Jordan University of Science and Technology

A Report Submitted in Partial Fulfillment of the Requirements

for the Degree of Bachelor of Science in Mechanical Engineering, Sub-Branch Mechatronics Engineering

January, 2014

System Modeling of a Variable Pitch Quadrotor

Yazen Alali

Hassan Omari

Supervisors: Dr. Khaled Hatamleh

Eng. Yazan Al-Rihani

Abstract

Variable pitch quadrotors are quadrotors that have extra four actuators to change propeller angle of attack, and they are one of the methods proposed to solve maneuverability limitation in fixed pitch conventional Quadrotors.

This report focuses on the hardware and interfacings that were used in building a variable pitch quadrotor, and discusses some of the problems that were faced and recommends solutions, and then it introduces a mathematical model for the system using Simulink Matlab. Last parts of the report go through the tests implemented to identify the system, the first test is the static test, which was conducted to find thrust and torque coefficients, and the second one is the dynamic test, although this test was not finished, the report discusses some of obtained results.

Key words: Hardware and interfacing, System Identification.

إهداء

نحمد الله تعالى على ما وفقنا لأجله، و ما يسيرها لنا، وما أبعده عنا, اللهم لك الحمد حتى ترضا، ولك الحمد إذا رضيت، ولك الحمد بعد الرضا.

نهدي هذا العمل المتواضع لوالدينا الكرام، اللذين أضاءوا لنا عتمة الطريق بنور حنانهم.

نشكر د.خالد حتاملة لصبره، وثقته التي منحنا إيها في أوقات كادت أن تنطفئ هي في أنفسنا.

شكرا خاص موصول لكل من المهندس يزن ريحاني، والمهندس حمزة الزعبي، والمهندس يحيى المجالي، والمهندس سامر، والمهندس عمرو زريقات.

Acknowledgment

In the name of Allah, the merciful, the compassionate First and foremost all praise and thanks go to Allah (God) for giving me the strength and patience, and providing me the knowledge to accomplish this research study.

Special Thanks to our parents whom assisted us with our project.

A Special note of Thanks to our supervisor Dr.Khaled Hatamleh for his expert guidance and contribution with the project.

Thanks to King Abdullah II Design and Development Bureau for the financial support and special thanks for Eng. Yazan Al-Rihani, Eng. Hamzeh Al-Zoubi, Eng. Yhea'a Al-Majali, Eng. Samer, Eng. Umro Al-Zereqat.

Table of Contents

List of Figures

List of Tables

List of symbols

ω: Rotation Speed.

α: Propeller Pitch Angle.

ms: Millisecond.

μs: Microsecond.

g: Gravity acceleration, 1g = 9.81 m/s^2.

A: Ampere.

V: Voltage.

ρ: Air Density.

c: propeller Cord.

R_P: Propeller Radius.

C_L: Lift Coefficient.

C_t: Estimated Thrust Coefficient.

ab1, ab2, ab3: Estimated Torque Coefficients.

C_{D0}: Parasitic Drag Coefficient.

C_{Di}: Thrust-Induced Drag Coefficient.

Φ: Roll angle (around x-axis).

θ: Pitch angle (around y-axis).

ψ: Yaw angle (around z-axis).

J_r: Rotor inertia.

ω_r: Overall residual propeller angular speed.

l: Horizontal distance, (propeller center to center of gravity).

I_{xx}, I_{yy}, I_{zz}: Inertia Moments.

m: Overall mass.

Acronyms

UV: Unmanned Vehicle.

UAV: Unmanned Aerial Vehicle.

IMU: Inertial Measurement Unit.

GPS: Global Positioning System.

IO: Input /Output.

IDE: Integrated Development Environment.

HZ: Hertz (1 Hz = 1/s).

KHz: Kilo Hertz.

MHz: Mega Hertz.

GHz: Giga Hertz.

KB: Kilo Bytes.

bin: Binary.

rad: Radian.

mAh: Mili-Ampere Hour.

PWM: Pulse Width Modulation.

HEX: Hexadecimal.

TTL: Transistor-Transistor Logic.

ESC: Electronic Speed Controller.

USB: Universal Serial Bus.

TX/RX: Transmit and Receive in Serial communication.

ADC: Analog to Digital Converter.

DCV: Direct Current Voltage.

KADDB: King Abdullah ll Design and Development Bureau.

DMM: Digital Multi Meter.

PID: Proportional Integral Derivative.

SISO: Single Input, Single Output System.

MIMO: Multiple Inputs, Multiple Outputs System.

RPM: Revolutions per minute.

MT: Motion Tracker, it is the IMU that was used, the Xsense MTi-G-700 IMU.

3D: The Three Dimensions.

UART: Universal Asynchronous Receiver/Transmitter.

GPIO: General-purpose input/output.

IEEE: Institute of Electrical and Electronics Engineers.

0x##: means ## is represented in hexadecimal format.

DOF: Degree Of Freedom.

Li-Poly: Lithium Polymer.

NI: National Instruments, a well-known company in the engineering field.

LabVIEW: Laboratory Virtual Instrument Engineering Workbench.

ELVIS: Educational Laboratory Virtual Instrumentation Suite.

DC: Direct Current.

Chapter 1: Introduction

1.1. Quadrotors in general

With today's enormous technology advancement, especially in the industry of microcontrollers and high accuracy sensors, Unmanned Arial Vehicles (UAVs) became a dream come true for engineers and hobbyists, and when we're talking about unmanned aerial vehicle (UAVs), quadrotors are no exception, in fact they opened a wide field for researchers and engineers to design and develop new kinds of autonomously controlled vehicles to serve the growing military, civilian and industrial applications. One of the most interesting recently announced is the Amazon PrimeAir delivery service; where a UAV delivers the costumers online order from the house wares to the destination address within half an hour by means of a Quadrotor UAV.

Figure (1-1): Amazon PrimeAir UAV for shipping purposes

Quadrotors and (UAVs) are examples of closed-loop controlled robots, that take feedback from Inertial Measuring Units (IMUs), which are basically a fusion of more than one sensor providing 3-axis linear acceleration measurements, angular rates measurements, and some of them provide GPS data, temperature and angular positions (roll, pitch and yaw), using embedded microcontrollers and filtering algorithms, most commonly the Kalman filter.

1.2. Concept of Operation

What makes Quadrotors so popular is their simple mathematical model compared to other (UAVs), as well as the ease of orientation control (roll, pitch and yaw).This is done by changing each rotor torque and thrust, which are functions of rotor speed of rotation and propeller parameters (they're constants in the case of conventional fixed pitch Quadrotors). Various movements are shown in Figure (1-2).

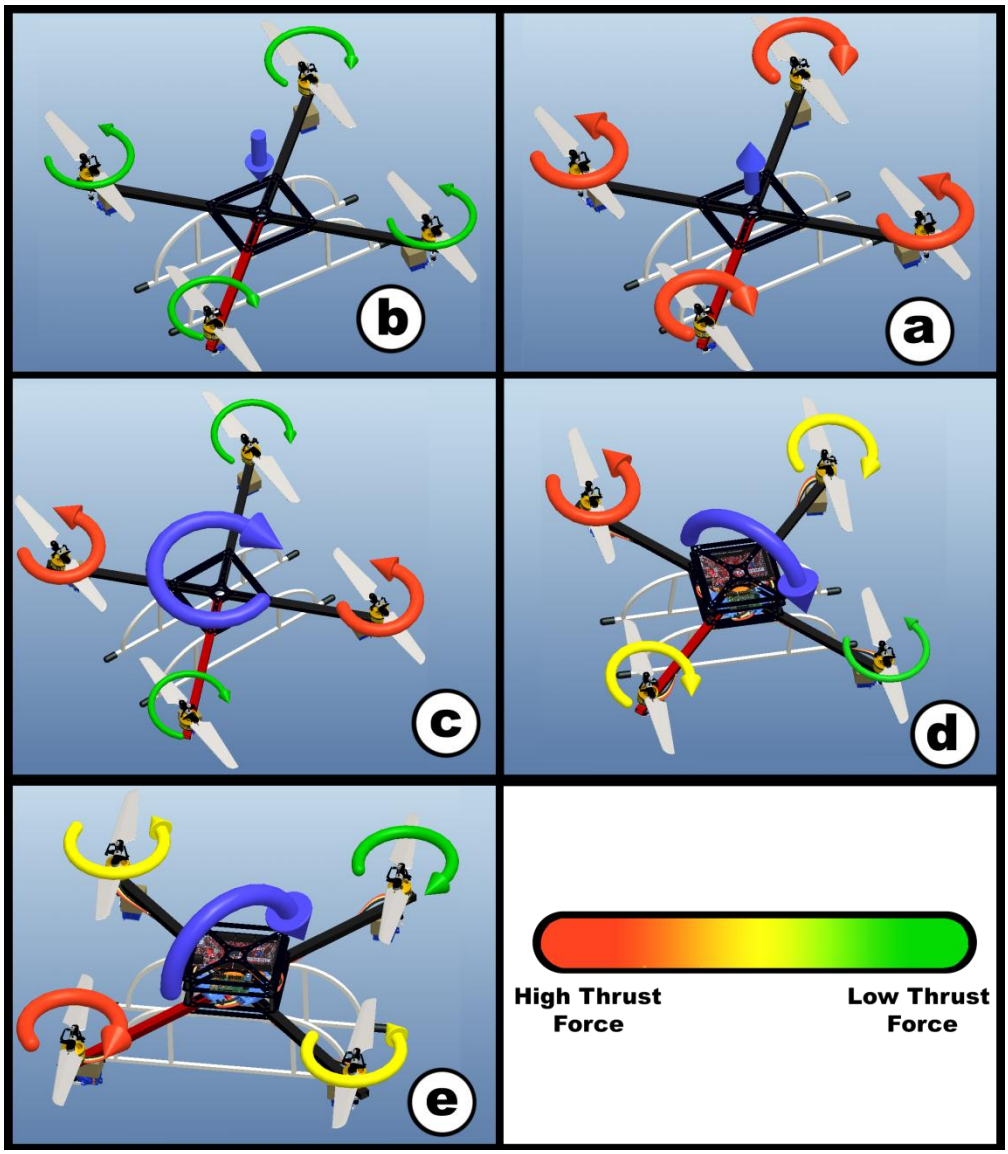


Figure (1-1): Quadrotor various movements

In conventional Quadrotors, changing the propeller speed **ω** is the only way to change both thrust and torque, so the controller receives inputs and feedback signals, and then calculate moments and forces which are related to **ω** of each rotor.

Once **ω** is calculated, the controller sends orders to the motor driver to change the motor speed accordingly, and because of the response limitations any system has (mechanical and electrical response), this change is delayed, a phenomenon widely referred to as phase shift by control references.

The sensors then close the loop again by providing feedback to the controller which then calculates the error, the error after that is converted into desired moments and forces, and the loop goes on.

1.3. Variable Pitch Quadrotors

On the other hand, in a variable pitch quadrotor, there are two factors affecting rotor thrust and torque, which are **ω** and **α** (propeller pitch angle), in this case the controller has the choice either to change **ω** or **α** in order to reach the desired moments and forces, this will create the need for control allocation and increase overall system complexity, meanwhile it significantly improves system response and maneuverability.

Changing **ω** will affect **α** and the opposite is true, and because they are coupled, the system is considered a multiple input, multiple output (MIMO) system, which is harder to be mathematically modeled, this is the main reason why system complexity increases in variable pitch Quadrotors.

This project will implement a single arm of a variable pitch Quadrotor, using a brushless DC motor to drive the propeller, and a servo motor to drive the pitch angle. Moreover, several static and dynamic tests will be performed to acquire the input/output data necessary to extract the lift and drag coefficients as well as a mathematical model for the Quadrotor arm.

Chapter 2: Hardware and Interfacing

2.1. Choosing the Controller

Several aspects were considered before choosing the chipKIT™ Max32™microcontroller board. The most critical one was the microcontroller's Clock speed, since it will deal with a highly unstable system, code loop time must be as short as possible. The second aspect was the controller available (IO) ports, number of analog inputs and numbers of serial communication ports, all of these are needed for the sensors and drivers used.

Figure (2-1): Max32™ Board

Another key aspect was the availability of resources and libraries on the internet, and an easy to use (IDE[1]).Arduino IDE is a very good one, and there are huge number of tutorials and recourse on the internet, but locally available Arduino boards did not fulfill the project requirements. The following are features summaries for two of the Arduino boards (locally available in Jordan stores).

2.1.1. Arduino Uno:

Figure (2-2): Arduino Uno R3 board [2]

Table (1): Arduino Uno R3 Characteristics

Clock Speed	16 MHz
Memory	32 KB
Digital IO Pins	14
Analog Pins	6
Serial Ports	1

1-*"An integrated development environment (IDE) or interactive development environment is a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of a source code editor, build automation tools and a debugger. Most modern IDEs offer intelligent code completion features* "Wikipedia

2-*For more information visit* http://arduino.cc/en/Main/arduinoBoardUno

2.1.2. Arduino Mega 2560:

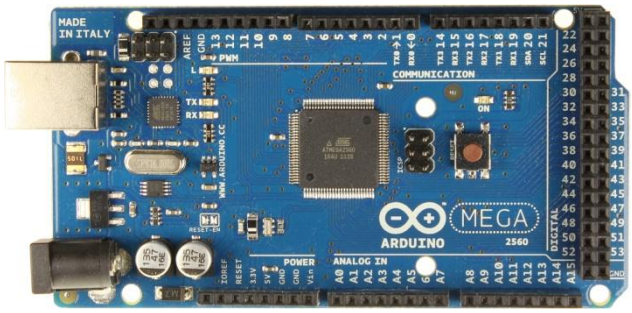


Figure (2-3): Arduino Mega 2560 board [1]

Table (2): Arduino Mega 2560 Characteristics

Clock Speed	16 MHz
Memory	256 KB
Digital IO Pins	54
Analog Pins	16
Serial Ports	4

2.1.3. ChipKIT™Max32™Board:

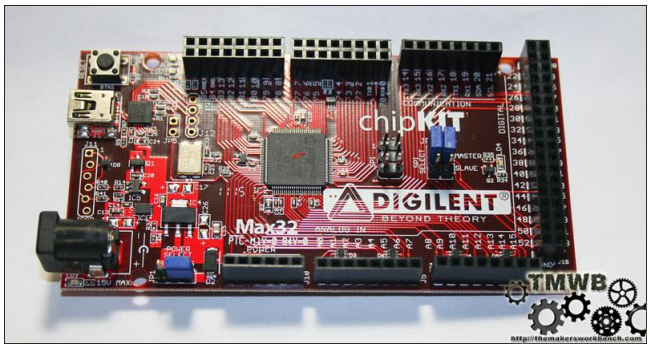


Figure (2-4): ChipKIT MAX32 board [2]

Table (3): MAX32™ Board Characteristics

Clock Speed	80 MHz
Memory	512 KB
Digital IO Pins	54
Analog Pins	16
Serial Ports	4

***1**-For more information visit http://arduino.cc/en/Main/ArduinoBoardMega2560*

***2**-For more information visithttp://www.digilentinc.com/Products/Detail.cfm?Prod=CHIPKIT-MAX32*

2.2. Xsens (IMU)

For outdoor-intended UAVs, onboard IMU is a necessity, and it is advisable to use high quality ones, as they are needed to provide accurate feedback for the system states. The more accurate the feedback is, the more robust the controller could become. One of the available IMUs on the market and yet is a very accurate one is the Xsens MTi-G-700 IMU, shown in Figure (2-5), and because of that it will be used in the project.

Figure (2-5): Xsense MTi-G-700 IMU

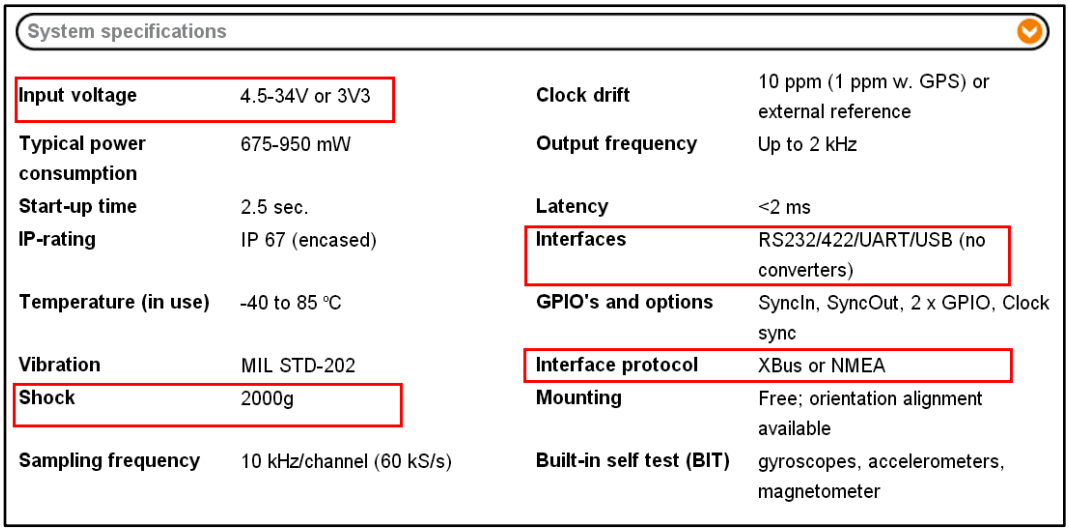
2.2.1. Overview of the Xsense MTi-G-700

Xsense MTi-G-700 is a high performance, accurate and reliable GPS-aided IMU, also referred to as the MT (Motion tracker). It contains gyroscopes, accelerometers, temperature sensor and magnetometers in 3D. It also has an embedded low-power microprocessor running an advanced sensor fusion algorithm that provides more accurate measurements than those provided by the extended Kalman filter.

To communicate with the MT, four interfaces are available, USB, UART, RS422 and the RS232 interfaces, all of them can be used without the need for converters (the host device, of course, should be able to communicate through the used interface, if it is not able to do that, a converter is needed).

USB interface was used to connect the MT with the computer in order to configure it as desired, using the MT Manager, which is special software that comes within its package.

The RS232 Interface was used to communicate between the Max32 board and the MT, a RS232-TTL converter was used as it was needed to achieve the serial communication. Figure (2-6) shows some of the MTi-G-700 IMU specifications.



System specifications

Input voltage	4.5-34V or 3V3	**Clock drift**	10 ppm (1 ppm w. GPS) or external reference
Typical power consumption	675-950 mW	**Output frequency**	Up to 2 kHz
Start-up time	2.5 sec.	**Latency**	<2 ms
IP-rating	IP 67 (encased)	**Interfaces**	RS232/422/UART/USB (no converters)
Temperature (in use)	-40 to 85 °C	**GPIO's and options**	SyncIn, SyncOut, 2 x GPIO, Clock sync
Vibration	MIL STD-202	**Interface protocol**	XBus or NMEA
Shock	2000g	**Mounting**	Free; orientation alignment available
Sampling frequency	10 kHz/channel (60 kS/s)	**Built-in self test (BIT)**	gyroscopes, accelerometers, magnetometer

Figure (2-6): Xsense MTi-G-700 Specification[1]

The MT uses the XBus protocol, which is a low-level communication protocol used for most of the Xsense products, knowledge of this protocol is needed since it will be used to communicate directly with the MT.

Before going through the details of this protocol, it is advised to read more about the single-precision floating-point (IEEE 754) data format discussed in Appendix (C.1), the code for the function that converts a 4 bytes value represented in this format to a floating point variable is listed in Appendix (A.5).

2.2.2. MT States

The MT can be in one of two states, the first state is the configuration state, where many parameters can be configured, such as selecting the desired outputs and formats, the MT can be configured using the MT Manager installed on a computer, or it can be directly configured by the host device during the configuration state, the second state is the measurement state, in this state the MT sends the data to the host device according to how it was configured.
At power-up, the MT sends a **WakeUp** message, if the host device did not respond to it within 500 ms by a **WakeUpAck** message, the MT enters the measurement state, and at any time, changing the current state can be done using a certain message, this is explained in Figure (2-7), in this project the MT was configured using the MT Manager since this method is a lot easier, so what is important here is the measurement state.

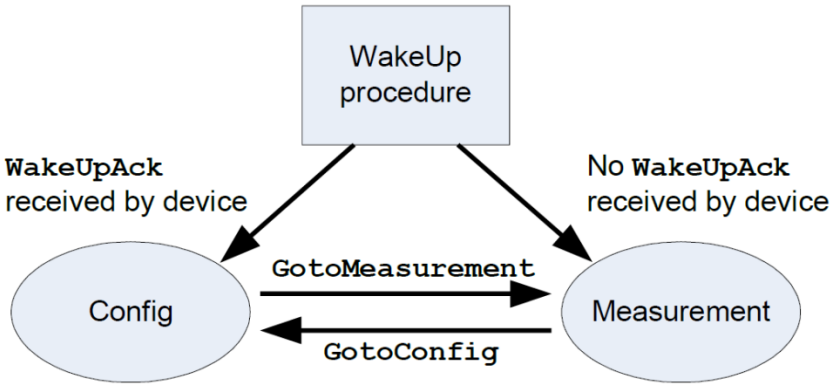
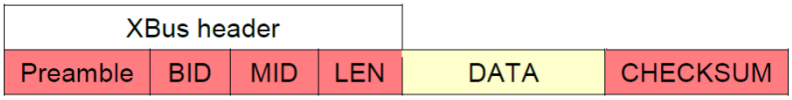


Figure (2-7): MT states

2.2.3. XBus Protocol

The XBus communication protocol that the MT uses is done by messages that are in a standard structure, depending on the size of the data, if the data are smaller than 254 bytes, the MT sends the messages using its standard length messages structure, where as if the data are larger than 254 bytes the messages are sent using the extended length messages structure, the data size on the other hand depends on the configurations.

The standard length messages structure starts with the XBus header followed by the data and ended by the Checksum. Extended length messages structure is not discussed here, since the needed data for this project do not exceed the size of a standard length message.



Field	Field width	Description
Preamble	1 byte	Indicator of start of packet → 250 (0xFA)
BID	1 byte	Bus identifier or Address → 255 (0xFF)
MID	1 byte	Message identifier
LEN	1 byte	For standard length message: Value equals number of bytes in DATA field. Maximum value is 254 (0xFE) For extended length message: Field value is always 255 (0xFF)
DATA (standard length)	0 – 254 bytes	Data bytes
Checksum	1 byte	Checksum of message

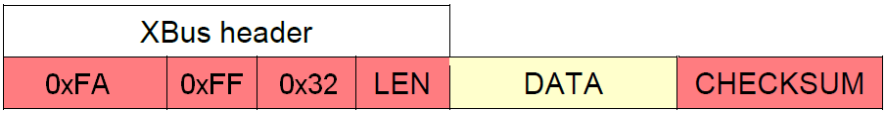
Figure (2-8): Standard length MT message structure

According to the datasheet [#], and since no messages will be sent to the MT (the host device in the project will be only receiving data) the value of the BID byte is always 0xFF.

2.2.4. MT messages

There are many messages that can be sent or received by the MT, they are all listed in its datasheet [#], and since it was used just to provide feedback for the system states, sensors measurements are the only needed data. These measurements which represent the system states can be received from the MT by the **MTData** message.

The **MTData** message has a MID of 0x32, and it is valid in the measurement state, it has the following structure:



The Data in this message depends on the current configuration, for instance the user has the choice to enable/disable temperature data, GPS data and velocity data, etc, shown in Figure (2-9). However, the output data order is always the same as follows:

1. Temp (4 bytes).
2. GPS PVT data (44 bytes).
3. Calibrated data output (36 bytes).
4. Orientation data output (12 bytes).
5. Auxiliary data output (4 bytes).
6. Position (12 bytes).
7. Velocity (12 bytes).
8. Status (1 byte).
9. Sample counter (2 bytes).

Detailed information for each output data can be seen on page 39 in the datasheet [#].

