UNIVERSITÀ DEGLI STUDI DI MILANO
FACOLTÀ DI SCIENZE MATEMATICHE,
FISICHE E NATURALI

PROGETTO DI

ARCHITETTURE DIGITALI

Il Comelicottero

Docente: Prof. Federico Pedersini

Progetto di:

- Giovanni Intorre
- Simone Castellani
- Andrea Toscano

Anno Accademico 2014/2015

Sommario

Lo scopo di questo progetto è di costruire e programmare un quadrirotore, velivolo costituito da una struttura rigida ad X sulle cui estremità sono installati quattro motori con le relative eliche.
Il drone monta a bordo un microprocessore per ricevere i comandi da terra via seriale con cavo USB, un microcontrollore per il sistema di controllo e la board di sensori da cui preleva i dati che garantiranno la stabilità di volo.
Nella presente documentazione verranno illustrate tutte le fasi dello sviluppo del progetto.

Indice

Capitolo 1

Quadrirotori

1.1 Introduzione

I quadricotteri sono degli aeromobili costituiti da quattro rotori, generalmente progettati per volare senza pilota a bordo e per questo vengono chiamati APR (aeromobile a pilotaggio remoto) o più comunemente droni.
I quadricotteri utilizzano dei sensori per monitorare la propria posizione ed un algoritmo di controllo per la stabilità del volo.
Data la relativa facilità nella manovrabilità, sono largamente utilizzati sia in ambito militare sia in ambito civile.

1.2 Controllo del volo

Prima di descrivere la dinamica del volo di un quadricottero, è bene evidenziare che esistono due possibili configurazioni di volo: a "+" e ad "X", mostrate in Figura 1.1.
Per il nostro progetto è stata utilizzata una configurazione a X.

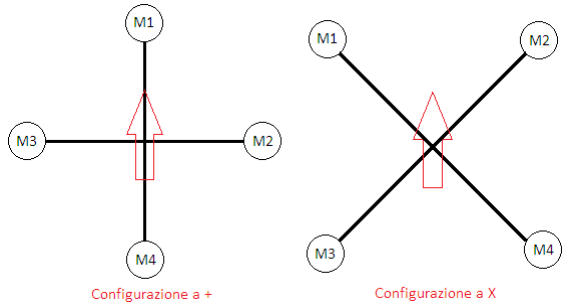

Figura 1.1: Possibili configurazioni di un quadricottero

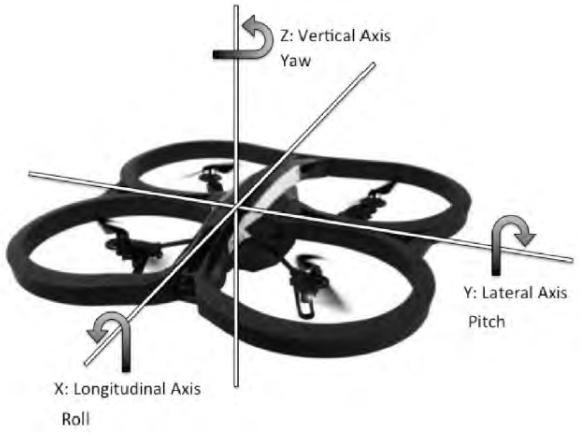

Figura 1.2: Rotazioni di un velivolo attorno agli assi

Un'altra considerazione importante, che si evince dalla Figura 1.2, sono i 6 gradi di libertà di un quadricottero:

- X: movimento frontale del velivolo;
- Y: movimento laterale del velivolo;
- Z: movimento verticale del velivolo;
- Beccheggio (pitch): oscillazione attorno ad un asse trasversale (pitchaxis);
- Rollio (roll): oscillazione attorno ad un asse longitudinale (rollaxis);

- Imbardata (yaw): oscillazione attorno ad un asse verticale (yawaxis)

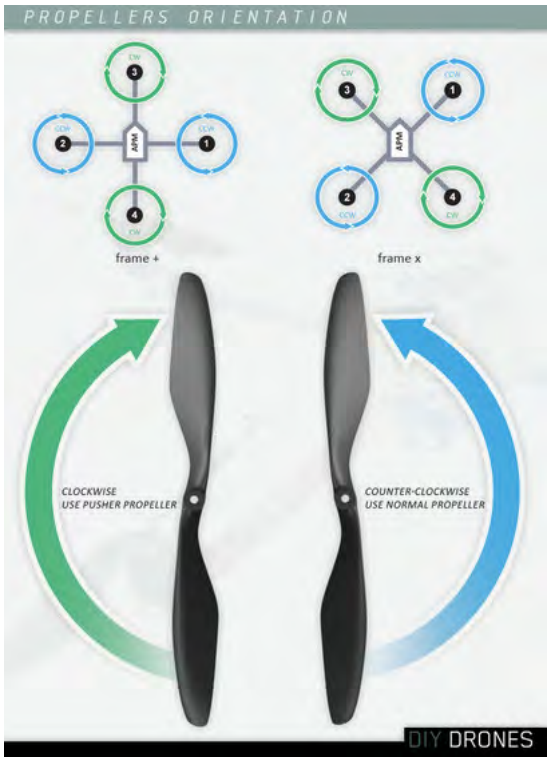

Figura 1.3: Senso di rotazione delle eliche

Nei droni, come mostrato in Figura 1.3, i rotori posti sullo stesso asse hanno lo stesso senso di rotazione mentre è contrario per i restanti. Questo produce sui due assi una coppia di momenti che, a pari velocità di tutti i motori, si eguagliano non facendo ruotare il drone. Un incremento (o decremento) della velocità di un singolo rotore, o di una coppia di questi, provoca uno sbilancio di forze o momenti tali da provocare il movimento del velivolo. In particolare abbiamo che:

- un incremento(decremento) di egual potenza in tutti e quattro i rotori, provoca una salita(discesa) verticale in quota del drone come si può vedere dalla Figura 1.4.a.

- Un incremento di potenza di due motori contigui, genera una rotazione attorno all'asse trasversale o longitudinale. Questa manovra permette di eseguire il pitch e il roll, mantenendo costante lo yaw. Infatti, come si può vedere dall'immagine 1.4.b un aumento della velocità della coppia di motori M2-M3 oppure M1-M4 provoca una rotazione lungo l'asse longitudinale, ovvero un roll. Per poter eseguire il pitch, ovvero una rotazione lungo l'asse trasversale, bisogna aumentare la velocità della coppia di motori M1-M2 o M3-M4, come mostrato in Figura 1.4.c.

- Un incremento pari di potenza di due rotori posti sullo stesso asse, porta il drone a ruotare rispetto all'asse verticale di yaw, come si nota dall'immagine 1.4.d in riferimento alla coppia M2-M4 o M1-M3. La rotazione sarà concorde al senso di rotazione dei motori che hanno maggior potenza.

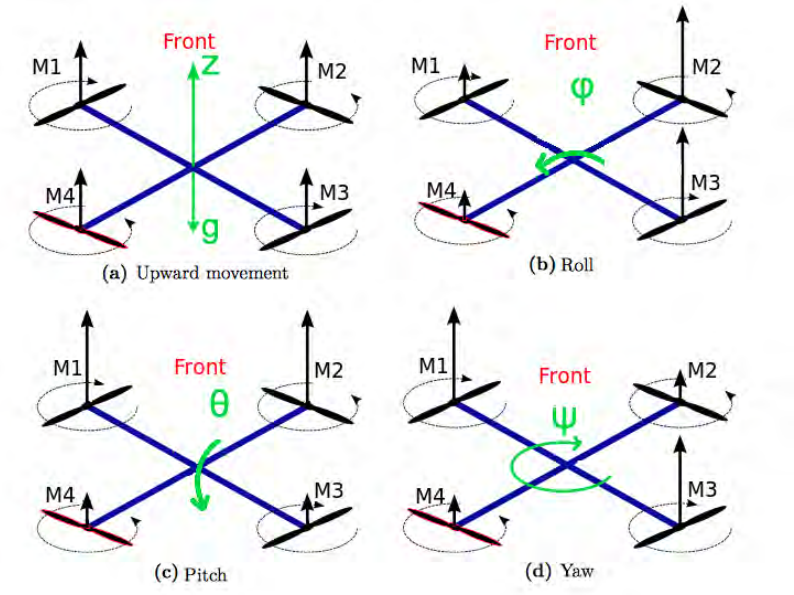

Figura 1.4: Throttle, Pitch, Roll e Yaw del quadricottero

Capitolo 2

Componenti

In questo capitolo verranno riportate le scelte di progettazione e i materiali utilizzati per la costruzione del drone. Inizialmente, per realizzare il drone, si erano scelti i seguenti componenti: motori brushless da 750Kv, ESC da 25A, eliche da 12", frame in fibra di carbonio, una power distribution board (per portare l'alimentazione ai 4 ESC) e due batterie Li-Po da 2 celle.
La scelta dei materiali è stata effettuata sulla base di un progetto esistente reperito online ma, durante una serie di prove, ci si è resi conto che questi componenti presentano una serie di limiti. In particolare gli ESC scelti necessitano di un cambio nel firmware che regola la traduzione dei segnali da inviare ai motori: la versione installata prevede l'utilizzo nei modelli di elicotteri. La coppia batteria/motori scelta non era sufficientemente potente per sollevare il frame scelto.
Le componenti selezionate in prima istanza sono state quindi sostituite con quelle riportate di seguito, che sono largamente utilizzate anche nei droni commerciali.

2.1 Microcontrollore

All'inizio del nostro progetto si era pensato di utilizzare come prototyping board l'*Arduino Yun* [1], scheda provvista di un microcontrollore *ATMega32u4* e del processore *Atheros AR9331* che supporta la distribuzione Linux *OpenWrt-Yun* in grado di eseguire script python, cURL e di gestire la parte network (sia Ethernet che WiFi) della board. La comunicazione tra i due processori avviene tramite la libreria *Bridge*, come mostrato in Figura 2.4. L' *ATMega32u4*, nel nostro caso, gestisce il controllo di volo mentre l'*Atheros AR9331*, grazie al modulo WiFi, instaura una comunicazione punto-punto tra il drone e il PC. [1]).

La scheda è stata scelta confrontando le caratteristiche delle principali board di prototipazione attualmente presenti sul mercato. Ai nostri scopi è fondamentale che la piattaforma disponga di un modulo WIFI, svolga calcoli in *Real Time*, consumi poca energia.

Come si nota dalla Figura 2.1 l'Arduino Yun risulta avere un processore meno performante ma tuttavia adatto al progetto. Raspberry PI e Intel Galileo risultano inadeguate poiché poco inclini a svolgere calcoli in Real Time disponendo di un unico processore.

Le due alternative risiedono nell'Arduino Yun e nel BeagleBone. La prima, come già anticipato, dispone di un microcontrollore, quindi è la candidata ideale, mentre la seconda è programmabile in Real Time solamente utilizzando l'unità *BeagleBone Programmable real-time Unit*.

La Figura 2.2 riassume il consumo energetico delle schede.

Arduino Yun con WIFI operativo è molto prestante in quanto richiede poco

	Arduino Yun	Beaglebone Black	Intel Galileo	Raspberry Pi
Picture				
SoC	Atheros AR9331	Texas Instruments AM3358	Intel Quark X1000	Broadcom BCM2835
CPU	MIPS32 24K and ATmega32U4	ARM Cortex-A8	Intel X1000	ARM1176
Architecture	MIPS and AVR	ARMv7	i586	ARMv6
Speed	400mhz (AR9331) and 16mhz (ATmega)	1ghz	400mhz	700mhz
Memory	64MB (AR9331) and 2.5KB (ATmega)	512MB	256MB	256MB (model A) or 512MB (model B)
FPU	None (Software)	Hardware	Hardware	Hardware
GPU	None	PowerVR SGX530	None	Broadcom VideoCore IV
Internal Storage	16MB (AR9331) and 32KB (ATmega)	2GB (rev B) or 4GB (rev C)	8MB	None
External Storage	MicroSD (AR9331)	MicroSD	MicroSD	SD card
Networking	10/100Mbit ethernet and 802.11b/g/n WiFi	10/100Mbit ethernet	10/100Mbit ethernet	None (model A) or 10/100Mbit ethernet (model B)
Power Source	5V from USB micro B connector, or header pin.	5V from USB mini B connector, 2.1mm jack, or header pin.	5V from 2.1mm jack, or header pin.	5V from USB micro B connector, or header pin.
Dimensions	2.7in x 2.1in (68.6mm x 53.3mm)	3.4in x 2.1in (86.4mm x 53.3mm)	4.2in x 2.8in (106.7mm x 71.1mm)	3.4in x 2.2in (85.6mm x 56mm)
Weight	1.4oz (41g)	1.4oz (40g)	1.8oz (50g)	1.6oz (45g)
Approximate Price	$75	$55 (rev C), $45 (rev B)	$80	$25 (model A), $35 (model B)

Figura 2.1: Schede a Confronto

meno di 300mAh in maniera costante.

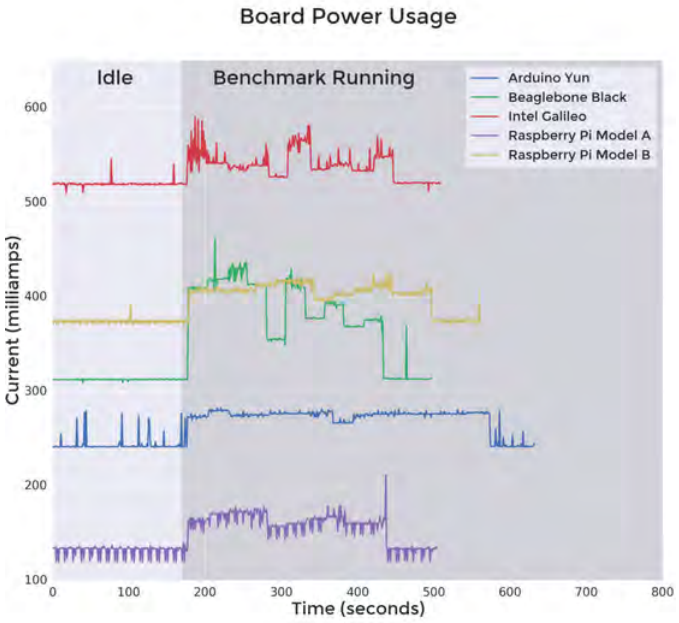

Figura 2.2: Consumi energetici delle board

In definitiva Arduino Yun è stata la prima scelta considerando anche la facilità di utilizzo e la attiva community online utile alla soluzione di molte problematiche.

Durante numerosi test ci si è accorti che la libreria *Bridge*, che mette in comunicazione i due processori, introduceva un ritardo che rallentava il sistema di controllo dell'assetto. Alla fine si è deciso di utilizzare la board priva di WiFi, *Arduino Uno*, e di comunicare in seriale (tramite cavo USB) per testare la bontà del controllo di volo.

L'*Arduino Uno* [2], mostrato in Figura 2.3, è una scheda che utilizza il microprocessore ad 8 bit *ATMega32u4*. Presenta 14 pin digitali di input/output (di cui 6 possono essere usati come output PWM), 6 pin analogici di input, un connettore USB e un header ICSP. Può essere alimentato tramite la porta USB (5V) o tramite un alimentatore esterno (7-12V) dotato di plug

center-positive da 2,1 mm da collegare al power jack della board. La board è provvista di un uscita a 5V utilizzata per alimentare circuiti esterni, come ad esempio un *IMU*. Alcuni de pin presenti sulla board, però, hanno delle funzioni speciali:

- **Serial pin: 0(RX) e 1(TX)**. Usati per ricevere (RX) o trasmettere dati seriali TTL.
- **External Interrupts pin: 2 e 3**. Questi pin possono essere configurati per attivare un interrupt.
- **PWM pin: 3, 5, 6, 9, 10, e 11**. Generano un output PWM a 8 bit utilizzando la funzione *analogWrite()* messa a disposizione da Arduino.
- **SPI pin: 10(SS), 11(MOSI), 12(MISO), 13(SCK)**. Questi pin supportano la comunicazione SPI, utilizzando la libreria *SPI library* messa a disposizione da Arduino.
- **TWI (I2C) pin: pin A4 (o SDA) e pin A5 (o SCL)**. Supportano la comunicazione TWI (I2C) usando la libreria *Wire library* messa a disposizione da Arduino.

