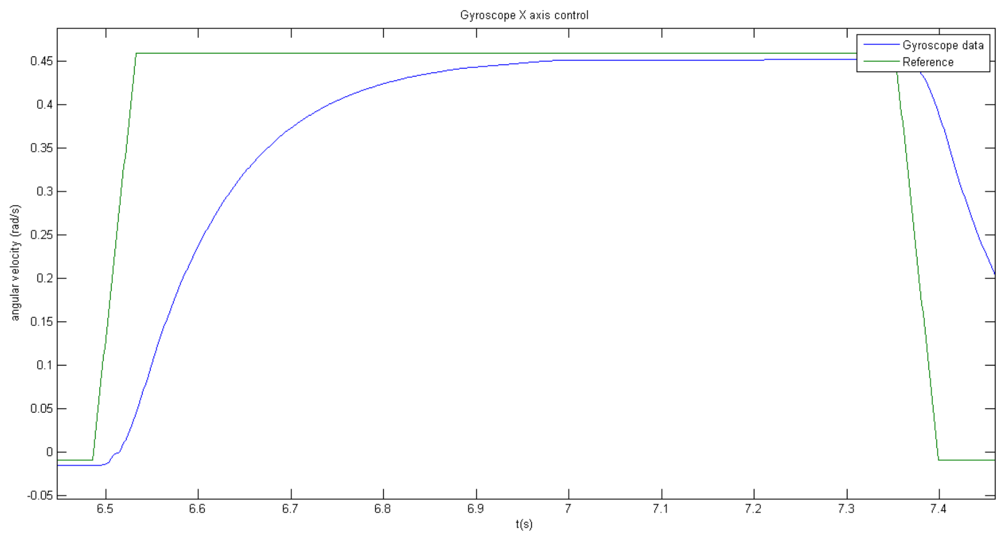
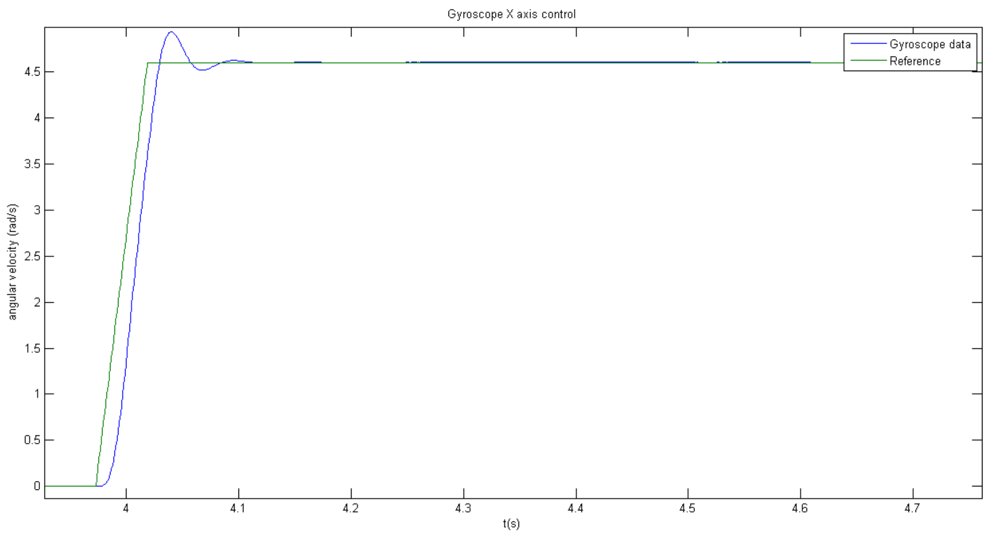


Figure 6.3 - Step response with higher gain



It is important to note that the system poles are directly related to the DC motors dynamics. Hence, Figure 6.4 shows the step response with slower motors. The resistance R in Equation (5.64) was increased and the controller gain was set to 100. In comparison with Figure 6.2 that used the same controller, we can see in Figure 6.4 that both poles of the system already have influence in its response. In Figure 6.5 we can see the step response of the system with the controller gain set to 500 (half of the controller gain in step response of Figure 6.3). With such configuration, one can notice the high overshoot and system oscillation. Therefore, depending on the physical parts of the quadcopter, the P controller gain might need a careful tuning.

Figure 6.4 - Step response with slower motors

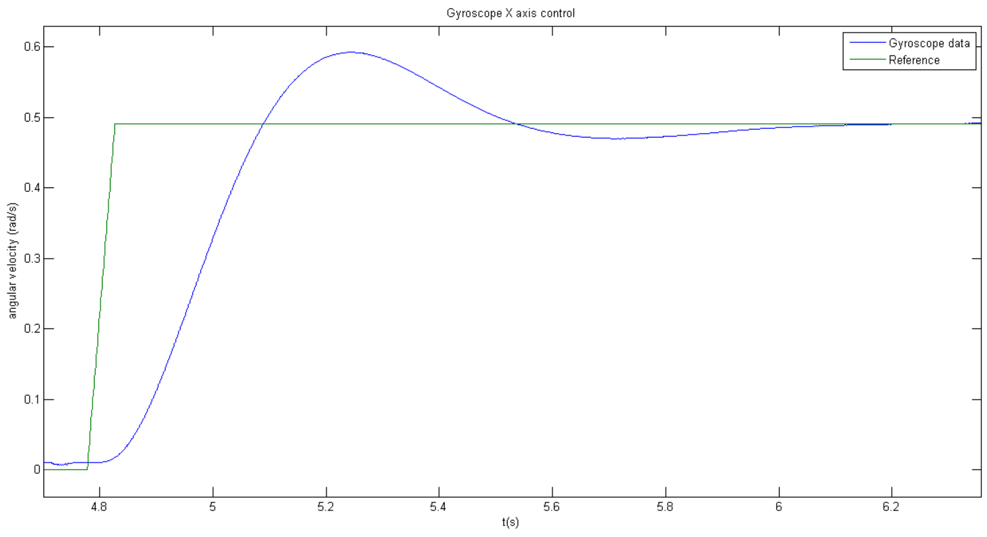
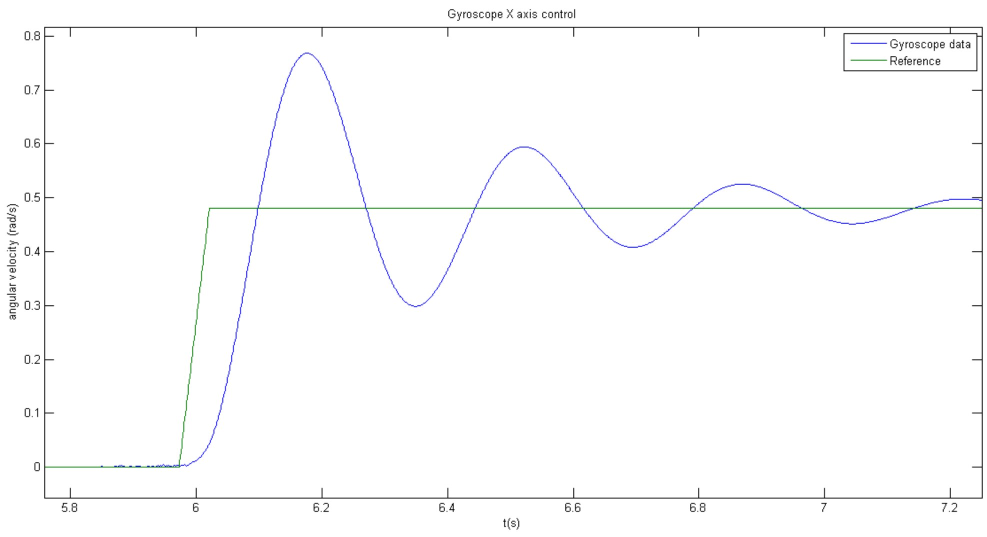


Figure 6.5 - Step response with slower motors and increased gain



Despite the system performance, the above tests show that the system is following the reference correctly. Thus, with a flyable quadcopter we can use the adaptive methods proposed in previous chapters to design a better controller. From an artificial intelligence point of view we can use an ANN to learn the quadcopter dynamics so we can design a predictive controller that always finds the best control signal based on the dynamics prediction (SOLOWAY, 1997). The following results were obtained from the theoretical model from Equation (5.2). The signal used to excite the system as well as its respective output (which is used to train the network) are shown in Figure 5.10. Figure 6.6 and Figure 6.7 show two other input and output signals used for verifying purposes. That is, we used those inputs to our trained network and compared its output with the real system output.

Figure 6.6 - Verification signal 1

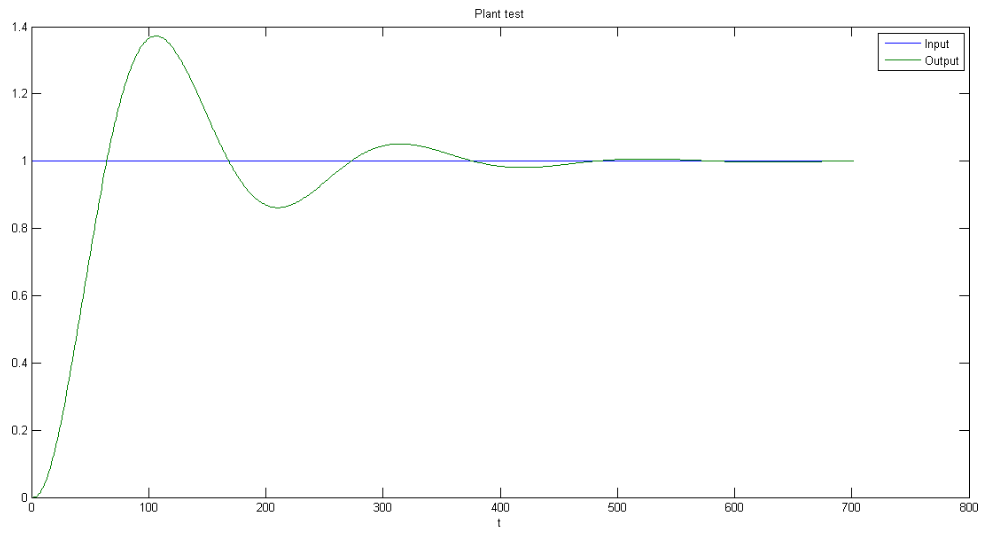
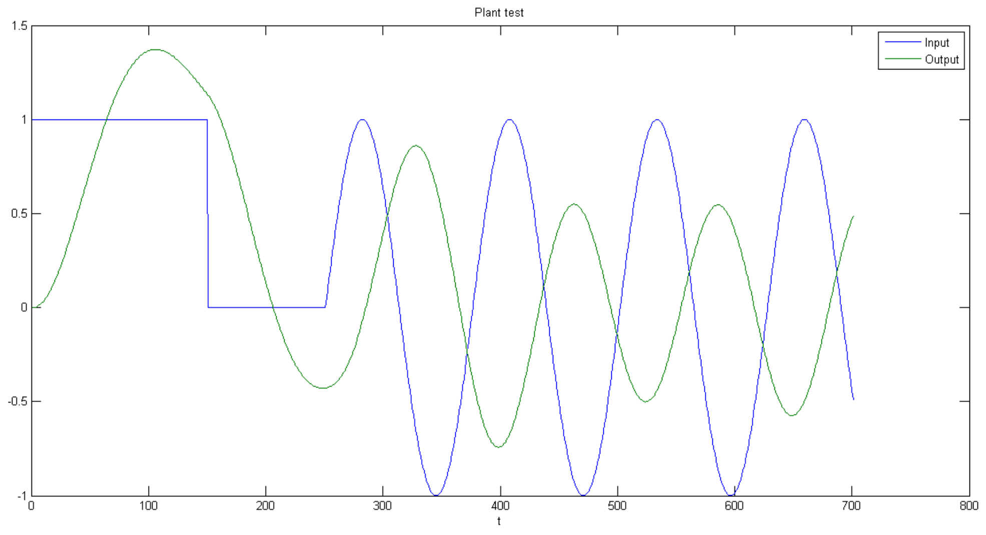


Figure 6.7 - Verification signal 2



Initially the networks structures that we created have only one output, which means they were only trained to predict the very next sample (1-step ahead prediction). The results from the traditional recurrent multi-layer perceptron trained with our Levenberg-Marquardt algorithm in a Series-Paralell configuration is shown in Figure 6.8 for the training signal and in Figure 6.9 and Figure 6.10 for the verification signals.