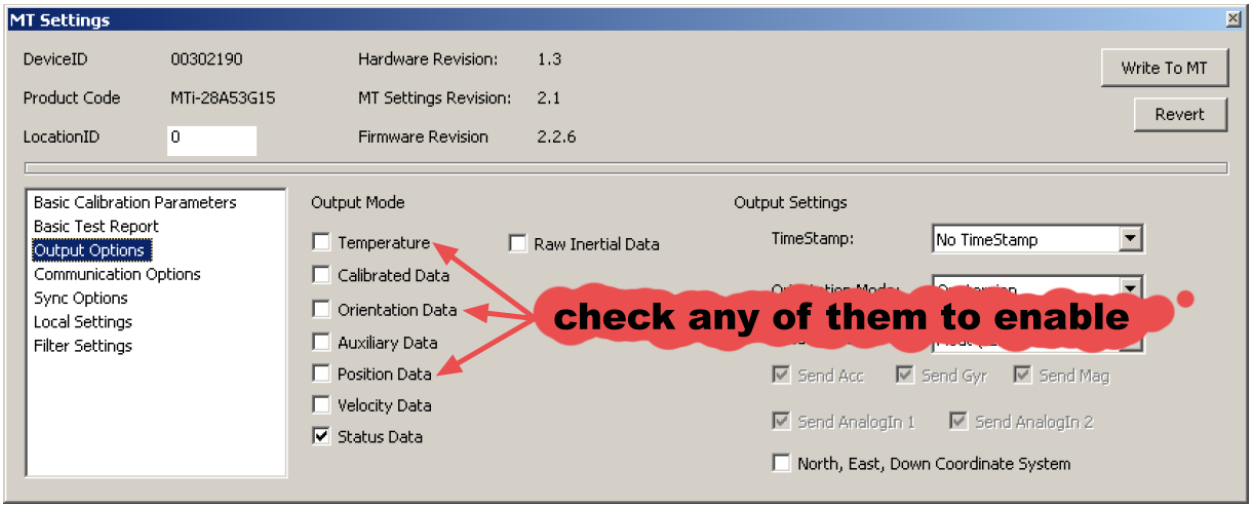


Figure (2-9): MT Manager user interface

The Arduino code that interprets this massage is listed in Appendix A.1.

2.3. PWM Generator

A PWM signal is necessary to actuate the servos and the brushless motors of a quadrotor, the (AXI 2208/20) brushless motor Figure (2-10.a) is connected to an (ESC) Figure (2-10.b) by its three output wires. Two signals to run this speed controller one is the supply voltage 12v and the other is PWM signal with a duty cycle between 1000µs-2000µs. Another two signals for the (HS-55) mini servo Figure (2-10.c) are needed to actuate it, a supply voltage with 5-6v and a PWM signal with the same duty cycle range of the brushless motor.

a)  b)  c) 

Figure (2-10): Components of PWM signal a) Brushless Motor. b) ESC. c) HS-55 mini servo

The PWM signal can be generated by the PWM pins of MAX32™, but this will not work with our case because generating these signals for 4 brushless motors and 4 servos will suspend the microchip and holds up the whole code, a large amount of delays such like this one must be avoided, especially when dealing with Quadrotors or (UAVs). So we must include another component that can generate PWM signals and communicate with the MAX32™ without causing a delay in the code.

2.3.1. Mini Maestro 18

Figure (2-11): Mini Maestro 18

Channels	18
Analog input channels	12
Analog output channels	6
Width	1.10"(2.79 cm)
Length	1.80"(4.57 cm)
Weight	4.9 g
Configurable pulse rate	1-333 Hz
Pulse range	64-4080 µs

Table (): Characteristics of Mini Maestro 18

The Mini Maestro 18 is a PWM generator Figure (2-11) with three control methods, USB for direct connection to a PC computer, TTL serial for use with embedded systems and internal scripting for self-contained, host controller-free applications. The Maestro's USB interface provides access to all configuration options by the Pololu Maestro Control Center software. To match all the channels that are used for our components the Mini Maestro 18 is configured in servo mode, pulse rate at 100Hz which makes the period 10ms, the range of duty cycle is between 1000µs-2000µs, the speed, acceleration, 8-bit neutral and 8-bit range(+/-) were all set by default Figure (2-12).

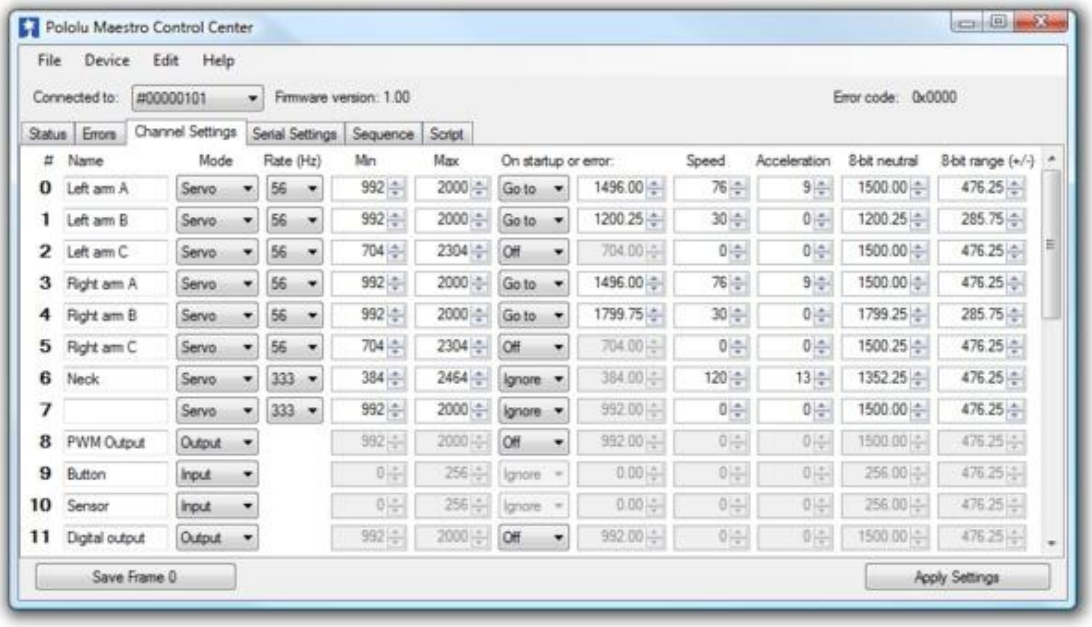


Figure (2-12): Pololu Maestro Control Center

1- *For more information visit http://www.pololu.com/catalog/product/1354*

The (ESCs) of the brushless motors and the servos must be connected directly to one of the channels pins on the Maestro, the servo power pins Figure (2-13) were attached to a supply voltage 5-6v for powering the servos, on the other hand, the supply signal for the (ESCs) was taken from another source which means that they just take the PWM signal from the Maestro.

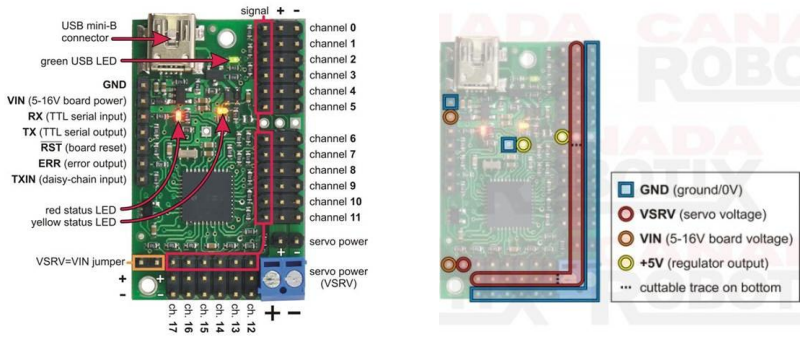


Figure (2-13): Maestro Pins

2.3.2. Maestro Serial

Since the Mini Maestro 18 is an embedded system it does not consume high currents such as the servos or the Brushless Motors, so it is powered directly from the MAX32™ (VIN) and (GND) pins. From its three control ways the TTL serial communication is used for this project with connecting the TX of MAX32™ to RX of the Maestro, the most important of this serial connection is to set both the MAX32™ and the Maestro at the same baud rate Figure (2-14).

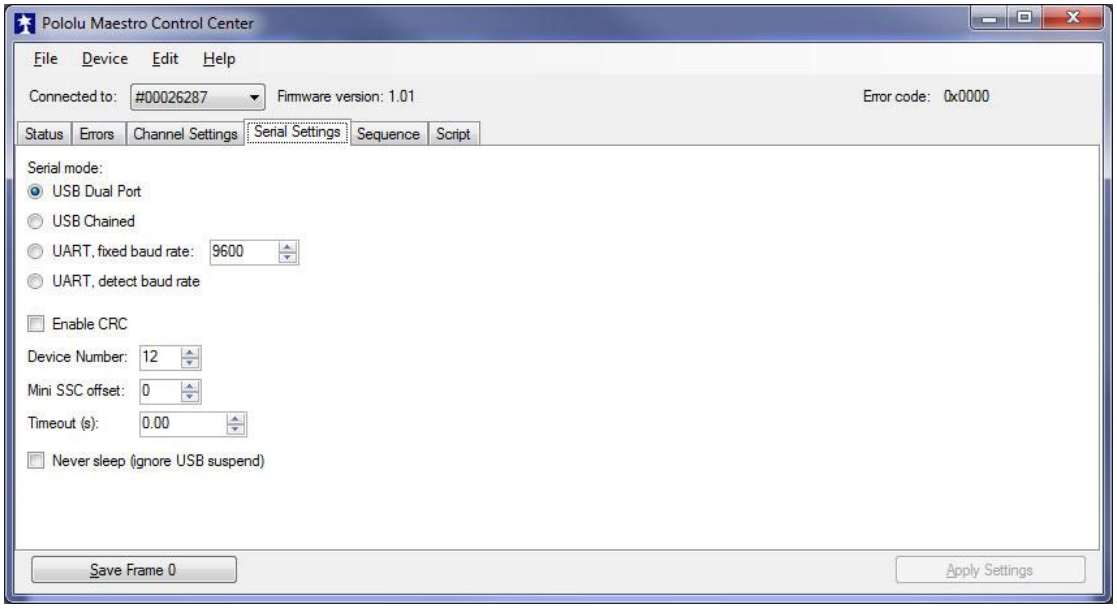


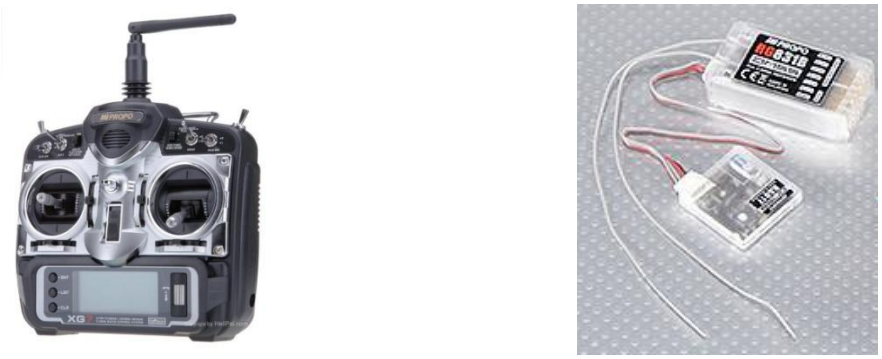
Figure (2-14): Baud rate configuration

A code was written to communicate serially between the MAX32^{TM} and the Maestro PWM generator in Appendix (A.2). The data type that was used with this code is in Appendix (C.2).

2.4. The Ground Station

2.4.1. Transmitter

Any (UAV) needs a ground station to let the user control the vehicle. For the Quadrotor in hand available at (KADDB) the JRXG7 joystick controller is the one used for this project, see Figure (2-15), it has 7-channels and uses 2.4GHz radio communication with its receiver which will be onboard.



(a) (b)

Figure (2-15): a) JRXG7 Joystick. b) Receiver

2.4.2. Receiver

The receiver consists of two modules, the first one is the receiver itself which receives radio wave signals transmitted from the transmitter, then it sends them to the second module which is the PWM generation using TTL serial communication protocol, as the name implies, this module breaks out the serial massage into PWM signals on each corresponding channel, each channel is related to a certain button or joystick on the transmitter.

In order to save code loop time, so that the controller doesn't have to calculate duty cycle of each input, a hardware low pass filter was used to solve this dilemma, the filter will convert the duty cycle into an easy to read analog voltage, the next section is dedicated for the used low pass filter implantation.

2.4.3. Low Pass Filter

A first order low pass filter consists of a resistor and a capacitor. In order to get it right, certain values for the resistor and the capacitor should be chosen taking into account two factors; the first one is the response time, or how fast the filter should be, the second factor is the ripple.

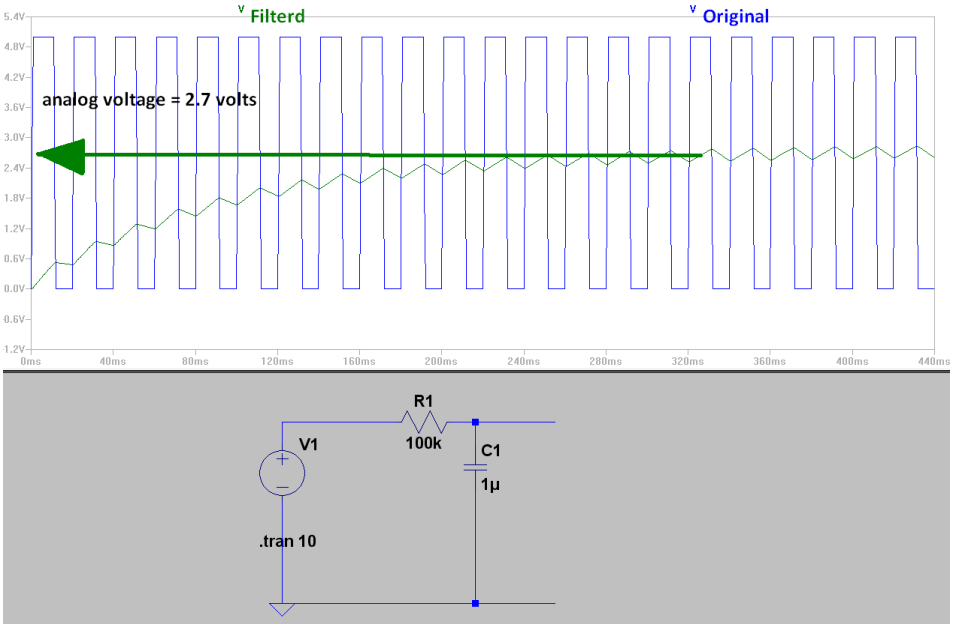


Figure (2-16): Circuit of a first order low pass filter, and a graph that shows the filtered signal and the original one

Choosing the appropriate values are a compromise, if you increase the capacitance, ripple will be reduced and response time therefore will increase, and vice versa. So, in order to get the optimum values without getting into the details of low pass filtering, we used the LTspice[1] simulation program, and changed both values, R and C, to meet the components that were available in the laboratory, and reducing the ripple as much as possible so that the response time is reasonable.

With the first order low pass filter, everything worked the same as simulated by the program used, but the problem was in the ripple, a fine signal is required without any ripples that might change the digital representation of the analog voltage, so the ripple must be smaller than 4.883 mV [2], which is the resolution of the 10 bit ADC in MAX32 board, increasing the order of the low pass filter might solve the problem, Figure (2-18) clarifies the advantage of using a higher order low pass filter.

1-For more information visit http://www.linear.com/designtools/software/

2- Assuming that the ADC reference is 5 V, this value will change if an external reference is used.

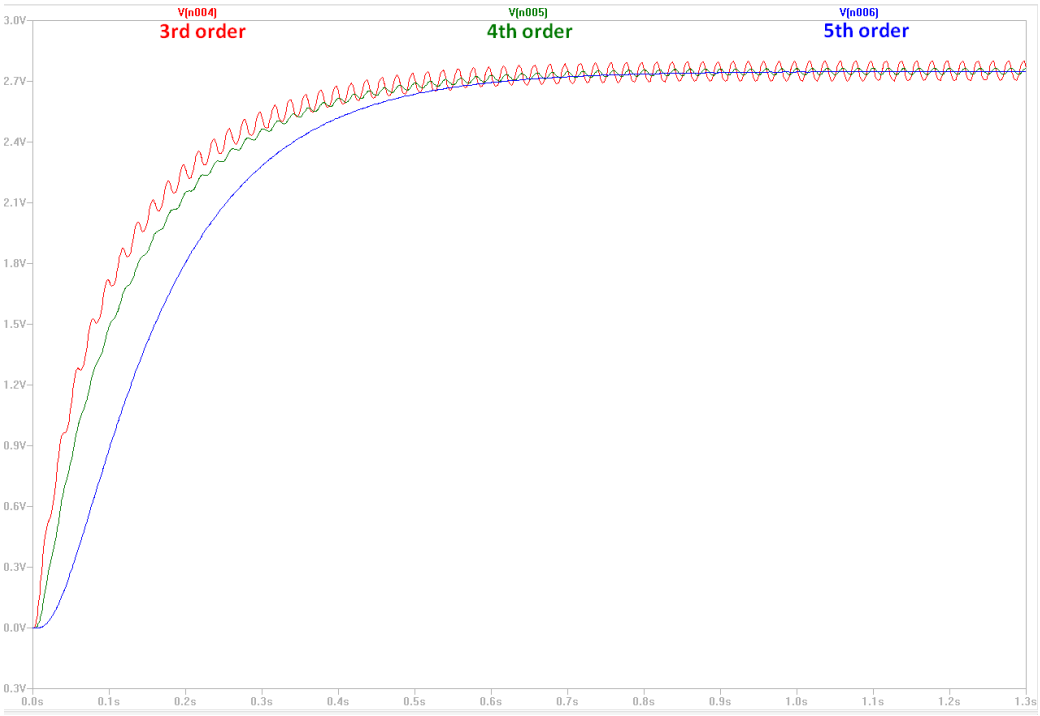


Figure (2-17): Using Higher order low pass filter significantly reduced the ripple

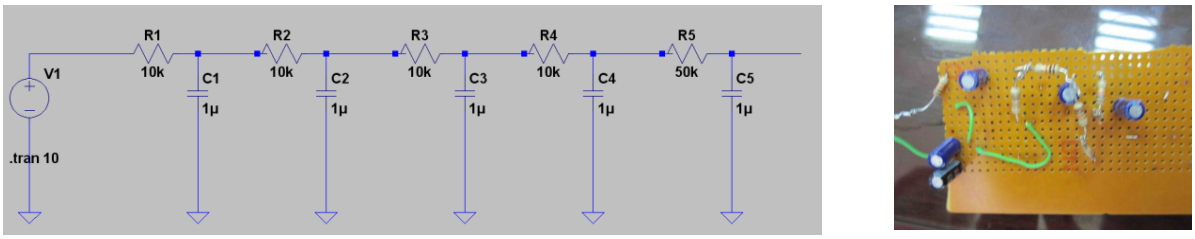


Figure (2-18): Higher order low pass filter implementation

Although the higher order filter managed to get rid of the ripple, the main problem response time, had not been solved yet; the final filter needed 0.4-0.5 second to reach the desired voltage!

Using low pass filters to convert a PWM signal into an analog voltage is useful in applications where fractions of a second don't really matter, for the current case this delay really matters, hence the low pass filters option was dropped, the alternative is discussed at the following section.

2.4.4. Using Serial Communication

It was noticed that the receiver, shown in figure, communicate with the PWM generation module using TTL serial communication, which is the same of the microcontroller uses to communicate through its serial pins, Consequently, the solution, this time, was to interpret the serial communication and directly read the messages through the serial pins on the MAX32^{TM} board.

The receiver didn't have any datasheet, nor we had any clue about its protocol for sending serial messages, by experimental trial and error it was clear that the message contains 16 bytes of data. The address which is the first two bytes is 0301 Hex, and the rest of the 14 bytes are the main data that gives values for seven different channels. These channels can be defined according to the configuration of the joystick controller.

Figure (2-19): Receiver

The thrust, roll, pitch and yaw movements are necessary channels to control the quadrotor. This means that four different channels must be connected and defined to the microcontroller from the joystick controller. The code for this serial connection reads the message and defines the address, then it takes the 14 bytes of data in an array called *Buffer* and separates each two bytes alone which represent a single channel, as shown in the Table (4), finally mapping the values according to desired numbers that the microcontroller needs for its calculations Appendix (A.4).

Table (4): Values of *Buffer*

Channel	***Buffer* array**	**Min Value in Hex**	**Max value in Hex**	**Description**
Thrust	[2]-[3]	14AA	1756	
	[8]-[9]	00AA	0356	
Roll	[0]-[1]	04AA	0758	
Pitch	[4]-[5]	08AD	0B56	
Yaw	[6]-[7]	0CA9	0F54	

The rest of the *Buffer* array is defined into two switches of the joystick controller. This solution was more functional for the Quadrotor due to its high response as well as the code will not delay the whole program of the quadrotor thus it will not affect the update for the motors and servos.

2.5. The Servos

2.5.1. Identifying the Servos

To change the propeller pitch angle of a variable pitch quadrotor a servo propeller mechanism, Section (2.5.3), must be installed to each motor, the main component of this mechanism is the servo that will actuate the propeller. The objective of testing experimentally the servos was to figure out the properties differences between each servo even if all the servos is same type and produced from the same factory. The potentiometer of each servo will give an indication of the real servo angle. To get the signal of the potentiometer we needed to open each servo and take the output signal of it then connecting it to any analog pins of MAX32^{TM} and to a DMM, the arm of each servo was attached with a protractor to record the angle.

It is well known that the servo works with PWM signal between 1000µs to 2000µs duty cycle which comes from the PWM generator, the program code will take all the range of the servo duty cycle by steps of 50µs Appendix (B.2), records of all analog voltage were acquired by MAX32^{TM}, voltage by DMM, and the degree by the protractor. The test was performed on 5 servos (HS-55) four of them for the quadrotor and the fifth for static test Chapter (4) (Thrust and Torque), Table (5) shows the readings for the servo number four.

Table (5): Reading's servo number 4

Servo PWM (µs)	Analog pin MAX32™	Analog Voltage (Volts)	Arm Degree
1000	248	0.8038	0
1050	260	0.8446	4.5
1100	272	0.8856	8.5
1150	286	0.9272	12.5
1200	299	0.9672	17
1250	312	1.0086	21.5
1300	324	1.0498	25.5
1350	337	1.0909	29.5
1400	350	1.1308	33.5
1450	363	1.173	38
1500	376	1.2131	41.5
1550	388	1.2532	46
1600	401	1.2946	50.5
1650	414	1.335	54
1700	427	1.377	58.5
1750	440	1.4177	63.5
1800	453	1.4574	67.5
1850	464	1.4981	72
1900	478	1.538	76.5
1950	491	1.5784	81.5
2000	504	1.6199	85.5

After recording all the readings for all servos it can be noticed that differences between them can be negligible. By taking the average equation for the PWM versus Degree Figure (2-20), and the Analog Voltage Figure (2-21), it can be used to get the ratio between the servo angle and the actual angle (Angle of Attack) and indicates what servo angle value when recording readings for Static test. The actual angle was also directly measured using an accelerometer; the following section is dedicated for it.