L'Arduino può essere programmato usando l'IDE ufficiale, software che include un *serial monitor* che permette di inviare e ricevere dati testuali con la board. Maggiori specifiche tecniche sulla board, sono riportate in tabella 2.1.

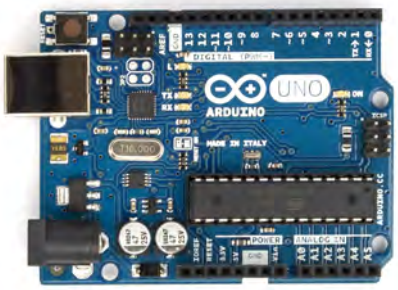

Figura 2.3: Arduino Uno [2]

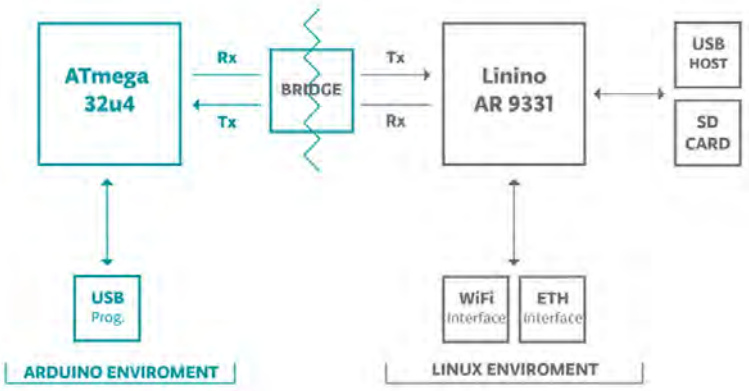

Figura 2.4: Comunicazione tra ATMega32u4 e Atheros AR9331 [1]

2.2 Motori

I motori inizialmente scelti erano i *Turnigy D2836/11 Brushless* da 750KV. Sono stati poi sostituiti in quanto non garantivano un'adeguata spinta, infatti per far decollare il drone bisognava portarli quasi alla massima velocità. Ci si è resi conto che se avessimo voluto aggiungere al drone altri sensori, una camera o qualsiasi altro strumento che ne avrebbe aumentato il peso, questi motori non sarebbero stati in grado di farlo volare. Dopo aver fatto delle prove con dei motori della Turnigy da 1100KV presi in prestito, abbiamo deciso di sostituire i nostri. La scelta è ricaduta su motori *DJI 2212 920KV* ,

Microcontroller	ATMega32u4
Operating Voltage	5V
Input Voltage (recommended)	7-12v
Input Voltage (limits)	6-20v
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pin	6
DC Current per I/O Pin	40mA
DC Current per 3,3V Pin	50mA
Flash Memory	32 KB (ATMega32u4) 0.5 KB used by bootloader
SRAM	2KB (ATMega32u4)
EEPROM	1KB (ATMega32u4)
Clock Speed	16 MHz
CPU	8 bit AVR
Length	68.6 mm
Width	53.4 mm
Weight	25 g

Tabella 2.1: Specifiche microcontrollore Arduino Uno [2]

mostrati in figura 2.2, in quanto garantivano una spinta maggiore dei 750KV e potevano essere alimentati con batterie LiPo a 3 celle (a differenza dei 1100KV che richiedevano batterie a 4 celle). La parola "brushless" significa "senza spazzole" e sta ad indicare che questo tipo di motore non presenta il collettore e le spazzole che sono invece tipiche del motore in corrente continua. Il motore brushless è costituito da un rotore, su cui alloggiano dei magneti permanenti e da uno statore su cui sono disposti numerosi avvolgimenti.
I Brushless hanno una serie di vantaggi rispetto ai motori a spazzole: al contrario di questi, non necessitano di manutenzione periodica dovuta al contatto strisciante tra spazzole e collettore, hanno dimensioni inferiori e bassa inerzia rotorica.
Il *DJI 2212* è anche *outrunner* quindi, a differenza degli *inrunner*, la parte rotante è la calotta esterna. I primi, però, risultano lenti ma sono in grado

di produrre un maggior momento torcente.

Figura 2.5: *DJI 2212 920KV* [3]

Model	DJI 2212 920KV
Battery	3 4 Cell /11.1 18.5V
Motor speed (KV)	920 rpm/V
Current	15-25 A
Max Current	30 A
Recommended battery	Li-Po 3S-4S
Recommended propellers	10x4.5 inch (Li-Po 3S) / 8X4.5 inch (Li-Po 4S)
Max thrust	600 g/axis
Weight	50 g

Tabella 2.2: Specifiche motore *Dji 2212 920KV Brushless Outrunner Motor*

La velocità dei motori da noi scelti è di 920 KV, dove KV è una sigla che indica la velocità del motore per Volt. Questo vuol dire che se il motore è 920KV, girerà 920 volte al minuto per volt. Se la batteria è da 10V allora la velocità massima teorica del motore è 9200 giri al minuto.
Maggiori specifiche sono riportate in tabella 2.2.

2.3 Eliche

Le eliche utilizzate con i motori da 750KV erano da 12" e realizzate in materiale plastico. Durante le prove fatte con i motori da 1100KV, invece, abbiamo utilizzato delle eliche da 10" in fibra di carbonio e nylon, molto resistenti ma per le quali occorreva eseguire una procedura di bilanciamento per eliminare eventuali vibrazioni indesiderate trasmesse al frame.

La bilanciatura è un'operazione svolta attraverso il bilanciatore, mostra-

Figura 2.6: Bilanciatura eliche

to in Figura 2.6, al fine di rendere omogenei i lati delle pale *carteggiandole* e appesantendo il rotore centrale. Quando abbiamo sostituito i motori con i *DJI 2212*, abbiamo dovuto cambiare anche le eliche scegliendo le *DJI 9450* da 9,4", mostrate in Figura 2.7.

Queste eliche sono realizzate in materiale plastico resistente e flessibile, non necessitano di bilanciature e inoltre sono autoavvitanti.

Figura 2.7: *DJI 9450* [4]

2.4 ESC (Electronic Speed Controller)

Gli ESC (*Electronic Speed Controller*) sono dei circuiti elettronici il cui scopo è quello di far variare la velocità e la direzione dei motori. Essi ricevono un segnale PWM il cui dutycycle va da 1ms a 2ms ed in base ad esso variano la velocità del motore.

I primi ESC utilizzati erano i *Turnigy AE-25A* da 25A. Il primo grosso problema di questi regolatori è stato la scarsa documentazione, non abbiamo trovato delle specifiche tecniche e delle informazioni relative al range di frequenze del segnale PWM accettato in input. Questi ESC, inoltre, avevano una serie di altri limiti. Essi nascevano per essere utilizzati sugli elicotteri e non sui droni quindi per poterli utilizzare avremmo dovuto *flashare* il firmware interno. Sono stati quindi sostituiti con i *DJI E300* da 15A, mostrati in Figura 2.8. che, oltre ad essere forniti di specifiche tecniche, sono largamente usati in diversi droni commerciali.

Maggiori specifiche sono riportate in tabella 2.3.

Model	DJI E300
Current	15A OPTO
Battery	Li-Po 3 4 Cell
Voltage	11,1V 14,8V
Signal Frequency	30Hz 450Hz

Tabella 2.3: Specifiche ESC *DJI E300*

Figura 2.8: *DJI E300*

2.5 Frame

Il frame inizialmente utilizzato era il *Turnigy Talon Quadcopter (V2.0) 550mm*, realizzato in fibra di carbonio e scelto per il suo peso contenuto. Il telaio è costituito da un blocco centrale a cui sono ancorate 4 braccia di forma cilindrica.

Il sistema di fissaggio delle braccia al nucleo centrale è risultato poco solido, infatti in caso di urti le braccia si spostavano costringendoci ogni volta a riassemblare il frame.

Il frame è stato sostituito con uno più solido agli urti, il *DJI F450* mostrato in figura 2.9, realizzato in *Nylon 6-6* che oltre ad avere l'alloggiamento per la batteria è provvisto di una distribution board integrata per un wiring de-

gli ESC molto più semplice e sicuro. Maggiori specifiche sono riportate in tabella 2.4.

Figura 2.9: *DJI F450*

Model	DJI F450
Frame Weight	282g
Diagonal Wheelbase	450mm
Takeoff Weight	800g 1600g

Tabella 2.4: Specifiche frame *DJI F450*

2.6 Batterie

All'inizio del progetto avevamo adottato due batterie LiPo da due celle collegate in serie (con una tensione nominale di 14,8V). Dopo il danneggiamento di una di queste, la sostituzione dei ESC e dei motori, abbiamo adottato una batteria unica da 3 celle, mostrata in Figura 2.10. Le batterie LiPo hanno il vantaggio di essere più leggere e compatte di altri tipi di batterie e sono composte da celle con una tensione nominale di 3,7V.

Figura 2.10: *LiPo Battery 5400mAh*

La batteria adottata ha una tensione nominale di 11.1 V (3,7V*3=11,1V) e una capacità di 5400mAh. Il fattore di scarica della batteria è 30C, quindi è in grado di erogare in uscita una corrente massima di 162A (5400mAh*30). Come caricabatterie è stato scelto l'*IMAX B6 50W 5A Charger/Discharger* , mostrato in figura 2.11, in grado di caricare fino a 6 celle con corrente da 0.1A a 5.0A.

Figura 2.11: *Caricabatterie IMAX B6 50W 5A Charger/Discharger 1-6 Cells* [5]

2.7 Abbinamento componenti

Trovare la combinazione efficiente fra le componenti batteria, eliche, motori, ESC e frame non è facile perché dipende dai diversi parametri che caratteriz-

zano le singole parti. Vi sono tool online come eCalc [6] che aiutano l'utente a trovare la configurazione più efficiente in base alle proprie disponibilità.

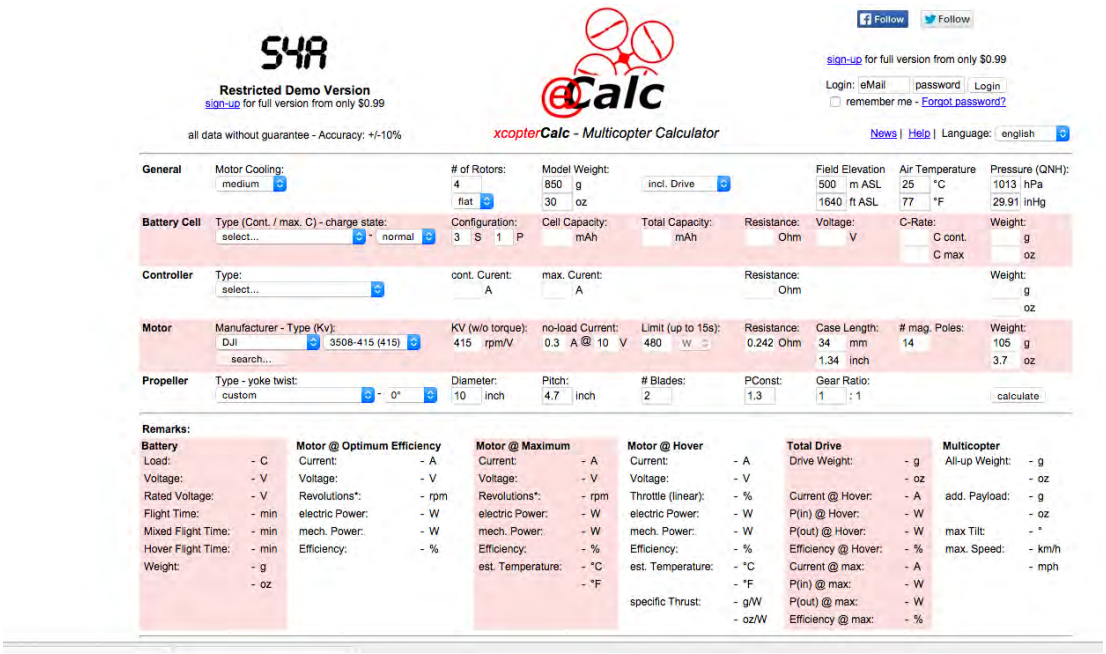

Figura 2.12: *eCalc Multicopter Calculator*

Come si vede dalla Figura 2.12 capacità, voltaggio, peso, fattore di scarica caratterizzano la batteria, resistenza e corrente massima l'Esc, Kv, grandezza, peso e resistenza il motore e diametro, pitch, numero e materiale le eliche. Si può arrivare ad un abbinamento basilare semplicemente considerando i Kv dei motori, amperaggio degli ESC, numero di celle della LiPo e diametro×pitch delle eliche.

La situazione riassunta dalla terza riga della Tabella 2.5 rispecchia maggiormente il nostro caso.

Motori	ESC	Batteria/Eliche
1500 kv	35A	Lipo2/9x6 o Lipo4/7x3
1110 kv	25A	Lipo2/11x7 o Lipo4/7x3
950 kv	15A	Lipo2/12x6 o Lipo3/9x6
750 kv	15A	Lipo2/12x6 o Lipo3/9x6

Tabella 2.5: Abbinamento Motori-ESC-Batteria/Eliche

2.8 IMU

L'IMU (Inertial Measurement Unit) è un dispositivo elettronico in grado di misurare la velocità angolare e l'accelerazione di un oggetto grazie alla presenza di un giroscopio e di un accelerometro. Unendo i dati di questi sensori è possibile ottenere informazioni relative al Pitch, Roll e Yaw del drone. Nel nostro progetto è stato utilizzato un 10 Degree of Freedom (10DOF) IMU: il *GY-80*, mostrato in Figura 2.13.

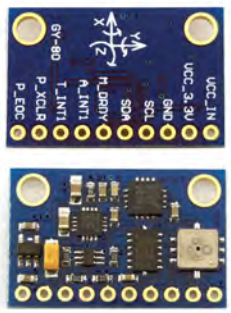

Figura 2.13: *10DOF GY-80 IMU*

Quest'IMU è costituito da un accelerometro triassiale (ADXL345), un giroscopio triassiale (L3G4200D), un magnetometro (HMC5883L), un barometro e un sensore di temperatura (BMP085). E' stato scelto il GY-80 in quanto utilizza l'accelerometro ADXL345 e il giroscopio L3G4200D che sono larga-

mente usati e ben documentati.

In Figura 2.14 viene mostrato il collegamento tra l'IMU e l'Arduino UNO.

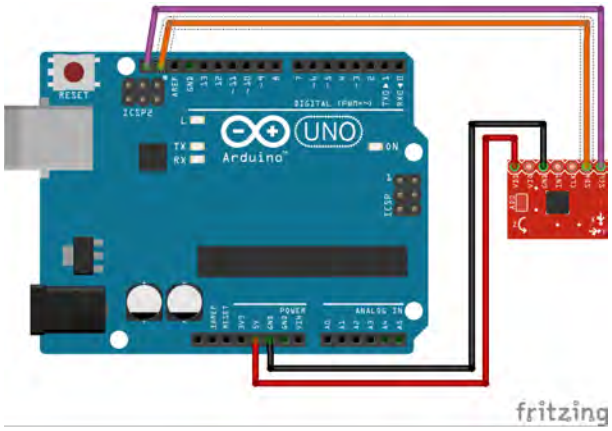

Figura 2.14: *Collegamento tra l'Arduino Uno e il GY-80*

Tutti i sensori dell'IMU supportano la comunicazione I2C ed è possibile accedere ai singoli sensori specificando il loro indirizzo I2C:

- indirizzo L3G4200D: 0x69
- indirizzo ADXL345: 0x53
- indirizzo MC5883L: 0x1E
- indirizzo BMP085: 0x77

Per la comunicazione I2C è sufficiente collegare all'Arduino i pin GND, 5V, SDA (I2C clock) e SCL (I2C data).