Figure 6.8 - Recurrent MLP 1-step ahead prediction of training signal

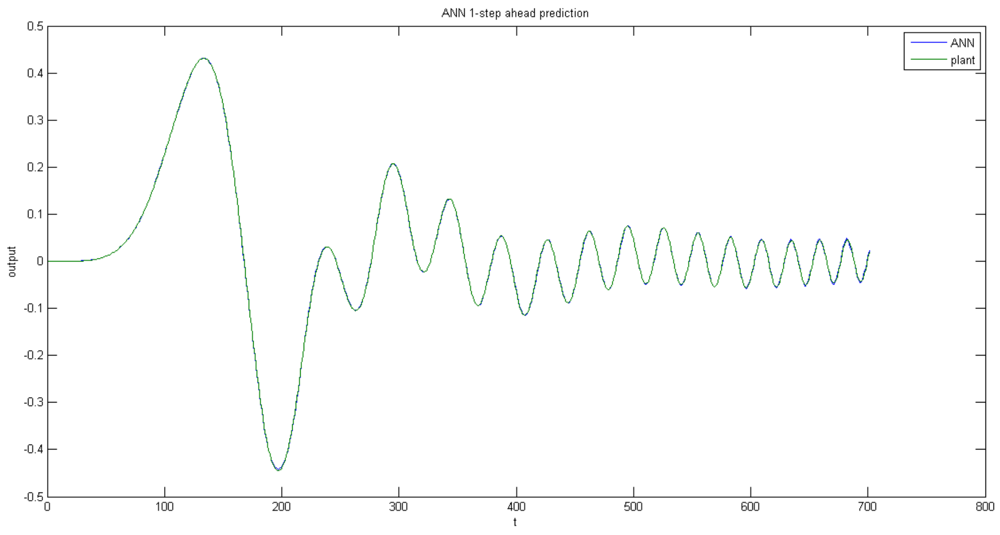
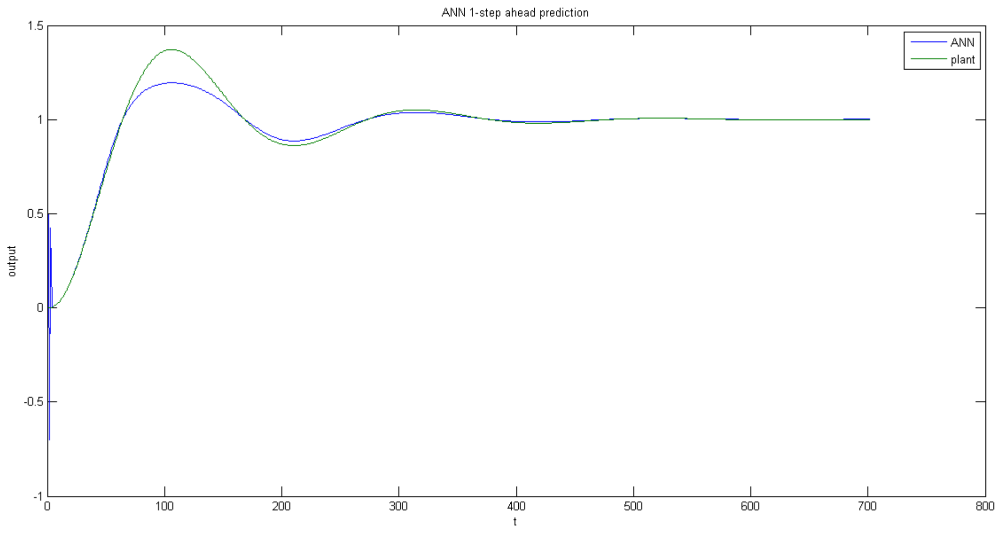


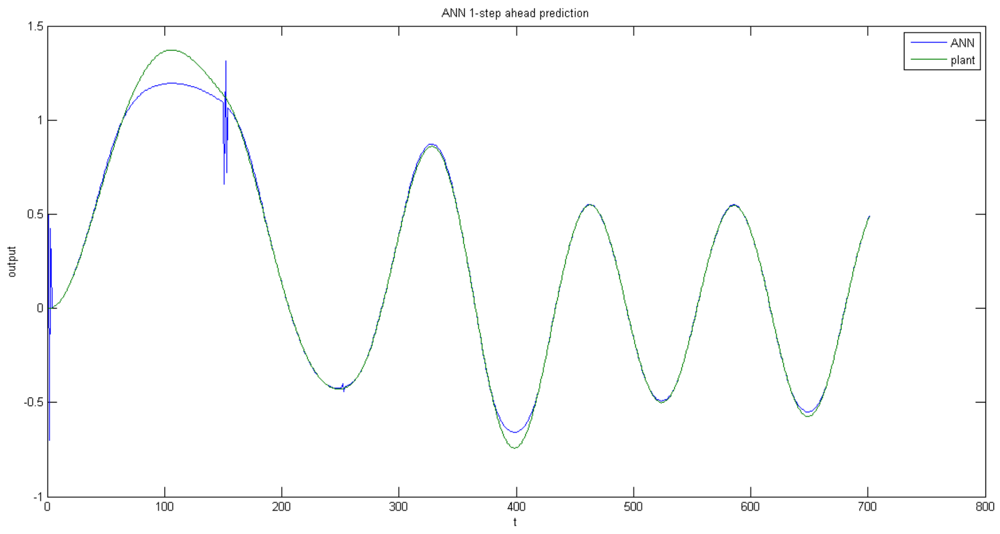
Figure 6.9 - Recurrent MLP 1-step ahead prediction of verification signal 1



One should notice that even though the network could predict the system dynamics reasonably well, significant errors still can be seen at some regions. However, it is worth mentioning that these errors occur mostly when the input signal has a step variation in its composition. Hence, when comparing the first verification signal with the network output one should notice significant error at initial moments since the input signal is a pure step signal.

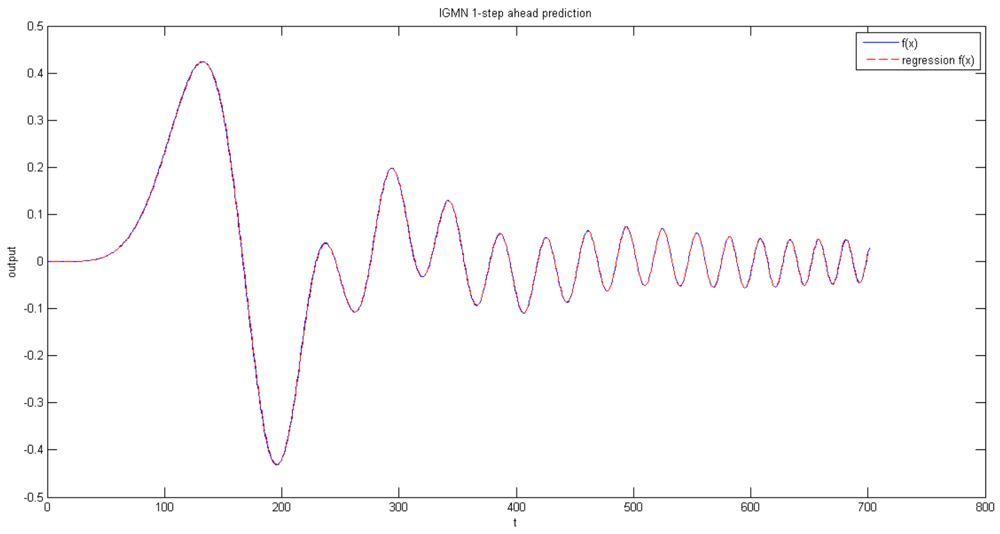
Yet, major errors in the second verification signal can be seen also in initial moments where the input signal is composed by a sequence of two step signals. This occurs because the step signal is composed by more frequencies than the frequencies that compose our training signal.

Figure 6.10 - Recurrent MLP 1-step ahead prediction of verification signal 2



The IGMN was also used to predict a single step ahead so we can compare to the recurrent MLP results. The same training signal and verification signals were used. Figure 6.11 shows the network prediction of the training signal. Figure 6.12 and Figure 6.13 show the network prediction of the verification signals.

Figure 6.11 - Recurrent IGMN 1-step ahead prediction of training signal



Although the IGMN results are somehow similar to the MLP results, one can point remarkable advantages on using the IGMN network for our purposes. First, the regression itself achieved better results than with the MLP. Second, due to its architecture and training process, a single use of each sample was enough to produce such results. Even though the LM algorithm almost always need less than 20 iterations (which is considered to be a fast convergence) to find good results, a single training iteration is considerably better. Lastly, also due to its structure, the IGMN can use each single sample at a time without the need of keeping track of more samples to perform the training process. Besides the memory savings, one should notice that as the LM algorithm needs all the samples at once, then the training process and calculations can become very costly depending on the training set size. In contrast, the IGMN can continue to improve its parameters at each new sample without keeping track of past samples or past training processes.

Figure 6.12 - Recurrent IGMN 1-step ahead prediction of verification signal 1

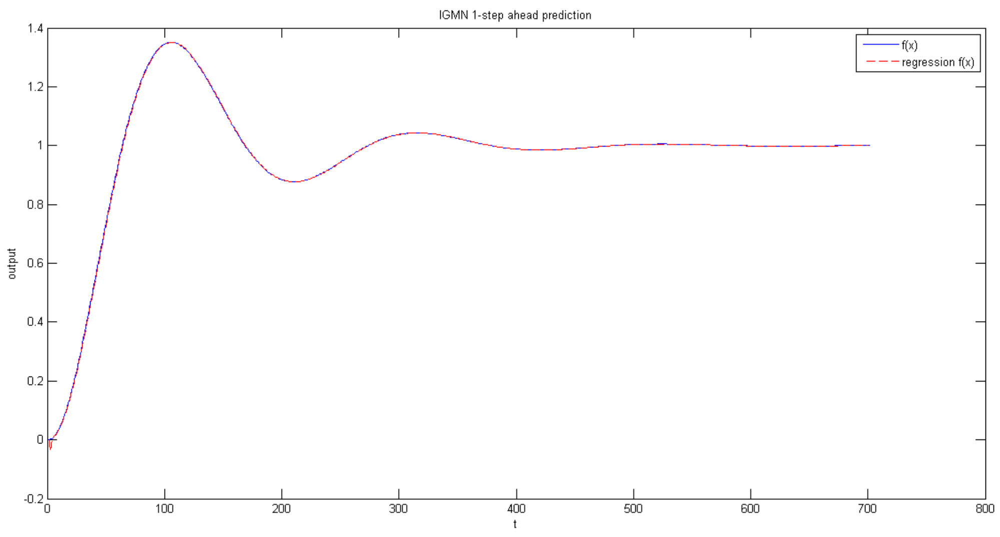
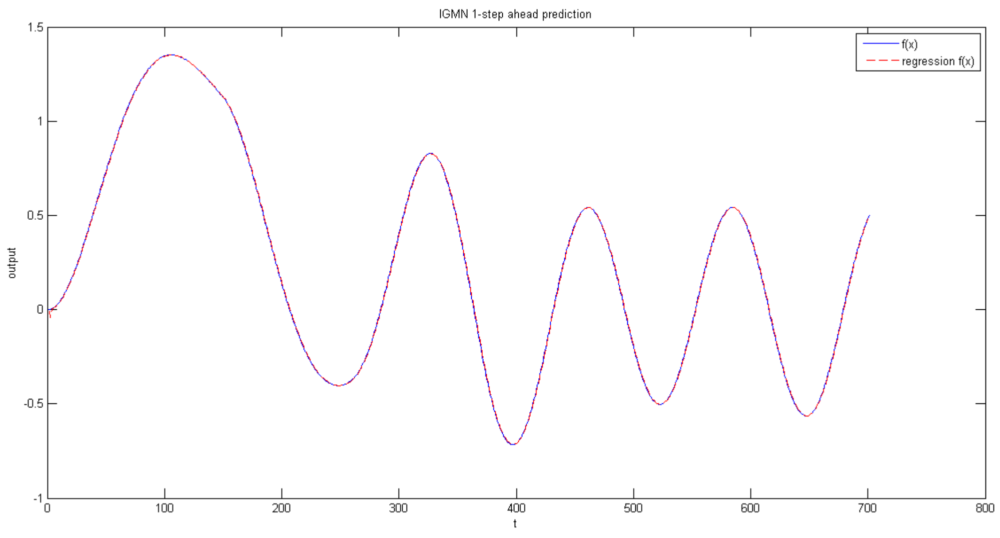
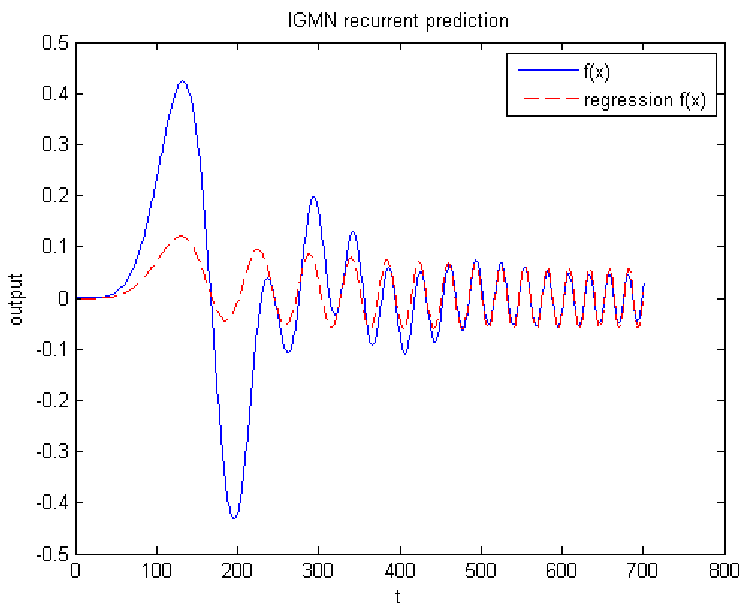