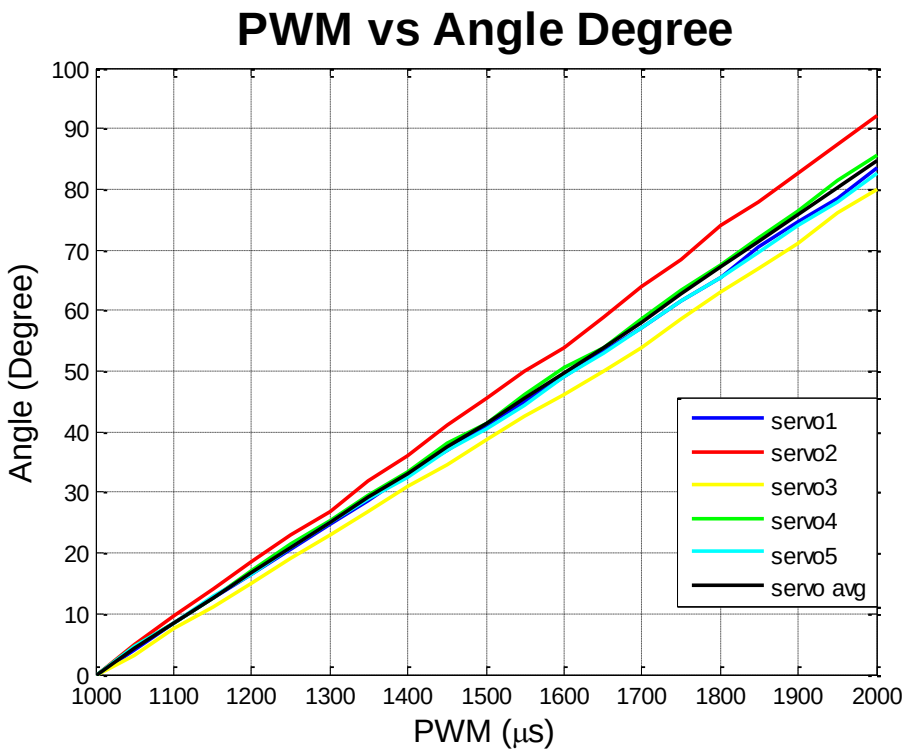


Figure (2-20): PWM versus. Servo Angle

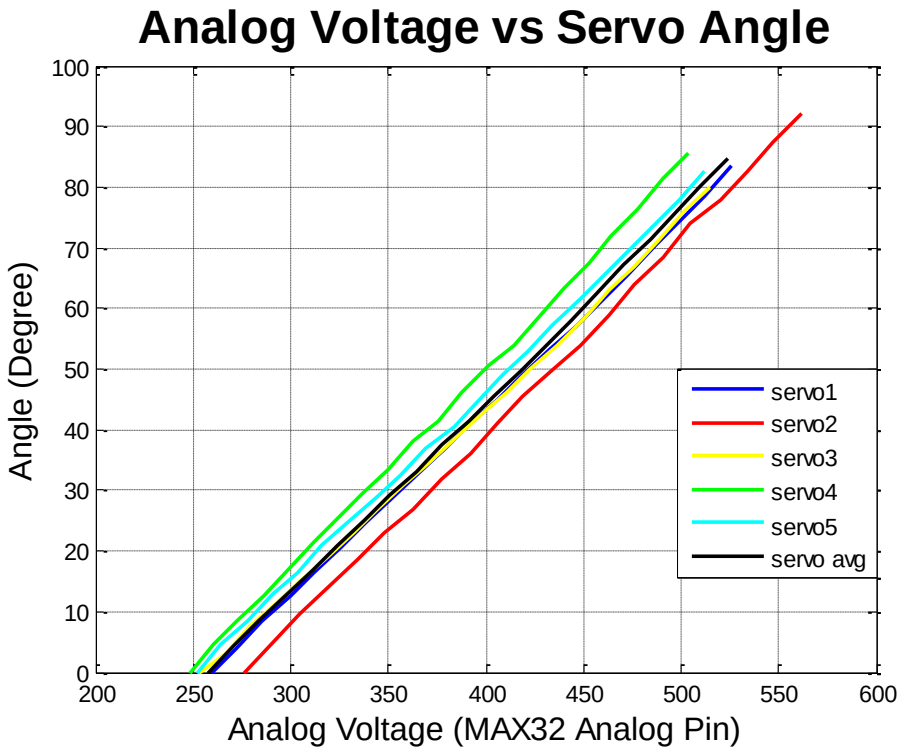


Figure (2-21): Analog Voltage versus. Servo Angle

2.5.2 Angle of Attack

Angle Calculations

Although the feedback was directly taken from the servo's potentiometer for the pitch angle, the row data taken from it are useless until they are mapped or converted into real angles, one way to do so, is by using an accelerometer mounted on a blade and measuring the tilt angle which equals the pitch angle, see Figure (2-22)

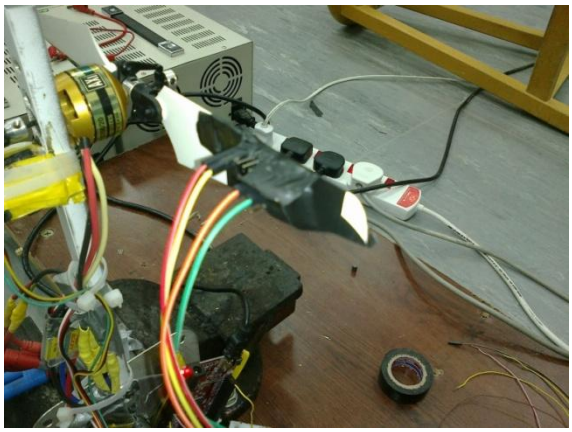


Figure (2-22): accelerometer mounted on a blade

The ADXL335 triple axis accelerometer was used, this sensor measures the acceleration on each axis. The output is an analog voltage on each corresponding pin. It could be used to measure tilt angle providing that the accelerometer is completely stationary such that the only acting force is the gravity.

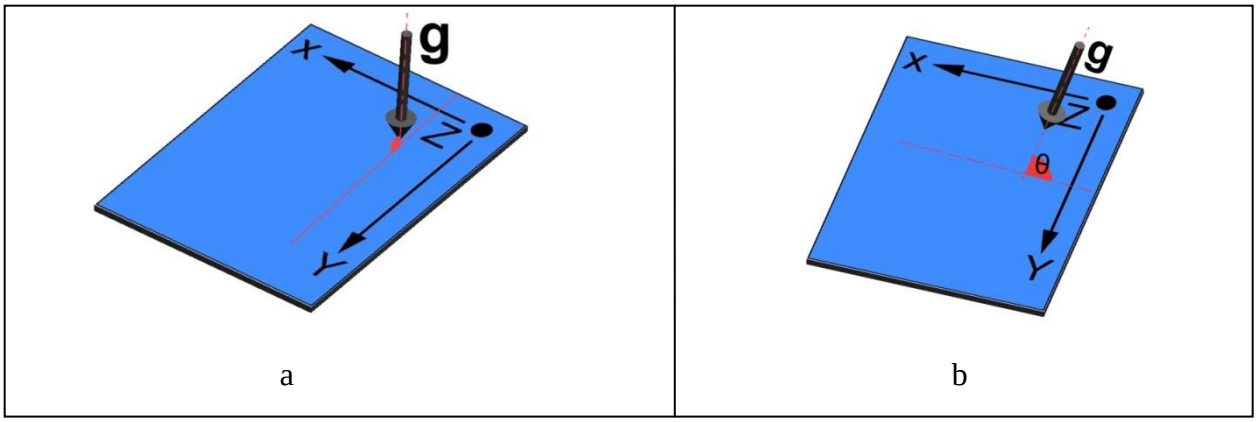


Figure (2-23): Accelerometer Orientations

In Figure (2-23.a), when the accelerometer is horizontally oriented (parallel to the ground), the gravitational force will act only on the z-axis, in this case the sensor will measure 1 g on this axis, and 0 g on the other two axes, in Figure (2-23.b) the accelerometer is tilted by an angle θ, and therefore, the gravitational force will act on two axes. In this orientation, it will act on the z-axis and the x-axis and the sensor will measure the following:

On the z-axis	**On the x-axis**
$z = g*\sin(\theta)$	$x = g*\cos(\theta)$

The only unknown is θ, and it can be easily calculated using $\tan^{-1}(z/x)$, $\tan^{-1}$was not used and $\sin^{-1}$was used instead, as it caused a problem whenever x=0, so basically:

$\Theta=\sin^{-1}(z/g)$

Accelerometer Sensitivity and Offset

According to its datasheet, the offset equals half of Vin (input voltage that drives the sensor), and the output sensitivity varies proportionally to the supply voltage, in other words we have to find the sensitivity on our own. Another problem was encountered when a cheap and imitative accelerometer was used, it was found that even the offset does not exactly equal half of Vin, so both of them were experimentally measured by fixing the sensor horizontally so that the x-axis will measure zero g, and the output analog voltage was read from the x-pin, this reading represents the offset.

In a similar way, the sensitivity was found by orienting the sensor such that the x-axis will measure 1g, thus the sensitivity is found by subtracting the reading of the analog voltage from the offset that was already found, and this procedure was repeated for the other two axes.

Mapping

By taking measurements of θ and the potentiometer analog voltage, a curve could be fitted to these points using least square method Appendix (B.1). The equation of the curve is θ as a function of analog voltage, shown in Table (6) and Figure (2-24).

We used Arduino Uno for this test. Both Arduino and Matlab (used for least square regression) codes are listed in Appendix (B.3).

Table (6): Reading's Angle of Attack

Servo PWM (µs)	**Analog pin MAX32™ (potentiometer feedback)**	**θ (degree)**
1900	478.65	23.58
1860	467.99	20.865
1820	458.345	18.155
1780	448.18	16.545
1740	438.205	13.71
1700	428.03	11.145
1660	417.4	8.54
1620	407.68	6.255
1580	396.515	2.9
1540	386.98	-0.07
1500	376.12	-2.7
1460	366.48	-4.89
1420	355.5	-7.545
1380	345.26	-9.325
1340	335.105	-11.425
1300	325.955	-13.025
1260	315.735	-15.195
1220	304.495	-17.075
1180	294.97	-18.55
1140	284.03	-20.325
1100	273.485	-21.6

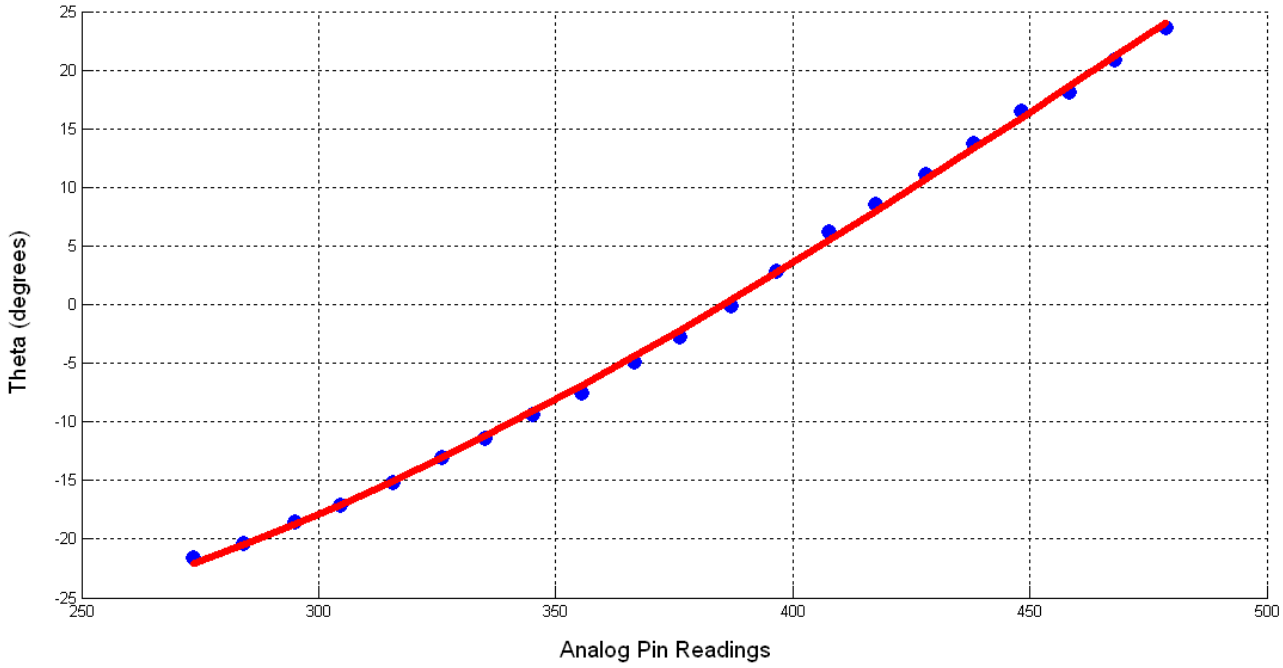


Figure (2-24): Theta versus Analog pin MAX™

2.6. Assembly

2.6.1. Motors and Propellers

Motors and propellers assembly was easy since most of the parts were pre-assembled by the manufacturer. The EVP unit for AXI 2208, provided from Model Motors Ltd [1] was used. The package contains the brushless DC motor with its propeller mechanism but without the servo, which is needed to change the pitch angle as discussed before.

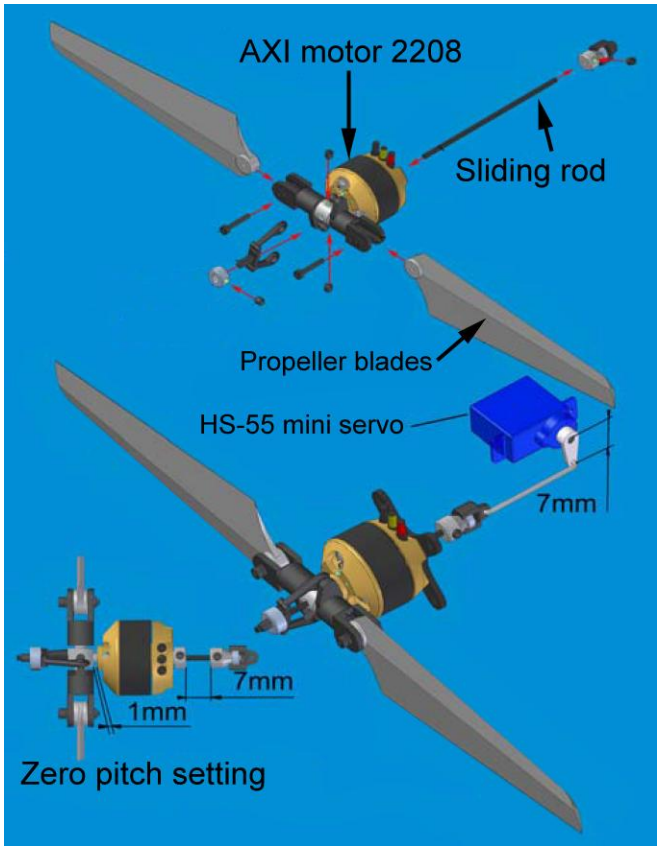


Figure (2-25): Installation instruction for the propeller, extracted from its manual guide [1]

1- For more information visit http://www.modelmotors.cz/index.php?page=101

2.6.2 Servo-Propeller Mechanism

The HS-55 mini servo was used, it is light and yet it provides the needed torque for changing the pitch angle, Figure (2-26).

Figure (2-26): Hi-tech, HS-55 mini servo

One of the problems that happened was the servo-propeller mechanism, or how can the servo be attached to the rod which is connected to the blades. The other thing was the servo mounting on the frame, since due to time limit to design a base that can be easily mounted on the frame that the servo can be attached to it without adding unnecessary parts, which will add extra weight, the only solution was to fix the servo on the frame using adhesive and plastic bands, Figure (2-28).

After trying different mechanisms for connecting the servo with the blade rod, it was finally connected using a 2 bar-linkage mechanism, Figure (2-27). This mechanism allows the servo to rotate its arm (the first link) without being resisted by the sliding rod (blade rod).

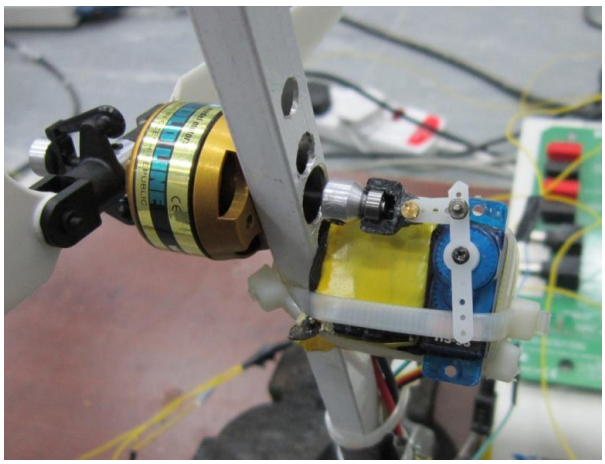


Figure (2-27): It shows servo mounting and the mechanism used

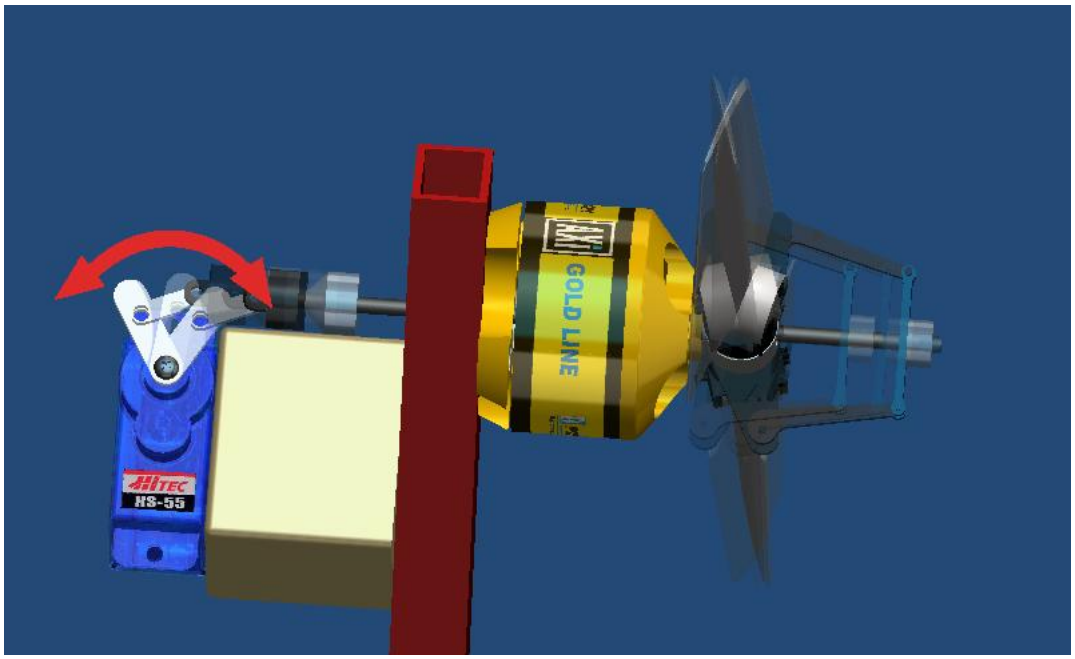


Figure (2-28): Servo can rotate without being resisted by the rod

Previous Mechanisms and Problems

The first mechanism that was used consisted of a bar that connects the servo arm to the sliding rod (which is connected to the blades), Figure (2-29). This mechanism restricted the servo rotation on certain angles since the sliding rod can only move in one direction which is along the motor axis, as a result, servo couldn't hold the angle, especially on high RPM values and it drew more current than expected, this caused a problem during the static test. Each time the motor was going up to high RPM values the servo couldn't hold the blades pitch angle and therefore the angle kept changing during the test!

Figure (2-29): First servo-propeller mechanism

We tried to solve this using another mechanism, which consists of one bar (servo arm) that has a slot, which gives the servo the freedom to rotate without being restricted, Figure (2-30).

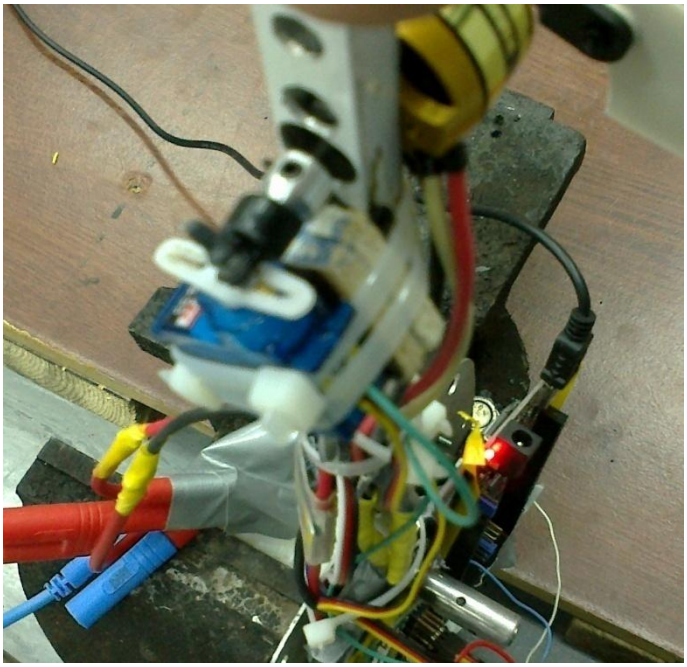


Figure (2-30): Second servo-propeller mechanism

This mechanism Figure (2-30) had no problems except that it was not done in a good way, the slot was manually graved and had not had a smooth surface that allows sliding, and otherwise it would work just fine.

Recommendations

It is recommended using 3D printed linkages dedicated for the mechanism. It is also recommend using metal gear servos instead of plastic ones, since they wear out with time and the metal ones hold the blades better. Another thing is the servo mounting, a base should be designed to mount the servo on instead of using adhesive and plastic bands.

2.6.3. The Frame

As the name implies, a variable pitch quadrotor have four brushless motors, so a frame of four aluminum arms and two of plastic base is assembled to mount these motors in the way they meant to be as in Figure (2-30). The aluminum was chosen due to its lightness and its ability to hold these motors and other components. One of the arms is painted with red color which indicates the x-axis of the body fixed frame, shown in Figure (3-6).

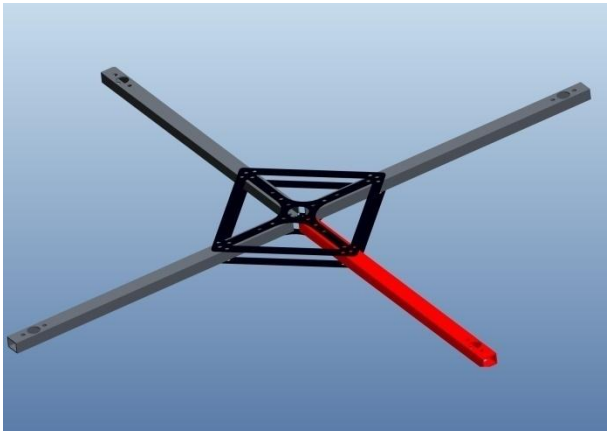


Figure (2-30): Frame of Quadrotor

2.7. Miscellaneous

2.7.1. Speed Controller

The brushless motors that is utilized for the variable pitch quadrotor needs a device that converts the PWM signal from both the generator chapter (2.3) and the 12 V from the power supply, to a three-phase signal, the brushless motor will receive this signal from an (ESC) Electronic Speed Controller, shown in Figure (2-31).

Figure (2-31): Electronic Speed Controller

2.7.2. Batteries

As the brushless motor consumes high currents up to 16 A with only 12 V, it needs a rechargeable long life battery that will afford this duty. A 3-cell Li-Poly battery is used for the variable pitch quadrotor with 11.1 V and 4500 mAh; batteries like this weigh about 400 grams.

Chapter 3: Mathematical Model

There are two methods to construct a variable pitch quadrotor from scratch. The first method is by gathering all the hardware parts of the quadrotor and all the software's needed for it, such that a PID controller with initial guesses of the gains is added to the whole code. The final step is tuning the gains of the controller experimentally (trial and error) while the quadrotor is attached to a test bench; this step will consume a lot of time and in most of the cases it will not reach the optimum solution for the values of the gains of PID controller. This method is based on the engineering sense because it deals with the system regardless of its mathematical model and properties.

The second method starts with identifying the system of the variable pitch quadrotor which will specify the entire necessary coefficient and characteristic to complete a whole mathematical model, this model expresses the system by equations of motion and transfer functions, and then it is built on the Simulink–Matlab. The Simulink will determine the optimum values of the PID controller gains; by this the motion of the quadrotor could be predicted. After placing these values in the code of the quadrotor, it will be attached to a test bench to compare the actual motion with the simulation motion which will not be exactly the same, but it will accomplish the purpose of this method and gives the best solution. However, the second method is more scientifically approached and reliable since it was used for this project.

Figure (3-1) shows the planned variable pitch Quadrotor 3D model.

Figure (3-1): 3D Model of a Variable Pitch Quadrotor

3.1. System Modeling with Simulink-Matlab:

The concept of system modeling for the variable pitch quadrotor is to describe mechanical components and physical effects in mathematical equations, the parameters for these theoretical equations are determined by experimental tests and the 3D model.

The Simulink-Matlab was used to build the mathematical model of the variable pitch quadrotor Figure (3-2). The control allocation part was not reached due to the problem with the servo propeller mechanism see Section (2.6.2), which had taken several months to solve it. So, the gain values for the PID controller were not estimated from this model.