2.9 L'Accelerometro

L'accelerometro è un dispositivo in grado di rilevare e misurare un'accelerazione [7]. Possiamo immaginarlo come un cubo con al centro una sfera sospesa da 3 molle che la attraversano, agganciate a loro volta al centro di ogni faccia del cubo, come mostrato in Figura 2.15

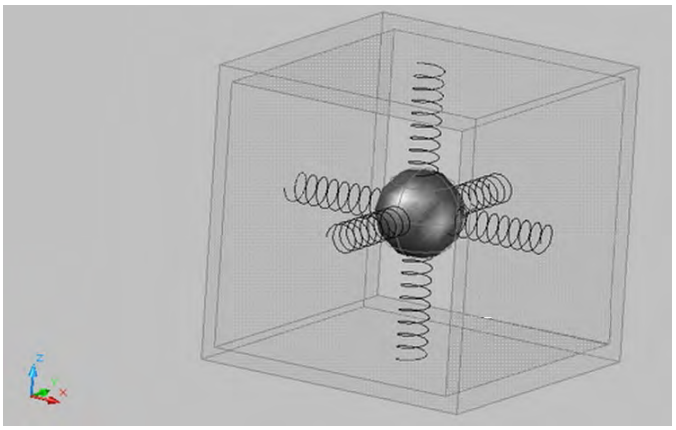

Figura 2.15: schema ideale di un accelerometro a 3 assi [7]

Muovendo il cubo nello spazio con una accelerazione diversa da 0, la sfera si muove al suo interno allungando e comprimendo le molle che la tengono sospesa. La misurazione del grado di compressione delle molle permette di stabilire l'entità dell'accelerazione nella direzione delle molle. Nella realtà, il comportamento dell'accelerometro può essere rappresentato dalla Figura 2.16.

Si può vedere che a causa della forza di gravità, la sfera comprime la parte inferiore della molla orientata lungo l'asse Z. Quindi, l'asse Z, subisce un'accelerazione costante verso il basso pari all'accelerazione gravitazionale terrestre $G = 9.81 m/s^2$.

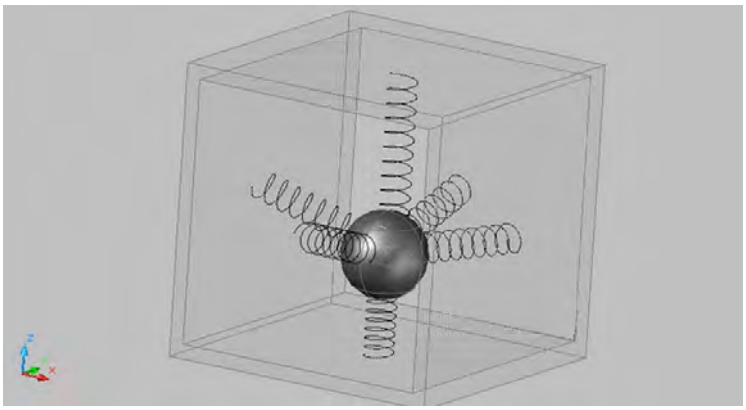

Figura 2.16: schema reale di un accelerometro a 3 assi [7]

2.9.1 Accelerometro ADXL345

L'ADXL345 [8] è un accelerometro triassiale in grado di misurare la proiezione del vettore gravità g sugli assi di riferimento X, Y, Z.

Ha una risoluzione settabile di 10 o 13 bit e permette misure fino a $\pm 16g$ (è possibile settare il range di misura a $\pm 16g$, $\pm 8g$, $\pm 4g$ e $\pm 2g$). Le misurazioni ottenute dal dispositivo sono espresse in G e sono composte da 16 bit (8 bit MSB e 8 bit LSM). E' possibile accedere al sensore tramite I2C o SPI e grazie alla sua alta risoluzione (3,9 mg/LSB) è possibile ottenere misurazione con precisione inferiore ad $1°$.

2.10 Il Giroscopio

Il giroscopio, inventato da Lèon Foucault, è un dispositivo rotante che per effetto della legge di conservazione del momento angolare, tende a mantenere il suo asse di rotazione orientato in una direzione fissa.

E' costituito principalmente da un rotore che ruota intorno al suo asse. Quando il rotore è in rotazione, il suo asse tende a mantenersi parallelo a sé stesso

e ad opporsi ad ogni tentativo di cambiare il suo orientamento.
Nel nostro caso si utilizza il principio del giroscopio per rilevare la velocità angolare di un corpo.

I giroscopi MEMS presentano una piccola massa vibrante che oscilla per esempio a 10kHz. La massa è sospesa in un sistema a molla, la lettura avviene tramite un sistema capacitivo come negli accelerometri. Quando il giroscopio viene ruotato, la rotazione esercita una forza di Coriolis perpendicolare sulla massa, che risulta tanto maggiore quanto la massa è lontana dal centro della rotazione. La massa oscillante fornisce quindi una lettura diversa su ciascun lato dell'oscillazione, che è una misura per la velocità di rotazione

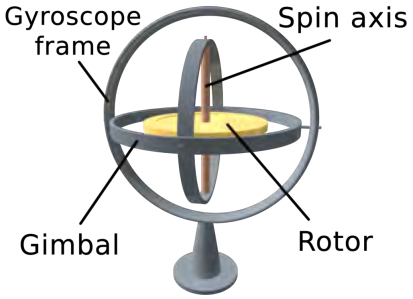

Figura 2.17: Esempio di Giroscopio

2.10.1 Giroscopio L3G4200D

L' L3G4200D è un giroscopio MEMS triassiale che permette di misurare la velocità angolare sui tre assi ed ha un scala di misurazioni settabile di ±250, ±500 e ±2000 dps. Le misurazioni ottenute dal dispositivo sono composte da 16 bit (8 bit MSB e 8 bit LSM) ed è possibile accedere al sensore tramite I2C o SPI.

Capitolo 3

Sensori

Come detto, i sensori sono indispensabili al drone in quanto permettono di conoscere la posizione del velivolo e le forze agenti sui tre assi di riferimento. In base alle informazioni fornite dai sensori, il controllore PID sarà in grado di garantire la stabilità del drone.
L'IMU necessita di procedure specifiche di inizializzazione, in base alla tipologia di utilizzo. Occorre impostare correttamente i registri all'interno dei singoli sensori a bordo del GY-80. Inoltre, accelerometri e giroscopi di fascia bassa, come quelli utilizzati nel progetto, hanno bisogno di una calibrazione accurata per correggere errori di fabbricazione.

3.1 Diagramma di flusso dei Sensori

Per avere un quadro generale del flusso di esecuzione che coinvolge i sensori, ha senso presentare un diagramma che riassume le operazioni svolte su di essi. I dettagli di tali calcoli sono descritti in maniera più approfondita nelle sezioni successive.

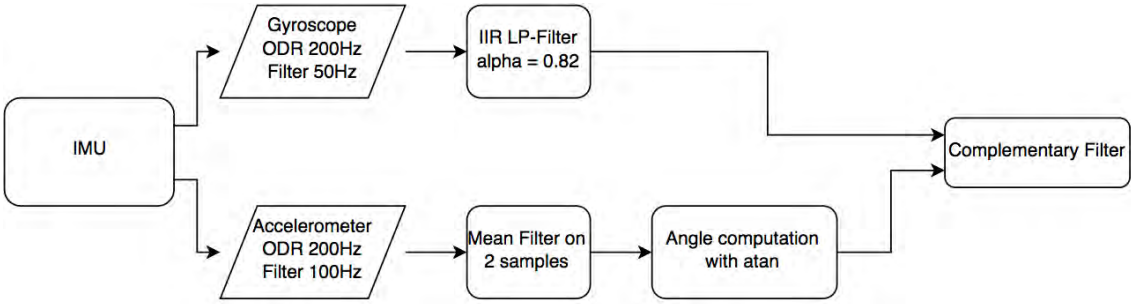

Figura 3.1: Diagramma di flusso per i Sensori

Il giroscopio ha un output data rate (ODR) di 200Hz e viene filtrato via hardware sull'IMU a 50Hz. Successivamente passa attraverso un filtro passa-basso per poi entrare nel filtro complementare.
L'accelerometro, sempre con un ODR di 200Hz viene filtrato via hardware a 100Hz. Quindi ne viene fatta una media su 2 campioni e successivamente, per l'asse pitch e roll, viene calcolato l'angolo corrispondente prima di entrare nel filtro complementare.

3.2 Calibrazione dell'accelerometro ADXL345

In alcuni casi può capitare che gli assi del sensore non siano perfettamente ortogonali tra loro. Per la maggior parte delle applicazioni commerciali, la sensibilità del sensore può essere soddisfacente. Nei casi in cui sia necessaria una precisione maggiore, come ad esempio il calcolo della posizione di un corpo, questi errori di fabbricazione possono causare letture sbagliate che influenzano di conseguenza il comportamento della catena di processamento

dei segnali campionati. Nel caso di disallineamento tra due assi, si misura una presenza di valori diversi da 0 anche sull'asse di rotazione; nel caso di un drone definire gli angoli di pitch e roll diventa quindi difficile ed impreciso.
Si può definire la relazione tra le misurazioni normalizzate A_{x1}, A_{y1}, A_{z1} e le letture raw del sensore A_x, A_y, A_z come:

$$\begin{bmatrix} Ax1 \\ Ay1 \\ Az1 \end{bmatrix} = \Big[A\_m\Big]_{3x3} \begin{bmatrix} 1/A\_SC_x & 0 & 0 \\ 0 & 1/A\_SC_y & 0 \\ 0 & 0 & 1/A\_SC_z \end{bmatrix} \begin{bmatrix} Ax - A\_OSx \\ Ay - A\_OSy \\ Ax - A\_OSz \end{bmatrix}$$
$$= \begin{bmatrix} ACC_{11} & ACC_{12} & ACC_{13} \\ ACC_{21} & ACC_{22} & ACC_{23} \\ ACC_{31} & ACC_{32} & ACC_{33} \end{bmatrix} \begin{bmatrix} Ax \\ Ay \\ Az \end{bmatrix} + \begin{bmatrix} ACC_{10} \\ ACC_{20} \\ ACC_{30} \end{bmatrix} \tag{3.1}$$

dove [A_m] è la matrice di disallineamento 3x3 tra gli assi dell'accelerometro e gli assi del dispositivo, A_SC$_i$ (i= x,y,z) è la sensibilità (o scale factor) e A_OS$_i$ è lo zero-g level (o offset) 3.1. L'obiettivo della calibrazione è quello di determinare i 12 parametri, da ACC10 a ACC33, in modo tale da ottenere i valori normalizzati A_{x1}, A_{y1} e A_{z1}, per ogni misurazione raw in posizioni arbitrarie, tale che:

$$|A| = \sqrt{A_{x1}^2 + A_{y1}^2 + A_{z1}^2} = 1 \tag{3.2}$$

Tuttavia, la scelta è stata di normalizzare tutto a 255, posticipando la trasformazione dei bit in m/s^2 a dopo la normalizzazione.
Il procedimento utilizzato è piuttosto semplice. Il sensore, fissato su un supporto contenente una bolla, è stato disposto in modo tale da poter cam-

pionare la forza agente in modo perpendicolare su ogni asse: sia con verso positivo che con verso negativo. Per ogni posizione sono stati misurati i valori presenti su tutti gli assi. Il rilevamento è stato effettuato calcolando una media su 20 iterazioni di un ciclo con all'interno una chiamata alla funzione *computeAccOffset()* per un totale di 6000 campioni.
In tabella 3.1 sono riportate le medie dei valori delle misurazioni per ogni posizione.

X up	X down
$A_x = 277.73$	$A_x = -249.62$
$A_y = 6.60$	$A_y = 2.25$
$A_z = -17.21$	$A_z = -23.30$

Y up	Y down
$A_x = 13.54$	$A_x = 14.52$
$A_y = 272.20$	$A_y = -258.03$
$A_z = -21.64$	$A_z = -15.15$

Z up	Z down
$A_x = -1.41$	$A_x = 10.01$
$A_y = 8.42$	$A_y = 9.68$
$A_z = 232.93$	$A_z = -268.39$

Tabella 3.1: Medie dei valori campionati nelle differenti posizioni

I 18 valori ottenuti dalle letture sono stati messi a sistema imponendo i valori normalizzati a ± 255, di seguito è riportato un esempio del sistema utilizzato per calcolare i coefficienti dell'asse Z:

$$\begin{cases} Ax_{1up} = (ACC_{11} \cdot Ax + ACC_{12} \cdot Ay + ACC_{13} \cdot Az) + ACC_{10} \\ Ay_{1up} = (ACC_{21} \cdot Ax + ACC_{22} \cdot Ay + ACC_{23} \cdot Az) + ACC_{20} \\ Az_{1up} = (ACC_{31} \cdot Ax + ACC_{32} \cdot Ay + ACC_{33} \cdot Az) + ACC_{30} \\ Ax_{1down} = (ACC_{11} \cdot Ax + ACC_{12} \cdot Ay + ACC_{13} \cdot Az) + ACC_{10} \\ Ay_{1down} = (ACC_{21} \cdot Ax + ACC_{22} \cdot Ay + ACC_{23} \cdot Az) + ACC_{20} \\ Az_{1down} = (ACC_{31} \cdot Ax + ACC_{32} \cdot Ay + ACC_{33} \cdot Az) + ACC_{30} \end{cases}$$

Riscrivendo il sistema:

$$\begin{cases} 0 = (ACC_{11} \cdot (-1,41) + 0 \cdot 8,42 + 0 \cdot 232,93) + ACC_{10} \\ 0 = (0 \cdot (-1,41) + ACC_{22} \cdot 8,42 + 0 \cdot 232,93) + ACC_{20} \\ 255 = (0 \cdot (-1,41) + 0 \cdot 8,42 + ACC_{33} \cdot 232,93) + ACC_{30} \\ 0 = (ACC_{11} \cdot 10,01 + 0 \cdot 9,68 + 0 \cdot (-268,39)) + ACC_{10} \\ 0 = (0 \cdot 10,01 + ACC_{22} \cdot 9,68 + 0 \cdot (-268,39)) + ACC_{20} \\ -255 = (0 \cdot 10,01 + 0 \cdot 9,68 + ACC_{33} \cdot (-268,39)) + ACC_{30} \end{cases}$$

In maniera analoga, il procedimento può essere fatto per gli altri assi. In tabella 3.2 sono riportati i Gain e gli Offset per ogni asse, ricavati dalle soluzioni dei sistemi appena introdotti.

Asse	Gain	Offset
X	0.9671	-13.5926
Y	0.9618	-6.8146
Z	18.0369	1.0173

Tabella 3.2: Valori di Gain e Offset per ogni asse

3.2.1 Inizializzazione dell'accelerometro ADXL345

Per inizializzare l'ADXL345 occorre abilitare il *measurement mode* (modalità che permette la misurazione delle accelerazioni) nel registro $POWER\_CTL$, specificare la risoluzione e il range di misurazione, scegliere l'output data rate (ODR) e la relativa larghezza di banda (bandwidth). Per abilitare questa modalità bisogna settare a 1 il bit D3 del registro $POWER\_CTL$, la cui struttura è mostrata in figura 3.2.

Register 0x2D—POWER_CTL (Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0
0	0	Link	AUTO_SLEEP	Measure	Sleep	Wakeup	

Figura 3.2: Registro $POWER\_CTL$ [8]

Per impostare la risoluzione e il range di misurazione, occorre settare opportunamente i bit D3, D1 e D0 del registro $DATA\_FORMAT$ mostrato in figura 3.3.