Figure 6.13 - Recurrent IGMN 1-step ahead prediction of verification signal 2



As our sampling period is small when compared to the system dynamics (i.e. the sampling frequency is much higher than the normal operating frequency range of the quadcopter dynamics), then a single step prediction is not much useful in a predictive controller since it does not give sufficient information of future movements. One possibility to try to work around this problem is closing the network loop (use the network in Series configuration) so it uses a predicted sample to predict another one (recursively). Figure 6.14 shows that this approach did not produce good results even when using the training signal. The reason is that each sample prediction has an associated error when compared to real system output, so when we feedback the predicted output we are also providing an error to the network input. Hence, this positive error feedback will make the network output quickly diverge from the real output.

Figure 6.14 - IGMN recursive prediction



Another strategy can be to increase the network output size so we train the network to predict more samples at once. Since the IGMN achieved better results than the MLP, we used it again but now with 10 outputs (10-step ahead prediction). To visualize this result, Figure 6.15 shows the network output curve composed by sets of 10 predicted samples sequentially. That is, at the end of each 10th predicted sample, we use the real system output (which is known) in the input again to start a new prediction. One can notice that the results are pretty good and very close to the 1-step ahead prediction. Figure 6.16 and Figure 6.17 show the same 10 steps prediction when feeding the same network with the verification signals. Again, reasonable results can be seen but errors are more visible and can be majorly associated to the input signal characteristics.

Figure 6.15 - IGMN 10 steps prediction of the training signal

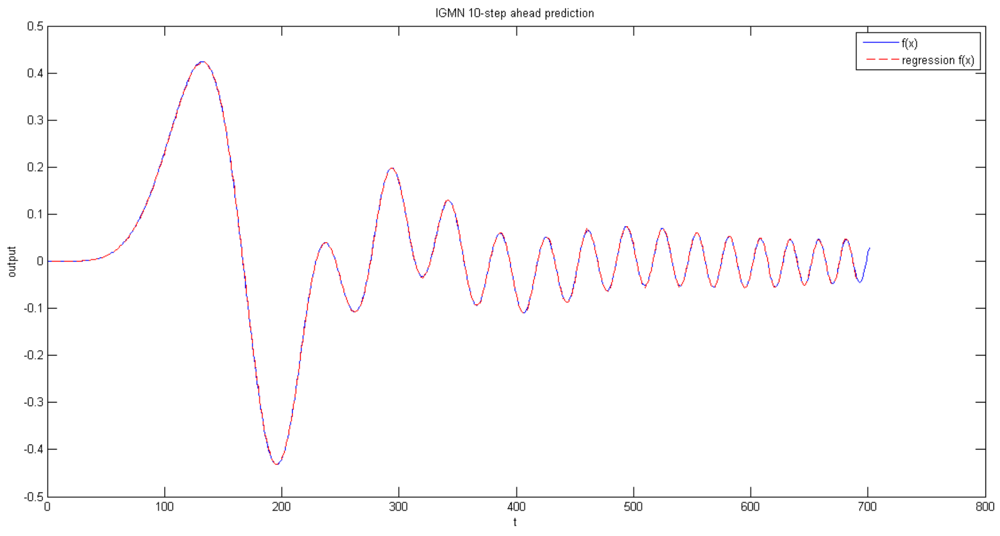
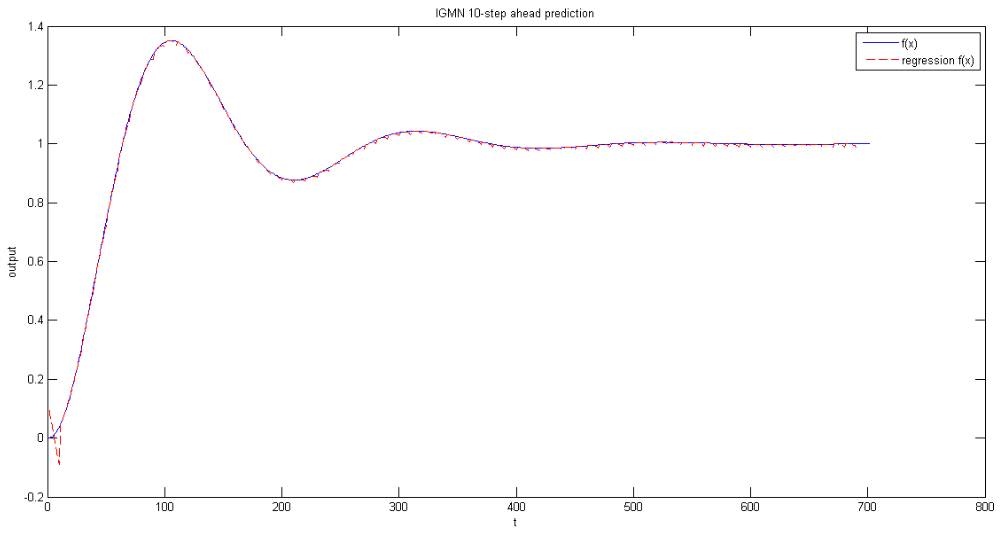
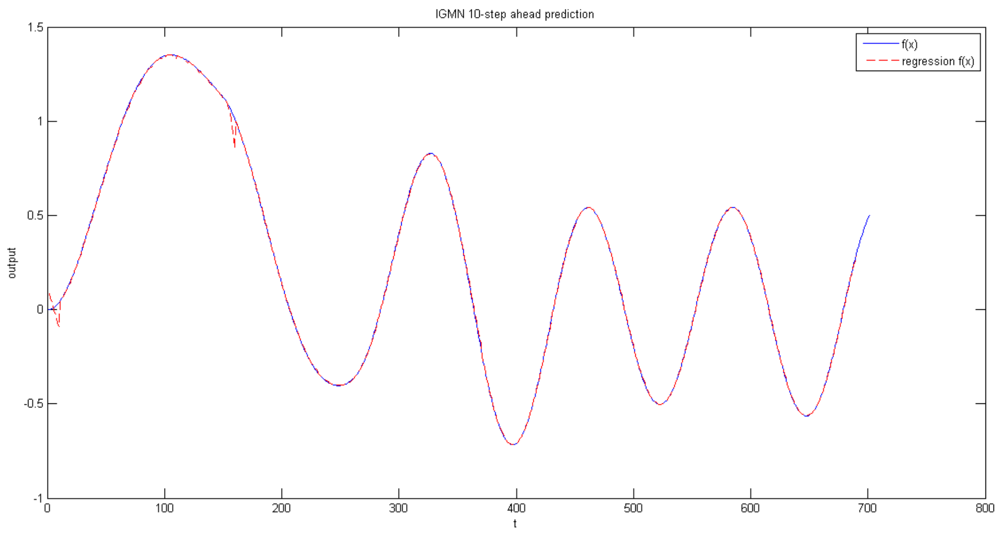


Figure 6.16 - IGMN 10 steps prediction of verification signal 1



Since our sampling period was $1ms$, even predicting the next 10 samples may not be enough to design a robust controller. Thus, even though our sampling period is small, we could use more spaced samples to train the network so each new sample can represent more significant data from the system dynamics point of view. One must be careful with the frequency range that the quadcopter dynamics work so between a pair of samples we do not lose information. This issue is strictly related to Nyquist sampling theorem but we cannot find a precise maximum frequency since it depends on the mechanical parts of the quadcopter and how fast it can react as well as how fast we need to control it. Using more spaced samples and a N-step ahead prediction may let us estimate significant future behavior of the system. Due to these issues, a neural network based predictive controller was not implemented in the 3D simulator. Further discussions can be seen in the next chapter.

Figure 6.17 - IGMN 10 steps prediction of verification signal 2



The implemented controller was based on the VRFT method. The main feature of this method is to avoid the trial and error tests when tuning the controller like the P controller mentioned previously. In this particular situation we could easily find some suitable P values for the system, but depending on the system complexity and number of controller parameters this task can become extremely difficult.

The first tests were made with MATLAB software only. Based on the acquired data from previous tests with a P controller, it was possible to approximate the quadcopter dynamics by a linear transfer function. One should notice that even though the methods used here should not be model-based, this approximation is only to test the algorithm results. Again we can approximate the system dynamics by a second order transfer function. With slower motors we could make a fast and reasonable estimation with one pole at 0.9999 and the other pole at 0.995. Therefore, our $G(z)$ will be:

$$G(z) = \frac{Y(z)}{R(z)} = \frac{(1-0.9999)(1-0.995)}{(z-0.9999)(z-0.995)} \tag{6.1}$$

where $Y(z)$ represents the angular velocity of one axis, $R(z)$ the angular velocity reference and Figure 6.18 shows the response of the real plant and of our estimation for the same input signal. Clearly we do not need an exact approximation since we are using this function for testing purposes only. Figure 6.19 depicts the VRFT result where the plant response in an open-loop configuration, the $M(z)$ response (desired system response in a closed-loop configuration) and the designed system (using VRFT) response are shown in the same plot.

One can easily see that the plant without a controller has a very slow response compared to a plant with a controller. In this case, the $M(z)$ pole was set to 0.999 and we see that the resulting system responds almost as the desired system. The next tests will be made using a $M(z)$ with its pole set to 0.96 (calculated in Chapter 5). This implies a faster response, so the next figures will not show the open-loop plant response in the same plot because of the time scale.