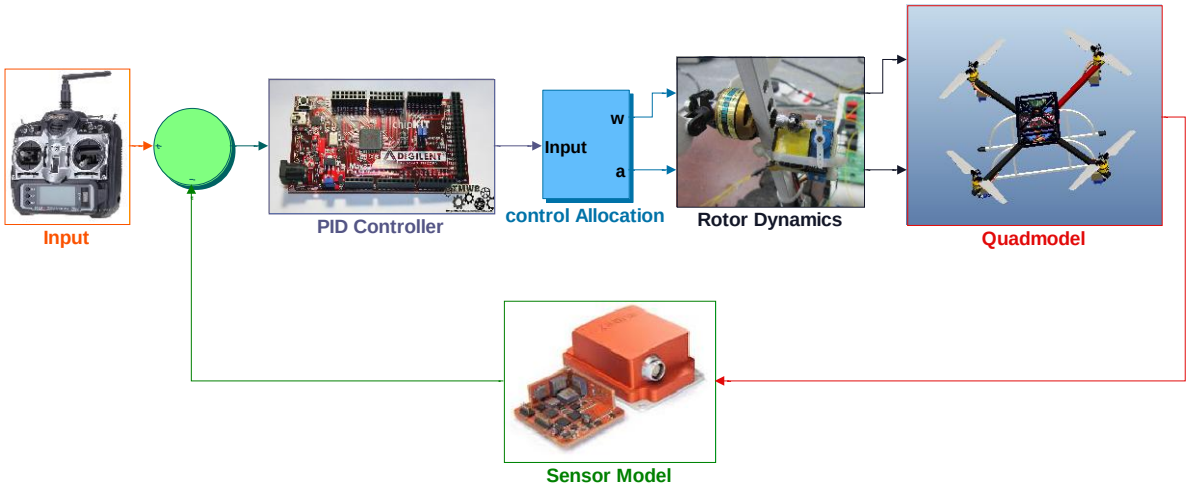


Figure (3-2): System Model

3.2. Input

The ground station (joystick) Section (2.4) will let the user control the variable pitch Quadrotor. In this model it is necessary to built a block that will represent the desired inputs from the user, also it will be the refreneces for the PID controller. The inputs block contains of the four desired parameters which are the z-axis, roll angle, pitch angle and yaw angle that are assumed to be constants in this model as in Figure (3-3).

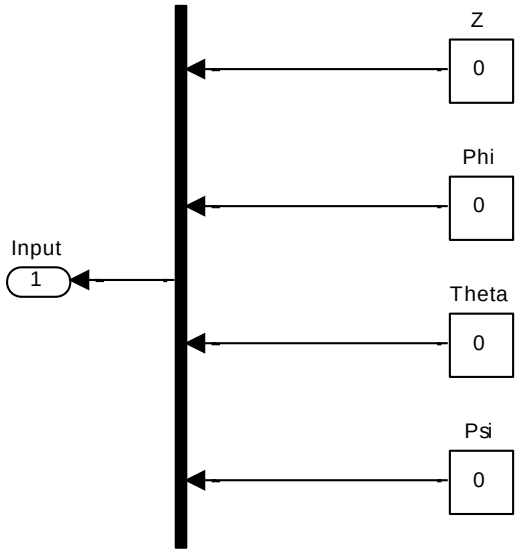


Figure (3-3): Input Block

3.3. PID Controller

The error or comparison signal which is between the feedback values and the desired input values needs to be corrected by a PID controller or in the case of a variable pitch Quadrotor it is also called a stability control. In this model the target of the PID controller is to control four error values which are the z-axis position, roll angle, pitch angle and yaw angle as shown in Figure (3-4).

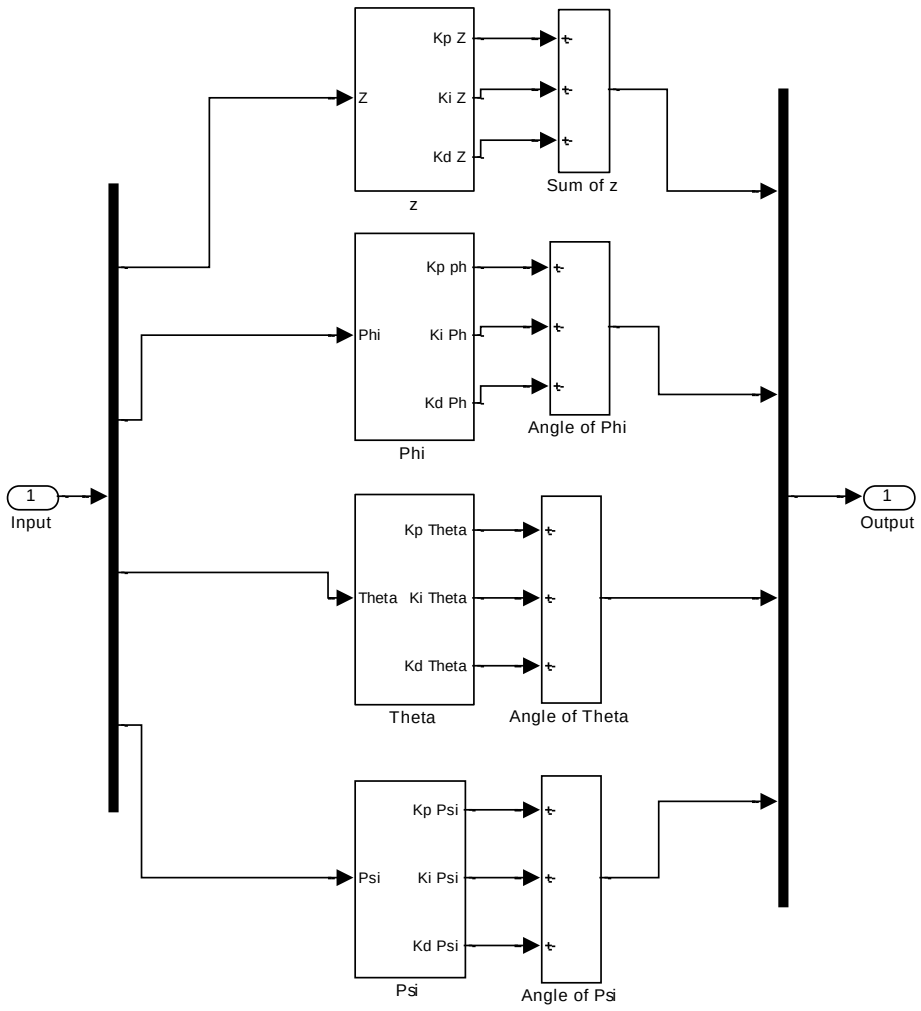


Figure (3-4): Quadrotor PID controller

The PID controller algorithm involves three separate constant parameters which called: the proportional (P), the integral (I) and derivative (D) values. These values can be interpreted in terms of time such that P depends on the present error, I on the accumulation of past errors, and D is a prediction of future errors, based on current rate of change. The weighted sum of these three actions is used to adjust the process via a control element. Figure (3-5) shows the part of controlling the z-axis position.

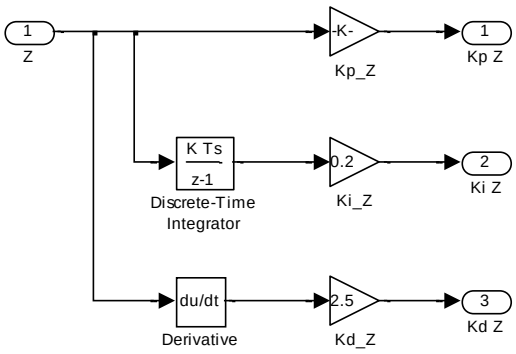


Figure (3-5): z-axis Position Controller

3.4. Control Allocation

The variable pitch Quadrotor is an overactuated system that a classic PID controller can not control the system. It requires a control allocator which decide the values of the anguler propeller speed ω and the pitch angle α. These values is calculated from the corrected parameters after the PID controller according on a cost function or weighted function. This part was not completed due to lack of time.

3.5. Rotor Dynamics

The rotor dynamics in this model represent the actuators mechanical delay and the phase shift responses between the desired inputs (ω and α) and the actual output. The variable pitch Quadrotor is a MIMO system, as it has two coupled actuators which are the brushless and servo motors. That means whenever α is changed ω is affected by some type of function. These responses and effects will be expressed by transfer functions that will be determined in the dynamic test Chapter (5) and due to lack of time these functions was not defined.

3.6. Quadrotor Model

A rigid body model must be built to describe the variable pitch quadrotor structure which is a complex mechanical system; it collects physical effects from aerodynamics and mechanics domains, the model of the quadrotor should consider all the important effects such as aerodynamics effects, gravity effect and the gyroscopic effects [7].

Let's consider a body fixed-frame such that the origin is at the center of gravity for the variable pitch quadrotor, by this three main Rotation angles can be defined as Φ around x-axis, θ around y-axis and ψ around the z-axis Figure (3-6).

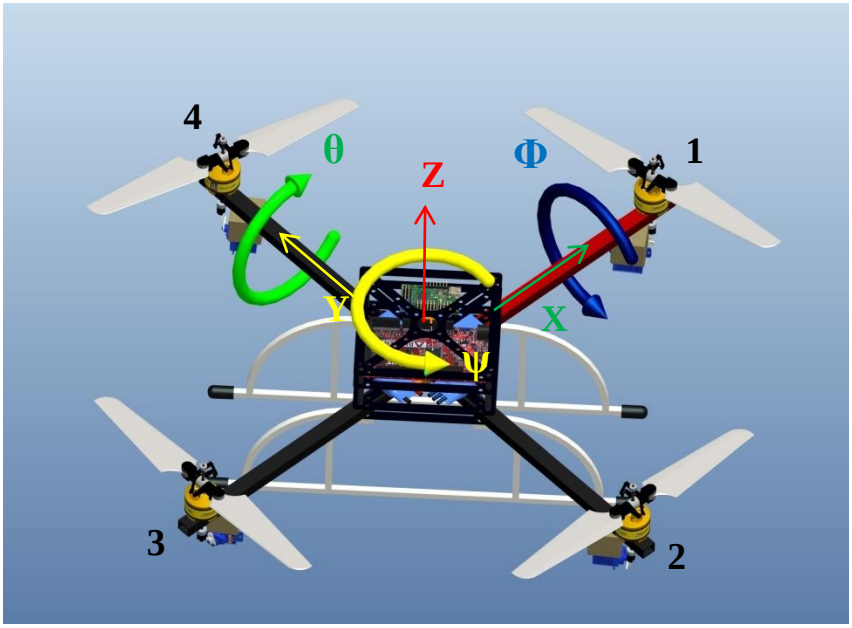


Figure (3-6): Rotation of Angles

Quadrotor motion is caused by several forces and moments that comes from different physical effects, the summation of these effects are the model of the system, here the Newton-Euler method was used to derive the mathematical model and for this model it is assumed the following:

- Vehicle structure is assumed to be rigid and symmetrical (i.e. on both x and y axes).
- Propeller is also assumed to be rigid at all angular speeds.
- Body fixed frame is coincident with the center of gravity.
- The hub forces and the friction are neglected.
- Thrust and drag are functions of both propeller angular speed and blade angle (pitch angle).

3.6.1. Equation of Motion:

As the variable pitch quadrotor is an (UAV), the aerodynamics effects must be considered first, in this model two major of these effects are modeled by equations which are the trust force (3.1) and the rotor torque (3.2), both of them depends on the two variable inputs which are the angular propeller speed ω and the pitch angle α [4].

$$T = C_t \omega^2 \alpha \tag{3.1}$$

Where **Ct** is the thrust coefficient.

$$Q = ab1\ \omega^2 + ab2\ \omega^2 \alpha^2 + ab3\ \omega\alpha \tag{3.2}$$

Where **ab1**, **ab2** and **ab3** are the torque coefficients.

The variable pitch quadrotor motion is obviously caused by a series of forces and moments, this model considers the following points (with c:cos, s:sin) [6].

Rolling Moments

Body gyro effect	$\dot{\theta}\dot{\psi}(I_{yy} - I_{zz})$
Propeller gyro effect	$J_r \dot{\theta} \omega_r$
Roll actuators action	$l(-T_2 + T_4)$

Pitching Moments

Body gyro effect	$\dot{\phi}\dot{\psi}(I_{zz} - I_{xx})$
Propeller gyro effect	$J_r \dot{\phi} \omega_r$
Pitch actuators action	$l(T_1 - T_3)$

Yawing Moments

Body gyro effect	$\dot{\theta}\dot{\phi}(I_{xx} - I_{yy})$
Inertial counter-torque	$J_r \dot{\omega}_r$
Counter-torque unbalance	$\sum_{i=1}^{4} (-1)^i Q_i$

Forces Along x-Axis

Actuators action	$(s\psi s\phi + c\psi s\theta c\phi) \sum_{i=1}^{4} T_i$

Forces Along y-Axis

Actuators action	$(-c\psi s\phi + s\psi s\theta c\phi)\sum_{i=1}^{4} T_i$

Forces Along z-Axis

Actuators action	$(c\psi c\phi)\sum_{i=1}^{4} T_i$
Weight	mg

Resultant of forces [6]

$$\begin{cases} U_1 = C_t(\omega_1{}^2\alpha_1 + \omega_2{}^2\alpha_2 + \omega_3{}^2\alpha_3 + \omega_4{}^2\alpha_4) \\ U_2 = C_t(\omega_4{}^2\alpha_4 - \omega_2{}^2\alpha_2) \\ U_3 = C_t(\omega_3{}^2\alpha_3 - \omega_1{}^2\alpha_1) \\ U_4 = -Q_1 + Q_2 - Q_3 + Q_4 \end{cases} \tag{3.3}$$

Where U1, U2, U3 are the resultant of forces in the z-axis, and U4 is the resultant of torques around the z-axis, all of these forces and torques in (3.3) are due to the four propellers of the variable pitch quadrotor.

By the summation of all these forces and moments the equation of motions can be considered as follows (with c:cos, s:sin) [6]:

$$\begin{cases} I_{xx}\ddot{\phi} = \dot{\theta}\dot{\psi}\left(I_{yy} - I_{zz}\right) + J_r\dot{\theta}\Omega_r + lU_2 & M_x \\ I_{yy}\ddot{\theta} = \dot{\phi}\dot{\psi}(I_{zz} - I_{xx}) - J_r\dot{\phi}\Omega_r + lU_3 & M_y \\ I_{zz}\ddot{\psi} = \dot{\theta}\dot{\phi}\left(I_{xx} - I_{yy}\right) + J_r\dot{\Omega}_r + U_4 & M_z \\ m\ddot{x} = (s\psi s\phi + c\psi s\theta c\phi)U_1 & F_x \\ m\ddot{y} = (-c\psi s\phi + s\psi s\theta c\phi)U_1 & F_y \\ m\ddot{z} = mg - (c\psi c\phi)U_1 & F_z \end{cases} \tag{3.4}$$

These equations of motions (3.4) are written as a function in the Simulink model to express the actual system of the variable pitch quadrotor appendix (D.1), with this function the 6-DoF (Euler Angles) block is used to calculate the positions and the rotation angles of the variable pitch quadrotor which is needed to be controlled Figure (3-7), this block needs to get the calculated equation of motions from the previous function and the overall mass with the moments of inertia around all three axes.

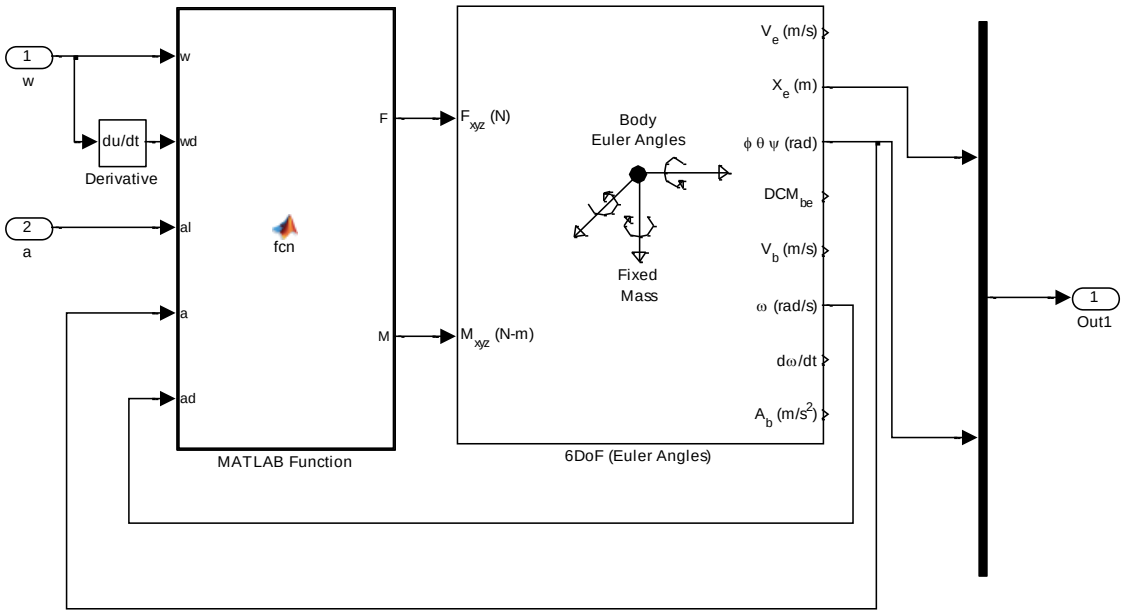


Figure (3-7): Quadrotor Model

3.7. Sensor Model

The PID controller in this model needs a feedback for the four parameters which are z-axis position, roll angle, pitch angle and yaw angle, In reality the controller will get the feedback from the Xsense MTI-G-700 sensor Section (2.2). As known that any kind of sensor could not give an exact value for what it is sensing and each sensor and to simulate this, a sensor model is built with consideration of the four parameters as in Figure (3-8).

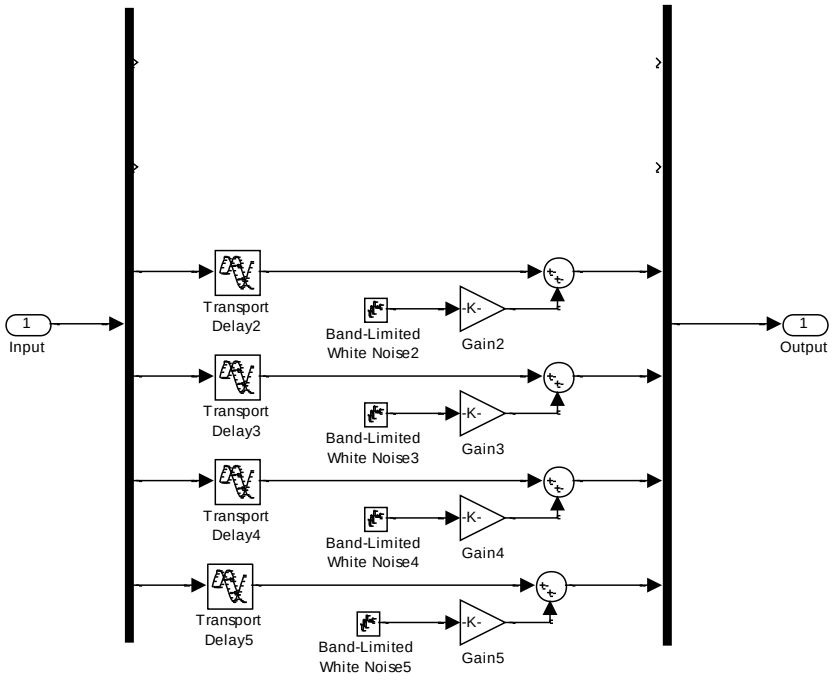


Figure (3-8): Sensor Model

The method of building the sensor model and all the values of the Gains, Band-Limited White Noises and the Transport Delays in Figure (3-8) were taken from Eng. Hamzeh Al-Zoubi at (*KADDB*).

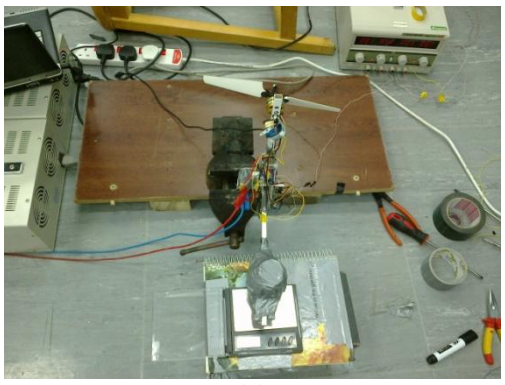
Chapter 4: Static Test

This chapter is devoted for the tests used to determine the coefficients of thrust and torque equations that are needed for the quadrotor model which was discussed in Section (3.6). The thrust equation has one coefficient symbolized by **Ct**, and the torque equation has three different coefficients symbolized by **ab1**, **ab2**, and **ab3** respectively. All data was collected in the steady state region of the system and the transient response was ignored, as it will be discussed in Chapter (5).

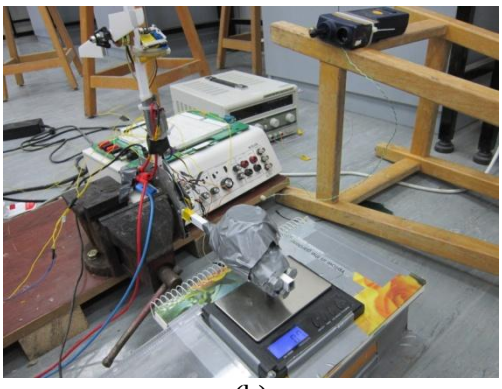
4.1. Setup for Static Test

An appropriate setup is essentially needed for any test or experiment in order to collect required data. The variable pitch quadrotor has four frame arms and due to its geometrical symmetry the static test will be dealing with only one single arm, such that the results obtained can be projected on the other three arms.

This single arm contains two beams which are perpendicularly attached to each other by means of a square-shaped base. A roll bearing is connected to this base, and this arm is attached to a vise from the side of the bearing that it is free to rotate around the bearing axis (one degree of freedom). For the thrust test the brushless motor with its servo propeller mechanism was assembled to this arm at the same direction of motion as shown in Figure (4-1.a), but in torque test it will be perpendicular to the motion of the arm as in Figure (4-1.b). Some weight is added at the side of the scale to remove unwanted vibration during the test.



(a)



(b)

Figure (4-1): Setup for Static test a) Thrust setup. b) Torque setup

The target of the static test was to determine thrust and torque coefficients by obtaining four required types of data which are:

1- The brushless motor current which was measured by a DMM (in amperes).
2- The angular speed of the propeller (in RPM) which was measured by a laser Tachometer that requires a reflector to be aligned with it.
3- The analog voltage of the potentiometer for the servo which is measured from the MAX32™ microcontroller (the value was between 0-1023 according to the 10-bit ADC).
4- The weight of the frame arm which is measured by a scale (in grams).

The (ESC) of the brushless motor and the servo will be both connected to the PWM generator with different power sources. However, the generator was serially connected to the MAX32™ see Section (2.4.4). So, a code was needed for this test that will allow controlling each of the motor and the servo at individual points which is required to obtain the data see Appendix (B.4).

As the system has two variable inputs, it must be considered to recover a reasonable range for them. For the angular speed of the propeller, all the ranges should be covered unless the brushless motor's current reached its maximum at (16 A). However, for the pitch angle of the propeller the range will be between -15 to 15 degree angle according to Ashley Duncan [3]. The variable pitch quadrotor will operate around this range. The experimental data has emphasized and indicated that it was the proper range. On the other hand, a wider range, which is between -17 to 19 degree was adopted to make sure that within the proper range (Figure 4-2).