Register 0x31—DATA_FORMAT (Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0
SELF_TEST	SPI	INT_INVERT	0	FULL_RES	Justify	Range	

Figura 3.3: Registro $DATA\_FORMAT$ [8]

Nel progetto è stato modificato il bit $FULL\_RES$ in modo da usare l'accelerometro in *full resolution mode*. Questa modalità permette di ottenere una risoluzione di $4mG/LSB$, quindi ogni bit letto rappresenta $4mG$ di accelerazione. I bit $RANGE$ sono stati impostati in modo da ottenere un range di misura di $\pm 8G$ ed avere quindi un range più elevato, a discapito della risoluzione.

Per impostare l'output data rate e la bandwidth desiderata, bisogna settare in modo opportuno i bit del registro BW_RATE.

Register 0x2C—BW_RATE (Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	LOW_POWER	Rate			

Figura 3.4: Registro BW_RATE [8]

Dopo diverse prove con data rate inferiori, è stato scelto un ODT di 200 Hz e una frequenza di taglio del filtro passa basso di 100 Hz. Questo valore si è rivelato ottimale allo scopo di ottenere la stabilità del velivolo.
Di seguito viene riportato il codice relativo all'inizializzazione dell'accelerometro con il relativi valori esadecimali scritti all'interno di ogni registro.

```
// ADXL345 INIT                                                                    1
writeCmd(ADXL_ADDRESS, REG_DATA_FORMAT, RANGE_8);//RANGE_8 = 0x0A                  2
writeCmd(ADXL_ADDRESS, REG_PWR_CTL, MEASURE_MODE);//MEASURE_MODE = 0x08           3
writeCmd(ADXL_ADDRESS, REG_BW_RATE, MODE_200_100);//MODE_200_100 = 0x0B          4
```

Listing 3.1: Inizializzazione accelerometro ADXL345

L'accelerometro è dotato di altri tre registri da 8 bit: OFSX, OFSY e OFSZ. Questi registri permettono di settare l'offset (con un fattore di scala di 15 mG/LSB) che verrà automaticamente sommato all'accelerazione in output. Tuttavia, non si sono impostati offset aggiuntivi perché il sensore è stato fissato in modo solido al centro del frame, in piano, in modo tale da considerare tutto come un unico corpo.

3.2.2 Lettura dei valori

I dati acquisiti dell'accelerometro, vengono memorizzati *raw* nei registri DATAX0, DATAX1, DATAY0, DATAY1, DATAZ0 e DATAZ1. In questo modo, per ogni asse, la misurazione ottenuta viene divisa in due blocchi ha 8 bit: MSB (Most Significant Bit) e LSB (Least Significant Bit). Una volta letti i dati raw dall'accelerometro, bisogna convertirli in Gs. Per fare questo, basta semplicemente moltiplicare il dato ottenuto per una costante che dipende dal range e dalla risoluzione impostati nel registro $DATA\_FORMAT$. Di seguito è riportata una tabella con i valori delle costanti:

Range, risoluzione	mG/LSB
qualsiasi range, full resolution	3,9
$\pm 2g$, 10 bit	3,9
$\pm 4g$, 10 bit	7,8
$\pm 8g$, 10 bit	15,6
$\pm 16g$, 10 bit	31,2

Tabella 3.3: Valori dei coefficienti di conversione delle misure dell'accelerometro

Nella listato 3.2 viene mostrata la procedura di lettura dei valori degli assi X,Y e Z; questi vengono sommati e un contatore tiene in memoria il numero dell'iterazione.

```
void readAcc() {
// Declaring buffer for acc data
float buffer[3];
float tempX, tempY, tempZ;
// Reading values from ADXL345
gy80.readAccData(buffer);

```

```
tempX = -1 * buffer[XAXIS] ;
tempY = buffer[YAXIS];
tempZ = -1 * buffer[ZAXIS];

sumAccX += tempX;
sumAccY += tempY;
sumAccZ += tempZ;
accCounter++;
}
```

Listing 3.2: Lettura valori dell'accelerometro

3.2.3 Calcolo angoli Pitch e Roll

L'accelerometro può anche essere utilizzato per l'operazione di *tilt sensing* ovvero il calcolo della pendenza dello sensore rispetto agli assi di pitch e roll. In assenza di accelerazione lineare, l'uscita dell'accelerometro è una misura vettoriale del campo di rotazione gravitazione e può essere utilizzato per determinare il proprio angolo di orientamento del pitch e del roll. L'accelerometro è insensibile alla rotazione del vettore della forza di gravità (asse yaw) e l'equazione per il calcolo degli angoli di pitch e roll presenta delle instabilità quando gli assi di questi sono allineati con la gravità [9].

Nel progetto sono stati considerati solamente due assi per il calcolo del tilt. I dati grezzi espressi in bit, letti dall'accelerometro, ci permettono di ricavare gli angoli di *pitch* e *roll* del Comelicottero. Per calcolare, ad esempio, l'angolo di *pitch* che il drone assume dobbiamo considerare la proiezione del vettore g sugli assi X e Z come mostrato in figura 3.5.

Il valore di accelerazione sull'asse X è proporzionale al seno dell'angolo di inclinazione, mentre il valore di accelerazione sull'asse Z è proporzionale al

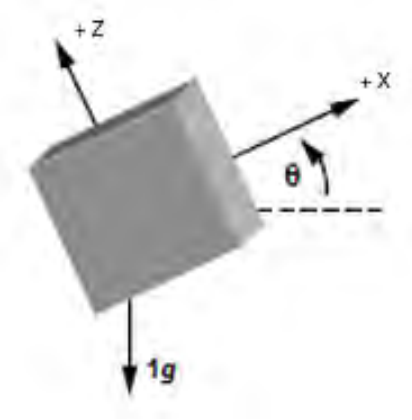

Figura 3.5: Angolo di Pitch

coseno dell'angolo di inclinazione.

Di conseguenza le due misure si compensano ed il loro rapporto è una costante.

$$\frac{A_x}{A_z} = \frac{\sin(\theta)}{\cos(\theta)} = \tan(\theta) \tag{3.3}$$

L'angolo θ si può facilmente ricavare:

$$\theta = \arctan\left(\frac{A_x}{A_z}\right) \tag{3.4}$$

3.2.4 Inizializzazione del giroscopio L3G4200D

Per inizializzare il giroscopio L3G4200D occorre abilitare i tre assi alla lettura, definire l'output data rate (ODR), la bandwidth, il range di misura e la sensibilità. Per abilitare i tre assi, impostare l'ODR e la bandwidth bisogna settare opportunamente i bit del registro $CTRL\_REG1$. Nel nostro caso

DR1	DR0	BW1	BW0	PD	Zen	Yen	Xen

Figura 3.6: Registro $CTRL\_REG1$ [10]

è stato settato il valore 0x6F all'interno del registro. In questo modo, oltre ad aver abilitati gli assi, è stato impostato un valore di ODR pari a 200 Hz e una bandwidth pari a 50 Hz. Il range di misura viene impostato tramite il registro $CTRL\_REG4$ e in base al valore scelto varia la sensibilità della misurazione, come riportato in tabella.

BDU	BLE	FS1	FS0	-	ST1	ST0	SIM

Figura 3.7: Registro $CTRL\_REG4$ [10]

Per il progetto abbiamo utilizzato una scala di 2000 dps (degree per second) per avere una minore sensibilità (8,75 mdps/LSB). Nel registro è stato scritto il valore 0x30.

```
// L3G4200D                                                                    1
writeCmd(L3G4_ADDRESS, CTRL_REG1, L3G4_BW_200_50);                             2
writeCmd(L3G4_ADDRESS, CTRL_REG2, L3G4_HPF1);                                  3
writeCmd(L3G4_ADDRESS, CTRL_REG3, L3G4_INTERRUPT);                             4
writeCmd(L3G4_ADDRESS, CTRL_REG4, MODE_2000);                                  5
writeCmd(L3G4_ADDRESS, CTRL_REG5, L3G4_LPF2);                                  6
```

Listing 3.3: Inizializzazione giroscopio L3G4200D

3.2.5 Lettura dei valori del giroscopio L3G4200D

I dati raw delle velocità angolari vengono salvati in sei registri da 8 bit: $OUT\_X\_L, OUT\_X\_H, OUT\_Y\_L, OUT\_Y\_H, OUT\_Z\_L$ e OUT_Z_H. Anche in questo caso, come per l'accelerometro, il valore di ogni asse è espresso su 16 bit, divisi in MSB (Most Significant Bit) e LSB (Least Significant Bit). Una volta letti i dati raw dal giroscopio, vanno convertiti in dps. La conversione avviene moltiplicando il valore ottenuto per una costante che di-

pende dal range selezionato, come riportato in tabella 3.4. Nel nostro caso, avendo scelto un range di ±2000, la costante moltiplicativa è 8,75 mdps/LSB.

Range (dps)	mdps/LSB
±250	8,75
±500	17,50
±2000	70

Tabella 3.4: Costanti di conversione dei dati da raw a dps

Nel listato di codice 3.4 è mostrata parte della procedura di lettura dei valori del giroscopio *readGyro()*; le velocità angolari contenute nelle variabili relative ai tre assi non sono quelle istantanee, ma sono invece il risultato di un filtraggio dei valori letti dal giroscopio mediante un filtro IIR con coefficiente $ALPHA\_IIR = 0.82$. Quest'ultimo determina un filtro passa basso con frequenza di taglio $f_c = (1 - ALPHA\_IIR)/T_s \simeq 36Hz$.

```
gyroXAverage = (ALPHA_IIR * gyroXAverage + (1.0 - ALPHA_IIR) * tempX);  1
gyroYAverage = (ALPHA_IIR * gyroYAverage + (1.0 - ALPHA_IIR) * tempY);  2
gyroZAverage = (ALPHA_IIR * gyroZAverage + (1.0 - ALPHA_IIR) * tempZ);  3
```

Listing 3.4: Snippet di codice per il filtraggio IIR dei valori campionati dal giroscopio

3.3 Sensor Fusion

Per ottenere la posizione angolare del drone attraverso l'accelerometro e il giroscopio, si necessita di uno filtro matematico atto a "fondere" insieme i segnali dei due sensori.

L'uso di questo filtro è necessario in quanto entrambi i sensori hanno caratteristiche che ne limitano le potenzialità: il giroscopio tende ad avere un drift, una deriva, che nel tempo fa rilevare valori errati; l'accelerometro, seppur fedele quando l'accelerazione è progressiva, soffre moltissimo di eventuali vibrazioni alle alte frequenze, che influenzano il calcolo degli angoli precedentemente introdotto. Tra i vari filtri utilizzati a tale scopo, i più diffusi sono due: il Kalman filter e il Complementary filter. Il primo risulta essere migliore ma il calcolo ha un costo computazionale molto più elevato. Nel progetto è stato scelto il complementary filter, che si dimostra essere di semplice implementazione e poco avido di CPU.

Il filtro complementare prevede l'uso di un filtro passa basso e di un filtro passa alto. Il filtro passa basso è applicato ai segnali rilevati dall'accelerometro, per eliminare le vibrazioni alle alte frequenze; il filtro passa alto, invece, filtra i segnali con bassa frequenza per correggere il drift del giroscopio.

Nella forma più semplice, lo pseudocodice del filtro è il seguente:

$$angolo(n) = \alpha * [gyroData(n) * dn + angolo(n-1)] + (1-\alpha) * accData(n) \tag{3.5}$$

dove α è il coefficiente del filtro e segue la relazione:

$$\tau = \frac{a * dt}{1-a} \leftrightarrow a = \frac{\tau}{\tau + dt} \tag{3.6}$$

dove τ rappresenta la costante di tempo del filtro, ovvero la durata relativa del segnale su cui si sta agendo [11]. Nel dominio delle frequenze determina la larghezza di banda del filtro. Per il filtro passa-basso, segnali di durata

maggiore alla costante di tempo passeranno inalterati mentre segnali di durata inferiore verranno filtrati. Viceversa per il filtro passa alto.

Nel progetto è stata utilizzata una costante di tempo $\tau = 0.49$ e un dt che dipende dal tempo di calcolo delle procedure precedenti, con cui è possibile calcolare il coefficiente a dinamicamente ad ogni iterazione; in questo modo, i coefficienti del filtro permettono di avere una banda passante coerente ad ogni calcolo.

Di seguito viene riportato il codice per il calcolo degli angoli di pitch e roll, mediante la fusione dei dati rilevati dai due sensori. Il valore letto dal giroscopio viene integrato per il dt calcolato nell'iterazione corrente e viene sommato all'angolo ricavato dall'applicazione del filtro all'iterazione precedente.

```
void Filter::complementary(float *output, float *angleAcc, float *gyroRate,
    unsigned int *dTime) {
     float dt = ((float)*dTime / 1000.0);
     if (dt > 0) {
          alpha = tau / (tau + dt);
          lastOutput = alpha * (lastOutput + *gyroRate * dt)+(1-alpha)* *
              angleAcc;
          *output = lastOutput;
     }
}
```

Listing 3.5: Sensor fusion

Capitolo 4

Sistema di Controllo

In questo capitolo verrà trattato il sistema di controllo adottato dal Comelicottero, prima in linea generale e poi in maniera approfondita. L'algoritmo di controllo è necessario affinché il quadrirotore mantenga la stabilità in volo.

4.1 Controllore PID

Il controllo adottato dal Comelicottero è il PID (Proporzionale, Integrativo, Derivativo) che è un sistema in retroazione negativa. É stato scelto questo tipo di controllore perché la sua implementazione non è molto difficile, è comunemente utilizzato in questo tipo di progetti e in diversi ambiti industriali.
Il PID riceve come *Input* il valore attuale della grandezza misurata e un dato di riferimento, detto *Setpoint*, ne calcola la differenza determinando così l' *Errore*.
Il controllore corregge sia errori positivi che negativi tendendo sempre verso l'errore nullo.

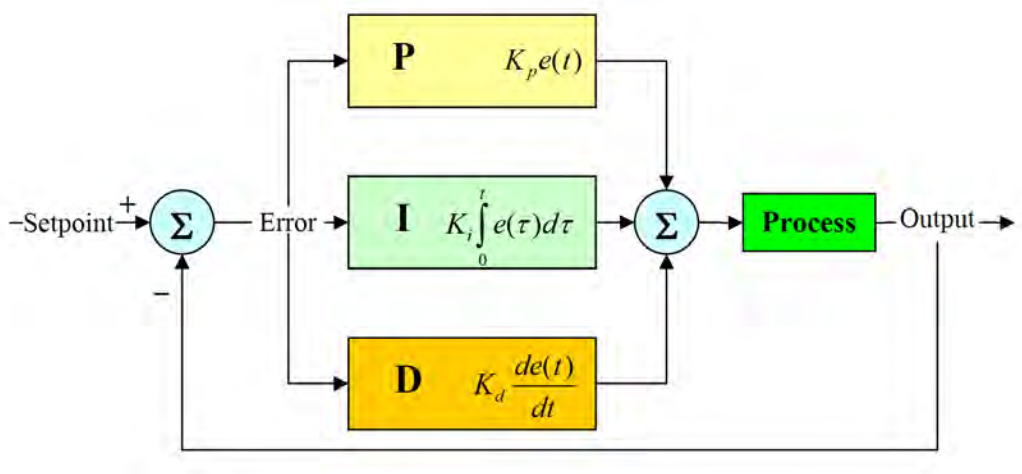

Figura 4.1: Schema a blocchi di un PID[12]

Come suggerisce il nome, l'uscita del PID è determinata dalla somma delle tre azioni: proporzionale, integrativa e derivativa.

I PID sono generalmente facili da implementare ma presentano alcuni limiti:

- Non sono in grado di adattarsi a cambiamenti nei parametri del processo.
- Non sono stabili a causa dell'effetto Windup nell'azione integrale
- Esistono regole di taratura che reagiscono male in alcune condizioni.