Figure 6.18 - Real and estimated plant comparison

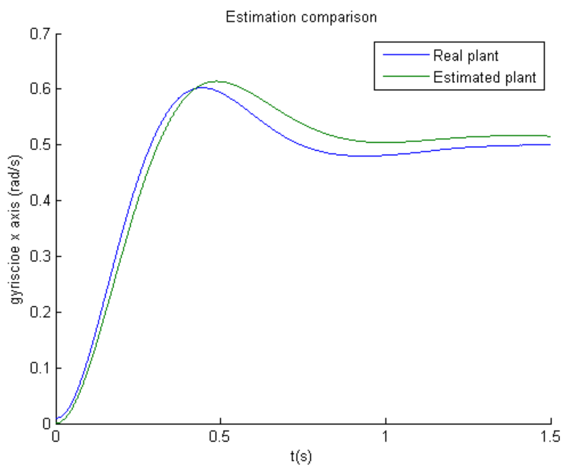
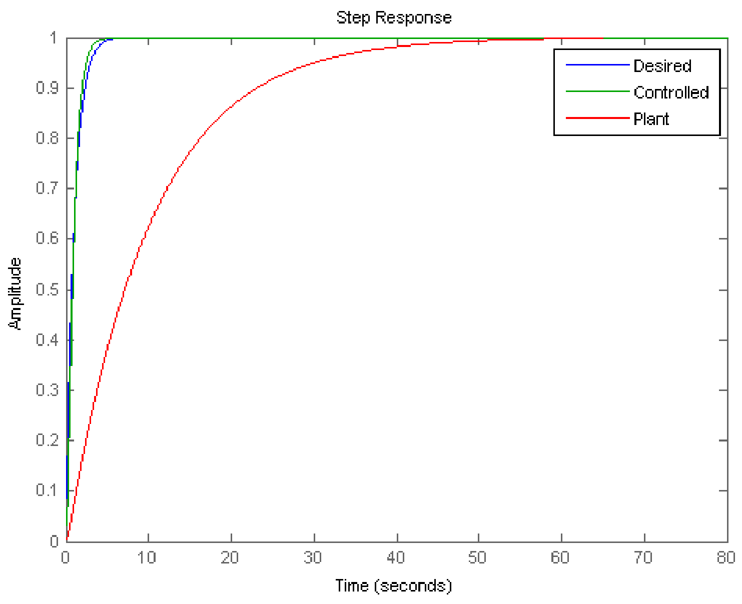
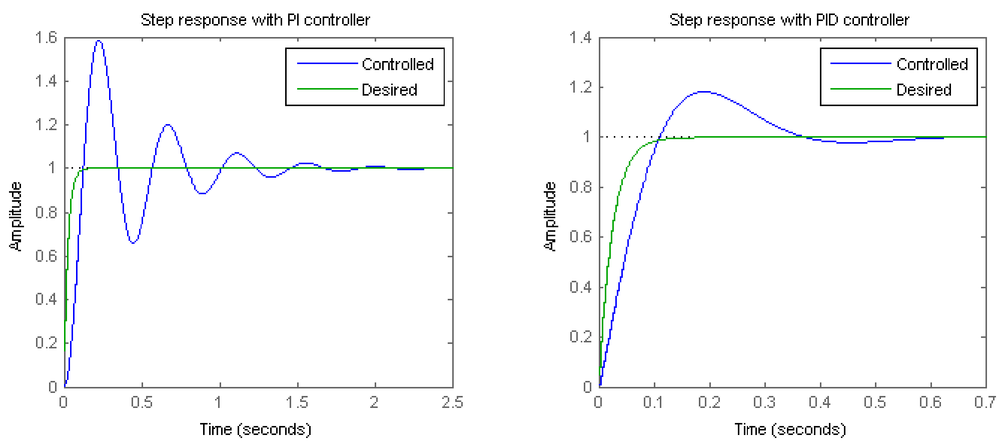


Figure 6.19 - Step response of the plant without a controller (open-loop) , desired closed-loop system and the plant with the tuned controller (closed-loop)



It can be seen in Figure 6.20 the comparison between a PID controller and a PI controller, both tuned by the VRFT algorithm using the desired system transfer function $M(z)$ with its pole at 0.96. One should notice that as we moved the $M(z)$ pole away from the unit circle, then we want a faster system. However, the closed-loop system clearly has limitations due to the plant dynamics. Hence, even though we want the closed-loop system to behave as $M(z)$, the best controller within the possible tunable parameters may result in oscillatory or slower response. In Figure 6.20 we can see that the closed-loop system with the tuned controller responds with an initial slope that is close to the desired response slope. Nevertheless, this results in a step response with overshoot. Yet, there is a significant difference in performance between the PI and the PID controller.

Figure 6.20 - PID vs PI controller tuned by VRFT algorithm



After verifying that the VRFT algorithm is working fine, further tests were made although they were not made directly in the 3D simulator. Instead, data was acquired in the simulator and used to calculate the controller parameters externally with MATLAB. These parameters were then used in the 3D simulator as the controller parameters. Again, the PI controller is compared with a PID controller when controlling the gyroscope data while flying the quadcopter in the simulator. Figure 6.21 shows the comparison between both controllers that were tuned by the VRFT algorithm, using fast response motors and setting the $M(z)$ pole to 0.96. It can be seen that the system is very well behaved and the response time is very close to what we expected when the pole of $M(z)$ was calculated in Chapter 5. Figure 6.22 shows the same comparison but now using slower motors. Clearly the system response is much worse since the plant is much slower and we are trying to obtain the same fast response as before (since we are using the same $M(z)$). Yet, the PID response is slightly better than the PI response although they are similar. Finally, as the system with fast motors is very well behaved, another experiment was made with a faster response in $M(z)$. Now, its pole was set to 0.9. Figure 6.23 shows that the system response began to show some overshoot as we are trying to make it respond faster than it is possible with this type of controller. Still, the PI and PID responses are very similar.

Figure 6.21 - PI and PID controllers tuned by VRFT using fast motors and M(z) pole at 0.96

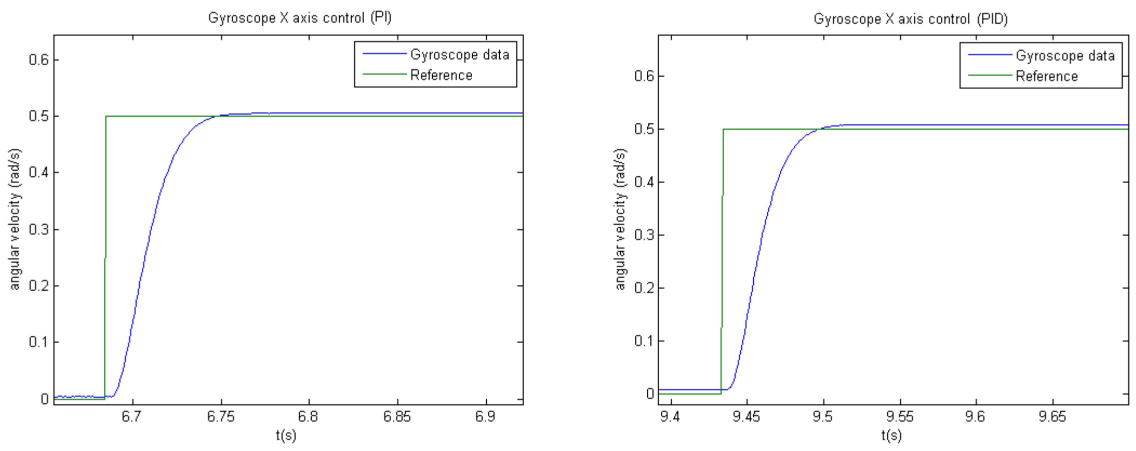


Figure 6.22 - PI and PID controllers tuned by VRFT using slow motors and M(z) pole at 0.96

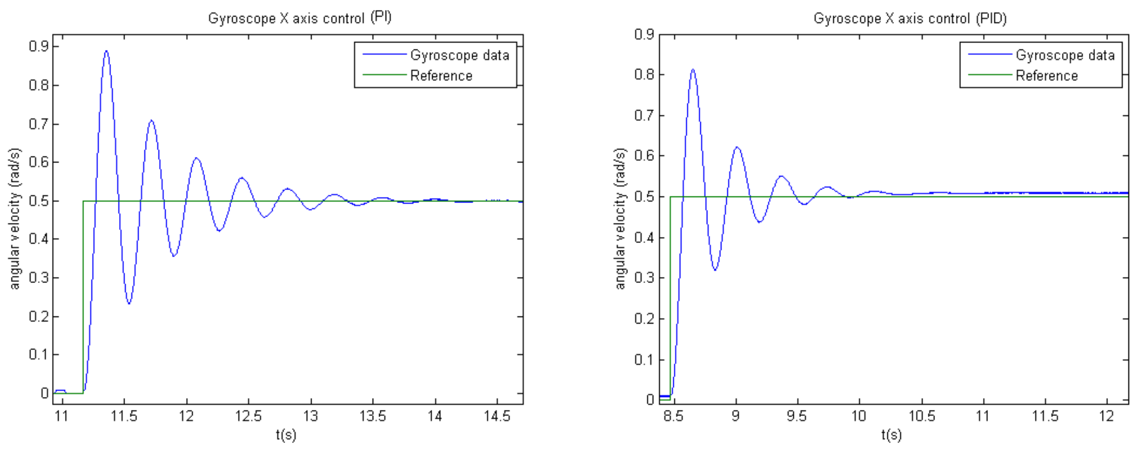
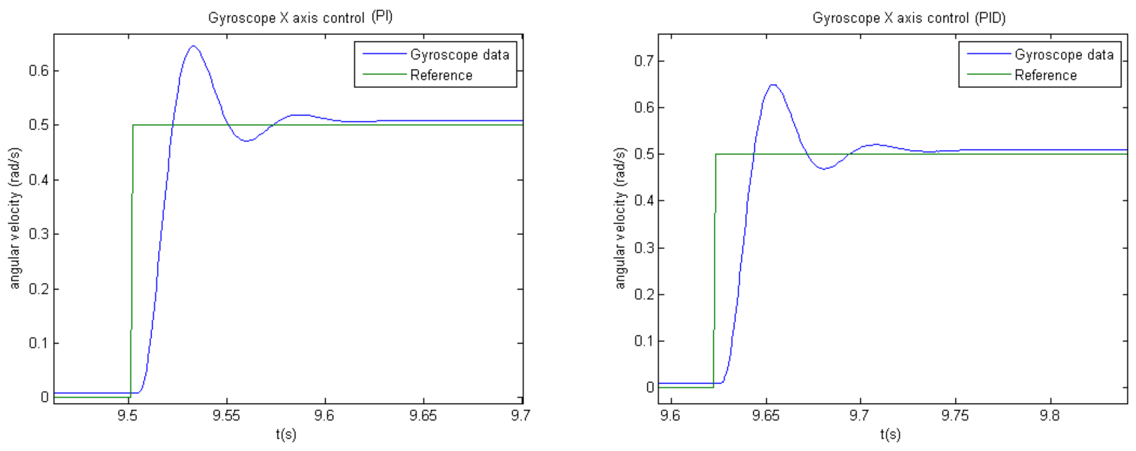


Figure 6.23 - PI and PID controllers tuned by VRFT using fast motors and M(z) pole at 0.9



It is also important to notice that in all cases the system showed some steady state error due to controller signal quantization. This happens because the controller internally converts the calculated control signal to integer values. The conversion is needed because the control signal must be a PWM duty cycle value which is specified in an integer range from

5000 to 10000. Therefore, whenever the error value is low at some point that the generated control signal is between two integers, nothing is done since the integer part of the control signal will remain unchanged. Hence, no actions occur in the system. As our controllers have an integral term, this small error is accumulated until the control signal can reach the next integer value. Then, the system will respond the other way around and the same thing will happen again. Thus, some kind of oscillation around the reference will occur like when one uses a bang-bang type controller. This action cannot be seen in our experiments since the oscillation period is too large and its amplitude is small. However, it also can be seen that the PID controllers show a slightly larger steady state error when compared with PI controller. This happens because the gains used with PI controllers are larger than the PID controller gains. Thus, the control signal truncation problem will be less apparent with PI controllers since its higher gains make the control signal exceeds the next integer value more easily.

Even though satisfactory results were obtained, we still do not have a purely online control tuning since MATLAB was used to run the VRFT algorithm. Hence, the same algorithm was implemented in Unity3D in order to directly find the controller parameters during the flight. However, bad results were found due to rounding issues. That is, the VRFT algorithm requires high precision both with small values and with very large values. Single-precision float type was not enough to obtain good results. Although double-precision float type could be used, some calculations were performed using Unity3D Matrix4x4 type which is natively composed by single-precision terms.