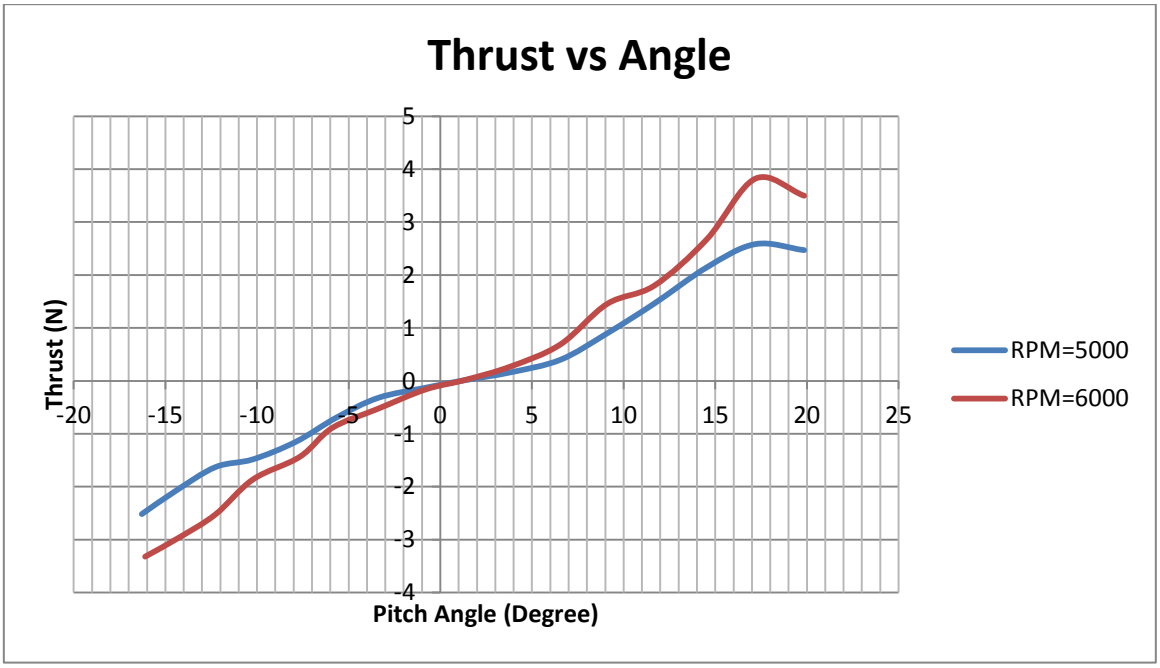


Figure (4-2): Thrust versus Pitch Angle

The procedure of recording the experimental data was performed by fixing one of the inputs at different point such that each of these points will cover the whole range of the second input. In our project the pitch angle range is divided into several angles and the whole range of the angular speed is recorded for each angle. So, as long as the angle points were increased the more the specific range will be covered and this leads to more accurate coefficients.

4.1.1. Problems

Three major problems were faced, one of them was discussed previously in Section (2.6.2), the second problem was with the scale that was used, and it was a digital scale with error of (1gram). However, the main issue of it was the lack of precision and a long delay time to give the value. So, to solve this problem another accurate digital scale was bought, that read the value faster and more accurate at (0.1gram) error Figure (4-3).

Figure (4-3): Digital scale

The third problem was the power supply for the brushless motor. This motor can draw maximum current (16 A) with (12 DCV), so it needs a power supply can cover at least its maximum current. During the training course in (*KADDB*) facilities a (50 A) power supply was used and was appropriate to do the job. After the training course in KADDB and back to the university, the maximum current of power supply was up to (10 A). The solution was to connect two of them in parallel that should raise the current to (20 A) to meet experimental needs Figure (4-4).

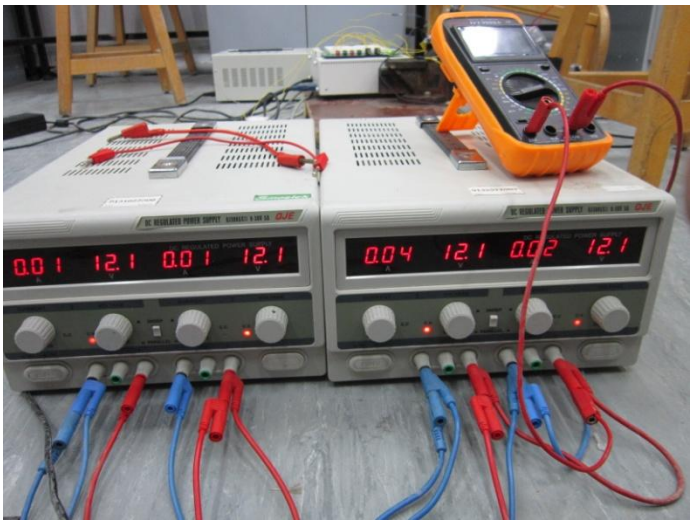


Figure (4-4): Parallel Connection

4.2. Future Work

To give the static test more accuracy and more reliability it can be computerized and performed automatically. So, a force cell can be used instead of using a digital scale to measure the weight, also the photodiode with a laser pointer aligned at it can be used instead of the tachometer to measure the propeller angular speed. These values with the analog voltage of the servo potentiometer that represent the pitch angle can be connected to the NI-ELVIS platform Section (5.2.1) and measured at the same time. By this configuration the required readings are performed faster and can obtain more accurate values of the unknown coefficients.

4.3. Thrust Test

The target of the thrust test is to define the coefficient **Ct** and an optimum range of thrust force that can be produced by the motor-propeller combination. These two values were obtained by analyzing experimental data. Sample of thrust data is displayed in Appendix (B.5). The way of mounting the setup for the thrust test will let any force derived by motor-propeller combination translated to weight reading at the scale, Figure (4-5).

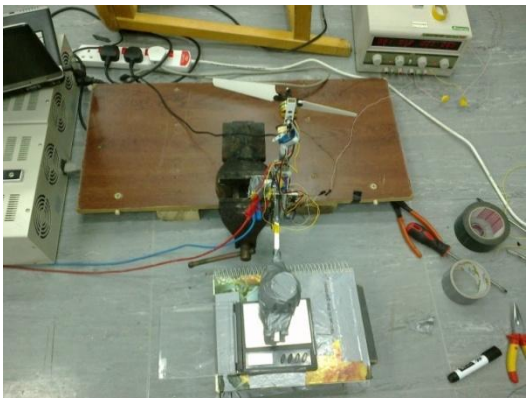
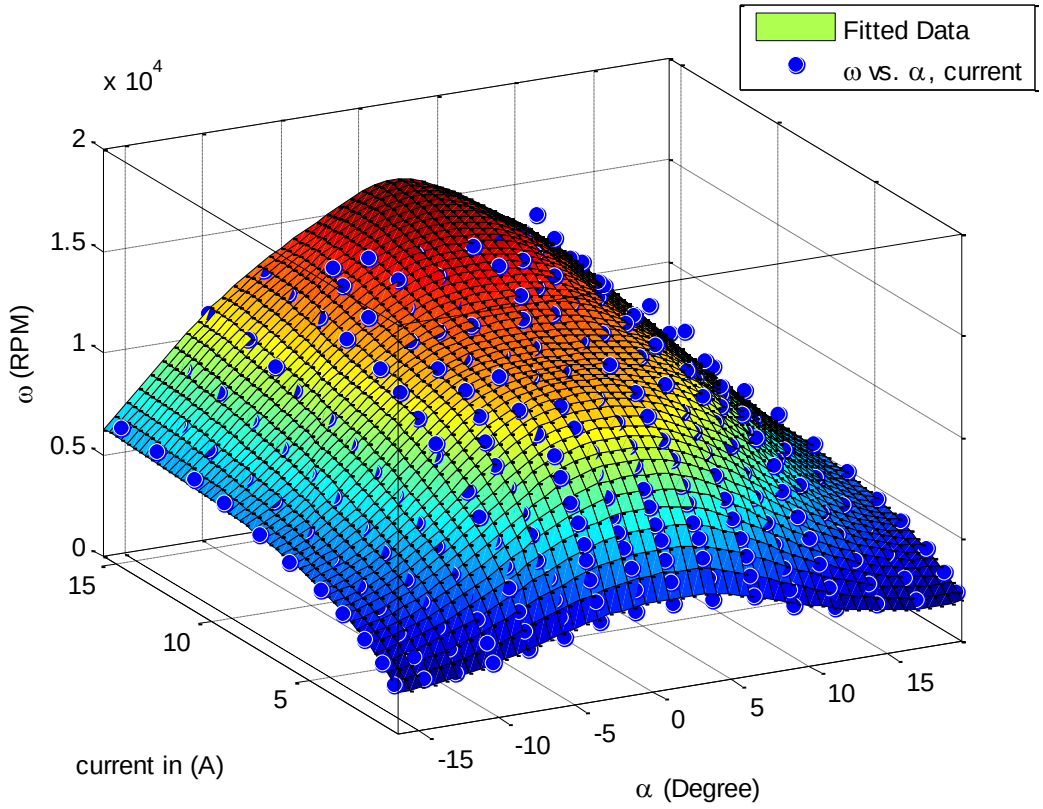


Figure (4-5): Setup for Thrust Test

As it was mentioned previously that four types of data were recorded to be analyzed, the actual or instantaneous pitch angle was calculated via regression method from the analog voltage feedback of the potentiometer of servo Section (2.5.2). For the thrust force it can be noticed that any forces produced by the motor-propeller combination will affect the beam which is mounted on it and there will be a moment on the square base. Since the distance of the two beams is equal, the produced force on the scale is also equal to the force of the motor-propeller combination, Figure (4-5). In order to convert the reading of the scale to Newton force it should be multiplied by the gravity acceleration.

The thrust force optimum range that the variable pitch quadrotor will operate depends on an important factor, which is the consumed current by the brushless motor. As it was mentioned previously, a 3-cell Li-poly battery was used for the quadrotor, so the controller must limit the consumption of current by the brushless motors that it does not overload the battery and at the same time it gives an enough thrust force to do the wanted movement.

Figure (4-6) shows the consumed current by the motor-propeller combination at different ω (RPM) and α (Degree).



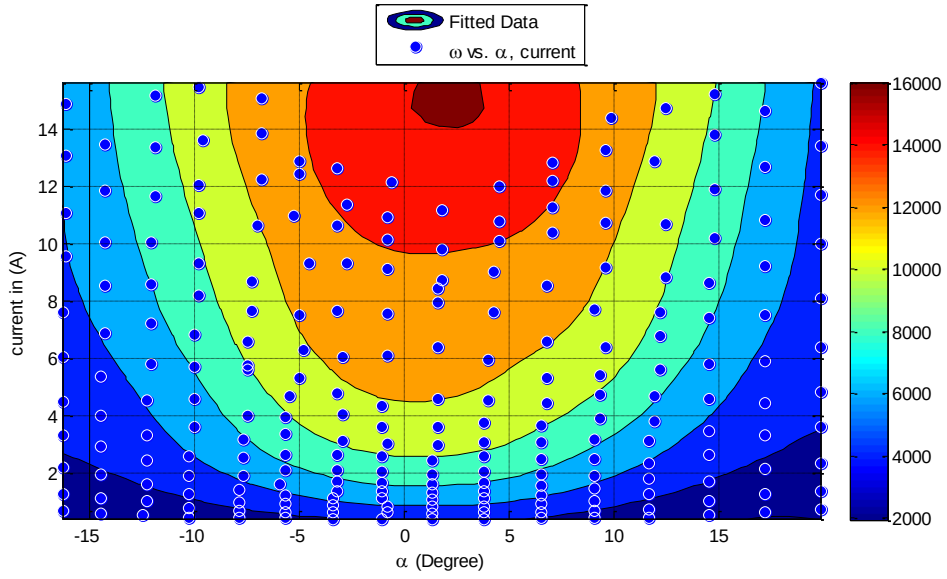


Figure (4-6): ω versus current,α

Figure (4-7) shows the consumed power by the motor-propeller combination at different thrust force (N) and α (Degree).

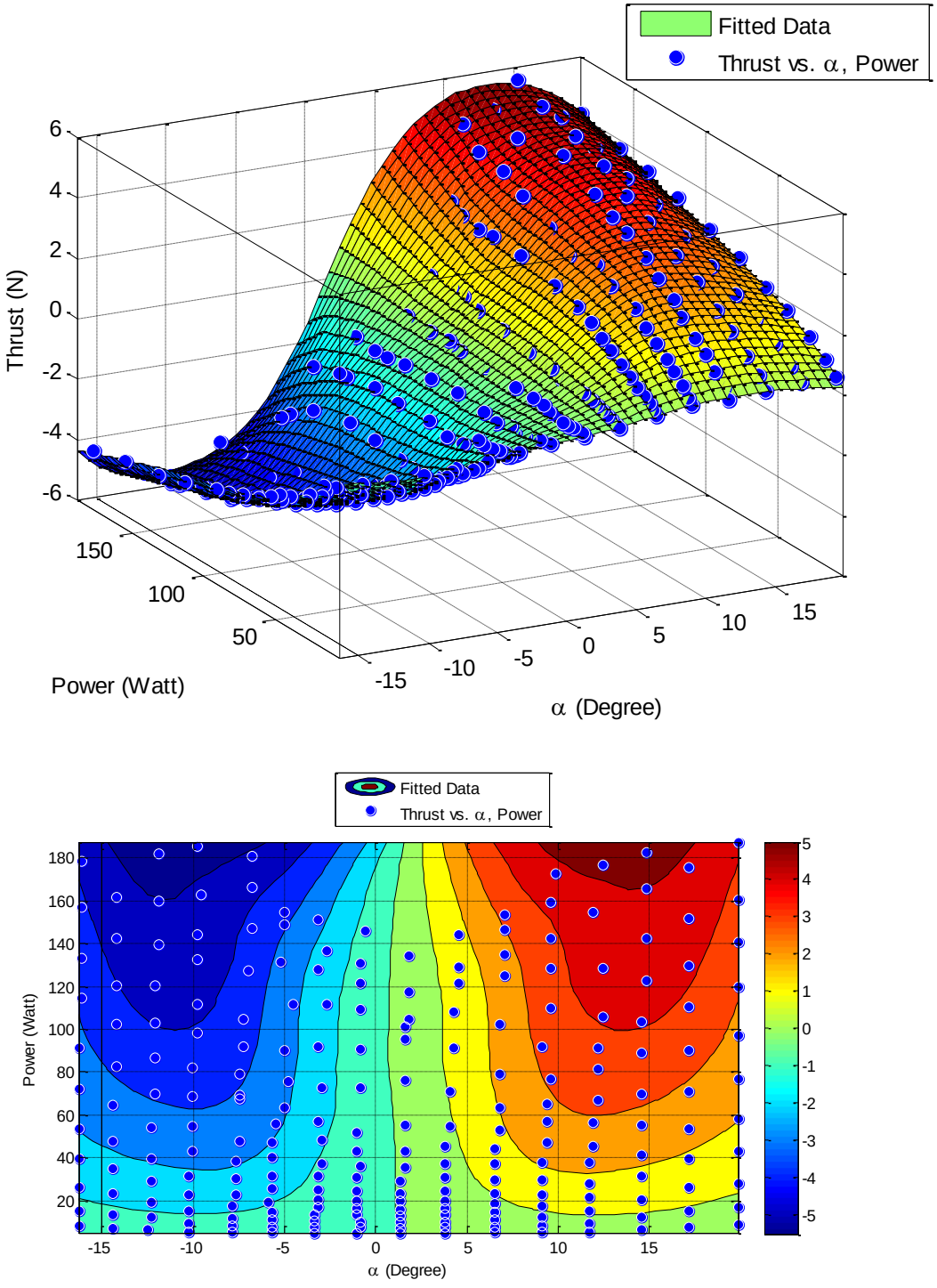


Figure (4-7): Thrust versus power, α

4.3.1. Thrust Coefficient

The thrust coefficient **Ct** is one of the parameters that will be used to derive a prime mathematical model Chapter (3). In a variable pitch quadrotor, the thrust force is the vertical force acting on the propeller element; it can be described by [4]:

$$T = \rho c R_P^3 C_L \omega^2 \frac{\alpha}{3} \tag{4.1}$$

This equation shows that the thrust force depends on the angular speed of the propeller ω and the propeller pitch angle α, for experimental tests the rest of the equation was constants, which are assumed as **Ct**, so the thrust force is [5]:

$$T = C_t \omega^2 \alpha \tag{4.2}$$

Two different methods in order to determine this coefficient, first method were calculated by Microsoft Excel, which had shown to have a high error and did not give an accurate solution. The second method was performed by using the least square method see Appendix (B.1). Therefore, the Matlab code could take all the required data with ω in rad/s, α in rad and thrust force in (N) and after that compute the coefficient **Ct**, see Appendix (B.5). This method gave better accuracy and more reliable value.

Figure (4-8) shows the fitted experimental data with utilization of equation (4.2) versus the pitch angle α and the angular propeller speed ω.

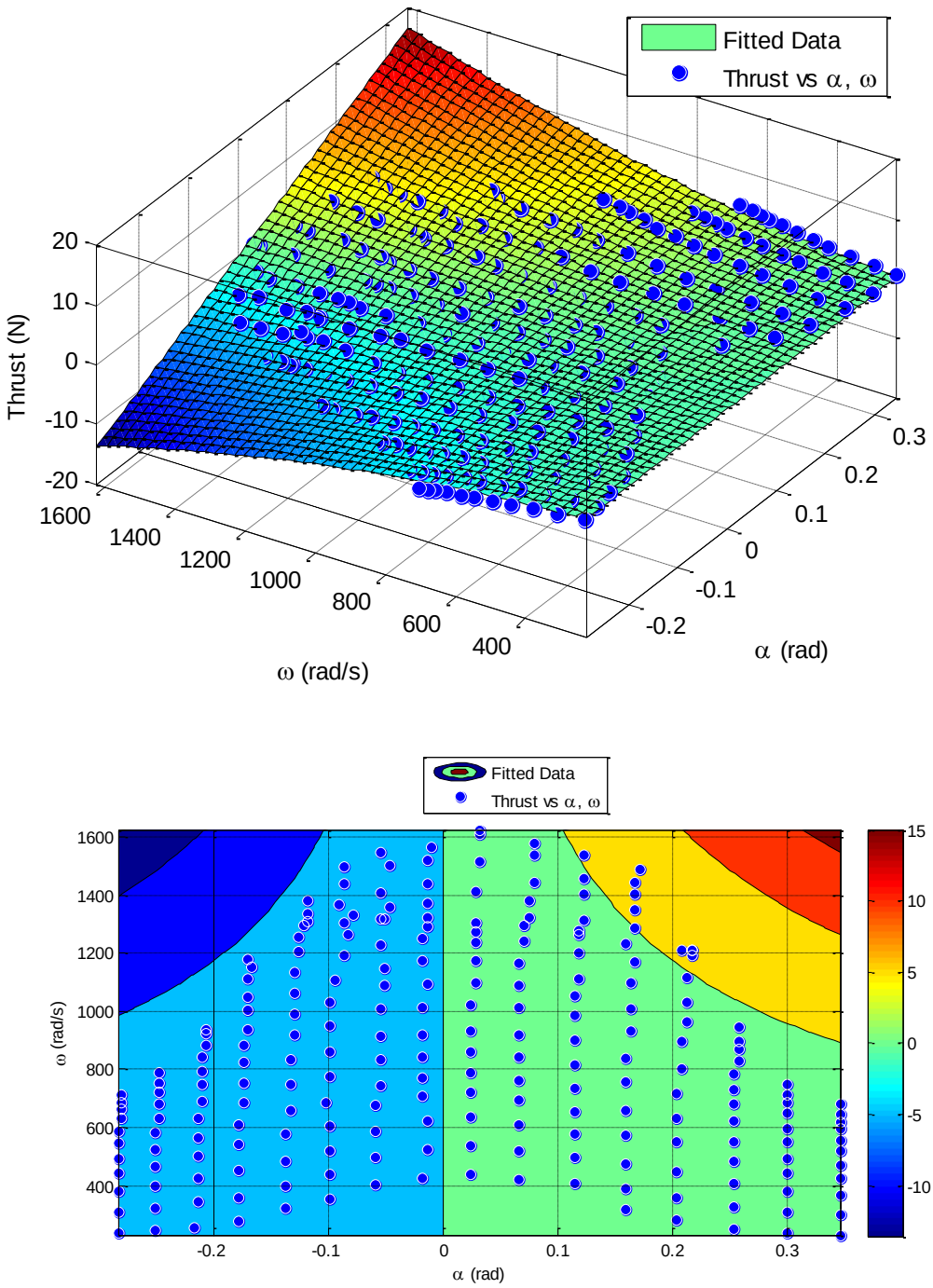


Figure (4-8): Thrust versus α, ω

The estimated **Ct** coefficient value was equal to:

$$C_t = 1.8096e - 05 \quad (4.3)$$

4.4. Torque Test

The target of the torque test is to define three different coefficients **ab1**, **ab2**, and **ab3** that were produced by the motor-propeller combination. These values were obtained by analyzing experimental data. See Appendix (B.6) which shows sample of torque data. The configuration of the torque test is different from the thrust test that the motor-propeller combination was perpendicular to the direction of motion as in Figure (4-9).

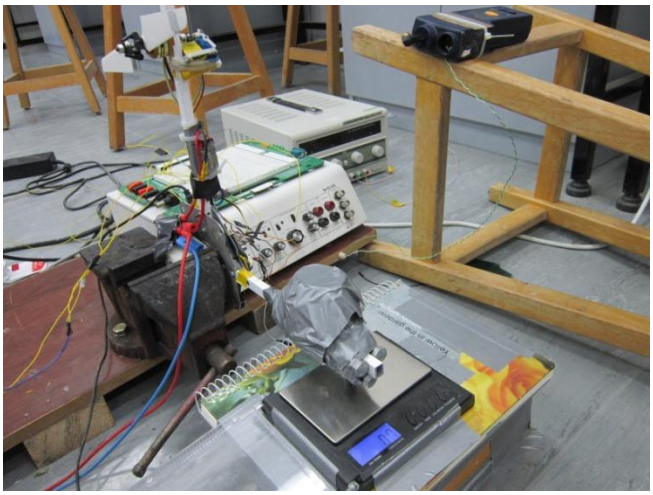


Figure (4-9): Setup for Torque Test

The way of mounting the torque test will let the produced torque by the motor-propeller combination derived to a moment at the beam that is mounted on the scale. By this configuration the scale will measure the weight of the moment in grams, and the value is multiplied by the gravity acceleration and the distance between the square base and the scale. As a result the unit of the torque will be in (N.m).

4.4.1. Torque Coefficient

The torque coefficients **ab1**, **ab2** and **ab3**, also called drag coefficients are important to derive a mathematical model Chapter (3), in a variable pitch quadrotor, the torque around the rotor shaft is caused by the aerodynamics forces acting in the blade elements, it can be descried by [4]:

$$Q = \rho c R_P^4 \omega^2 \left(\frac{C_{D0} + C_{Di}\alpha^2}{4} - \frac{C_L \alpha \omega}{3R_P} \right) \tag{4.4}$$

The equation (4.4) shows that the torque for a quadrotor depends on the angular speed ω and the pitch angle α. In the experimental tests, the remaining constants were assumed as **ab1**, **ab2** and **ab3**, and the torque equation is [5]:

$$Q = ab1\ \omega^2 + ab2\ \omega^2\alpha^2 + ab3\ \omega\alpha \tag{4.5}$$

The least square method Appendix (B.1) is used to estimate these three coefficients. This is the only numerical way to find these coefficients due to the fact that the equation has three different unknowns with one equation available. Thus, the Matlab code Appendix (B.6) will take all the required data with ω in (rad/s), α in (rad) and the produced torque in (N.m). With this method the estimated values have a high accuracy and a better view for the errors that may occur by using it.

The results of the torque coefficients are:

$$\begin{cases} ab1 = 1.3559e-08 \\ ab2 = 1.9398e-06 \\ ab3 = 1.5487e-04 \end{cases} \tag{4.6}$$

Figure (4-10) shows the fitted experimental data utilizing equation (4.5) versus the pitch angle α and the angular propeller speed ω.