La Figura 4.1 mostra lo schema a blocchi di un controllore PID, la cui legge di controllo può essere riassunta dalla seguente equazione:

$$Output(t) = K_P e(t) + K_I \int_{t_0}^{t} e(\tau)d\tau + K_D \frac{de}{dt} \tag{4.1}$$

Azione Proporzionale

L'azione proporzionale è ottenuta moltiplicando l'errore per una opportuna

costante K_P:

$$Output_P = K_P e(t) \tag{4.2}$$

Con un controllore di questa forma, è possibile regolare un processi sia stabili che instabili.

Non è garantito, però, che l'errore converga a zero perché l'azione di controllo risulta applicabile solo se l'errore è diverso da zero. Nel caso di un quadrirotore, la componente P è fondamentale perché determina cosa è più importante: il controllo manuale o i valori misurati dai sensori [13].

Più alto è il coefficiente K_P maggiore risulterà la sensibilità e la reattività a modifiche della sua posizione. Viceversa, se è troppo basso il quadricottero risulterà lento nel cambiamento di assetto e sarà difficile da manovrare.

Azione Integrale

L'azione integrale è proporzionale all'integrale nel tempo del segnale di errore $e(\tau)$ moltiplicato per la costante K_I :

$$Output_I = K_I \int_{t_0}^{t} e(\tau) d\tau \tag{4.3}$$

L'azione integrale fa sì che il controllore abbia memoria dei valori del segnale di errore nel tempo; in particolare, il valore dell'azione integrale non è necessariamente nullo se è nullo l'errore. Questa proprietà dà al PID la capacità di portare il processo esattamente al punto di riferimento richiesto, dove la sola azione proporzionale risulterebbe nulla.

L'azione integrale è anche detta elemento metastabile di un PID perché un

ingresso costante non convergerà ad un determinato valore.

Nel caso di un drone, l'azione integrale può aiutare ad aumentare la precisione della posizione angolare del velivolo.

Per esempio quando il quadricottero è disturbato ed il suo angolo cambia di 20 gradi, il controllore si ricorda, in teoria, di quanto l'angolo è cambiato e tornerà indietro di 20 gradi. In pratica, se si comanda al drone di andare avanti e poi di fermarsi improvvisamente, esso continuerà per un certo periodo a contrastare l'azione e ad aumentare il valore dell'integrale.

Senza questo termine l'opposizione non durerebbe tanto a lungo. Il fattore integrale è specialmente utile in caso di vento irregolare e *ground effect*.

In ogni caso, quando I diventa troppo grande, il quadricottero inizia a rispondere sempre meno velocemente e diminuisce l'effetto dell'azione proporzionale.

Il problema più comune legato alla componente integrativa è il cosiddetto *windup*.

L'insorgenza del windup può verificarsi quando il comando di attuazione è vincolato a dei valori massimi e minimi [14]; in tal caso, supponendo di applicare al sistema di riferimento un disturbo a gradino, l'integratore inizierà ad accrescere il suo valore in uscita, per via dell'errore non nullo. L'output del controllore è allora tale da saturare il comando di attuazione e questo rimarrà costante, anche se l'uscita dell'integratore crescerà fino a quando l'errore non sarà nullo.

Se il sistema è stabile, dopo un dato intervallo di tempo, l'errore diventerà nullo e l'integratore inizierà a scaricarsi e, fino a quando il valore di uscita non sarà inferiore alla saturazione (relativa all'attuatore), il segnale di controllo

rimarrà costante. Questo fenomeno fa sì che il sistema, dopo aver raggiunto la condizione di errore nullo, si allontani in direzione opposta, creando un effetto di sovraelongazione dalle caratteristiche non lineari.

Per prevenire questo problema, nell'implementazione reale dell'integratore, bisognerebbe prevedere delle soglie di saturazione, correlate alle soglie dell'attuatore [15]. Una soluzione pratica è scalare il segnale di controllo, garantendo che al massimo valore di uscita del controllore, corrisponda il massimo valore di attuazione. In questo modo si elimina completamente l'effetto di windup.

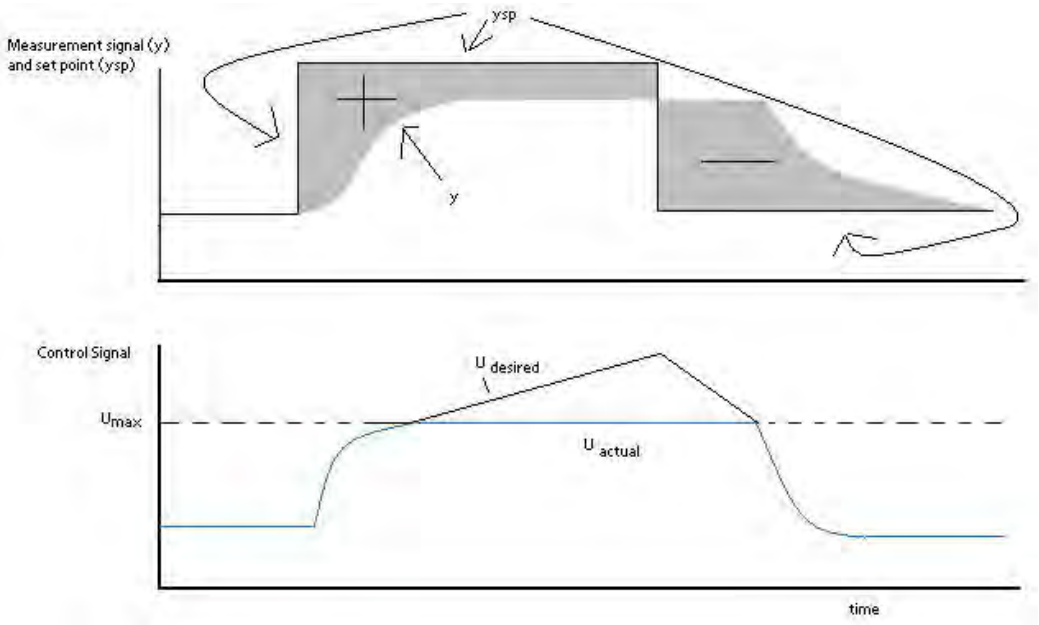

Figura 4.2: Fenomeno del Windup

I grafici della Figura 4.2 mostrano l'effetto windup dell'integrale. L'errore e è mostrato nel primo disegno come $e = y_{sp} - y$, dove y_{sp} è il setpoint mentre y è il segnale misurato. Il secondo grafico mostra il segnale di controllo u.

In questo caso specifico si verifica un cambio del setpoint e il controllo del

segnale satura al suo massimo u_{max}. L'errore del sistema non è ancora eliminato perché il segnale di controllo è troppo piccolo per mandare l'errore a zero.

Questo causa invece un aumento della componente integrale che a sua volta causa un aumento del segnale di controllo. Rimane ancora dunque una differenza fra il segnale di controllo desiderato e quello attuale.

Azione Derivativa

L'azione derivativa è proporzionale alla derivata dell'errore nel tempo:

$$Output_D = K_D \frac{de}{dt} \tag{4.4}$$

Questo apporto compensa rapidamente le variazioni del segnale di errore: se vediamo che l'errore sta aumentando, l'azione derivativa cerca di "frenare" questa deviazione, in ragione della sua velocità di cambiamento, senza aspettare che l'errore diventi significativo (azione proporzionale) o che persista per un certo tempo (azione integrale)[16].

L'azione derivativa è spesso tralasciata nelle implementazioni dei PID perché li rende troppo sensibili: un PID con azione derivativa, per esempio, subirebbe una brusca variazione nel momento in cui il riferimento venisse cambiato quasi istantaneamente, risultando in una derivata di e tendente ad infinito.

Ciò sconsiglia l'applicazione dell'azione derivativa in tutti i casi in cui l'attuatore fisico non deve essere sottoposto a sforzi eccessivi.

Il termine derivativo permette al drone di raggiungere più velocemente l'angolo desiderato. L'azione derivativa viene chiamata anche il *parametro acce-*

leratore perché amplifica l'input dell'utente.

Diminuisce velocemente l'azione del controllo con il rapido diminuire dell'errore.In pratica aumenta la velocità di reazione ed in certi casi aumenta l'effetto di della componente P.

Occorre prestare attenzione con il tuning della componente D poiché rende il quadricottero molto sensibile. Questo vuol dire che se il drone oscilla, questa azione peggiorerà il movimento oscillatorio.

4.2 Implementazione del Sistema di Controllo

4.2.1 Prime sperimentazioni

Prima di implementare il controllore nella forma utilizzata nel progetto abbiamo testato diverse configurazioni di PID.

Ci siamo chiesti innanzitutto se fosse possibile controllare il drone utilizzando unicamente le informazioni relative alla posizione assoluta provenienti dal sensor fusion. A questo scopo i tre PID, uno per asse, ricevono in ingresso come *setpoint* l'angolo assoluto dato dall'utente e come *input* il valore di fusion relativo al proprio asse.

Purtroppo, con questo sistema non è stato possibile trovare dei coefficienti PID adatti a stabilizzare il drone.

L'apporto del giroscopio, che lavora su velocità angolari, è indispensabile per aumentare la rapidità con cui il drone cerca di stabilizzarsi.

Il test successivo prevedeva una configurazione PID che utilizzasse solo l'ap-

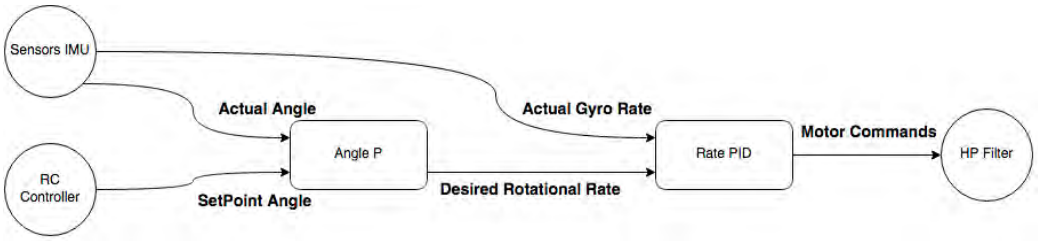

Figura 4.3: Schema logico del PID per pitch e roll

porto diretto dei giroscopi, senza considerare gli angoli calcolati con il fusion filter.

Anche in questo caso il risultato non era soddisfacente, il drone stentava a stabilizzarsi nonostante numerosi tentativi con diversi coefficienti.

Il problema, in questo caso, era dovuto al fatto che dalla sola informazione data dai giroscopi non è possibile conoscere la posizione assoluta del drone lungo l'asse di pitch e roll, perciò risulta molto difficile trovare dei coefficienti adatti a stabilizzare il velivolo con un certo grado di precisione.

Esistono droni stabilizzati mediante i soli giroscopi e vengono chiamati normalmente *acrobatici* proprio per la loro indole reattiva. Sono più difficili da controllare e richiedono costantemente una correzione manuale da parte del pilota.

4.2.2 Implementazione finale

Per il controllo dei movimenti rotazionali di *pitch* e *roll*, abbiamo adottato uno soluzione di controllori *in cascata* che prevede una componente proporzionale in quello più esterno e un PID completo in quello interno. La Figura 4.3 mostra il flusso dei segnali in questo tipo di controllore.

Dal punto di vista implementativo, il sistema è un unico blocco regolatore e non è stato necessario implementare i controllori come entità separate. Per quanto riguarda le rotazioni di pitch e roll, il controllore può essere rappresentato dallo schema in figura 4.4. Si può notare come l'intero regolatore è di fatto la somma tra un *Rate PID* e un regolatore proporzionale che contribuisce, con il suo output, alle sue tre azioni.

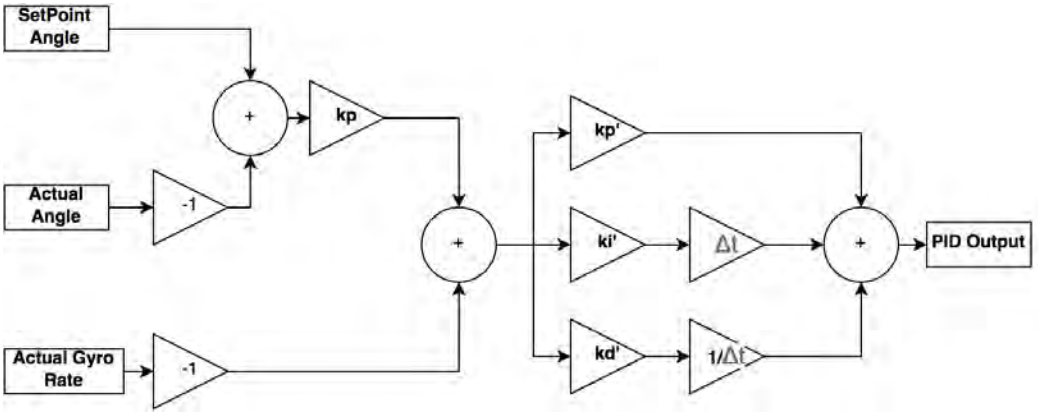

Figura 4.4: Schema equivalente ai regolatori in cascata

Possiamo chiamare l'angolo attuale θ, il valore desiderato dell'angolo $\hat{\theta}$, la velocità rotazionale misurata dal giroscopio $\dot{\theta}$, l'uscita del primo controllore PID u e l'uscita del secondo controllore PID U, tutte funzioni del tempo t. Definiamo l'errore sull'angolo come:

$$\epsilon(t) = \hat{\theta}(t) - \theta(t) \tag{4.5}$$

si può allora definire l'uscita del primo controllore PID come:

$$u(t) = K_P \epsilon(t) \tag{4.6}$$

dove K_P è il coefficiente proporzionale del primo PID.

Questa semplice azione, ha l'obiettivo di correggere in maniera proporzionale l'errore $\epsilon(t)$ e fornisce tale risultato come setpoint al PID successivo.

Quest'ultimo lavora, a differenza del primo, sul valore attuale del giroscopio. Definiamo l'errore tra il setpoint della velocità rotazionale, ovvero $u(t)$, e la velocità rotazionale del drone come:

$$\epsilon'(t) = u(t) - \dot{\theta}(t) \tag{4.7}$$

Considerando queste grandezze, la cui relazione può essere dedotta dalla figura 4.4, possiamo scrivere l'equazione che relaziona l'uscita del rate PID $U(t)$ con le altre grandezze come:

$$U(t) = K'_P \epsilon'(t) + K'_I \int_0^t \epsilon'(t)dt + K'_D \frac{\epsilon'(t)}{dt} \tag{4.8}$$

dove K'_P, K'_I e K'_D sono i coefficienti del secondo controllore PID.