7 CONCLUSION AND FUTURE WORKS

Starting with a simple P controller it can be seen that one can easily find suitable parameters depending on the variable that we are controlling. Best results were obtained when the controlled variable was the angular velocity of each axis of the quadcopter. Although this configuration already allows us to fly the quadcopter, the flying operation becomes more difficult as we are controlling the angular velocity and not the angles itself. Besides, the angular velocity control will not reach zero steady-state error with only a P controller since neither the controller nor the plant has a pole at origin. Null error can be obtained if the controlled variable is the angle of each axis, but a suitable parameter for the controller is more difficult to find besides the slow response of the closed-loop system as we saw in Chapter 5. Therefore, adaptive controllers are indeed very useful in our situation.

The neural network based controller can be a good option although it was not implemented due to reasons mentioned in previous chapter. One of the main problems in the neural networks training processes was the selection of a suitable input signal to maximize the model dynamics information within this signal. Future works may investigate the improvement of the neural network predictions when training it with richer signals like white Gaussian noise for instance. Since not all frequencies are equally relevant to the system dynamics (i.e. too high frequencies are not as relevant as lower frequencies since the system can barely respond to higher frequencies) and a pure white Gaussian noise may not be easily fed into the system in real time, one may investigate training methods where each movement is monitored and mapped to a frequency map so whenever a monitored signal correspond to a new region in this frequency map, it is used in the training process. One can make use of the wavelet transform to continuously map pieces of the acquired data to frequency regions. Once all regions are covered, the network should have learned all necessary frequencies to operate correctly.

Also based on previous chapter, one may investigate the use of predictive controllers with a N-step ahead prediction neural network using higher sampling period. Once the network is well trained, one can opt between various types of controller based on the model predictions. An interesting method would be to create a cost function to be minimized in order to find an optimal controlling signal as proposed by Soloway (1997). The LM algorithm can even be used in the minimization process.

We saw in the previous chapter the great advantages of using the IGMN instead of a traditional MLP. The main features of the IGMN make it the best choice for our application. Besides our use of the IGMN in previous sections, one can imagine other uses of this network due to its facility to operate online, besides learning the system model directly. For example, if our plant is linear, one can easily learn its transfer function from few well selected experiments and then predict any output of the plant by a superposition of the known experiments scaled by some factors. In practice, most plants are nonlinear and this is also our case. However, even though it is not linear, most regions of operation are somehow close to be linear. Thus, dealing with this plant as a linear process might make it works reasonably in some situations but might present bad behavior under other conditions. Therefore, as most of these nonlinearities usually are not abrupt, one might use an IGMN to learn the nonlinear behavior of the system. Hence, we can start again by learning some relation between the system output and the system input under certain conditions with a few experiments and so predict other outputs by a combination of these experiments but now with the weighting of the nonlinearity learned by the IGMN.

At least, VRFT algorithm proved to be very useful in our experiments. Theoretical simulations showed that it could find well-tuned controllers. When using the 3D simulator, the calculated controllers also showed nice results although they were calculated with MATLAB. To make a true auto-adjustable controller one must solve the rounding problems presented by Unity3D. One way of doing this is by replacing all uses of single-precision float numbers inside the VRFT algorithm and implementing all matrix operations by hand with double-precision float type. Thus, the quadcopter system could autonomously find its best controller after first movements. One should notice that these first movements also depend at least in a poorly tuned controller so it can fly. Otherwise it will be much more difficult to make an open-loop flight to acquire enough data to use in VRFT. Hence, when we make our data acquisition with a closed-loop with a poorly tuned controller, the plant input signal will present some noise that is correlated with the output signal since one depends in the other due to the closed-loop configuration. This correlation can lead to worse results when using the VRFT algorithm, depending on the system and on the signal-to-noise ratio. Also, depending on the controller precision needed and on the signal-to-noise ratio in the data, one might consider the elaboration of the $L(z)$ filter as suggested by Campi (2002) as well as other few extra steps of the algorithm when dealing with noisy data. Furthermore, if the user wants more precision in the system output, it should be investigated the best way to minimize the control signal quantization problem. That is, investigate how the rounding of the control signal to integer values will influence less in the system performance. Also, when implementing the controller in the real system other issues must be taken care of like the controller saturation or even the actuators saturation. Finally, one might experience some difficulties when choosing a suitable $M(z)$. That is, one might not know what the best $M(z)$ order is or where to put its poles. We saw that an excessively fast $M(z)$ (when compared to the real plant) can make a closed-loop system that oscillates too much when trying to be as fast as the $M(z)$ response. Hence, future works might investigate methods that adjust iteratively the $M(z)$ pole position in order to find the best balance between overshoot and settling time.

REFERENCES

ACHTELIK, Michael et al. **Adaptive Control of a Quadcopter in the Presence of large/complete Parameter Uncertainties**. Infotech@ Aerospace 2011. 2011.

ARGENTIM, Lucas M., et al. **PID, LQR and LQR-PID on a quadcopter platform**. Informatics, Electronics & Vision (ICIEV), 2013 International Conference on. IEEE, 2013.

BAZANELLA, Alexandre Sanfelice; DA SILVA JUNIOR, João Manoel Gomes. **Sistemas de Controle: princípios e métodos de projeto**. UFRGS, 2005.

BAZANELLA, Alexandre Sanfelice; ECKHARD, Diego; CAMPESTRINI, Lucíola. **Data-Driven Controller Design**. 2012.

BURKA, Alex; FOSTER, Seth. **Neato Quadcopters**. Swarthmore, 2012.

CAMPI, Marco C.; LECCHINI, Andrea; SAVARESI, Sergio M. **Virtual reference feedback tuning: a direct method for the design of feedback controllers**. Automatica, v. 38, n. 8, p. 1337-1346, 2002.

CHAN, K. C.; LEONG, S. S.; LIN, G. C. I. A **neural network PI controller tuner**. Artificial intelligence in engineering, v. 9, n. 3, p. 167-176, 1995.

HAYKIN, Simon S. et al. **Neural networks and learning machines**. Upper Saddle River: Pearson Education, 2009.

HEINEN, Milton Roberto; ENGEL, Paulo Martins. **IGMN: An incremental neural network model for on-line tasks**. WORKSHOP ON MSC DISSERTATION AND PHD THESIS IN ARTIFICIAL INTELLIGENCE (WTDIA), V, São Bernardo do Campo, 2010. Proceedings... São Bernardo do Campo. 2010. p. 732-741.

LEVENBERG, Kenneth. **A method for the solution of certain problems in least squares**. Quarterly of applied mathematics, v. 2, p. 164-168, 1944.

MARQUARDT, Donald W. **An algorithm for least-squares estimation of nonlinear parameters**. Journal of the Society for Industrial & Applied Mathematics, v. 11, n. 2, p. 431-441, 1963.

NGUYEN, Derrick; WIDROW, Bernard. **Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights**. Neural Networks, 1990., 1990 IJCNN International Joint Conference on. IEEE, 1990. p. 21-26.

NICOL, C.; MACNAB, C. J. B.; RAMIREZ-SERRANO, A. **Robust neural network control of a quadrotor helicopter**. Proceedings of the Canadian Conference on Electrical and Computer Engineering. 2008. p. 1233-1237.

NORTHFLEET, Christian. **Aprendizado on-line e formação incremental de conceitos em um controlador híbrido para futebol de robôs**. UFRGS, 2011.

OPPENHEIM, Alan V. et al. **Discrete-time signal processing**. Englewood Cliffs: Prentice-hall, 1989.

RANGANATHAN, Ananth. **The levenberg-marquardt algorithm**. Tutoral on LM Algorithm, p. 1-5, 2004.

ROWEIS, Sam. **Levenberg-marquardt optimization**. Notes, University Of Toronto, 1996.

SEBORG, Dale; EDGAR, Thomas F.; MELLICHAMP, Duncan. **Process dynamics & control**. John Wiley & Sons, 2006.

SMITH, Steven W. et al. **The scientist and engineer's guide to digital signal processing**. 1997.

SOLOWAY, Donald; HALEY, Pamela J. **Neural generalized predictive control: a Newton-Raphson implementation**. Langley Research Center, Hampton, Virginia, 1997.

SONG, Yang; WU, Shenyi; YAN, Yuying. **Development of Self-Tuning Intelligent PID Controller Based on BPNN for Indoor Air Quality Control**. International Journal of Emerging Technology and Advanced Engineering, v. 3, 2013

SUZUKI, Michiyo; YAMAMOTO, Toru; TSUJI, Toshio. **A design of neural-net based PID controllers with evolutionary computation**. IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences, v. 87, n. 10, p. 2761-2768, 2004.

YU, Hao; WILAMOWSKI, Bogdan M. **Levenberg-marquardt training**. The Industrial Electronics Handbook, v. 5, p. 1-15, 2011.

APPENDIX A: MEDIAN FILTER COMPLEXITY COMPARISON

A.1 Original filter

<u>Standard Median Filter complexity analysis</u>

$M = Window\ size$
$N = Input\ signal\ length$

$1. Create\ Msize\ Window\ vector$	O(1)
$2. For\ loop\ (1 \rightarrow N)$	O(N)
$2.1. Fill\ window\ vector$	O(M)
$2.2. Sort\ window\ vector$	O(M*log*(M))
$2.3. Find\ window\ median$	O(1)
$2.4. End\ for\ loop$	O(1)

When considering only the operations inside the loop that starts in line 2 (since both algorithms have this loop), the worst operation will be the sorting operation in 2.2 whose complexity is $O(Mlog(M))$.

A.2 Modified filter

Our Median Filter Algorithm complexity analysis

$N = Input\ signal\ length$
$M = Window\ size$
$x = Input\ signal\ vector$
$I = M_{size}\ auxiliar\ vector\ (indices\ of\ w\ elements)$
$w = M_{size}\ window\ vector\ (elements)$

```
1. $Create\ w$                                                  O(M)
2. $Create\ I$                                                  O(M)
3. $For\ (i = 1 \rightarrow N)$                                 O(N)
  3.1. $next\_val = x[i]$                                       O(1)
  3.2. $new_{index} \leftarrow 1$                               O(1)
  3.3. $For\ (j = 1 \rightarrow (M - 1))$                       O(M-1)
    3.3.1. $w[j] \leftarrow w[j + 1]$                           O(1)
    3.3.2. $If\ next\_val > w[j + 1]$                           O(1)
      3.3.2.1. $new_{index} + +$                                O(1)
      3.3.2.2. $End\ if$                                        O(1)
    3.3.3. $I[j] - -$                                           O(1)
    3.3.4. $If\ I[j] == 0$                                      O(1)
      3.3.4.1. $I[j] = length(w)$                               O(1)
      3.3.4.2. $old_{index} = j$                                O(1)
      3.3.4.3. $End\ if$                                        O(1)
    3.3.5. $End\ for$                                           O(1)
  3.4. $j \leftarrow length(I)$                                 O(1)
  3.5. $I[j] - -$                                               O(1)
  3.6. $If\ I[j] == 0$                                          O(1)
    3.6.1. $I[j] = length(w)$                                   O(1)
    3.6.2. $old_{index} = j$                                    O(1)
    3.6.3. $End\ if$                                            O(1)
  3.7. $If\ new_{index}\ ! = old_{index}$                       O(1)
    3.7.1. $If\ new_{index} > old_{index}$                      O(1)
      3.7.1.1. $For\ (j = old_{index} \rightarrow (new_{index} - 1))$      O(M)
        3.7.1.1.1. $I[j] \leftarrow I[j + 1]$                   O(1)
        3.7.1.1.2. $End\ for$                                   O(1)
    3.7.2. $Else$                                               O(1)
      3.7.2.1. $For\ (j = (old_{index} - 1) \rightarrow new_{index})$      O(M)
        3.7.2.1.1. $I[j + 1] \leftarrow I[j]$                   O(1)
        3.7.2.1.2. $End\ for$                                   O(1)
      3.7.2.2. $End\ if$                                        O(1)
    3.7.3. $End\ if$                                            O(1)
  3.8. $I[new_{index}] \leftarrow length(w)$                    O(1)
  3.9. $w[length(w)] \leftarrow next\_val$                      O(1)
  3.10. $median \leftarrow w[I[ceil(length\left(\frac{I}{2}\right))]]$      O(1)
4. $End\ for$                                                   O(1)
```

When considering only the operations inside the loop that starts in line 3 (since both algorithms have this loop), the worst operations will be other loops whose complexities are linear with respect to the window size.