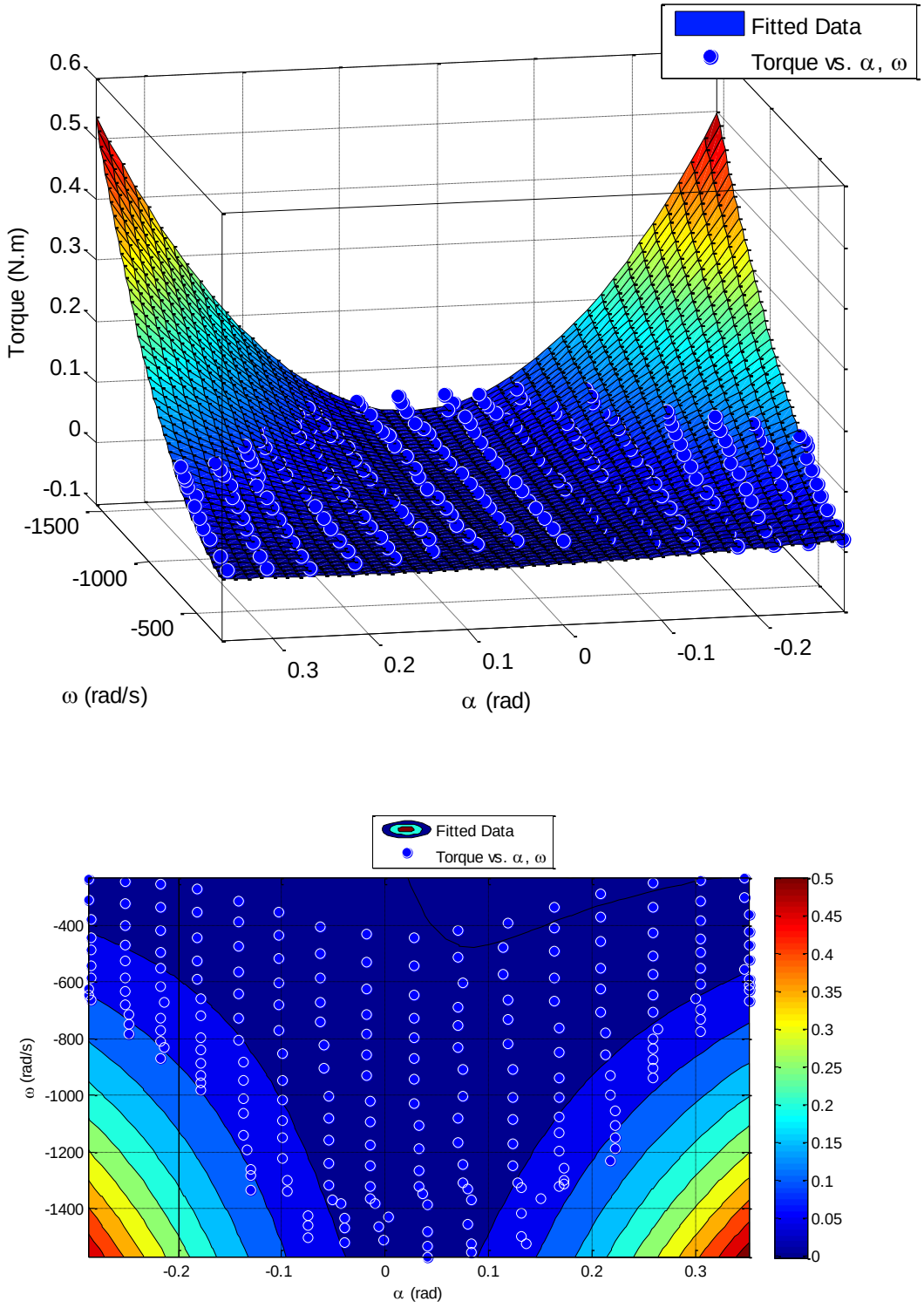


Figure (4-10): Torque versus ω, α

Chapter 5: Dynamic Test

The static test results are insufficient to describe or model the Quadrotor system, as discussed previously, all static test measurements were conducted in the steady state region of the system ignoring the system real behavior or its response.

In a control problem, system transfer function and output are known, but the input is to be determined, in a simulation problem, system transfer function and input are known, but this time the output is to be determined, on the other hand, in a system identification problem both system inputs and outputs are known but its transfer function is not, so the dynamic test was conducted to get readings for system outputs that are associated with known inputs in order to identify the whole system and find its transfer functions, identifying a completely unknown system is called black box system identification.

A variable pitch quadrotor has eight actuators; four identical brushless DC motors, and four identical servo motors, so only two transfer functions are needed to model the system (servo transfer function and DC motor transfer function).

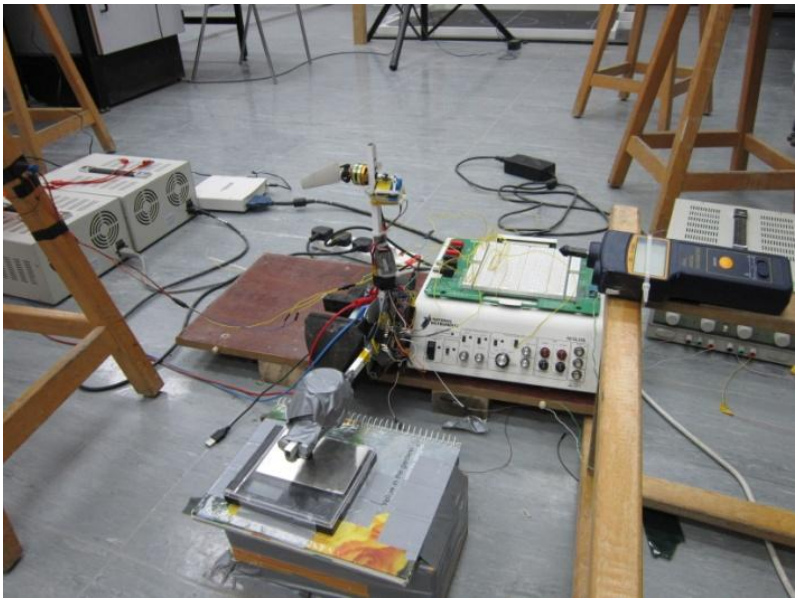


Figure (5-1): Dynamic test setup

5.1. Methodology

The method used to identify the system is done by applying different sine wave signals with different frequencies on the actuators as inputs, and measuring the corresponding outputs, then after processing these row time domain data, they are entered to the System Identification Matlab Tool Box. In order to obtain transfer functions, phase shift can be calculated by comparing two corresponding normalized input and output signals. Bearing in mind the frequency of the sine wave for the servo motor is expected to be more than that of the brushless DC motor.

Note: Dynamic test was not finished and transfer functions were not obtained.

5.2. Dynamic test setup, software and hardware

Considering only one arm of the Quadrotor, the system has two input signals, one for the DC motor and the other is for the servo motor. The source of these signals is the PWM generator that follows MAX32^{TM} orders, in order to acquire these signals without using junctions which might disturb them; extra channels from the PWM generator were used to provide additional signals that are parallel to the input signals. The same applies for the two output signals (i.e. DC motor and servo motor output signals).

5.2.1. Data acquisition using NI LabVIEW

To get all system signals a four-channel oscilloscope is needed, a more efficient way is by using the NI ELVIS platform, this platform integrates many commonly used laboratory instruments such as oscilloscopes, it only needs to be connected to a computer that has LabVIEW software installed on. The computer receives data from this platform and process the data according to a graphical LabVIEW code, Figure (5-3) shows the code used for data acquisition.

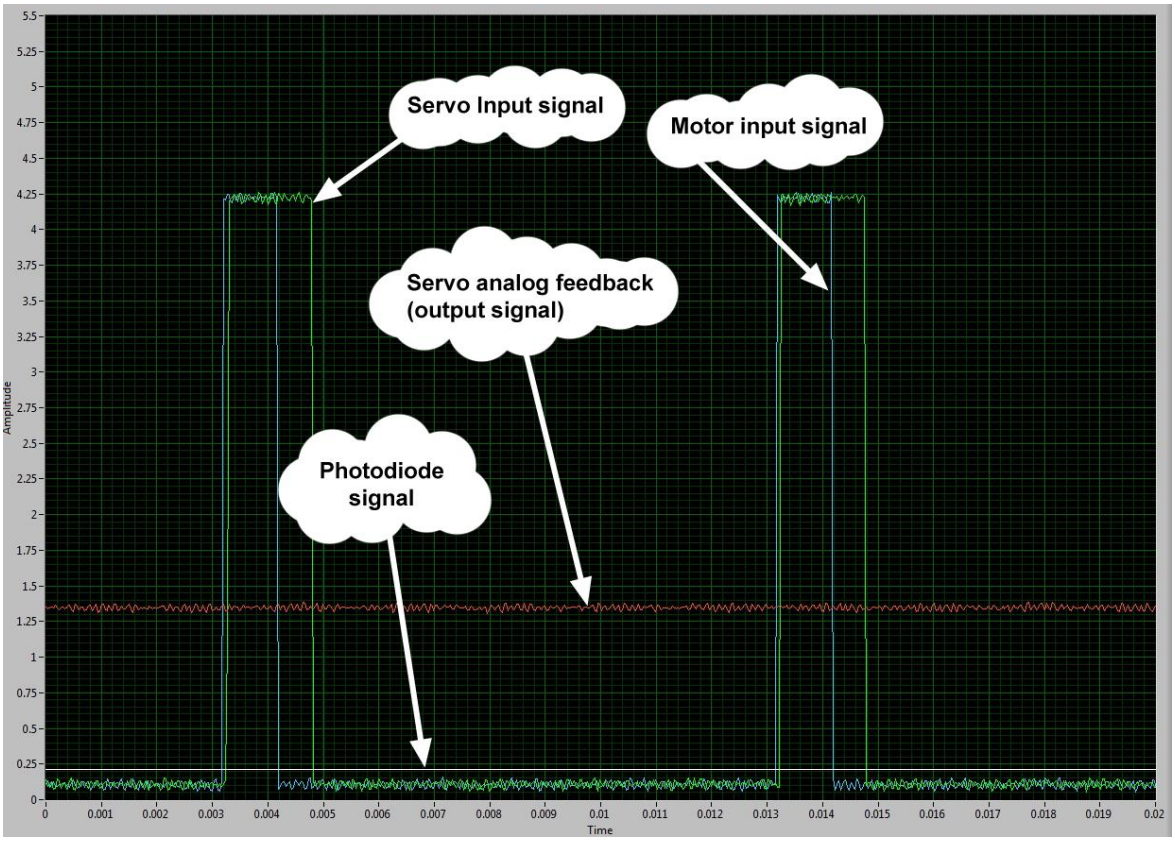


Figure (5-2): A screen shot of running LabVIEW code

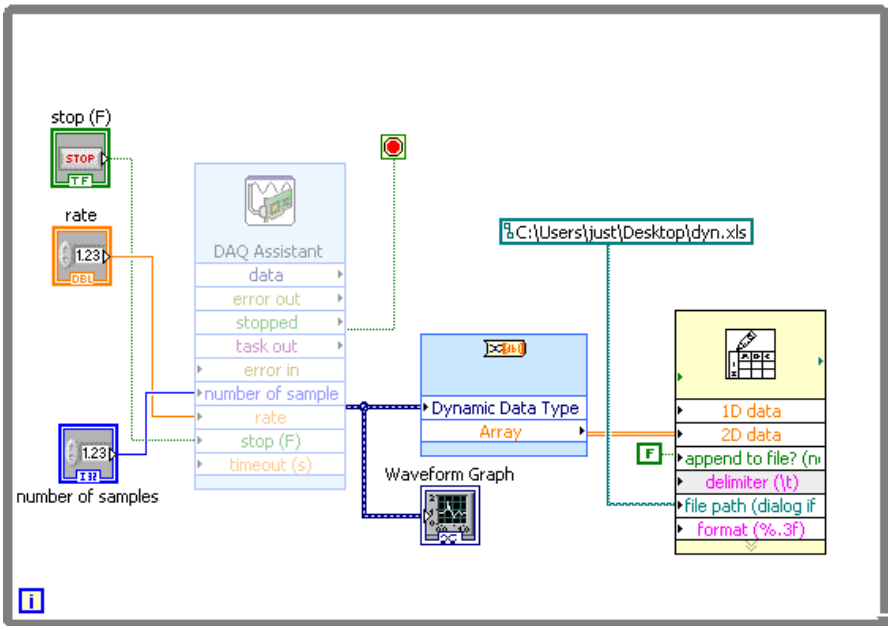


Figure (5-3): LabVIEW Graphical Code

Figure (5-4): NI-ELIVS Platform

A LabVIEW code consists of two parts, the first part is the code itself, consists of tool boxes that severe as functions, the second part is the GUI that is associated with the code, this a very helpful feature in LabVIEW, visualization of real time data is needed to fix errors that might occur before processing the data.

5.2.2. Inputs (Brushless DC and Servo Motors)

The DC and servo motor were actuated using PWM signals (from the PWM generator in Section (2.3)) with their width varying as a sine wave that has a certain frequency, these signals were measured by two channels of the NI ELIVS platform that are connected to the PWM generator. The specified signals frequency and amplitude are defined by the microcontroller code Appendix (E.1). The range of the amplitude for servo motor is between 8 to 11 degree and for the brushless motor between 1300µs to 1700µs where the variable pith Quadrotor will operate within these ranges. This code sends the DC motor signal to two channels of the PWM generator, one for the DC motor and the other for measuring the signal (the parallel one), the same thing for the servo. By this configuration it can be ensured that the process of measuring the signal will not disturb the signal that is going to the DC motor and to the servo, as discussed previously.

To produce a sine wave of the duty cycle (or the width) which will be useful when it is utilized to identify the system, the PWM signal must be processed and then curve fitted by a Matlab code. Figure (5-5) shows the fitted curve with the PWM sine wave of the duty cycle data for the DC motor with arbitrary data Appendix (E.2), and Figure (5-6) shows the fitted curve with the PWM sine wave of the duty cycle data for the servo with arbitrary data Appendix (E.3).

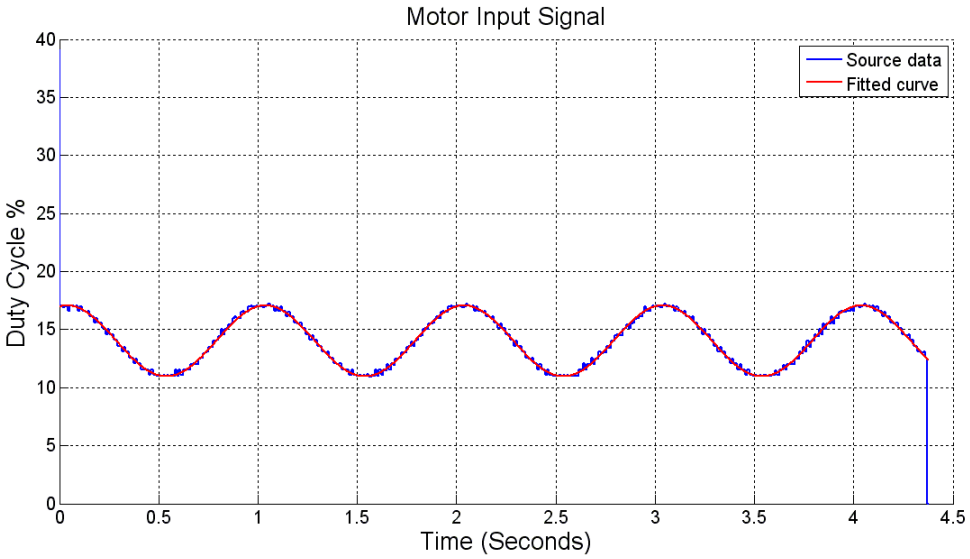


Figure (5-5): Motor Input Signal

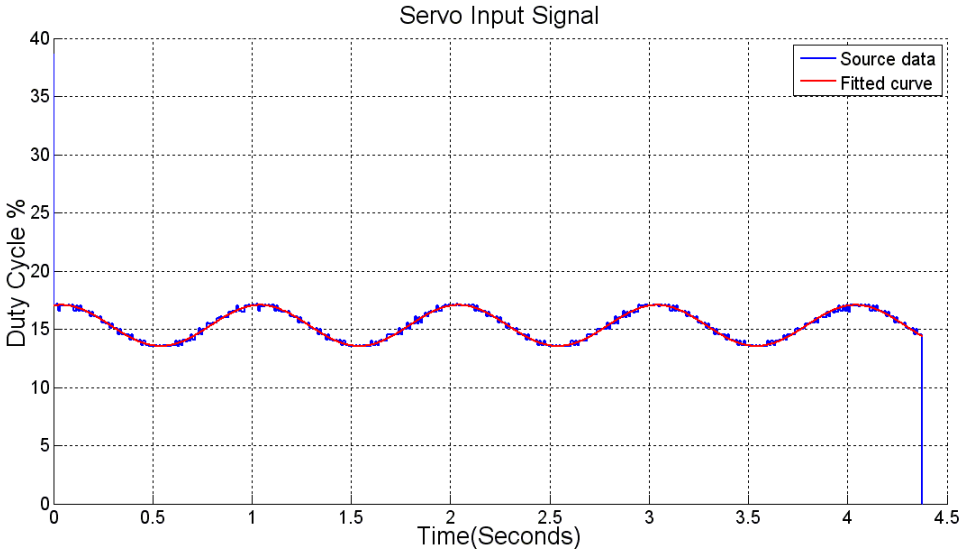


Figure (5-6): Servo Input Signal

5.2.3. Brushless DC Motor Output Signal

This signal was obtained using a laser pointer and a photodiode that are aligned together, when laser light beam is pointing at the photodiode the output voltage is nearly zero volt, when the light beam is cut by the blade, the output jumps to Vin, which is the photodiode input voltage (usually Vin=5 V, so the output in that case is 5 V), so photodiode output can be in one of two states and thus the output signal obtained is a square wave with its width related to the speed of the motor.

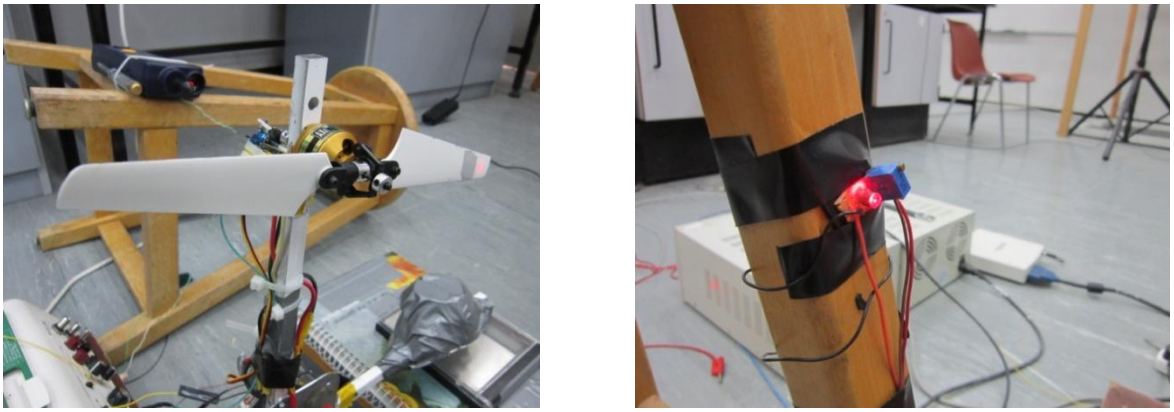


Figure (5-7): Photodiode-laser pointer alignment

Row output signal must be processed in order they become useful (sinusoidal trend), since the row data are a square wave with its duty cycles related to the speed. A listing code in Appendix (E.4) converts the PWM (square) signal into the speed as a function of time, and since the input is a sine wave the converted data will have a sinusoidal trend.

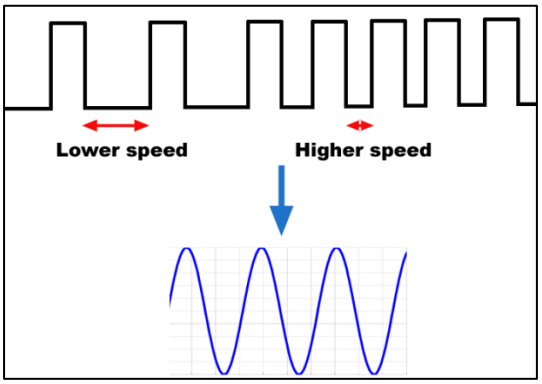


Figure (5-8): Data Processing

After the conversion of the square wave, a sine wave curve was fitted to get its function and its parameters. The following Figure (5-9) shows the source data (after conversion) and the fitted curve.

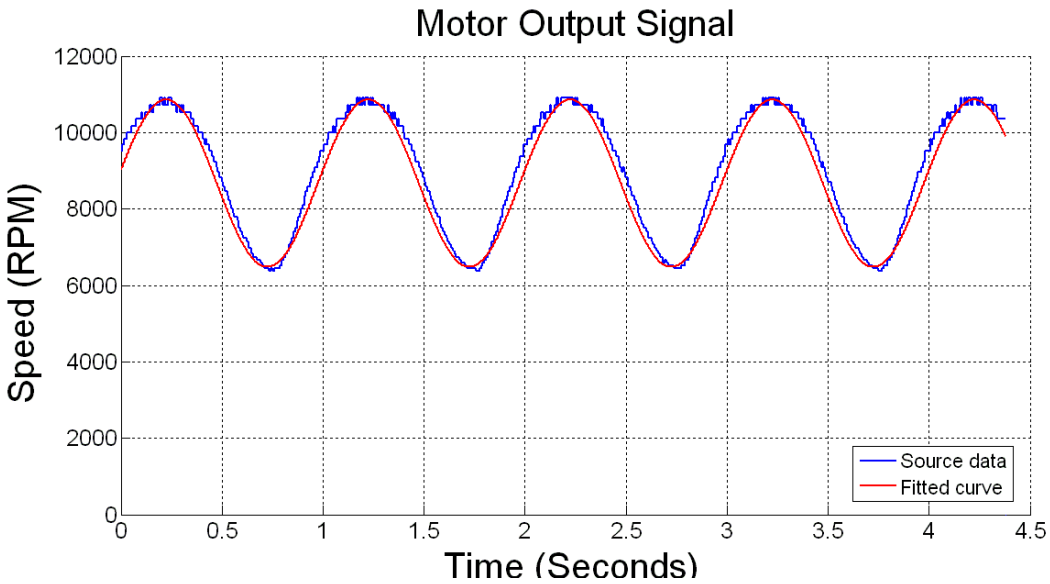


Figure (5-9): Motor output signal processing

5.2.4. Servo Motor Output Signal

The servo output signal is measured from the feedback signal of the potentiometer in the servo; this signal indicates the instantaneous pitch angle by an analog voltage. It was noticed that when the feedback signal of the potentiometer is connected to the NI ELVIS platform, the signal becomes disturbed and this had a small effect on the servo operation which can be neglected. The measured data does not need to be processed, but it must be curve fitted, Appendix (E.5) shows how an arbitrary data can be curved. This curve fit will obtain the sine wave voltage signal at the concentrated points of data; by this the disturbance is terminated as in Figure (5-10).

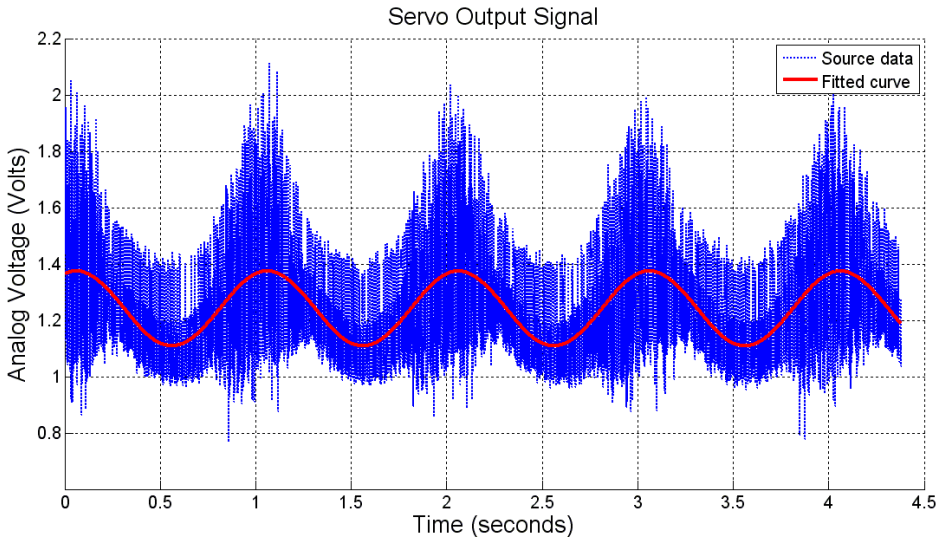


Figure (5-10): Servo Output Signal

5.3. Data Observation

Though the project team ran out of time before finding the DC motor and the servo motor transfer functions, the preliminary analysis of the obtained data from the dynamic test describe the actuators behavior by observing phase shifts between input and output signals for both motors. The following results were obtained at a 30 KHz sampling rate, and all sine waves were normalized. As predicted, as a higher frequency is given to the actuators, larger phase shifts are noticed. Moreover, servos have faster response than DC motors.