Sostituendo 4.5 in 4.6, 4.6 in 4.7 e 4.7 in 4.8, esplicitiamo $U(t)$:

$$\begin{aligned} U(t) = {} & K'_P \left[\left(K_P \hat{\theta}(t) - \theta(t) \right) - \dot{\theta}(t) \right] \\ & + K'_I \int_0^t \left[\left(K_P \hat{\theta}(t) - \theta(t) \right) - \dot{\theta}(t) \right] dt \\ & + K'_D \frac{\left[\left(K_P \hat{\theta}(t) - \theta(t) \right) - \dot{\theta}(t) \right]}{dt} \end{aligned} \tag{4.9}$$

Sotto l'ipotesi che $\theta = \int_0^t \dot{\theta} dt$, si possono rappresentare tutti fattori in termini di singole grandezze:

$$\begin{aligned} U(t) = {} & K'_P K_P \hat{\theta}(t) - K'_P K_P \theta(t) - K'_P K_P \dot{\theta} \\ & + K'_I K_P \int_0^t \hat{\theta}(t) dt - K'_I K_P \int_0^t \theta(t) dt - K'_I K_P \theta(t) \\ & + K'_D K_P \frac{\hat{\theta}(t)}{dt} - K'_D K_P \dot{\theta(t)} - K'_D \ddot{\theta}(t) \end{aligned} \tag{4.10}$$

in cui:

$$\begin{aligned} -K'_I K_P \theta(t) &= -K'_I K_P \int_0^t \dot{\theta}(t) dt \\ -K'_D K_P \dot{\theta}(t) &= -K'_D K_P \frac{\theta(t)}{dt} \\ -K'_D \ddot{\theta}(t) &= -K'_D \frac{\dot{\theta}(t)}{dt} \end{aligned}$$

In caso di hovering possiamo imporre il setpoint del PID più esterno $\hat{\theta} = 0$ e raccogliere θ e $\dot{\theta}$, ottenendo:

$$\begin{aligned} U(t) = {} & -(K'_P K_P + K'_I)\theta \\ & - K'_I K_P \int_0^t \theta(t) dt \\ & - (K'_P + K'_D K_P)\dot{\theta} \\ & - K'_D \ddot{\theta}(t) \end{aligned} \tag{4.11}$$

appare evidente che il PID agisce sia in funzione diretta delle componenti lette dai sensori, cioè l'angolo e la velocità angolare, che in funzione dell'accelerazione angolare con cui la posizione del drone viene modificata.
I coefficienti proporzionale K_P, integrativo K_I, derivativo K_D e derivativo di

secondo grado K_{DD}, del controllore completo, possono essere definiti dalle relazioni:

$$\begin{aligned} K_P &= -K'_P K_P - K'_I \\ K_I &= -K'_I K_P \\ K_D &= -K'_P - K'_D K_P \\ K_{DD} &= -K'_D \end{aligned} \tag{4.12}$$

e per garantire l'hovering, l'uscita del PID per i movimenti di roll e pitch può essere definita come:

$$U(t) = K_P\theta(t) + K_I \int_0^t \theta(t)dt + K_D\dot{\theta}(t) + K_{DD}\ddot{\theta}(t) \tag{4.13}$$

Dato che dall'accelerometro non possiamo ricavare informazioni utili per determinare la rotazione dell'asse di *yaw* del velivolo, il controllo può essere effettuato solo in base al giroscopio.

Le rotazioni sull'asse dello *yaw* sono controllate mediante un solo controllore PID. Lo schema di controllo si semplifica e in ingresso al PID forniamo una velocità angolare desiderata $\hat{\phi}$ specificata dall'utente e la velocità angolare letta dal giroscopio $\dot{\theta}$, uguale a quella in input nei PID definiti precedentemente. La figura 4.5 mostra il flusso dei segnali per questo tipo di controllo. Nel caso in cui $\hat{\phi} = 0$, ovvero quando si desidera che il drone non ruoti lungo l'asse Z, definendo l'errore come $e(t) = \hat{\phi}(t) - \dot{\theta}(t) = -\dot{\theta}(t)$, l'uscita del PID viene descritta sostituendo $e(t)$ nell'equazione 4.1:

$$U(t) = -K_P\dot{\theta}(t) - K_I \int_0^t \dot{\theta}(t) - K_D \frac{\dot{\theta}(t)}{dt} \tag{4.14}$$

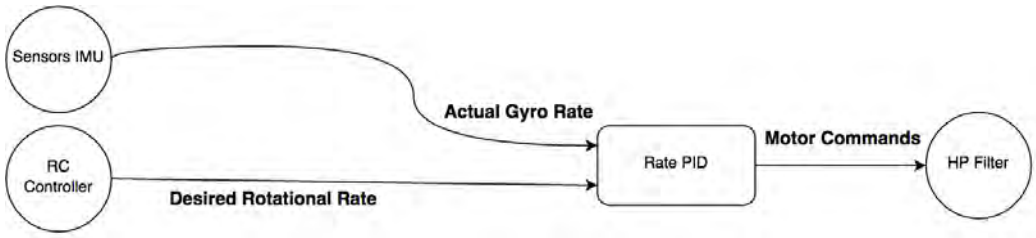

Figura 4.5: PID Logic Layout for Yaw

che può essere riscritta come:

$$U(t) = -K_P\dot{\theta}(t) - K_I\theta(t) - K_D\ddot{\theta}(t) \tag{4.15}$$

si può notare che anche in questo caso il controllo è calcolato come combinazione lineare di fattori in funzione dell'angolo, della velocità angolare lungo l'asse Z e l'accelerazione angolare con cui il drone si muove.

4.2.3 Approssimazioni

Semplificazione delle costanti temporali L'intervallo temporale dt con cui viene calcolato il PID, quando fissato, lo si può includere direttamente nella definizione dei coefficienti. In questo modo non è necessario il calcolo delle costanti di integrazione, o derivazione, necessarie ad ogni iterazione dell'algoritmo.

Di conseguenza i coefficienti K_I e K_D possono essere calcolati come:

$$K_I = k_i dt$$
$$K_D = \frac{k_d}{dt}$$

in cui k_i e k_d sono i coefficienti scelti per il PID.

Approssimazione dell'integrale Il controllore PID implementato è a tempo discreto. Occorre trovare un metodo per approssimare al meglio la sua versione continua.
L'esatto mapping fra il dominio di Laplace e il dominio Zeta, dove T_s è il tempo di campionamento, è nella forma [17]:

$$z = e^{sTs} \tag{4.16}$$

La conversione purtroppo coinvolge una funzione trascendente e la risultante funzione di trasferimento non può essere rappresentata come un rapporto fra polinomi. Questo rende difficile l'implementazione di un algoritmo di controllo su un processore.
Esistono diverse regole di quadratura che possono essere derivate costruendo funzioni interpolatrici polinomiali, più facili da integrare.
Se si utilizza la *regola del rettangolo*, mostrata in figura 4.6, la funzione interpolatrice è un polinomio di grado zero che passa attraverso il punto $\left[\frac{a+b}{2}, f(\frac{a+b}{2})\right]$:

$$\int_a^b f(x)dx \approx (b-a) f\left(\frac{a+b}{2}\right) \tag{4.17}$$

Con questo metodo di approssimazione numerica, la stabilità del controllo è strettamente dipendente dal loop-time. La soluzione preserva la stabilità del modello a tempo continuo solo se il loop è sufficientemente veloce.

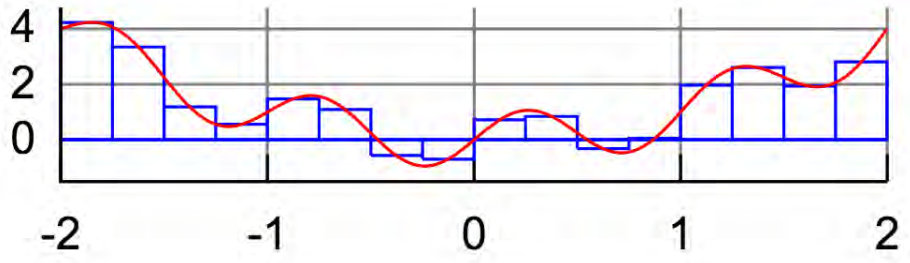

Figura 4.6: Rectangle Rule

Per questo motivo è preferibile utilizzare il metodo di Tustin che approssima un integrale come somma delle aree di trapezoidi, come mostrato in figura 4.7. Questo metodo preserva le proprietà di stabilità della forma a tempo continuo, indipendentemente dalla velocità del loop.
Per approssimare l'azione integrale nei controllori PID utilizzati nel progetto è stata utilizzata l'approssimazione:

$$\int_a^b f(x)dx \approx (b-a)\frac{f(a)+f(b)}{2} \tag{4.18}$$

La corrispondente trasposizione fra Laplace e Zeta [18] è nella forma:

$$s = \frac{2}{Ts}\frac{z-1}{z+1} \tag{4.19}$$

Questa trasformazione mappa l'intera metà sinistra del piano s nella circonferenza unitaria nel piano z.

Approssimazione della derivata La componente derivativa del PID è approssimata con un filtraggio dell'azione derivativa mediante un filtro passa basso, la cui frequenza di taglio è 5∼10 volte più grande del rapporto K_P/K_D, per evitare interferenze con lo zero del PID posizionato in $-K_P/K_D$.

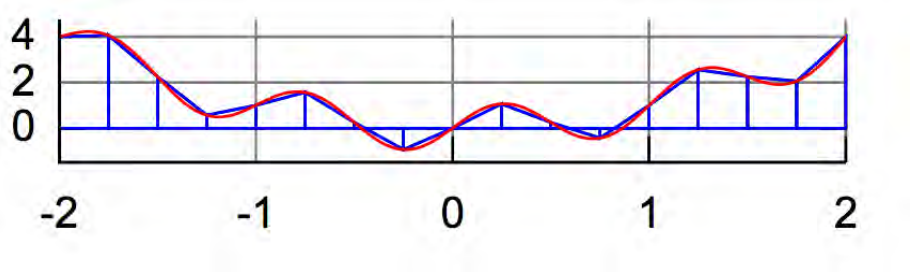

Figura 4.7: Trapezoidal Rule

Nel dominio di Laplace, la funzione di trasferimento del filtro è descritta come:

$$\dot{s} = K_D \frac{s}{1 + \frac{s}{N}} \tag{4.20}$$

Il filtro può essere discretizzato con il metodo di Tustin e assume la forma:

$$D(t) = k_f D(t-1) + K_D(\epsilon(t) - \epsilon(t-1)) \tag{4.21}$$

dove $D(t)$ è l'azione derivativa del PID, $\epsilon(t)$ è l'errore e i coefficienti K_f e K_D sono definiti come:

$$k_f = (2 - Ndt)/(2 + Ndt)$$
$$K_D = (2k_d N)/(2 + Ndt)$$

4.2.4 Implementazione del PID

Come anticipato nell'introduzione di questo capitolo, la struttura a cascata può essere implementata come un unico controllore PID che include il contributo proporzionale esterno.

Dal seguente listato si può notare che il costruttore ha come parametri due bound che servono a limitare l'uscita totale del PID e l'apporto dell'azione

integrativa.

```
PID::PID(int maxV, int minV) {
     kpAngle = 0.0, kpRate = 0.0, ki = 0.0, kd = 0.0;
      I = 0.0, D = 0.0, previousError = 0.0;
     MaxValue = maxV, MinValue = minV;
}// Settings PID's coefficients
void PID::setParameters(float setKpAngle, float setKpRate, float setki,
    float setkd) {
     // PID_COMPUTER_TIMER is defined in config.h
     // In this way sample time is fixed
     float SampleTimeInSec = ((float)PID_COMPUTE_TIMER) / 1000.0;
     // sampleTime is inclued in each coefficient
     // -> less computation in compute method
     kpAngle = setKpAngle;
     kpRate = setKpRate;
     ki = setki * SampleTimeInSec * 0.5;
     kd = (2 * setkd * 100) / (2 + 100 * SampleTimeInSec);
     // N = 100
     kf = (2 - 100 * SampleTimeInSec)/ (2 + 100 * SampleTimeInSec);
}
```

Listing 4.1: Setting coefficienti PID

Il metodo *setParameters* ha come parametri i quattro coefficienti del PID: *setKpAngle* relativo all'angolo, *setKpRate*, *setki*, *setkd* relativi al giroscopio; questo metodo calcola il *dt* e imposta i reali coefficienti utilizzati nel calcolo. L'implementazione del PID è esposta nel seguente listato.

```
float PID::compute(float setPoint, float angle, float gyro) {
     float error = (kpAngle*(setPoint - angle) - gyro);
     float P = kpRate * error;
     // Computing integrative
     float deltaI = ki * (error + previousError);
     // Computing derivative
     D = kf * D + kd * (error - previousError);
```

```
    //  PID output =  P  +        I +  D                                    8
    float output = P + I + deltaI + D ;                                     9
    if ((error * output > 0) && ((output > MaxValue) || (output < MinValue) 10
        )) {
         I += ki * errorOldI;                                               11
         errorOldI = 0;                                                     12
    }                                                                       13
    else {                                                                  14
         I += deltaI;                                                       15
         errorOldI = error;                                                 16
    }                                                                       17
    // Updating previous error                                              18
    previousError = error;                                                  19
    // Output                                                               20
    output = P + I + D;                                                     21
}                                                                           22
```

Listing 4.2: Implementazione mainRoutine PID

Alla riga 2 viene computato l'errore relativo alla componente proporzionale sull'angolo, che verrà successivamente utilizzato nei calcoli successivi.

Alla riga 3 viene calcolata la componente proporzionale sul giroscopio.

Alla riga 5 viene computato l'attuale contributo dell'azione integrale, *deltaI*.

Successivamente viene calcolata l'azione derivativa D e l'uscita globale del PID *output*.

A questo punto, si passa alla valutazione della componente integrale. Se l'output e l'errore hanno il medesimo segno, e se l'output oltrepassa i valori limite, allora la componente integrale I non viene aggiornata, altrimenti viene aumentata del valore *deltaI* ottenuto precedentemente. Il problema del windup è risolto ponendo una soglia oltre la quale l'azione integrale non ha effetto.

Figura 4.8: Roll Tuning

4.3 Tuning dei coefficienti

I coefficienti dei PID sono stati trovati sperimentalmente.

La procedura di tuning per il pitch ed roll risulta diversa dal procedimento adottato per lo yaw.

Dalla Figura 4.8 si può vedere come il drone sia stato ancorato ad un bastone sospeso fra 2 sedie. Questo ha permesso di lavorare su un unico asse di rotazione disabilitando i PID relativi ai rimanenti.

Il drone viene perturbato lungo l'asse, simulando ad esempio un rotazione sull'asse del roll, in modo da osservare la reazione del PID.

La calibrazione è iniziata settando i valori dei coefficienti delle componenti proporzionali, relativi all'angolo assoluto e alla velocità angolare.

Il coefficiente proporzionale *KpAngle*, che agisce sull'angolo ottenuto dal-

l'algoritmo di sensor fusion, deve essere abbastanza elevato da smorzare le oscillazioni sull'asse di roll del drone senza però far andare in oscillazione il sistema.

La risposta con il solo termine proporzionale si rivela però lenta, per questo è necessario incrementare il coefficiente dell'azione proporzionale sul giroscopio $KpRate$.

Grazie a questo, il drone è capace di portarsi in hovering dopo poche oscillazioni. Il coefficiente dell'azione integrale Ki è il successivo parametro da aumentare affinché il drone si stabilizzi più velocemente. Infine abbiamo aumentato lievemente il coefficiente di derivazione kd che è stato necessario per smorzare rapidamente le oscillazioni prossime alla posizione desiderata.

I valori dei coefficienti dei PID di pitch e roll sono riportati in tabella 4.1.

Coefficienti Pitch/Roll	Valori
$kpAngle$	3.3
$kpRate$	0.12
ki,	1.35
kd	0.004

Tabella 4.1: Coefficienti PID Pitch e Roll

Per testare lo Yaw è invece necessario che il drone sia libero di muoversi lungo il piano XY. Per questo è stato necessario trovare un modo per ancorare il quadricottero; ci siamo serviti del disco rotante di un hard-disk per computer desktop (figura 4.9). Il tutto viene fissato ad una piattaforma rigida che il drone non è in grado di spostare durante la sessione di tuning (figura 4.10). Lo Yaw è regolato esclusivamente dal giroscopio che agisce sull'asse Z, per cui occorre settare solamente i 3 coefficienti legati al controllore interno.

Figura 4.9: Hard Disk utilizzato come piatto rotante

Figura 4.10: Comelicottero montato sul piatto rotante

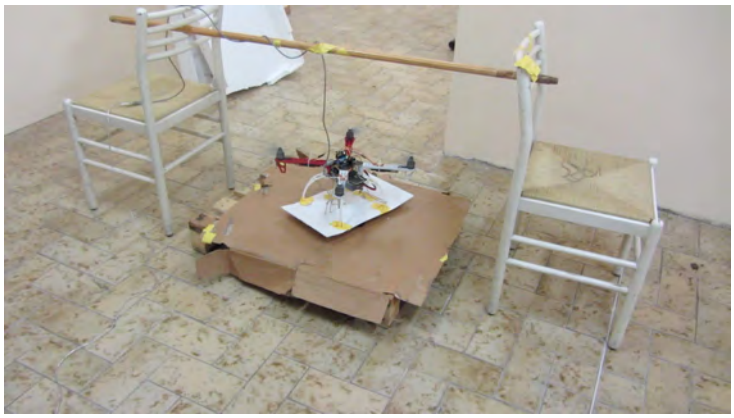

Figura 4.11: Yaw Tuning

Con le stesse regole di calibrazione adottate per le rotazioni di pitch e roll, i coefficienti dello yaw sono stati decisi nel seguente ordine *kp, ki, kd* e la tabella 4.2 ne mostra i valori.