APPENDIX B: INITIAL WORK DESCRIPTION (TG1 PAPER)

Controlador Adaptativo de Veículo Aéreo Não Tripulado (VANT) utilizando PID e Redes Neurais

Felipe Ribas Silva de Azevedo

Instituto de Informática, Universidade Federal do Rio Grande do Sul (UFRGS)
Porto Alegre – RS – Brasil

felipe.ribas@ufrgs.br

Orientador: Paulo Martins Engel

Co-Orientador: Renato Perez Ribas

Resumo *– Este trabalho mostra o desenvolvimento de um controlador eletrônico embarcado para Veículos Aéreos Não Tripulados (VANTs) que seja capaz de receber e enviar comandos para uma central e também manter a estabilidade do veículo durante todo seu trajeto de vôo. O diferencial deste controlador em relação aos controladores mais comuns encontrados no mercado (controladores PID) será o uso do conceito de redes neurais para realizar um auto-ajuste dos parâmetros do PID tradicional com finalidade de melhorar a estabilidade a cada iteração.*

1. Introdução

Os Veículos Aéreos Não Tripulados (VANTs) são cada vez mais o alvo de estudos e pesquisa devido à sua versatilidade e seu alto poder de realização de uma grande diversidade de tarefas. Uma de suas principais vantagens é o fato de não necessitarem o envolvimento de mão de obra humana presencial para realizar tais tarefas, o que geralmente envolve o aumento de custo. Entretanto, para realizar estas tarefas é necessário um sistema de controle robusto capaz de comandar e controlar seus componentes mecânicos que inclui o controle de sua estabilidade de vôo.

O foco deste trabalho será o desenvolvimento de um controlador eletrônico embarcado completo capaz de enviar e receber comandos ao VANT remotamente a partir de uma central sendo responsável, também, por sua estabilidade em vôo. Será utilizada uma topologia baseada nos controladores mais tradicionais encontrados no mercado atualmente (controladores PID), porém o grande desafio destes controladores é o ajuste de seus parâmetros, que geralmente são feitos por "tentativa e erro" ou então necessitam um conhecimento detalhado do modelo do sistema a ser controlado. Como desejamos controlar um VANT genérico e, portanto, não teremos o modelo do sistema, será desenvolvido um controlador adaptativo que seja capaz de se auto-ajustar ao longo do tempo. Far-se-á uso de uma rede neural artificial capaz de auxiliar no ajuste dos parâmetros do PID.

Cabe notar também que os VANTs englobam uma diversidade muito grande de tipos de veículos aéreos. Portanto, cada tipo de veículo possui uma dinâmica de vôo que podem ser bem diferentes entre si. Apesar deste trabalho tratar do desenvolvimento de um controlador genérico, será adotado um tipo específico de veículo denominado quadricóptero (ou quadrotor) que será detalhado a seguir. O auto-ajuste do controlador será suficiente para estabilizar modelos mecânicos diferentes desse mesmo tipo de veículo, que podem conter características diferentes (massa, tamanho, etc) sem que seja necessário conhecer seus valores. A genericidade do controlador poderá ser aplicada a outros tipos de VANTs, porém, ajustes deverão ser feitos e, por não ser o foco do trabalho, não serão cobertos neste trabalho.

2. Revisão teórica

Será descrito a seguir uma breve explicação teórica dos quatro principais componentes envolvidos na elaboração do projeto (quadricóptero, sensores, controlador PID e redes neurais).

2.1 Quadricóptero

O componente principal do projeto é o sistema mecânico que iremos controlar, denominado quadricóptero. Este é um tipo de VANT amplamente usado tanto na indústria como por hobbistas. Dentre os VANT's mais comuns, podemos dividi-los em dois grandes grupos: os de asa fixa e os de asa rotativa. Um exemplo de veículo com asa fixa são os aviões de turbina, cuja sustentação aerodinâmica se da por meio de asas fixas na estrutura do avião. Já os de asa rotativa podem ser helicópteros ou multicópteros (ou multi-rotor). Em geral, os helicópteros possuem apenas uma única hélice de sustentação e uma hélice secundária para evitar a rotação da aeronave originada do torque devido à rotação da hélice principal. Já os multicópteros são veículos aéreos com mais de dois rotores, onde cada rotor é um conjunto composto pelas partes mecânicas móveis que contém as hélices (responsáveis pela sustentação no ar).

Os multi-rotores se tornaram populares com popularização dos VANT's, também conhecidos como *drones*. Com o avanço da tecnologia, tornou-se fácil e barato a construção desse tipo de veículo, devido a sua simetria e simplicidade das peças envolvidas. O objeto de estudo deste trabalho será uma classe específica dentro dos veículos multi-rotores, que são os quadricópteros, ou seja, contém quatro rotores.

A seguir podemos ver a ilustração (Figura 1) de um quadricóptero genérico inserido num sistema de coordenadas XYZ.

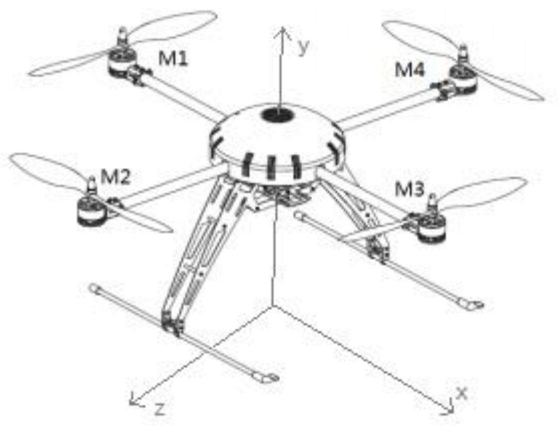


Figura 1 - Quadricóptero genérico

Considerado que a origem do sistema de coordenadas tenha sido deslocado para baixo para facilitar a visualização do mesmo, iremos fazer uma breve análise dos movimentos que o veículo é capaz de realizar.

Os rotores foram numerados de 1 a 4 (M1, M2, M3 e M4) e cada um pode ser controlado individualmente. Para atingir um estado de equilíbrio é necessário que motores com numeração par (M2 e M4) girem em sentido contrário em relação aos motores ímpares (M1 e M3) para haver um cancelamento da força resultante aerodinâmica das hélices que faria toda a estrutura ficar girando em torno do eixo Y, caso todos eles girassem suas hélices pro mesmo lado. Sendo assim, para executar um movimento de rotação em torno do eixo Y, basta criar um desequilíbrio entre a rotação dos motores pares e ímpares, porém mantendo a força resultante ao longo de Y, ou seja, caso os motores pares reduzam suas velocidades, os ímpares devem aumentar (e vice-versa). Se isto não ocorrer, a força resultante vertical se alterará, gerando um outro movimento (além da rotação), que seria a translação ao longo de Y.

Por fim, também deseja-se mover para os lados, mas nota-se que não existe como os rotores aplicarem uma força puramente lateral. Portanto, se faz necessária a combinação de dois movimentos para realizar uma translação lateral: uma inclinação (rotação ao longo de X ou Z) seguido de uma translação vertical (que agora não será mais ao longo apenas de Y já que a estrutura está inclinada). Assim, temos um total de quatro movimentos diretamente realizáveis ou seis movimentos que podem ser realizados direta ou indiretamente.

2.2 Sensores

Os sensores geralmente utilizados são o acelerômetro com três eixos (X, Y e Z) e o giroscópio também de três eixos. O primeiro mede a aceleração de translação em cada um desses eixos, e o segundo mede a velocidade angular da rotação em torno de cada um desses eixos. Pode-se usar também sensores ultrassom para medir distância e evitar colisões, assim como pequenas câmeras para reconhecimento de locais por processamento de imagens.

2.3 Controlador PID

Os controladores do tipo PID (*proportional-integral-derivative*) são um dos tipos de controlador mais utilizados na indústria atualmente, devido à sua simplicidade e eficácia ao mesmo tempo.

Supondo que nós possuímos um processo que deve seguir um sinal de referência desejado e que existam sensores no sistema capazes de nos informar a saída atual do processo a fim de comparar com a referência de entrada, então podemos representar esse nosso modelo como um diagrama de blocos visto na Figura 2.

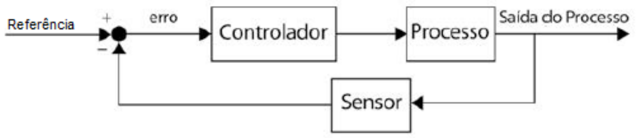


Figura 2 - Processo controlado

O bloco Controlador, neste caso, será nosso controlador PID, que terá como função comparar o valor lido pelos sensores na saída com a referência desejada de entrada (também chamado de *erro*) e atuar diretamente no processo a fim de minimizar esse erro. Para isso, ele fornecerá um sinal para o processo que possuirá três componentes: P (que será proporcional ao erro), I (que será uma integral do erro, ou seja, o cumulativo do erro no passado) e D (que será a derivada do erro, ou seja, proporcional à variação do erro). Matematicamente esse sinal de saída (denominado *u(t)*) poderá ser expresso por:

$$u(t) = K_p e(t) + K_i \int e(t)dt + K_d \frac{de(t)}{dt}$$

Onde K_p, K_i e K_d são as constantes do controlador que devem ser ajustadas para se obter o desempenho desejado.

2.4 Redes Neurais

Uma rede neural (ou rede neural artificial, RNA) é um recurso bastante poderoso na área de inteligência artificial, mas que pode ser usado em muitos outros campos de estudo. A idéia fundamental é a criação de um modelo que fosse baseado em uma rede neural biológica e que fosse capaz de aprender algo (assim como o cérebro animal).

Do ponto de vista matemático, é interessante notar que uma RNA pode modelar com bastante eficácia uma função não-linear conhecendo apenas alguns pontos da mesma (amostras). Ou seja, para um sistema cujo modelo matemático é desconhecido, podemos realizar ensaios com valores de entrada conhecidos e, baseados nas saídas obtidas, treinar uma RNA para modelar a função que representa este sistema.

O elemento principal da rede é o neurônio, que possui como entrada diversos sinais de saída de outros neurônios (com seus respectivos pesos) e uma função de ativação que, dependendo dessas entradas, ativa ou não a sua única saída. Reunindo vários desses elementos, formamos a topologia comumente utilizada para RNA's que possui uma camada de neurônios que recebe diretamente os sinais de entrada da rede (camada de entrada), uma ou mais camadas intermediárias (camadas ocultas) e a camada de saída, como mostra a Figura 3.

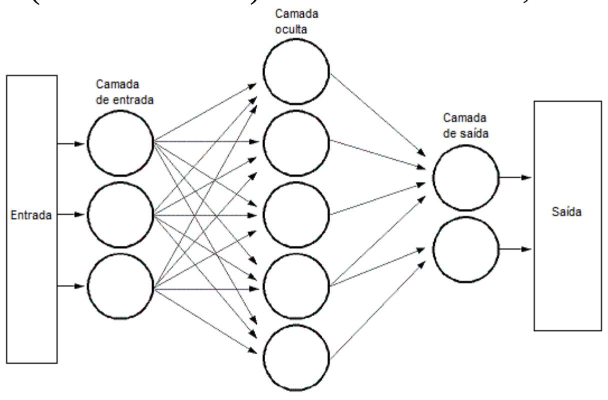


Figura 3 - Rede neural artificial típica

Cabe notar que o modelo visto na figura acima representa uma RNA com três entradas, duas saídas e apenas uma camada oculta contendendo cinco neurônios. Pode-se variar esses valores de acordo com a aplicação desejada.