5.3.1. Both DC and servo motors frequencies = 1 Hz

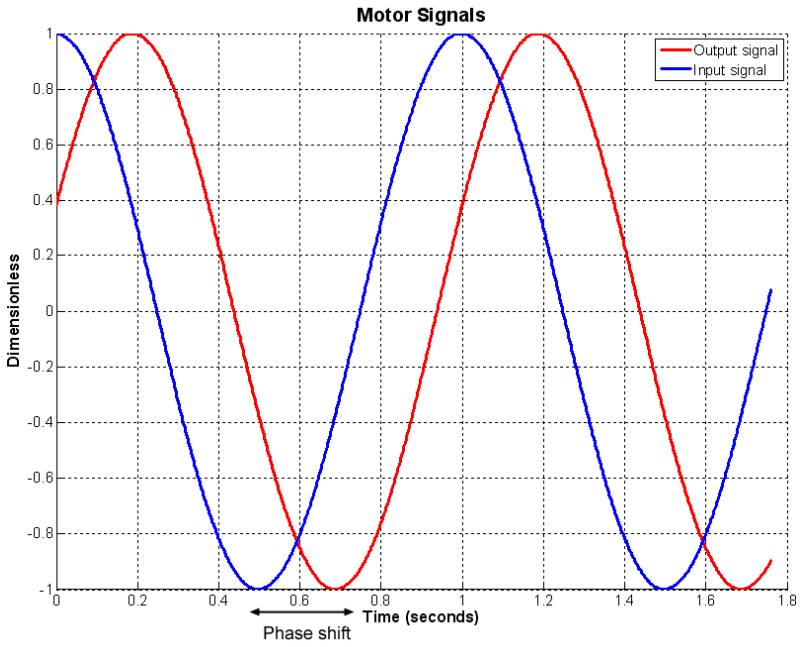


Figure (5-11): Normalized motor IO signals with frequency of 1 Hz

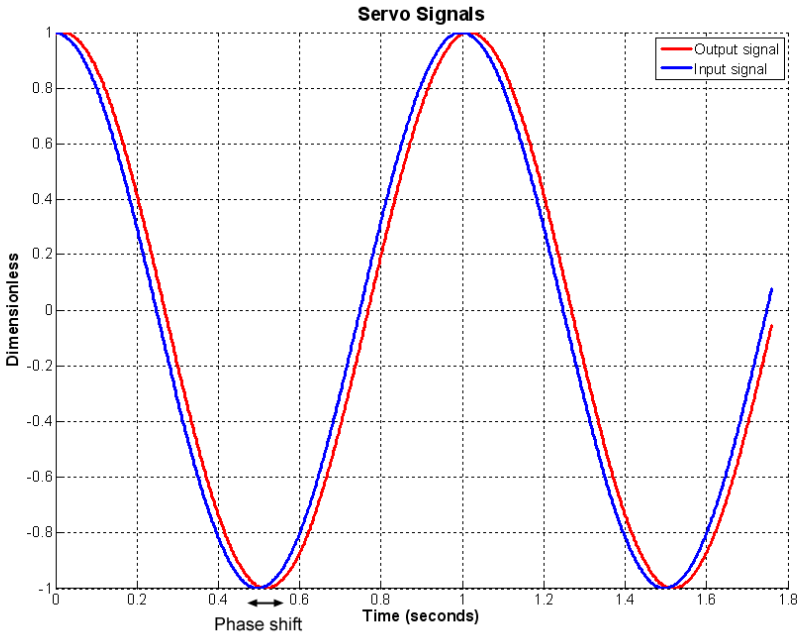


Figure (5-12): Normalized servo IO signals with frequency of 1 Hz

5.3.2. Both DC and servo motors frequencies = 2 Hz

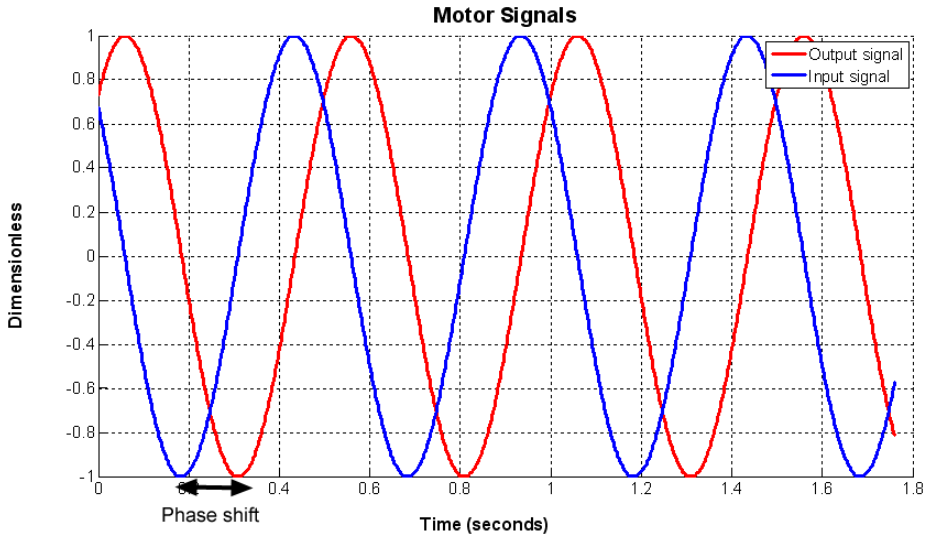


Figure (5-13): Normalized motor IO signals with frequency of 2 Hz

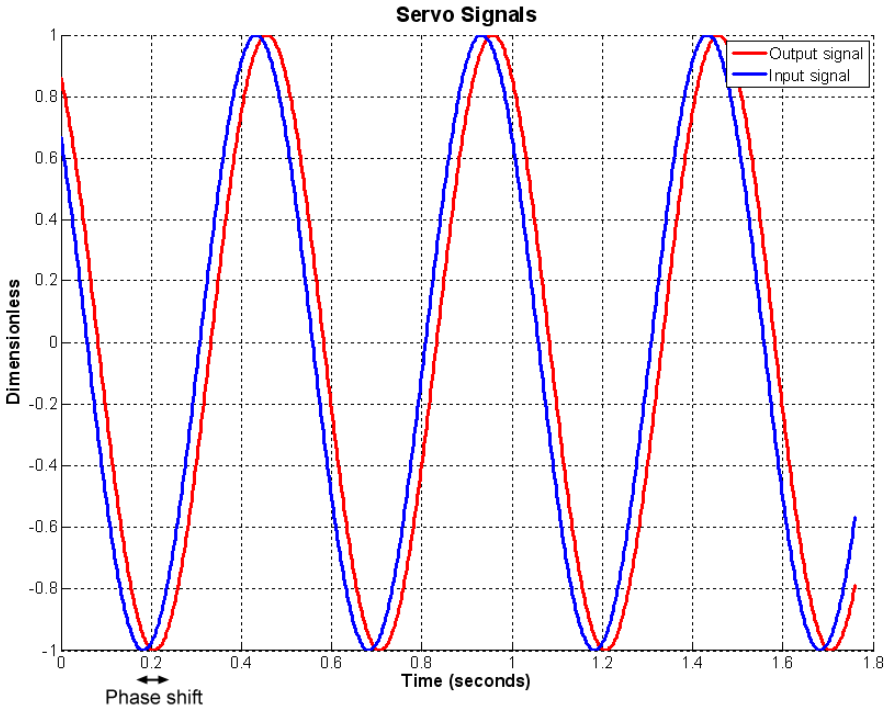


Figure (5-14): Normalized servo IO signals with frequency of 2 Hz

Appendices

Appendix A: Hardware Programming Codes

A.1: Xsens Code

```
/*
This code was written by:  Yazan Alali & Hassan Omari.
August 2013
Jordan
>>it was written for the Xsense MTi-G-700 IMU
>>tested on Arduino Mega and ChipKIT MAX32 board
>>it recieves a standarad length MTData message and interprets it to get
the useful values
>>this code is valid for most of the MT configurations, by only changing
the parameters of the config array
*/

// this is the configuration array, 0 >> disabled     1>>  enabled

//   temp   gps    calib.  euler   aux  pos   vel  ampler counter
byte config[9]={0  ,  0   ,  0  ,  1  ,  0  ,  0  ,  0 ,  0 ,  0  };

//Variables declarations
int w,n=0;
byte data[255];
byte temp[4];
byte gps[44];
byte accx[4];byte accy[4];byte accz[4];
byte gyrx[4];byte gyry[4];byte gyrz[4];
byte magx[4];byte magy[4];byte magz[4];
byte roll[4];byte pitch[4];byte yaw[4];// orientation Euler
byte auxiliary[4];
byte lat[4];byte lon[4];byte alt[4];
byte velx[4];byte vely[4];byte velz[4];
byte stat[1];
byte sc[2];
byte prea,bid,mid,length,cs;
byte mask=0xFF;byte datsum=0;byte sum=0;byte check;

float rollf,pitchf,yawf,gyrxf,gyryf,gyrzf,tempf,accxf,accyf,acczf,velxf,
velyf,velzf;
```

```
void setup() {
        Serial3.begin(115200); // RX3 of the arduino mega is connected to
TX3 //of the MT
         Serial.begin(115200); // for arduino-PC communication
       }

void loop() {
  //the following lines receives and checks the XBus header bytes
  w=0;n=0;

//preamble byte
h0:while(!(Serial3.available()>0)){}
     prea=Serial3.read();
 if (prea!=(0xFA)){goto h0;}

//Bus identifier byte
h1:while(!(Serial3.available()>0)){}
     bid=Serial3.read();
 if (bid!=(0xFF)){goto h0;}

//Message identifier byte
h2:while(!(Serial3.available()>0)){}
     mid=Serial3.read();
 if (mid!=(0x32)){ goto h0;}

//data length byte
while(!(Serial3.available()>0)){}
     length=Serial3.read();

//data bytes
for(int i=0;i<=length-1;i++){
while(!(Serial3.available()>0)){}
     data[i]=Serial3.read(); }


//checksum byte
 while(!(Serial3.available()>0)){}
     cs=Serial3.read();


//check if there is an error in the message
for(int i=0;i<=(length-1);i++){datsum+=data[i];}
sum=cs+bid+mid+length+datsum;
check=sum&mask;
//uncomment the next line to enable message checking
//if(check!=0){Serial.println("message error"); goto h0;}
```

```
//assigning each data output bytes the their corresponding varaible array

if(config[0]!=0){w=0;while(w<=3){temp[w]=data[n];w++;n++;}}

if(config[1]!=0){w=0;while(w<=43){gps[w]=data[n];w++;n++;}}

if(config[2]!=0){w=0;while(w<=3){
accx[w]=data[n];accy[w]=data[n+4];accz[w]=data[n+8];
gyrx[w]=data[n+12];gyry[w]=data[n+16];gyrz[w]=data[n+20];
magx[w]=data[n+24];magy[w]=data[n+28];magz[w]=data[n+32];
w++;n++;}n+=32;}

if(config[3]!=0){w=0;while(w<=3){
roll[w]=data[n];pitch[w]=data[n+4];yaw[w]=data[n+8];
w++;n++;}n+=8;}

if(config[4]!=0){w=0;while(w<=3){auxiliary[w]=data[n];w++;n++;}}


if(config[5]!=0){w=0;while(w<=3){
lat[w]=data[n];lon[w]=data[n+4];alt[w]=data[n+8];
w++;n++;}n+=8;}

if(config[6]!=0){w=0;while(w<=3){
velx[w]=data[n];vely[w]=data[n+4];velz[w]=data[n+8];
w++;n++;}n+=8;}

if(config[7]!=0){stat[0]=data[n];n++;}


if(config[8]!=0){w=0;while(w<=1){sc[w]=data[n];w++;n++;}}
//end of assigning


//calling the converter function to convert the desired 4 bytes into equi
vale//nt floating point value
//uncomment the desired value (the value you want to be printed on the se
rial //monitor

//tempf=convert(temp[0],temp[1],temp[2],temp[3]);
rollf=convert(roll[0],roll[1],roll[2],roll[3]);
//pitchf=convert(pitch[0],pitch[1],pitch[2],pitch[3]);
//yawf=convert(yaw[0],yaw[1],yaw[2],yaw[3]);
//gyrxf=convert(gyrx[0],gyrx[1],gyrx[2],gyrx[3]);
//gyryf=convert(gyry[0],gyry[1],gyry[2],gyry[3]);
//gyrzf=convert(gyrz[0],gyrz[1],gyrz[2],gyrz[3]);
//accxf=convert(accx[0],accx[1],accx[2],accx[3]);
//accyf=convert(accy[0],accy[1],accy[2],accy[3]);
//acczf=convert(accz[0],accz[1],accz[2],accz[3]);
Serial.println(rollf);}
```

```
//--------Converter Function-----------------------------------------------
------------------
//it was commented and explained in appendix A.5

/*
this function recieves 4 bytes (i.e b0(MSbyte) b1 b2  b3(LSbyte) and conv
erts them into 4 byte float variable (IEEE floating point standard)
*/
float convert(byte b0,byte b1,byte b2,byte b3){
unsigned long h;
h=(b0*pow(16,6))+(b1*pow(16,4))+( b2*pow(16,2));
h+=(b3);

unsigned long p31=2147483648; unsigned long p23=8388608; float
pf23=8388608.0;
long g1,g2; float value; unsigned long g3;

 g1=pow(-1,(h/p31));
 g2=((h%p31)/p23)-127.0;
 g3=h%p23;

 value=g1*pow(2,g2)*(1.0+(g3/pf23));
 return value;}
```

A.2: PWM Generator Code

```
int i;

void setup(){
Serial.begin(115200);  //Initialize Baud rate
Serial1.begin(115200);
}

void loop(){             //This code is for testing only
for(i=1150;i<1250;i++){
sendpwm(i,4);            //Go to Function (sendpwm)
delay(5);
 }

for(i=1250;i>1150;i--){
sendpwm(i,4);
delay(5);
 }
}

void sendpwm(int pwm,byte channel){  //Function Serial communication
  byte low,high;          //Define low and high bytes for the protocol
  low=(pwm*4)%128;        //Low equals to remainder of the value
  high=(pwm*4)/128;       //High equals to division of the value
  Serial1.write(0x84);    //Send the address of the protocol
  Serial1.write(channel); //Send the number of the pin
  Serial1.write(low);     //Send Low byte
  Serial1.write(high);    //Send High byte
}
```

A.3: Low Pass Filter Code

```
int x;

float y;


void setup(){
  //using lower voltage reference than the default one to increase
accuracy, //and since the output is impossible to reach the 5 volt

analogReference(EXTERNAL);

Serial.begin(115200);

y=analogRead(A0);

}



void loop(){

//a software low pass filter to enhance the hardware one

y=0.9*y+0.1*analogRead(A0);

// normalize the values so that they range from 0 ---> 100

x=map(y,55,92,0,100);

  Serial.print(x); Serial.print("---"); Serial.println(y);
  delay(10);
}
```

A.4: Receiver Code

```
unsigned int Buffer[14]={0};       //Define array called (Buffer) for 14
terms
unsigned int add1,add2;
unsigned int i;
int temp1,temp2,temp3,temp4;
void setup() {
  Serial3.begin(115200);   //Initialize   the   baud   rate   for   serial
communication
}
void loop() {
h0:while(!(Serial3.available()>0)){}  //Checking  the  first  byte  of  the
message
     add1=Serial3.read();
 if (add1!=(0x03)){goto h0;} //first byte must equal 03 HEX or return to
(h0)

   while(!(Serial3.available()>0)){}   //Checking   second   byte   of   the
message
     add2=Serial3.read();
 if (add2!=(0x01)){goto h0;} //second byte equal to 01 HEX or return to
(h0)

if(Serial3.available()>0){  //After  confirming  right  address  of  the
message
for(i=0; i<14; i++){
Buffer[i]=Serial3.read();           //Saves all bytes in Buffer array
}}
//Throttle
 temp1=word (Buffer[8],Buffer[9]);        //Saves bytes Buffer[8]-[9] in
temp1
if(temp1>=170 && temp1<=854){
if(temp1>854){temp1=854;}               //To limit values between min and
max
if(temp1<170){temp1=170;}              //and cut off all the errors
temp1=map(temp1,170,854,1000,2000);       //Mapping values between 1000-
2000µs
}
//Roll
 temp2=word (Buffer[0],Buffer[1]);        //Saves bytes Buffer[0]-[1] in
temp2
if(temp2>=1194 && temp2<=1880){
if(temp2>1880){temp2=1880;}               //limit values between min and
max
if(temp2<1194){temp2=1194;}             //and cut off all the errors
temp2=map(temp2,1194,1880,1000,2000);     //Mapping  values  between  1000-
2000
}
//Pitch
 temp3=word (Buffer[4],Buffer[5]);        //Saves bytes Buffer[4]-[5] in
temp3
if(temp3>=2221 && temp3<=2902){
if(temp3>2902){temp3=2902;}               //limit values between min and
max
if(temp3<2221){temp3=2221;}             //and cut off all the errors
```

```
temp3=map(temp3,2221,2902,1000,2000);     //Mapping values between 1000-
2000
}
//Yaw
 temp4=word (Buffer[6],Buffer[7]);        //Saves bytes Buffer[6]-[7] in
temp4
if(temp4>=3241 && temp4<=3924){
if(temp4>3924){temp4=3924;}              //To limit values between min and
max
if(temp4<3241){temp4=3241;}             //and cut off all the errors
temp4=map(temp4,3241,3924,1000,2000);     //Mapping values between 1000-
2000
}}
```

A.5: IEEE 754 to Floating Point converter function

```
/*
This function recieves 4 bytes (i.e b0(MSbyte) b1 b2  b3(LSbyte)(IEEE flo
ating point standard) and converts them into a 4 bytes float variable,
and returns it.
*/


float convert(byte b0,byte b1,byte b2,byte b3){
unsigned long h;

//collect single bytes into one 4 bytes variable (according to their weig
hts)
h=(b0*pow(16,6))+(b1*pow(16,4))+( b2*pow(16,2));
h+=(b3);

//p## is used to get the bytes of a certain group
unsigned long p31=2147483648; unsigned long p23=8388608;
float pf23=8388608.0;

//g# is group#
long g1,g2; float value; unsigned long g3;


 g1=pow(-1,(h/p31));               //h/p31 = bit31 (the sign bit)
 g2=((h%p31)/p23)-127.0;           //(h%p31)/p23  will  get  the  exponent,
then the bias (127) is subtracted
 g3=h%p23;                         //group 3 bits

 value=g1*pow(2,g2)*(1.0+(g3/pf23));
 return value;}
```

Appendix B: Static Test

B.1: Least Square Method (General Linear Regression)

Where substantial error is associated with data, the best curve fitting strategy is to derive an approximating function that fits the shape or general trend of the data without necessarily matching the individual points. A useful extension of linear regression which is the general least square method [1] is the case where y is a linear function of two or more independent variable, as in:

$$y = a_1 x_1 + a_2 x_2 + \cdots + a_n x_n$$

One approach to do this and to get the best values of the coefficients is to minimize the sum of the residual errors for all the available data, as in:

$$S_r = \sum_{i=1}^{m} \left(y_i - a_1 x_{1,i} - a_2 x_{2,i} - \cdots - a_n x_{n,i}\right)^2 \qquad m: number\ of\ data$$

Then differentiate with respect to each of the unknown coefficients:

$$\frac{\partial S_r}{\partial a_1} = -2 \sum_{i=1}^{m} x_{1,i} \left(y_i - a_1 x_{1,i} - a_2 x_{2,i} - \cdots - a_n x_{n,i}\right)$$

$$\frac{\partial S_r}{\partial a_2} = -2 \sum_{i=1}^{m} x_{2,i} \left(y_i - a_1 x_{1,i} - a_2 x_{2,i} - \cdots - a_n x_{n,i}\right)$$

$$\vdots$$

$$\frac{\partial S_r}{\partial a_n} = -2 \sum_{i=1}^{m} x_{n,i} \left(y_i - a_1 x_{1,i} - a_2 x_{2,i} - \cdots - a_n x_{n,i}\right)$$

The coefficients yielding the minimum sum of the square of the residuals are obtained by setting the partial derivative equal to zero and expressing the result in matrix form as:

$$\begin{bmatrix} \sum_{i=1}^{m} a_1 x_{1,i}^2 & \sum_{i=1}^{m} a_2 x_{2,i} x_{1,i} & \cdots & \sum_{i=1}^{m} a_n x_{n,i} x_{1,i} \\ \sum_{i=1}^{m} a_1 x_{1,i} x_{2,i} & \sum_{i=1}^{m} a_2 x_{2,i}^2 & \cdots & \sum_{i=1}^{m} a_n x_{n,i} x_{2,i} \\ \vdots & \vdots & & \vdots \\ \sum_{i=1}^{m} a_1 x_{1,i} x_{n,i} & \sum_{i=1}^{m} a_2 x_{2,i} x_{n,i} & \cdots & \sum_{i=1}^{m} a_n x_{n,i}^2 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{m} y_i x_{1,i} \\ \sum_{i=1}^{m} y_i x_{2,i} \\ \vdots \\ \sum_{i=1}^{m} y_i x_{n,i} \end{bmatrix}$$

The coefficient of determination can be defined as:

$$r^2 = \frac{S_t - S_r}{S_t} \qquad where\ S_t = \sum_{i=1}^{m} (y_i - \bar{y})^2$$

B.2: Servo Test

```
unsigned long x1;
int x2=1000;

void setup() {
  pinMode(A0,INPUT);       //Initialize Analog pin A0
  Serial.begin(115200);    //Initialize baud rate for Serial
  Serial2.begin(115200);
}

void loop() {
  sendpwm(x2,4);               //Start with zero Degree for Servo
   for (int i=0;i<3000;i++){
     x1=analogRead(A0);        //Save the Analog value in x1
   Serial.print("servo1:  ");
   Serial.print(x1);           //Print value of x1 on the monitor
   Serial.print("       ");
   Serial.println(x2);         //Print value of x2 on the monitor
   }
  delay(2500);                 //2.5 second to record the value
  x2=x2+50;                    //Go to next step
}

void sendpwm(int pwm,byte servo){  //PWM generator Serial code
  byte low,high;
  low=(pwm*4)%128;
  high=(pwm*4)/128;
  Serial2.write(0x84);
  Serial2.write(servo);
  Serial2.write(low);
  Serial2.write(high);
}
```

B.3: Angle of Attack test

Arduino Code

```
#include <math.h>

#define zpin A5                 //zpin of accelerometer is connected to pin
A5
#define xpin A2                 //xpin of accelerometer is connected to pin
A2
#define ypin A3                 //ypin of accelerometer is connected to pin
A3

float z,y,x,B;
float offsetz=512.0;            // z-axis offset

float offsetxy=497.0;           // y-axis and x-axis offset

float in=-2.6;                  // initial angle (angle between
accelerometer
                                 //and blade due to mounting error)

float th;
float factor=57.29577951;         //factor converts from radians to
degrees
float g=102.0;                    //sensitivity
int i;
void setup(){
analogReference(EXTERNAL);
Serial.begin(115200);


pinMode(zpin,INPUT);
pinMode(xpin,INPUT);
pinMode(ypin,INPUT);

}

void loop(){
z=0.99*z+0.01*((analogRead(zpin)-offsetz)/g);   //low pass filter on z-
axis
//
isolate readings from nosie due to small vibrations that can't be avoided

y=analogRead(ypin)-offsetxy;
x=analogRead(xpin)-offsetxy;
th=(0.99*th)+0.01*(asin(z)*factor); //low pass filter on th to Stabilize
readings


i++;
if(i==300){i=0;
Serial.print(x);Serial.print("           ");Serial.print(y);Serial.print("
");Serial.println(th-in);}
}
```

MatlabCode (used to fit a curve to the data, using least square method)

```
%Torque, alpha= K1*(A^3)    +   K2*(A^2)   +   K3*(A) + K4
%A>>analog pin A0 (servo analog feedback)
clear all;
close all;
clc;

% input any analog reading to compute the corresponding angle(alpha)
test=376.12;


load('A.mat');
load('alpha.mat');
```

%% Declarations

```
B=zeros(4,1);

H(1,1)=sum(A.^6);
H(1,2)=sum(A.^5);
H(1,3)=sum(A.^4);
H(1,4)=sum(A.^3);
H(2,1)=sum(A.^5);
H(2,2)=sum(A.^4);
H(2,3)=sum(A.^3);
H(2,4)=sum(A.^2);
H(3,1)=sum(A.^4);
H(3,2)=sum(A.^3);
H(3,3)=sum(A.^2);
H(3,4)=sum(A);
H(4,1)=sum(A.^3);
H(4,2)=sum(A.^2);
H(4,3)=sum(A);
H(4,4)=1;

B(1,1)=sum((A.^3).*(alpha));
B(2,1)=sum((A.^2).*(alpha));
B(3,1)=sum((A.^1).*(alpha));
B(4,1)=sum(alpha);
```