Coefficienti Yaw	Valori
kpRate	1
ki,	3
kd	0.003

Tabella 4.2: Coefficienti PID Yaw

Capitolo 5

Filtro di compensazione

Una volta implementato il controllore PID, con la configurazione appena introdotta, ci si è resi conto che non era sufficiente per garantire la stabilità del drone.

Inizialmente, la reattività del drone, durante i test effettuati vincolandolo lungo un asse, era abbastanza alta. La posizione di equilibrio era raggiunta velocemente anche se con oscillazioni, talvolta violente, intorno al valore di hover.

In un secondo momento abbiamo testato il drone liberamente, cioè senza vincolarlo ad alcun asse.

In questa situazione abbiamo avuto modo di constatare che la reattività dei motori non era sufficiente a sollevare e stabilizzare il velivolo.

É stato progettato un filtro di compensazione della risposta dei motori che è da collocarsi dopo l'azione dei PID e prima dell'effettiva scrittura dei valori del duty cycle sugli ESC, come mostrato dalla Figura 5.1.

L'obiettivo di tale filtro è di accelerare la risposta dei motori ad un cambia-

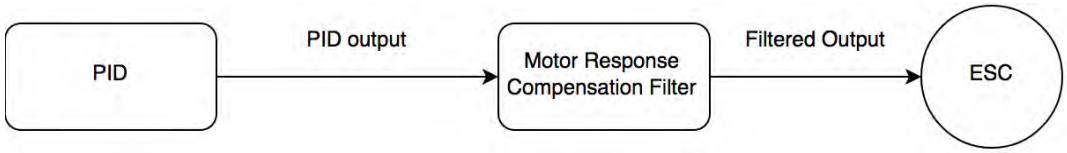

Figura 5.1: Schema a blocchi

mento di velocità in modo da rendere il sistema più reattivo.

Pur non avendo effettuato una stima della funzione di trasferimento tra comando e velocità dei motori, possiamo ipotizzare che i motori abbiano un polo a circa 2Hz.

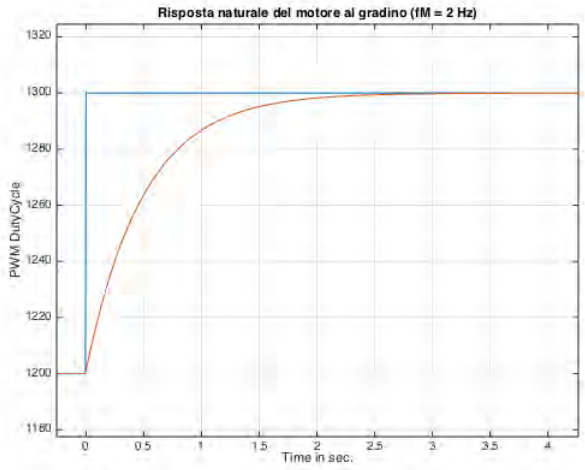

Figura 5.2: Risposta del motore

Nella Figura 5.2 viene mostrata la risposta del motore ad una variazione di 100 ms del dutycycle del segnale PWM inviato agli ESC. Come si può notare il tempo che i motori impiegano per andare a regime è prossimo ai 2 secondi. Il filtro deve perciò invertire la risposta e a questo scopo ci basta disegnare un filtro del primo ordine.

Lo zero A, poiché deve cancellare il polo del motore, sarà posizionato a 2Hz

mentre il polo B, per rendere il filtro realizzabile, sarà superiore. Nel nostro caso viene posizionato a 10Hz.

La funzione di trasferimento nel dominio zeta può essere rappresentata come segue:

$$H(z) = \frac{z - A}{z - B} \tag{5.1}$$

A è calcolato come $1 - (T \times FreqZero)$ mentre B come $1 - (T \times FreqPolo)$, nel nostro caso T è 0.005 secondi, $FreqZero$ 2Hz e $FreqPolo$ 10Hz. In Figura 5.3 è mostrato il posizionamento polo-zero del filtro.

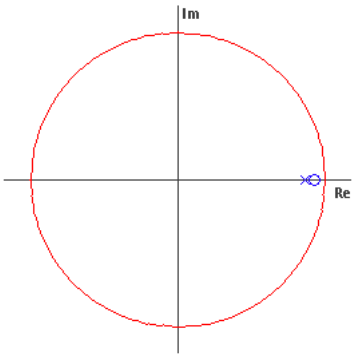

Figura 5.3: Zero Pole Placement. A = 0.99, B = 0.95

Analizzando il filtro come equazioni alle differenze possiamo scriverlo come somma di un filtro IIR e di un FIR. Il seguente diagramma chiarisce la relazione ingresso uscita del filtro.

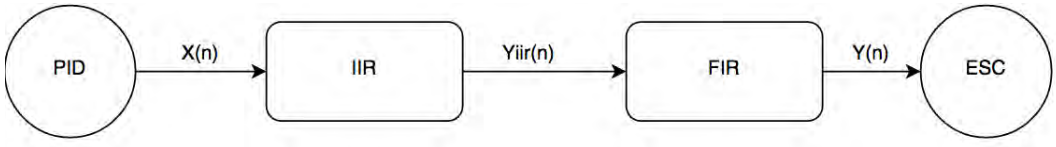

Figura 5.4: Risposta del motore

Il filtro è riassunto dalle equazioni:

$$Y_{IIR}(n) = (1 - B) \times X(n) + B \times Y_{IIR}(n - 1) \tag{5.2}$$

$$Y(n) = (\frac{1}{1 - A}) \times (Y_{IIR}(n) - A \times Y_{IIR}(n - 1)) \tag{5.3}$$

I fattori $(1 - B)$ e $(\frac{1}{1-A})$ rappresentano il guadagno del filtro.
La differenza fra il segnale filtrato e la normale risposta dei motori è mostrata dalla Figura 5.5. Il transitorio si è notevolmente accorciato permettendo ai motori di andare a regime in meno di 0.5 secondi.

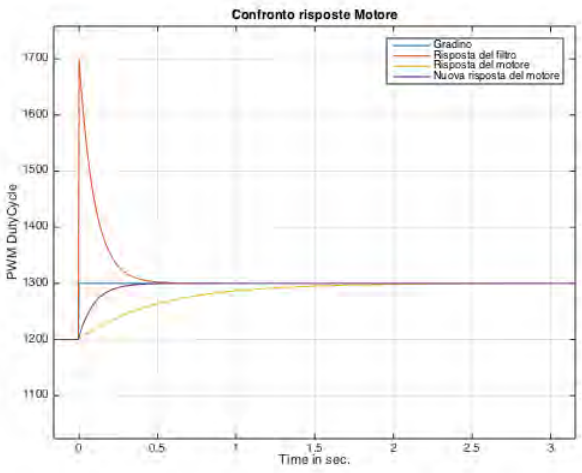

Figura 5.5: Risposta del motore

Capitolo 6

Struttura Firmware

In questo capitolo verranno introdotte le principali caratteristiche dei firmware per microcontrollori e la struttura software utilizzata all'interno del progetto.

6.1 Struttura generale dei firmware per microcontrollori

I firmware per microcontrollori utilizzano una struttura standard composta da due procedure. All'accensione del microcontrollore, viene invocata una procedura il cui compito è quello di inizializzare il microcontrollore e le varie periferiche, in modo che tutto funzioni correttamente ai fini dell'applicazione. In genere, durante questa fase, vengono settati i canali di input/output, viene settato il master-clock e vengono inizializzate la comunicazione e le variabili d'ambiente. L'inizializzazione si può omettere se le impostazioni di default del microcontrollore sono adeguate per eseguire correttamente il lavoro desi-

derato.

Al termine della fase di inizializzazione, il firmware avvia l'esecuzione di un ciclo infinito in cui all'interno vengono eseguite tutte le operazioni necessarie alla svolgimento di un determinato compito. Vengono perciò effettuate le letture degli input, le elaborazioni dei dati raccolti, le attivazioni delle uscite, ecc..

La normale esecuzione progressiva del programma codificato può essere interrotta qualora venga sollevata una eccezione. Il ambito microcontrollori, come nel software, il concetto di eccezione è molto utile e largamente utilizzato. Con questo meccanismo, nel momento in cui una eccezione è sollevata, la normale esecuzione viene messa in stand-by e viene eseguita una funzione, o una procedura, associata al tipo di eccezione. Ad esempio, un sensore provvisto di clock interno, sincronizzato con il master-clock, può inviare un segnale di interrupt al microcontrollore, "avvisando" che i suoi dati sono pronti per essere acquisiti; a questo punto la CPU, informata dall'interrupt, può interrompere momentaneamente l'esecuzione della porzione di programma per avviare la procedura di lettura dei dati del sensore.

Questo meccanismo è utile, tra gli altri, in applicazioni in cui il risparmio di energia è fondamentale, in quanto la CPU può essere "svegliata" al momento più opportuno; oppure quando l'esecuzione del core del programma ha una priorità maggiore rispetto alle altre funzionalità, che possono essere eseguite in maniera asincrona.

6.1.1 Sketch Arduino

Anche i programmi sviluppati per Arduino si adeguano alla struttura appena introdotta. Gli sketch Arduino prevedono le procedure *setup()* e *loop()*, che implementano sostanzialmente la procedure di inizializzazione e il ciclo infinito previste dalla struttura tipica descritta in sezione 6.1.
Il linguaggio di programmazione utilizzato nei progetti che utilizzano Arduino è il C/C++; si può parlare di entrambi i linguaggi, in quanto è possibile utilizzare, o meno, il supporto per le classi che il C++ mette a disposizione. Inoltre, il compilatore utilizzato è dedicato per i microcontrollori della famiglia AVR della Atmel e fa parte di una ben definita toolchain GCC.
L'ambiente di sviluppo di Arduino compie alcune trasformazioni dello sketch principale prima di passarlo al compilatore avr-gcc.
Al momento della compilazione dello sketch, tutti i file con estensione *.ino* vengo concatenati per formare il *main-sketch*, a cui viene aggiunto l'header *Arduino.h* che fornisce tutti le definizioni necessarie per il core standard di Arduino.

6.2 Struttura firmware del Comelicottero

In questa sezione vengono descritte le procedure *setup()* e *loop()* presenti nel firmware adottato per il controllo del drone.

6.2.1 setup()

Come mostra il diagramma di flusso in Figura 6.1, questa procedura ha il compito di inizializzare diverse parti del sistema.

Figura 6.1: Setup del firmware del Comelicottero

Inizializzazione della Seriale

Una comunicazione seriale è fondamentale in fase di progettazione e debugging del codice in quanto permette, attraverso una interfaccia a linea di comando di interagire con l'ATMega collegato tramite la porta USB al pc. Il boudrate è il numero di simboli al secondo che la comunicazione è in grado di gestire, deve essere dimensionato in base alla lunghezza del protocollo utilizzato per comunicare e in base alla quantità di "informazioni" che si desidera scambiare con il microcontrollore; nel nostro caso è stato scelto un baudrate pari a 115200 bit/s che equivale ad 11400 byte/s, ovvero la possibilità di scambiare senza perdite 57 caratteri ad ogni iterazione di loop().

Inizializzazione motori

Inizialmente i motori erano pilotati passando il valore di PWM utilizzando la funzione *analogWrite()* della libreria standard di Arduino. Questa accetta come parametro un intero tra 0 e 255 che determina il dutycycle del PWM. Ci si è accorti che, con questo metodo, la granularità con cui il duty cycle viene

modificato non è abbastanza fine, in quanto una variazione unitaria induce una variazione di velocità del motore troppo elevata. Nel progetto, invece, i motori sono utilizzati come servomotori: il duty cycle è deciso indicando i suoi millisecondi con un numero compreso tra 1000 e 2000 (0-5V). Inoltre, sia le aziende produttrici di droni che alcuni forum specializzati sostengono che per ottenere una buona risposta degli ESC ai cambi di velocità desiderati il PWM deve avere una frequenza di refresh prossima ai 400Hz. Purtroppo, Arduino non è in grado di fornire un numero sufficiente di PWM a queste velocità poiché per aumentare i rate bisogna diminuire il numero di PWM generabili con lo stesso timer. Con opportune modifiche alla libreria *Servo* siamo stati in grado di ottenere la generazione di quattro PWM ad una velocità di refresh di 133Hz che si è rivelata molto più robusta rispetto ai 50Hz standard previsti dalla libreria. In questa fase di inizializzazione vengono quindi chiamate le procedure per fare l'*attach* di ogni motore ad un oggetto Servo e la scrittura di un valore PWM minimo (1100ms) agli ESC che imposti i motori nello stato di *ready*.

Inizializzazione PID

Istanzia i tre controllori PID necessari ed imposta i coefficienti descritti nella sezione 4.3.

Inizializzazione IMU

Vengono settati i valori all'interno dei registri del giroscopio e dell'accelerometro, come descritto nelle sezioni 3.2.1 e 3.2.4.

Start Contatori

Vengono settati le variabili destinate a contare il tempo trascorso fra due iterazioni consecutive del loop.
Queste variabili sono fondamentali per sincronizzare le operazioni di lettura dei sensori e il calcolo delle correzioni da applicare ai motori.

6.2.2 loop()

Il main loop ha il compito di eseguire continuamente le funzioni "vitali" del quadricottero, come la lettura dei dati da sensori e l'algoritmo di controllo.
La figura 6.2 mostra il flusso di esecuzione della procedura.

Empiricamente è stato testato che per completare un ciclo di loop() occorrono meno di 5ms, di cui la lettura dei valori dall'IMU e conseguenti filtraggi impiega circa 3ms mentre l'algoritmo di controllo occupa circa 1ms. E' stato molto importante definire le frequenze di lavoro dei metodi $readIMU()$ e $flightControl()$ poiché da questi dipende il tempo di risposta del drone stesso.
Introduciamo ora i macroblocchi che compongono il loop.

Lettura dei comandi

E' il metodo che legge da seriale i comandi di throttle, yaw, pitch, roll e accensione/spegnimento inviati dall'utente.
Al termine di ogni ciclo di loop viene chiamata automaticamente la procedura *serialEvent()* che interrogata la comunicazione seriale per verificare se una nuova stringa contenente i comandi, inviati dall'applicazione desktop, è

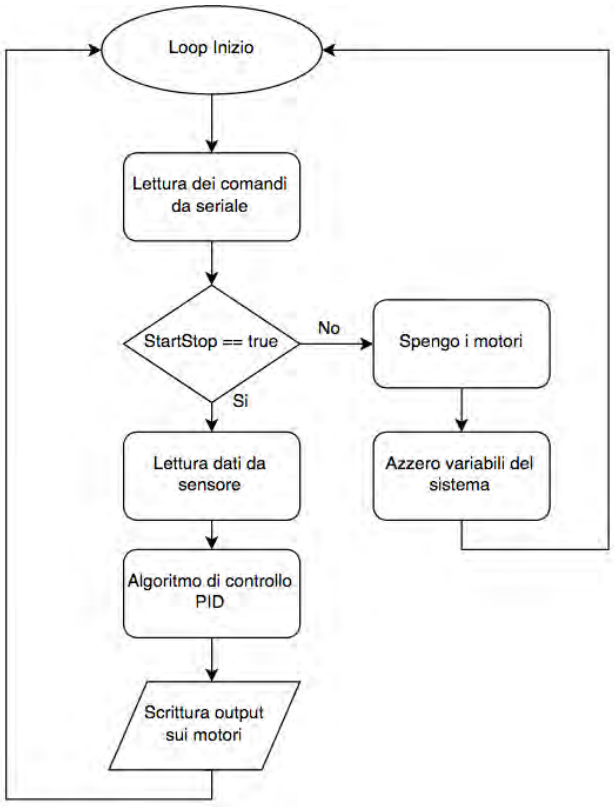

Figura 6.2: Main loop del nostro firmware

disponibile e ben formattata; in caso positivo aggiorna le variabili con i nuovi valori.