O aprendizado da rede (também chamado *treinamento*) pode-se dar ao longo da execução do processo (em tempo real, ou *online*), ou previamente antes de utilizar o controlador (treinamento *offline*). Para ambos casos, devemos ter valores de referência, ou valores de treinamento, que serão dados de entrada e da respectiva saída desejada. Com base nesses dados, a rede comparará os valores reais obtidos com os valores desejados e fará atualização de seus pesos internos com objetivo de minimizar este erro encontrado.

3. Trabalhos relacionados

É muito comum encontrar trabalhos sobre ajuste dos parâmetros de um controlador PID e também artigos sobre métodos de ajuste automático desses parâmetros, já que esse tipo de controlador é bastante utilizado atualmente. Apesar de sua simplicidade, o principal ponto negativo é justamente encontrar o melhor conjunto de parâmetros que forneça um resultado satisfatório. Muitas vezes não se conhece o modelo do sistema a ser controlado, o que impossibilita o equacionamento para encontrar uma solução. Pode-se aproximar um modelo do processo e assim tentar ajustar o controlador com algumas técnicas relativamente simples como mostra Sigurd Skogestad (2001). Além disto, a realização de ensaios com o sistema pode ser complicada, o que impossibilita extrair dados comportamentais do sistema para encontrar valores aproximados para os parâmetros do controlador.

Portanto, muitos trabalhos foram propostos a fim de mostrar métodos em que o PID se auto ajuste, como mostra, por exemplo, Feng Lin et al. (2000), Z. Iwai et al. (2006), Cláudio Ferrastro et al. (2007) e Varun Aggarwal et al. (2006).

Outros trabalhos mostram também a tentativa de controlar o processo diretamente com uma rede neural artificial, justamente para evitar o problema de ajuste de um controlador PID (ou outros problemas encontrados em outros tipos de controlador), como mostra C. Nicol et al. (2008) que utiliza um controlador baseado em rede neural para estabilizar um quadricóptero. Para isto, foi necessária uma descrição do modelo dinâmico do quadricóptero, restringindo bastante a genericidade de um controlador como o proposto neste artigo. Além disso, um dos principais problemas ao se utilizar RNA's, é a inicialização dos pesos da rede, que geralmente são aleatórios e podem proporcionar resultados ruins nos momentos iniciais de aprendizado. Outro ponto que pode ser negativo é a necessidade de se treinar a rede antes de utilizá-la, com dados de treinamento.

Para resolver este último problema citado, P. M. Engel (1996) propôs um tipo de RNA cujo treinamento fosse em tempo real e, portanto, o controlador melhora seu desempenho com o tempo. Ainda assim existe o problema de inicialização dos pesos e, para este, Derrick Nguyen e Bernard Widrow (1990) propuseram um método para escolher valores iniciais de peso e assim reduzir o tempo de treinamento de uma rede neural.

Com o objetivo de unir os pontos positivos de um controlador PID e de um controlador baseado em RNA (e também na tentativa de reduzir os pontos negativos de cada um deles), foram propostos muitos trabalhos onde essas técnicas foram utilizadas em conjunto. Algumas delas utilizaram redes neurais onde a função do PID estaria incorporada na rede, como mostra F. Shahraki et al. (2009), propondo uma rede neural simples com apenas uma camada oculta contendo três neurônios, sendo cada um deles a representação de um dos parâmetros do PID tradicional. Outro trabalho similar (Gary M. Scott et al., 1992) utiliza uma rede um pouco mais complexa que incorpora a função do PID e inclusive utiliza algumas técnicas de ajuste de PID's para inicializar alguns pesos da rede.

Uma outra maneira de unir essas técnicas que também é bastante utilizada, é utilizar uma rede neural juntamente com um controlador PID tradicional, onde a rede fica responsável por ajustar os parâmetros do PID, como mostra Yang Song et al. (2013) e Michiyo Suzuki et al. (2004). O método proposto a seguir se baseia nesta topologia.

4. Motivação e Proposta

Como vimos anteriormente, diversos métodos de controle foram propostos sendo que cada um deles contém algumas características que podem fazer com que sejam mais desejáveis para alguns tipos de processo do que para outros. A motivação principal deste trabalho é a realização de um controlador capaz de aprender e se adaptar com o tempo, porém livre de condições iniciais e treinamento prévio.

Vimos que para um controlador PID tradicional, ou se faz necessário o conhecimento do modelo do processo para se desenvolver o PID, ou de ensaios iniciais para se buscar valores adequados para os parâmetros. Além disto, o controlador PID tradicional não se adapta com o tempo.

Já as redes neurais podem ser treinadas para melhorar seu desempenho, porém isso requer dados e treinamentos iniciais. Caso seja adotada uma rede com treinamento em tempo real, se faz necessária uma inicialização aproximada dos pesos adequados da rede.

O objetivo deste trabalho é controlar um veículo aéreo sem treinamento prévio. Portanto, é conveniente que o controlador consiga fazer um auto-ajuste inicial para iniciar seu trajeto de vôo já de maneira estável ou próxima disso. Poder-se-ia usar um controlador PID caso conhecêssemos o modelo dinâmico do veículo (processo), mas como o controlador deve ser capaz de se adaptar para qualquer veículo semelhante (porém com características físicas que pode ser diferentes), então não podemos descrever um modelo preciso do processo a ser controlado.

Por outro lado, poderíamos utilizar uma rede neural que fosse controlando o processo e melhorando seu desempenho ao longo do tempo. Porém, utilizando os métodos conhecidos de treinamento (*offline*) iria fugir a proposta de auto-adaptação sem treinamento prévio, além de requerer dados iniciais do sistema que não teremos. Já utilizando um método de treinamento *online*, a rede poderia se adaptar ao longo do tempo, como proposto por P. M. Engel (1996). Pode-se, de alguma maneira, realizar pequenos movimentos, ainda em solo, a fim de estimar o comportamento do sistema que, mesmo que de maneira grosseira, já serviria de condição inicial de vôo para ser melhorada posteriormente. Entretanto, para que a rede consiga melhorar seu desempenho (treinar), necessitamos conhecer, mesmo que de maneira aproximada, o equacionamento cinemático do processo. Somente desta maneira poderíamos encontrar as equações necessárias que nos diriam qual é a correta atualização dos pesos da rede a fim de minimizar o erro na saída em relação à saída desejada. Ou seja, é um problema semelhante ao do projeto do controlador PID.

Para solucionar estes problemas, utilizaremos tanto os conceitos do controlador PID como de redes neurais em conjunto. Um controlador PID tradicional será responsável diretamente pelo controle do processo e, para resolver o problema visto acima de falta de conhecimento do processo para ajuste das constantes, utilizaremos uma rede neural. Será utilizada a idéia vista acima de que podemos realizar pequenos movimentos ainda em solo apenas para estimar, de maneira grosseira, um modelo de resposta do sistema. Esse modelo pode ser representado pela função de transferência do processo, que pode ser vista como uma função não-linear. Ou seja, uma rede neural pode ser capaz de aprender esta função. Podemos utilizar métodos tradicionais de treinamento *offline*, porém desta vez em tempo real, utilizando os dados de treinamento como sendo os resultados obtidos desses pequenos movimentos em solo. Dessa forma, mesmo que imprecisos, estes dados podem servir de treinamento inicial para que tenhamos apenas um modelo aproximado do processo e que, com ele, possamos calcular os coeficientes do controlador PID que apresentem um resultado satisfatório para iniciar o vôo.

À medida que o vôo for sendo executado, novos movimentos podem ser comandados a cada momento, para se gerar novos dados que serão utilizados no treinamento da rede. Com isso, à medida que o tempo for passando, a rede vai possuir um modelo cada vez mais aproximado do comportamento do processo. Desta forma, os coeficientes do PID podem ser cada vez melhor ajustados devido à maior precisão do modelo do processo. Isto se deve porque temos conhecimento de todos blocos e sinais envolvidos (vide Figura 2), exceto o próprio processo. Sabemos a referência (sinal de entrada que aplicaremos), sabemos a equação do controlador (pois estamos projetando) e sabemos o sinal de saída (que será lido pelos sensores). Sabemos também, por meio do modelo aproximado que a rede irá aprender, qual deve ser a resposta esperada para esta referência de entrada. Assim, podemos comparar a resposta obtida com a resposta esperada e ajustar novamente a rede baseada nesta diferença. No momento em que obtivermos um erro desconsiderável, quer dizer que a rede possui um modelo preciso do processo, ou seja, para qualquer entrada podemos prever a saída que será obtida e, assim, dimensionar os parâmetros da equação do controlador PID.

Para um controle ainda mais robusto pode-se utilizar uma segunda rede para agrupar certos tipos de comportamento. Um comportamento neste caso pode ser representado pelo conjunto de coeficientes do PID, que determinará se ele terá um comportamento mais agressivo ou mais suave. Isto porque é sabido que um dos quesitos desejados no projeto de um sistema de controle não é apenas seguir uma referência, mas também a rejeição a perturbações externas. Assim sendo, dependendo das condições do ambiente externo (mais perturbações ou menos), podemos utilizar um conjunto de parâmetros ou outro. Para armazenar um grupo de conjuntos de parâmetros, podemos tanto usar tabelas como uma segunda rede neural.

Para enviarmos os comandos e extrairmos os dados dos sensores remotamente, o controlador embarcado terá que enviar essa informação por meio de transmissores e receptores RF (rádio-frequência). Além disto, deveremos ter um microcontrolador embarcado responsável tanto pela execução do PID, como pela atualização de seus coeficientes, envio e recepção de dados para a central e leitura dos sensores. Este microcontrolador deve ser rápido o suficiente para não gerar atrasos indesejáveis no sistema, assim como o transmissor RF, o que pode acabar dificultando o controle do processo como um todo.

5. Metodologia para implementação da proposta

Como o processo a ser controlado neste trabalho se trata de um veículo aéreo, não é interessante que os primeiros testes sejam feitos diretamente no modelo real. Portanto, será útil desenvolver um simulador da dinâmica de vôo de um quadricóptero qualquer a fim de verificar se o modelo do controlador é capaz de se adaptar e controlar diferentes configurações de veículos.

Para desenvolvimento desse simulador, será utilizada a ferramenta Unity 3D, que proporciona uma grande flexibilidade na criação de simulações físicas com objetos em três dimensões e de maneira intuitiva. Em conjunto com esta ferramenta, o modelo tridimensional do quadricóptero será feito utilizando outra ferramenta chamada Blender. Esse modelo poderá se exportado diretamente do Blender para o Unity e conterá os traços principais de um quadricóptero genérico, onde suas propriedades físicas como peso e tamanho poderão ser alteradas em diferentes simulações dentro do Unity.

Apesar do Unity ser capaz de tratar scripts que podem conter todo o código que simule o controlador embarcado, um aplicativo feito na linguagem C# que será utilizado no modelo real, também se comunicará com o simulador por *sockets* a fim de simular uma comunicação à distância. Este aplicativo poderá plotar gráficos dos dados que estão sendo extraídos do controlador embarcado.

Se o controlador contido no simulador funciona com sucesso, não garante que o modelo real também funcionará, devido a outras variáveis externas e um detalhamento do mundo real que não são levados em consideração na simulação por questões de simplificação. Todavia, o sucesso na simulação nos dará fortes indicativos que o modelo real poderá ser capaz de realizar um vôo completo, mesmo que com pequenos desvios na estabilidade. Em outras palavras, a simulação neste caso nos servirá como um auxílio para evitar que os primeiros testes no modelo real sejam catastróficos. Para um robô em terra, na maioria dos casos os erros iniciais podem ser toleráveis visto que as colisões, em geral, são de pequeno impacto. Já para robôs aéreos, invariavelmente uma instabilidade no trajeto pode ocasionar uma colisão de alto impacto e trazer danos mais graves ao modelo real. Por este motivo, uma simulação simplificada prévia do controlador desenvolvido servirá não para garantir o funcionamento esperado, mas ajudar a evitar resultados iniciais muito aquém do esperado.