%% Results

```
K=(H)\B;
result=(K(1,1).*A.^3)+(K(2,1).*A.^2)+(K(3,1).*A)+K(4,1);

plot(A,alpha,'*',A,result,'r')
```

B.4: Static Test Code

```
int ch,counter=0;
int y=1540 ;
int x=1000;
float A;
void setup(){
Serial.begin(115200);   //Initialize Baud Rate
Serial2.begin(115200);
A=analogRead(A0);
}
void loop(){
  A=(0.99*A)+(0.01*analogRead(A0));  //Software low pass filter
  if(Serial.available()){    //To read from the serial monitor
  ch=Serial.read();}
  if(ch=='i'){x=x+50;ch=0;}
  if(ch=='1'){x=x+100;ch=0;}
  if(ch=='2'){x=x+200;ch=0;}
  if(ch=='3'){x=x+300;ch=0;}
  if(ch=='4'){x=x+400;ch=0;}
  if(ch=='5'){x=x+500;ch=0;}
  if(ch=='6'){x=x+600;ch=0;}
  if(ch=='7'){x=x+700;ch=0;}
  if(ch=='8'){x=x+800;ch=0;}
  if(ch=='9'){x=x+900;ch=0;}
  if(ch=='d'){x=x-50;ch=0;}
  if(ch=='c'){y=y-20;ch=0;}
  if(ch=='v'){y=y+20;ch=0;}
  if(ch=='s'){x=1000;ch=0;}
  if(ch=='z'){y=y+40;ch=0;}
  if(ch=='n'){y=y-40;ch=0;}
  if(ch=='q'){y=y+1;ch=0;}
  if(ch=='a'){y=y-1;ch=0;}
  if(counter>200){counter=0;  //Display the readings of:
  Serial.print("Analog=   "); //Analog potentiometer
  Serial.print(A);
  Serial.print("                  ");
  Serial.print("Speed= ");    //Propeller speed in µs
  Serial.print(x);
  Serial.print("                  ");
  Serial.print("Servo= ");    //Servo angle in µs
  Serial.println(y);}
  sendpwm(x,4);   //Send Propeller speed to generator
  sendpwm(y,1);   //Send Servo angle to generator
 counter++;
}
void sendpwm(int pwm,byte servo){ //PWM generator code
  byte low,high;
  low=(pwm*4)%128;
  high=(pwm*4)/128;
  Serial2.write(0x84);
  Serial2.write(servo);
  Serial2.write(low);
  Serial2.write(high);
}
```

B.5: Thrust Test

Sample of Thrust Data

Motor PWM (μs)	Analog Pin	Current (A)	ω (RPM)	Weight on Scale (gram)	α (Degree)	Force (N)
1100	422	0.46	3034	41.8	9.093744	0.409918
1150	422	0.69	3743	62.7	9.093744	0.614877
1200	422	1.04	4543	91.4	9.093744	0.896328
1250	422	1.46	5492	120.3	9.093744	1.17974
1300	422	1.92	6307	148.1	9.093744	1.452365
1350	422	2.5	7217	180.1	9.093744	1.766178
1400	422	3.16	8021	214.4	9.093744	2.102546
1450	423	3.93	8901	249.3	9.35033	2.444798
1500	423	4.75	9620	279.5	9.35033	2.740959
1550	423	5.43	10474	275	9.35033	2.696829
1600	424	6.4	11163	308.2	9.607274	3.02241
1650	422	7.68	11775	296.7	9.093744	2.909633
1700	424	9.18	12272	335.6	9.607274	3.291112
1750	424	10.72	12879	389.4	9.607274	3.81871
1800	424	11.85	13382	429.7	9.607274	4.213918
1850	424	13.27	13768	447	9.607274	4.383573
1900	425	14.38	14216	501	9.864568	4.913132

Matlab Code (Thrust)

```
%%Thrust, Th= Ct*(w^2)*(a)
%a>>alpha (degrees) , analog = servo analog feedback

clear all;
close all;
clc;

load('analog.mat');
load('Th.mat');
load('RPM.mat');
load('K.mat');
load('cr');

% Equation of the pitch angle
a=zeros(size(analog,1),1);

for i=1:size(analog,1)
a(i,1)=(K(1,1)*analog(i,1)^3)+(K(2,1)*analog(i,1)^2)+(K(3,1)*analog(i,1))+K(4,1);
end

%In rad
arad=a*pi/180;

% Equation to get w (rad/s) from RPM
w=(RPM*2*pi)/60;

% Declarations
A=sum(Th.*(w.^2).*a);
B=sum((w.^4).*(a.^2));
Arad=sum(Th.*(w.^2).*arad);
Brad=sum((w.^4).*(arad.^2));

% Result

Ctrad= Arad / Brad
p=12*cr;
```

```
%%Fit: Thrust Generated by Curve Fit Tool.
[xData, yData, zData] = prepareSurfaceData(arad, w, Th );

% Set up fit type and options.
ft = fittype( 'Ct*x*(y^2)', 'independent', {'x', 'y'}, 'dependent', 'z' );
opts = fitoptions( ft );
opts.Display = 'Off';
opts.Lower = -Inf;
opts.StartPoint = 0.157613081677548;
opts.Upper = Inf;

% Fit model to data.
[fitresult, gof] = fit( [xData, yData], zData, ft, opts );

% Plot fit with data.
figure(1);

h = plot(fitresult, [xData, yData], zData );
legend(h,'Fitted Data','Thrust vs \alpha, \omega','Location','NorthEast');

% Label axes
xlabel('\alpha (rad)' );
```

```
ylabel('\omega (rad/s)' );
zlabel('Thrust (N)' );
grid on
view( -57.5, 50 );

% Make contour plot.
figure(2);

h = plot(fitresult,[xData, yData],zData,'Style','Contour');
legend(h,'Fitted Data','Thrust vs \alpha, \omega','Location','NorthEast');

% Label axes

xlabel('\alpha (rad)' );
ylabel('\omega (rad/s)' );
grid on
```

```
%%Fit: Current Generated by Curve Fit Tool.
[xData, yData, zData] = prepareSurfaceData( a, cr, RPM );

% Set up fittype and options.
ft = fittype( 'loess' );
opts = fitoptions( ft );
opts.Normalize = 'on';

% Fit model to data.
[fitresult, gof] = fit( [xData, yData], zData, ft, opts );

% Plot fit with data.
figure(3);

h = plot(fitresult, [xData, yData], zData );
legend(h,'Fitted Data','\omega vs. \alpha, current','Location','NorthEast');

% Label axes
xlabel('\alpha (Degree)' );
ylabel('current in (A)' );
zlabel('\omega (RPM)' );
grid on
view( -27.5, 26 );

% Make contour plot.
figure(4);

h = plot(fitresult, [xData, yData], zData, 'Style', 'Contour' );
legend(h,'Fitted Data','\omega vs. \alpha, current','Location','NorthEast');

% Label axes
xlabel('\alpha (Degree)' );
ylabel('current in (A)' );
grid on
```

```
%%Fit: Power Generated by Curve Fit Tool.
[xData, yData, zData] = prepareSurfaceData( a, p, Th );

% Set up fittype and options.
ft = fittype( 'loess' );
opts = fitoptions( ft );
opts.Normalize = 'on';


% Fit model to data.
[fitresult, gof] = fit( [xData, yData], zData, ft, opts );
```

```
% Plot fit with data.
figure(5);

h = plot(fitresult, [xData, yData], zData );
legend(h,'Fitted Data','Thrust vs. \alpha, Power','Location','NorthEast');

% Label axes

xlabel('\alpha (Degree)' );
ylabel('Power (Watt)' );
zlabel('Thrust (N)' );
grid on
view( -27.5, 26 );

% Make contour plot.
figure(6);

h = plot(fit result, [xData, yData], zData, 'Style', 'Contour' );
legend(h,'Fitted Data','Thrust vs. \alpha, Power','Location','NorthEast');

% Label axes

xlabel('\alpha (Degree)' );
ylabel('Power (Watt)' );
grid on
```

B.6: Torque Test

Sample of Torque Test

Motor PWM (µs)	Analog Pin	Current (A)	ω (RPM)	Weight (gram)
1100	413	0.44	3725	1.2
1150	412	0.65	4566	2.6
1200	412	0.9	5454	3.6
1250	413	1.18	6329	4.4
1300	413	1.44	6983	4.2
1350	413	1.82	7810	5.3
1400	414	2.2	8680	6.6
1450	414	2.73	9619	7.4
1500	414	3.27	10371	8.6
1550	414	3.9	11166	9.8
1600	414	4.62	11918	12.2
1650	415	5.93	12493	14.9
1700	416	7.69	12672	15.5
1750	420	9.6	13033	17.7
1800	416	10.85	13524	16.9
1850	416	11.75	14302	19.4
1900	417	12.92	14560	20.5

Matlab Code (Torque)

```
%%Torque, T= E1*(w^2) + E2*(w^2)*(a^2) + E3*(w)*(a)
```

```
clear all;
close all;
clc;

load('analog.mat');
load('Te.mat');
load('w.mat');
load('K.mat');
```

```
%%Equation of the pitch angle in degrees
```

```
g=9.81;
a=zeros(size(analog,1),1);
for i=1:size(analog,1)
a(i,1)=(K(1,1)*analog(i,1)^3)+(K(2,1)*analog(i,1)^2)+(K(3,1)*analog(i,1))+K(4,1);
end
w=-1*w;
Te=0.25*g*Te;
% a in rads
arad=a*pi/180;
```

```
%%Declarations
```

```
Arad(1,1)=sum(w.^4);
Arad(1,2)=sum((w.^4).*(arad.^2));
Arad(1,3)=sum((w.^3).*(arad));
Arad(2,1)=sum((w.^4).*(arad.^2));
Arad(2,2)=sum((w.^4).*(arad.^4));
Arad(2,3)=sum((w.^3).*(arad.^3));
Arad(3,1)=sum((w.^3).*(arad));
Arad(3,2)=sum((w.^3).*(arad.^3));
Arad(3,3)=sum((w.^2).*(arad.^2));

Brad(1,1)=sum((w.^2).*(Te));
Brad(2,1)=sum(((w.^2).*(arad.^2)).*(Te));
Brad(3,1)=sum((w.*arad).*(Te));

% Result
Erad=(Arad)\Brad
```

```
%%Fit: Torque Fit.
```

```
[xData, yData, zData] = prepareSurfaceData(arad, w, Te );

% Set up fit type and options.
ft = fittype( 'a*y^2+b*x^2*y^2+c*x*y', 'independent', {'x', 'y'}, 'dependent',
'z' );
opts = fitoptions( ft );
opts.Display = 'Off';
opts.Lower = [-Inf -Inf -Inf];
opts.StartPoint = [0.196595250431208 0.251083857976031 0.616044676146639];
opts.Upper = [InfInfInf];

% Fit model to data.
[fitresult, gof] = fit( [xData, yData], zData, ft, opts );
```

```
% Plot fit with data.
figure(1);
h = plot(fitresult, [xData, yData], zData );
legend( h, 'Fitted Data', 'Torque vs. \alpha, \omega', 'Location', 'NorthEast' );
% Label axes
xlabel('\alpha (rad)' );
ylabel('\omega (rad/s)' );
zlabel('Torque (N.m)' );
grid on
view( -186.5, 20 );

% Make contour plot.
figure(2);
h = plot(fitresult, [xData, yData], zData, 'Style', 'Contour' );
legend( h, 'Fitted Data', 'Torque vs. \alpha, \omega', 'Location', 'NorthEast' );
% Label axes
xlabel('\alpha (rad)' );
ylabel('\omega (rad/s)' );
grid on
```

Appendix C: Data types

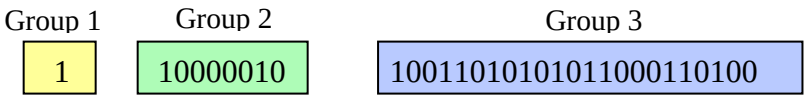
C.1: IEEE 754, Single-precision binary floating-point data format

It's a computer data format in which any value represented using it occupies 4 bytes (32 bits) in the memory, and it is used to represent signed floating point values, the conversion of this format into a floating-point decimal value can be done as explained in the following example:

Let's say we have the following value that is represented using this format:

11000001010011010101011000110100

It can be noticed it has 32 bits, the first step is to put them in 3 groups:



1- First Group: consists of 1 bit only (bit31, leftmost bit), it indicates the sign of the number, if it is 1, then the number is negative, and if 0, the number is positive. So in this case the number is negative.

   Sign = 1

2- Second Group: consists of 8 bits (bit23 to bit30), it indicates the exponent value, take it as is and convert it to the equivalent decimal value, in our case:

   $10000010_{bin} = 130_{decimal}$

   Then subtract the bias from it, the bias equals 127, since we have 8 bits value that ranges from 0 to 255, that represent an exponent value ranges from -127 to 128, so a value of 127 indicates the zero and therefore is the bias, so the exponent value in this case becomes:

Exponent= 130 – 127 = 3

3- Third Group: it consists of 23 bits (bit0 to bit 22), it indicates the fraction part, an easy way to convert this group to the equivalent decimal fraction number is to convert it to the equivalent decimal number, and then multiply the result by 2^{-23}, in our case:

$10011010101011000110100_{bin} = 5068340_{decimal}$

Fraction = 5068340×2^{-23}

= 0.6041932106

Now, the last step is to put the final expression:

Value = $(-1)^{sign} \times (1+fraction) \times 2^{exponent}$

= $-1.6041932106 \times 2^{3}$

= **-12.83354568**

Finally, an Arduino function that does the conversion was written and is listed in appendix A.5.

C.2: Pololu Maestro Data Format

A sequence of 4 bytes starts with the address byte, followed by a byte for the channel pin number and two bytes are for the data, all these bytes must be sent to the Mini Maestro 18 from the microcontroller by its program code, listed in Appendix (A.2), due to Maestro datasheet protocol [2]. The address byte is always 84 in Hex and the channel pin byte is any channel number available also in Hex. The two bytes of data contains the value of the duty cycle for the desired pin in Hex, to send these bytes it requires some of calculation, for example to make channel pin 2 at 1500µs:

$$1500 \times 4 = 6000 \quad in\ binary \quad 01011101110000$$

We can send the following byte sequence:

In Binary: 10000100, 00000010, 01110000, 00101110

In Hex: 0x84, 0x02, 0x70, 0x2E

Appendix D: Modeling

D.1: Quad Model Function (Matlab Code)

```
function [F,M]  = fcn(w,wd,al,a,ad)

m          %total mass(Kg)not determined yet
Ix    %inertia on x axis(Kg.m^2)not determined yet
Iy    %inertia on y axis(Kg.m^2)not determined yet
Iz     %inertia on z axis(Kg.m^2)not determined yet
l=0.245;       %arm length(m)
ct=1.8096e-5;    %thrust coefficient(N.s^2)
bd1=1.3559e-8;   %
bd2=1.9398e-6;  %%drag coefficients(N.m.s^2)
bd3=1.5487e-4;   %
g=9.81;        %gravity acceleration
Jr    %rotor inertia(Kg.m^2) not determined yet

w2=w.^2;
al2=al.^2;
Tl=bd1*w2 + bd2*(w2.*al2) + bd3*(w.*al);

U1=ct*(w2(1)*al(1)+w2(2)*al(2)+w2(3)*al(3)+w2(4)*al(4));
U2=ct*(w2(4)*al(4)-w2(2)*al(2));
U3=ct*(w2(3)*al(3)-w2(1)*al(1));
U4=-Tl(1)+Tl(2)-Tl(3)+Tl(4);

%---------------------------------Ta3deel--------------
Wr=w(2)+w(4)-w(1)-w(3);
Wrd=wd(2)+wd(4)-wd(1)-wd(3);

%-------------------------------------------------------
F=zeros(3,1);

ph=a(1);
th=a(2);
ps=a(3);

F(1)=(sin(ps)*sin(ph)+cos(ps)*sin(th)*cos(ph))*U1;        %Fx
F(2)=(-cos(ps)*sin(ph)+sin(ps)*sin(th)*cos(ph))*U1;       %Fy
F(3)=m*g-(cos(ps)*cos(ph))*U1;                           %Fz

%-------------------------------------------------
ph_d=ad(1);
th_d=ad(2);
ps_d=ad(3);

M=zeros(3,1);
M(1)=(th_d*ps_d*(Iy-Iz)+Jr*th_d*Wr+l*U2);         %Mx
M(2)=(ph_d*ps_d*(Iz-Ix)-Jr*ph_d*Wr+l*U3);         %My
M(3)=(th_d*ph_d*(Ix-Iy)+Jr*Wrd+U4);               %Mz
```

Appendix E: Dynamic Test

E.1: Dynamic Test Code

```
char ch='s';
unsigned long tmil,tmili;
float ts;
float pi=3.141592654;
float f=1.0;
float f1=1.0;
int servo;
int motor;
void setup(){
Serial.begin(115200);  //Initializing the Baud Rate
Serial2.begin(115200);
tmili=millis();        //Save real time (old time)
}

void loop(){
 if(Serial.available()){  //To control the test
 ch=Serial.read();}
  if(ch=='z'){f=f+0.5;ch=0;}
  if(ch=='n'){f=f-0.5;ch=0;}
while(ch=='s'){
  sendpwm(1000,4);  //The start up signal
  sendpwm(1000,5);
  sendpwm(1690,1);
  sendpwm(1690,2);
if(Serial.available()){
ch=Serial.read();}}
tmil=millis()-tmili;  //To get real time (new time-old time)
ts=tmil/1000.0;       //Convert to seconds
motor=1500+(200*sin(2*pi*f*ts));  //Sine wave for motor
servo=1690+(50*sin(2*pi*f*ts));   //Sine wave for servo
  sendpwm(motor,4);  //Sends the desired signal to pins
  sendpwm(motor,5);
  sendpwm(servo,1);
  sendpwm(servo,2);
}

void sendpwm(int pwm,byte servo){  //PWM Generator Code
  byte low,high;
  low=(pwm*4)%128;
  high=(pwm*4)/128;
  Serial2.write(0x84);
  Serial2.write(servo);
  Serial2.write(low);
  Serial2.write(high);
}
```

E.2: Brushless DC Motor Input Signal (Matlab Code)

```
% This code converts the PWM signal from the PWM generator into sine wave
clear all;
close all;
clc

load('mpwm3.mat');
A=mpwm3;
rate=20000;
t=zeros(size(A,1),1);
for i=1:size(A,1)
 t(i,1)= (i-1)*(1/rate) ;
end

d=zeros(size(A,1),1);
i=1;
b=1;
while(i~=size(A,1))
    z=0;
    f=0;
    while(A(i,1)<2 && i~=size(A,1))
        z=z+1;
        i=i+1;
    end
    while((A(i,1)>=1) && i~=size(A,1))
        f=f+1;
        i=i+1;
    end
    F(b,1)=f;
    Z(b,1)=z;
    b=b+1;
end
i=1;zz=1;
for i=1:size(Z,1)
 for  w=(zz):(Z(i,1)+F(i,1))+zz
    d(w)=(F(i,1)/(Z(i,1)+F(i,1)))*100;
 end
zz=w;
end

mpwmf=3.054*sin(2*pi*t+1.322)+14.01;

plot(t,d,t,mpwmf,'r')
```

E.3: Servo Motor Input Signal (Matlab Code)

```
%This code converts Servo input PWM signal into duty cycle % sine wave
clear all;
close all;
clc

load('spwm3.mat');
A=spwm3;
rate=20000;
t=zeros(size(A,1),1);
for i=1:size(A,1)
 t(i,1)= (i-1)*(1/rate) ;
end
d=zeros(size(A,1),1);
i=1;
b=1;

while(i~=size(A,1))
    z=0;
    f=0;
    while(A(i,1)<2 && i~=size(A,1))
        z=z+1;
        i=i+1;
    end
    while((A(i,1)>=1) && i~=size(A,1))
        f=f+1;
        i=i+1;
         end
    F(b,1)=f;
    Z(b,1)=z;
    b=b+1;
end
i=1;zz=1;
for i=1:size(Z,1)
 for  w=(zz):(Z(i,1)+F(i,1))+zz
    d(w)=(F(i,1)/(Z(i,1)+F(i,1)))*100;
 end
zz=w;
end

spwmf=1.763*sin(2*pi*t+1.309)+15.3;

plot(t,d,t,spwmf,'r')
```

E.4: Brushless DC Motor Output Signal (Matlab Code)

```
% This code converts a square wave signal with its width related to motor
% speed into a sine wave (speed is measured in RPM)

clear all;
close all;
clc
rate=20000;
load('pd3.mat');
A=pd3;

t=zeros(size(A,1),1);
for i=1:size(A,1)
 t(i,1)= (i-1)*(1/rate) ;
 end

d=zeros(size(A,1),1);

z=0;
b=1;
i=1;
```

%% for each two cycles get the number of ones> store in array Z (index,1)

```
while( i~=size(A,1))

   while((A(i,1)<2) && (i~=size(A,1)))
     i=i+1;
   end

  while(A(i,1)>2 &&(i~=size(A,1)))
     i=i+1;
     z=z+1;
  end

    while((A(i,1)<2) && (i~=size(A,1)))
     i=i+1;
     z=z+1;
    end

    while(A(i,1)>2 &&(i~=size(A,1)))
     i=i+1;
     z=z+1;
    end

    Z(b,1)=z;
    z=0;
    b=b+1;
end
```

%% each row value of matrix Z(index,1) is filled in the rows that correspond to the two cycles period>> store matrix d

```
b=1;
i=1;
while( i~=size(A,1))
```

```
    ii=i;
   while(A(i,1)<2 && (i~=size(A,1)))
     i=i+1;
   end

  while(A(i,1)>2 && (i~=size(A,1)))
     i=i+1;
  end

    while(A(i,1)<2 && (i~=size(A,1)))
     i=i+1;
    end

    while(A(i,1)>2 && (i~=size(A,1)))
     i=i+1;
    end
  % i-1>> 0              i>> 5

    for n=ii:i-1
        d(n,1)=Z(b,1);
        d(n,1)=60/(d(n,1)*(1/rate)*2);
    end
     b=b+1;
end
```

%% curve fitted data versus original

```
pdf=2190*sin(2*pi*t+0.1590)+8670;   % after using curve fit tool for matrix d

figure
hold
grid on
plot(t,d,t,pdf,'r')
```

E.5: Servo Motor Output Signal (Matlab Code)

```
% This code plots servo analog reading (feedback) (output signal) along with
% the its fitted curve
clear all
close all
clc;


load('sa3.mat');
q=sa3;
rate =20000;
```

```
%% Time vector (rate = 20000 Hz)
```

```
t=zeros(size(q,1),1);
for i=1:size(q,1)
 t(i,1)= (i-1)*(1/rate) ;

end

saf=0.1328*sin(2*pi*t+1.165)+1.242;

spwmf=1.763*sin(2*pi*t+1.309)+15.3;

hold
grid
plot(t,sa3,'b',t,saf,'r')
```

References

[1] Steven C. Chapra, "Applied Numerical Methods with MATLAB@ for Engineers and Scientists, Second Edition," *International Edition 2008, Berger Chair in Computing and Engineering Tufts University*, pp. 284-324.

[2] "PololuMaestro Servo Controller User's Guide," *@2001-2013 Pololu Corporation*, *http://www.pololu.com/docs/0J40/all*, pp. 33-39.

[3] Ashley Duncan Scillitoe, "Propulsion System Design and Optimisation for the Tumbleweed Novel Autonomous Hexrotor Micro Air Vehicle," *University of Manchester, 2010*, pp.64.

[4] Bristeau, P.Martin, P.Salaun, E.Petit, "The Role of Propeller Aerodynamics in the Model of a Quadrotor UAV," *European Control Conference, 2009.*

[5] Mark Cutler, N.KemalUre, Bernard Michini, Jonathan P.How, "Comparsion of Fixed and Variable Pitch Actuators for Agile Quadrotors," *Aerospace Controls Lab at MIT.*

[6] S. Bouabdallah, "Design and control of quadrotors with application to autonomous flying,"*Thesis 3727 (2007)*, pp.15-27.

[7] P.Mullhaupt, "Analysis and Control of Underactuated Mechanical Nonminimum-Phase Systems," *PhD Thesis, EPFL, 1999.*