Lettura sensori

In questo step viene invocata la funzione *readIMU()*. La procedura controlla, con una comparazione tra il tempo passato dall'ultima iterazione e IMU_COMPUTE_TIMER, se il periodo di refresh dei valori dell'IMU è passato. In caso positivo, chiama le procedure di lettura dei valori dai sensori. Viene effettuato un controllo sul contatore delle letture dell'accelerometro e quando questo è uguale ad *ACC_COMPUTE_TIMER*, vengono compu-

tate le funzioni di arcotangente per il calcolo degli angoli di roll ($degAccX$) e pitch ($degAccY$). Questo controllo permette di effettuare un aggiornamento dei valori degli angoli derivati dal sensore ogni $ACC\_COMPUTE\_TIMER$ volte rispetto all'aggiornamento dei valori del giroscopio.

La funzione $atan()$ richiede come argomento il rapporto tra i valori di accelerazione sugli assi interessati; l'angolo restituito è espresso in radianti e successivamente viene trasformato in gradi. Infine, vengono computati i valori di pitch e roll mediante il filtro complementare.

```
void readIMU() {
    dt = millis() - imuComputeTimer;
    if (dt >= IMU_COMPUTE_TIMER) {

    imuComputeTimer = millis();
    readGyro();

    readAcc();
    if (accCounter % ACC_COMPUTE_TIMER == 0) {
        accXAverage = sumAccX / ACC_COMPUTE_TIMER;
        accYAverage = sumAccY / ACC_COMPUTE_TIMER;
        accZAverage = sumAccZ / ACC_COMPUTE_TIMER;
        // Useful to compute pitch and roll angles
        degAccX = atan(accXAverage / accZAverage) * RAD_TO_DEG;
        degAccY = atan(accYAverage / accZAverage) * RAD_TO_DEG;
        accCounter = 0;
        sumAccX = 0, sumAccY = 0, sumAccZ = 0;
    }
// FIRST ORDER COMPLEMENTARY FILTER

pitchFilter.complementary(&pitchFusion, &degAccX, &gyroYAverage, &dt);
rollFilter.complementary(&rollFusion,  &degAccY, &gyroXAverage, &dt);
}
```

Listing 6.1: Calcolo degli angoli Pitch e Roll

IMU_COMPUTE_TIMER è settato a 5ms e questa procedura viene invocata ad un frequenza di 200Hz, frequenza oltre la quale non ha senso operare poiché verrebbero letti valori identici dai sensori dell'IMU.

Algoritmo di controllo PID

Effettuata la lettura dei sensori, viene invocata la procedura *flightControl()* la quale è incaricata di mappare, nei range di lavoro del quadricottero, i comandi di movimento impartiti dall'utente nelle variabili *throttle, UIr, UIp, UIy*; come viene mostrato nel listato 6.2, se il tempo passato dall'ultima computazione è maggiore di *PID_COMPUTE_TIMER* allora vengono calcolate le nuove correzioni da attuare ai motori per mantenere il drone in hovering o farlo muovere rispetto ai comandi imposti.

```
...
if ((millis() - pidComputeTimer) >= PID_COMPUTE_TIMER) {
      pidComputeTimer = millis();
      if(throttle > MIN_SPEED) {
      rollOut=PIDroll.compute(UIr, rollFusion, gyroXAverage); // ROLL
      pitchOut=PIDpitch.compute(UIp, pitchFusion, gyroYAverage); // PITCH
      yawOut = PIDyaw.compute(UIy, 0, gyroZAverage );     // YAW
}
else {
      resetI();
}
...
```

Listing 6.2: Sezione di codice del calcolo dei PID

Sulla base di altri progetti simili trovati online e informazioni su siti specializzati, la frequenza con cui calcolare la correzione e passare il valore PWM corrispondente agli ESC dovrebbe il più alto possibile. Anche se ciò può

significare scrivere più volte lo stesso valore, la reattività del drone può aumentare notevolmente al cambio di valore. Questa ipotesi è stato avvalorata durante i test effettuati vincolando il drone su un asse di rotazione, dove abbiamo constatato che una velocità operativa di 50Hz rendeva il sistema poco reattivo: il tempo impiegato per potarsi in hovering dopo uno squilibrio non era accettabile.

Per poter stabilire infine la frequenza ottimale di invocazione della procedura *flightControl*() abbiamo testato diversi valori, iniziando dal limite inferiore di 50Hz aumentando fino a 200Hz. Abbiamo scelto di adeguare il calcolo dei valori di PID alla velocità di lettura dei sensori, impostando a 5ms anche il valore di PID_COMPUTE_TIMER.

Scrittura sugli ESC

Una volta completato il calcolo delle correzioni da applicare ai quattro motori, prima dell'effettiva variazione del segnale PWM inviato agli ESC, i valori di uscita dei PID vengono filtrati dal filtro di compensazione. Il loop mostrato dal listato 6.3 mostra l'applicazione del filtro al segnale *motorValue[nMotor]* proveniente dai PID e la scrittura sugli ESC dell'output del filtro.

Le variabili *motorIIROld* e *motorIIROldOld* servono per tenere traccia dell'uscita del filtro IIR fino a 2 istanti di tempo prima dell'applicazione del filtro al tempo corrente.

```
for ( int nMotor = 0; nMotor < N_MOTORS; nMotor++ ) {                    1
    motorIIROldOld[nMotor] = motorIIROld[nMotor];                        2
    motorIIROld[nMotor] = motorIIR[nMotor];                              3
    motorIIR[nMotor] = (1-B)*motorValue[nMotor] + B*motorIIROld[nMotor]; 4
    motorValue[nMotor] = (1/(1-A))*(motorIIR[nMotor] - A*motorIIROldOld[ 5
        nMotor]);
```

```
    writeToMotor(motors[nMotor], constrain((int)motorValue[nMotor],
        MIN_SPEED, MAX_SPEED));
}
```

Listing 6.3: Implementazione Motor response compensation filter

Le costanti A, B sono definite nel file di configurazione del nostro progetto come segue:

```
#define F_NUM  2  // Hz
#define F_DEN  10 // Hz
#define T (IMU_COMPUTE_TIMER/1000.0) //Sampling time in sec
#define A (1- (T * F_NUM))
#define B (1- (T * F_DEN))
}
```

Listing 6.4: Definizione poli e zeri del filtro

Nel nostro caso T, il tempo di campionamento utilizzato per l'algoritmo di controllo ed il filtro di compensazione dei motori è di 5ms. Di conseguenza $A = 0.99$ mentre $B = 0.95$

Spegnimento Motori

Quando il drone non è in modalità On, si impone agli ESC un dutycycle sotto la soglia minima operativa in modo da spegnere i motori.

Azzeramento variabili di sistema

Nel momento in cui i motori sono spenti, vengono azzerate variabili del sistema come filtri, ultimi comandi di input dall'utente.

Capitolo 7

Interfaccia Grafica

7.1 GUI

Come scritto in precedenza, l'idea iniziale del progetto era quella di comandare il drone da Pc, sfruttando una connessione WiFi. Durante numerosi test, ci si è resi conti che i ritardi introdotti dalla libreria Bridge dell'Arduino Yun, rallentava il processo di controllo di assetto. In Figura 7.1 è mostrata l'interfaccia grafica, realizzata in *C#* usata durante i primi test in WiFi. Da questa interfaccia era possibile inviare comandi di throttle, pitch, roll, yaw e di leggere parametri relativi alla velocità dei motori, ai sensori e alle uscite dei PID grazie ai quali abbiamo testato il corretto funzionamento del sistema.

Quando si è decisi di adottare la comunicazione in seriale, questa interfaccia non è più stata utilizzata.

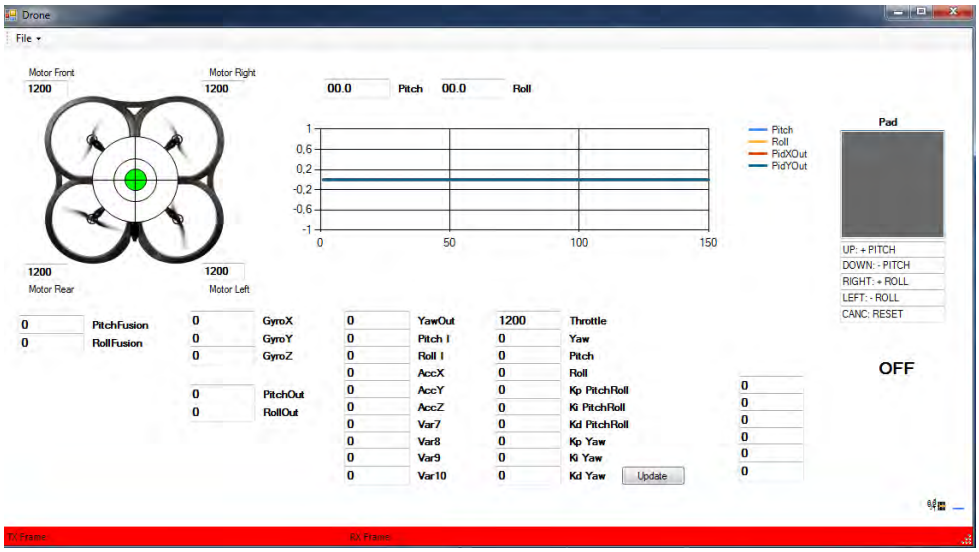

Figura 7.1: Interfaccia Grafica per il controllo

7.2 Controllo con Gamepad

Durante i test ci si è resi conto che impartire i comandi al drone tramite tastiera risultava scomodo e poco pratico. Abbiamo pensato di utilizzare un gamepad collegato tramite porta USB al computer. E' stato realizzato un script in Python con lo scopo di inoltrare all'Arduino i comandi eseguiti con il gamepad.

In Figura 7.2 è mostrata l'immagine del joypad. Tramite il tasto ANALOG B si incrementa/decrementa il throttle (movimento su-giù) e si controlla lo yaw (movimento destra-sinistra), mentre con il tasto ANALOG A si controlla il pitch (movimento su-giù) e il roll (movimento destra-sinistra). Il tasto 1 è stato utilizzato per l'accensione del drone mentre il tasto 2 per lo spegnimento. I valori di ogni asse del gamepad variano da -1 ad 1 e sono di tipo float. Mentre i tasti di accensione/spegnimento posso assumere solo i valori 0 o 1.

Figura 7.2: joypad

Di seguito è riportato il codice dello script un Phyton.

```
1  import pygame
2  import serial
3  import time
4  from sys import argv
5
6  #ask for Arduino port
7  if len(argv) == 1:
8      print 'Enter the port of Arduino device:'
9      port = raw_input()
10 else:
11     port = argv[1]
12
13 define serial port
14 print 'Selected Port: /dev/%s' % port
15
16 port = 'cu.usbmodem411'
17 serialPort = '/dev/' + port
18 # serialPort = '/dev/ttyUSB0'
19 boudRate = 115200
20
21 #open serial connection
22 connected = 0
23 print 'Try to connect:'
24 while connected == 0:
25     try:
26         ser = serial.Serial(serialPort, boudRate, timeout=1)
27         time.sleep(2)
28         print 'Connection established at port %s' % serialPort
29         connected = 1
30     except serial.SerialException:
31         pass
32
33
34 #initialize Joystick
35 num_button = 8
36 pygame.init()
37 j = pygame.joystick.Joystick(0)
38 j.init()
39 print 'Initialized Joystick : %s' % j.get_name()
40
41 #get 4 joystick data and 2 buttons
42 def getData():
43     try:
44         while True:
```

```
45              message = '['
46              pygame.event.pump()
47              for i in range(0, j.get_numaxes()):
48                  if i != 0: # avoid undefined axis
49                      #add 5 to avoid negative numbers
50                          message += '%.2f' % (j.get_axis(i) + 5)
51                          message += ';'
52              for i in range(0, num_button):
53                  if i == num_button - 1:
54                      message += '%s' % j.get_button(i)
55                      message += ']\n'
56              ser.write(message)
57              #print message
58              time.sleep(.1)
59      except KeyboardInterrupt:
60      j.quit()
61  #main routine
62  if __name__ == '__main__':
63      while True:
64          getData()
```

Bibliografia

[1] Arduino, "Arduino Yún." http://arduino.cc/en/Main/ArduinoBoardYun?from=Products.ArduinoYUN.

[2] Arduino, "Arduino Uno." http://arduino.cc/en/Main/arduinoBoardUno.

[3] DJI, "DJI 2212 Motors." http://www.dji.com/product/tuned-propulsion-system.

[4] DJI, "DJI 9" Propellers." https://store.dji.com/product/self-tightening-propellers.

[5] HobbyKing, "IMAX B6 50W 5A Charger/Discharger 1-6 Cells." http://www.hobbyking.com/hobbyking/store/uh_viewItem.asp?idProduct=12922.

[6] eCalc, "Multicopter Calculator." http://www.ecalc.ch/xcoptercalc.php?ecalc&lang=en.

[7] Giovanni Bernardo, "Cosa sono, come funzionano e a cosa servono gli Accelerometri." http://www.settorezero.com/wordpress/cosa-sono-come-funzionano-e-a-cosa-servono-gli-accelerometri/, 26 Sept 2011.

[8] Analog Devices, "ADLX345: 3-Axis, $\pm 2g/ \pm 4g/ \pm 8g/ \pm 16g$ Digital Accelerometer Data Sheet." http://www.analog.com/static/imported-files/data_sheets/ADXL345.pdf.

[9] "Tilt sensing using a three-axis accelerometer." http://www.freescale.com/files/sensors/doc/app_note/AN3461.pdf.

[10] STMicroelectronics, "L3G4200D MEMS motion sensor: three-axis digital output gyroscope." http://www.st.com/web/en/resource/technical/document/datasheet/CD00265057.pdf.

[11] S. Colton and F. Mentor, "The balance filter," *Presentation, Massachusetts Institute of Technology*, 2007.

[12] Wikipedia, "Pid Controller." http://en.wikipedia.org/wiki/PID_controller.

[13] Oscar Liang, "Quadcopter PID Explained and Tuning." http://blog.oscarliang.net/quadcopter-pid-explained-tuning.

[14] Ashwini Miryala, Kyle Scarlett, Zachary Zell, Brandon Kountz, "PID Downslides and Solutions." https://controls.engin.umich.edu/wiki/index.php/PIDDownsides#Windup.

[15] Brett Beauregard, "PID reset windup." http://brettbeauregard.com/blog/2011/04/improving-the-beginner%E2%80%99s-pid-reset-windup.

[16] Doug J. Cooper, "PID Control and Derivative on Measurement." http://www.controlguru.com/wp/p76.html.

[17] Michigan University, "PID Compensator with Bilinear Approximation." http://ctms.engin.umich.edu/CTMS/index.php?aux=Extras_PIDbilin.

[18] Wikipedia, "Controllo Digitale." http://it.wikipedia.org/wiki/Controllo_digitale.

[19] Wikipedia, "Quadrirotore." http://it.wikipedia.org/wiki/Quadrirotore.

[20] Christopher J. Fisher, "Using an Accelerometer for Inclination Sensing." http://www.analog.com/static/imported-files/application_notes/AN-1057.pdf.

[21] DJI, "DJI F450 Frame." http://www.dji.com/product/flame-wheel-arf/spec.