Deve-se certificar, também, que os algoritmos do controlador que funcionarem no simulador, também devem executar em tempo hábil para obter a resposta desejada no microcontrolador real, já que a capacidade de processamento é limitada em relação à CPU responsável pela simulação. Caso o controle do processo seja prejudicado por essa limitação, duas opções podem ser consideradas: um microcontrolador de maior capacidade de processamento ou a transmissão de uma parcela maior de informações para a central remota a fim que a mesma realize os cálculos remotamente e envie de volta os dados necessários ao controlador embarcado. Cabe notar que dependendo da informação que está sendo calculada remotamente, o controle do processo pode ser prejudicado pelo atraso embutido na comunicação dos transmissores RF. Assim sendo, uma estratégia válida a ser adotada é manter a execução contínua do PID no microcontrolador, mas permitir que os cálculos de aprendizado da rede e atualização dos coeficientes do PID possam ser feitos onde for mais conveniente, visto que um atraso nesses cálculos pode ser encarado apenas como um aprendizado mais lento, e não gera um atraso na malha de controle do processo diretamente.

6. Implementação e Resultados preliminares

Um modelo tridimensional foi feito utilizando a ferramenta Blender. A estrutura física do quadricóptero foi feita de maneira simples, porém com os principais componentes interessantes para simulação como os apoios inferiores que ficam em contato com o chão, o corpo central (estrutura) e braços que seguram os rotores (Figura 4). As hélices foram modeladas separadamente para que o objeto seja independente do resto da estrutura e possa girar em torno de si própria.

O modelo do quadricóptero foi importado para a ferramenta Unity juntamente com o modelo da hélice (replicada quatro vezes). Foi criada uma hierarquia física dos componentes para simular a junção entre eles. As características físicas dos objetos também podem ser ajustadas na ferramenta, como a massa e momento de inércia.

O mecanismo de física da ferramenta fica responsável pela simulação da dinâmica e interação entre os objetos. O resultado inicial apresentado se mostra coerente com o mundo físico real. Foi desenvolvido um mecanismo de comunicação por *sockets* que permite o simulador receber comandos externos a fim de representar a comunicação do controlador embarcado com a central. Basta introduzir o endereço de IP e Porta para estabelecer a comunicação (Figura 5).

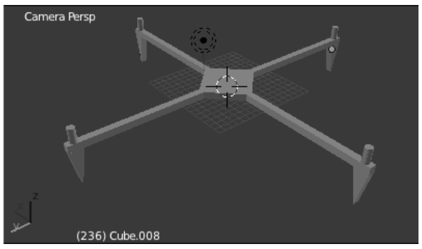


Figura 4 - Modelo (Blender)

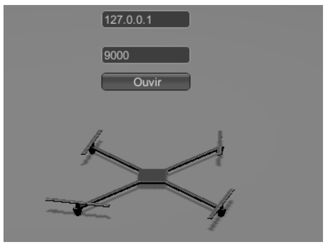


Figura 5 - Simulação (Unity 3D)

Um simples simulador do problema TBU (*Truck Backer-Upper*) utilizando o conceito de redes neurais com aprendizado *online* foi feito, como proposto por P. M. Engel (1996), a fim de testar variações nas condições reais e a resposta da rede a essas variações. O objetivo é analisar até quando uma rede é capaz de contornar e aprender variabilidades de perturbações e ainda manter um resultado satisfatório. Para tal, a dinâmica do modelo do caminhão foi incrementada com diferentes atrasos na resposta (inércia) e velocidades variáveis durante o percurso. O controlador apresentou oscilações notáveis com essas alterações e, portanto, o objetivo deste trabalho será a utilização dos controladores PID em conjunto as redes.

O microcontrolador escolhido inicialmente foi o modelo PIC24HJ128GP202 (Microchip), pois é um dos que permite quatro saídas de PWM (*Pulse Width Modulation*) independentes, que são utilizadas para controlar a velocidade dos rotores e com preço acessível para a aplicação. Além disto, possui uma capacidade de processamento alta em relação aos demais microcontroladores de baixo custo (40 MIPS) e possui ferramentas de otimização para códigos escritos na linguagem C, além de bibliotecas fornecidas pelo fabricante para controle de periféricos. Foi feito um teste de desempenho para o cálculo de uma iteração de aprendizado para a rede neural construída no simulador do TBU comentado anteriormente. Os cálculos realizados na central apresentaram uma velocidade 400 vezes superior ao mesmo cálculo realizado no microcontrolador. A rede utilizada possui uma topologia com três entradas, 25 neurônios na camada oculta e um único vetor de saída. Porém, a RNA que será utilizada para o problema do quadricóptero será um pouco mais complexa e envolverá mais cálculos. Além disto, cálculos para processamento de sinais (incluindo operações de convolução e filtragem) serão também necessários. Será avaliado a necessidade de um outro microcontrolador da família dos DSP’s (com operações otimizadas para este caso) ou a migração dos cálculos mais pesados para a central.

O dispositivo utilizado para realizar esta comunicação via RF será o NRF24L01+ (Nordic) devido ao seu tamanho reduzido, possibilidade de comunicação com garantia de entrega de pacotes e reenvio, baixíssimo custo, antena integrada, velocidade de transmissão e a alta flexibilidade na configuração de parâmetros como tempo de reenvio de pacote, possibilidade de transmissão bidirecional, escolha de freqüência de canal, dentre outros.

Na parte de sensoriamento, faremos uso de uma unidade inercial conhecida como 6DOF, popularmente usada no mercado de *drones* e de baixo custo, composta de um acelerômetro ADXL345 (Analog Devices) e um giroscópio ITG-3200 (InvenSense). A combinação desses sensores nos ajuda a manter o controle e estabilidade em vôo. Estes dados serão utilizados para alimentar o sistema de controle já que os comandos afetam diretamente a leitura dos mesmos. Sabemos, por exemplo, que a saída nula do giroscópio somada à saída do acelerômetro sendo um vetor vertical de módulo igual à aceleração gravitacional, corresponde ao estado de equilíbrio em vôo. Vale notar que nenhum desses dois sensores garante que se saiba se o quadricóptero possui uma velocidade de translação constante em qualquer direção, portanto, além de tentarmos utilizar a unidade inercial para minimzar este efeito, também serão utilizados sensores de distância ultrassônicos para evitar colisões.

Tanto o dispositivo de comunicação RF como a unidade inercial se comunicarão por meio do protocolo SPI (*Serial Peripheral Interface*), cujo microcontrolador tem suporte nativo. Apenas o sensor de ultrassom possui um protocolo próprio que poderá interagir com o microcontrolador por meio das demais portas de acesso.

7. Cronograma

Tarefas	Tempo de execução previsto
Elaboração da topologia da rede neural principal - Número de neurônios e camadas, funções de ativação.	1/2 semana
Equacionamento do treinamento da rede - Cálculo da atualização dos pesos da rede por descida do gradiente, a fim de minimizar o erro.	1/2 semana
Teste de convergência da rede para funções conhecidas - Utilizar funções de transferências conhecidas e verificar se a rede é capaz de aprendê-las com boa precisão.	1/2 semana
Implementação da RNA no simulador (Unity) - Escrever um script dentro do simulador que implemente a rede modelada.	1 semana
Testes iniciais da rede no simulador - verificar se a rede neural consegue aproximar um modelo do processo apenas com pequenos movimentos antes do vôo.	2 semanas
Equacionamento do PID juntamente com a malha de controle - Modelar as funções de transferência dos blocos da malha de controle sem utilizar a RNA.	1/2 semana
Teste de valores conhecidos do PID no simulador - Testar o controle de estabilidade com um PID tradicional e valores conhecidos, utilizando o equacionamento feito.	3/2 semanas
Elaboração do algoritmo de atualização dos parâmetros do PID baseado na rede - Integrar a rede neural projetada com o PID testado.	5/2 semanas
Teste do controlador PID juntamente com a RNA e inicialização randômica - Verificar se, para uma inicialização qualquer de parâmetros, o PID juntamente com a RNA consegue estimar parâmetros aceitáveis para iniciar um trajeto de vôo.	2 semanas
Avaliação da complexidade de todo o algoritmo e tempo de execução (CPU e PIC) - Verificar a complexidade do algoritmo e calcular o tempo de execução na central e no dispositivo embarcado e comparar.	1 semana
Construção do protótipo físico do sistema embarcado - Integrar os componentes reais (microcontrolador, sensores e transmissor) e verificar o funcionamento de todo o conjunto.	2 semanas
Fazer o teste do sistema de controle no modelo real - Integrar todo o estudo feito em simulador com o protótipo físico e verificar seu comportamento.	2 semanas

	Total: 16 semanas

8. Considerações finais

O grande desafio deste trabalho se encontra na união de diversas áreas de conhecimento, já que envolve desde a parte mais matemática e teórica, até a construção física do projeto, passando pela área de controle, inteligência artificial, programação e computação gráfica. Em geral os artigos e livros encontrados sobre esses assuntos, são focados na aplicação individual de cada tema, dificultando a união dos mesmos em uma única aplicação.

É sabido também que caso haja divergência entre a simulação e o teste real, ou seja, caso a simulação obtenha sucesso ao contrário do teste físico, ainda é pode ser possível realizar correções no modelo para que seja novamente testado na prática. Em outras palavras, por se tratar de um método diferenciado, a não obtenção de sucesso inicial nos testes reais não garante a invalidade da proposta. Sendo assim, essa integração pode inclusive sugerir trabalhos futuros que investiguem ainda mais a fundo as aplicações que se pode ter sucesso com esse sistema.

9. Referências

SKOGESTAD, S. (2001) “Probably the best simple PID tuning rules in the world”, In: AIChE Annual Meeting, Reno, Nevada.

LIN, F., BRANDT, R. D. and SAIKALIS, G. (2000) “Self-tuning of PID controllers by adaptive interaction”, In: American Control Conference. Proceedings of the 2000. IEEE, 2000. p. 3676-3681.

IWAI, Z., SHAH, S. L., MIZUMOTO, I., LIU, L. and JIANG, H. (2006) “Adaptive Stable PID Controller with Parallel Feedforward Compensator”, In: Control, Automation, Robotics and Vision, 9th International Conference on. IEEE, 2006. p. 1-6.

VERRASTRO, C. et al. (2007) “Fast self tuning PID controller specially suited for mini robots”, AMIRE Autonomous Minirobots for Research and Edutainment.

SUZUKI, M., YAMAMOTO, T., TSUJI, T. (2004) “A design of neural-net based PID controllers with evolutionary computation”, IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences, v. 87, n. 10, p. 2761-2768.

AGGARWAL, V., MAO, M., O'REILLY, U.-M. (2006) “A self-tuning analog proportional-integral-derivative (pid) controller”, In: Adaptive Hardware and Systems. AHS 2006. First NASA/ESA Conference on. IEEE. p. 12-19.

NICOL, C., MACNAB, C., RAMIREZ-SERRANO, A. (2008) “Robust neural network control of a quadrotor helicopter”, In: Proceedings of the Canadian Conference on Electrical and Computer Engineering. p. 1233-1237.

ENGEL, P. M. (1996) “Attentional Mode Neural Network: a new approach for real-time selflearning”, In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS(ISCAS’96). Anais... IEEE Press, 1996. p.45–48.

NGUYEN, D. and WIDROW, B. (1990) “Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights”, In: Neural Networks, 1990., 1990 IJCNN International Joint Conference on. IEEE. v.3 p. 21-26

SHAHRAKI, F., FANAEI, M. A. and ARJOMANDZADEH, A. R. (2009) “Adaptive System Control with PID Neural Networks”, Chemical Engineering transaction, v. 17, p. 1395-1401.

SCOTT, G. M., SHAVLIK, J. W. and RAY, W. H. (1992) “Refining PID controllers using neural networks”, Neural Computation, v. 4, n. 5, p. 746-757.

SONG, Y., WU, S. and YAN, Y. (2013) “Development of Self-Tuning Intelligent PID Controller Based on BPNN for Indoor Air Quality Control”, International Journal of Emerging Technology and Advanced Engineering.