

LAVORO DI Maturità

FISICA E APPLICAZIONI DELLA MATEMATICA

Quadricottero autostabilizzante

Autore:

Nevò MIRZAI HAMADANI

Supervisore:

Prof. Mattia BERGOMI



LICEO CANTONALE LUGANO 1

Anno Scolastico 2019/2020

Abstract

Il progetto consiste nella costruzione di un drone quadricottero a stabilizzazione autonoma. Viene studiato il metodo di estrazione ed elaborazione dati dal sensore di movimento MPU-6050, oltre che il suo funzionamento specifico, e la trasformazione dell'errore ricavato in un segnale di tipo PWM trasmesso ai motori grazie al sistema di controllo PID.

Indice

Indice abbreviazioni e glossario	III
Elenco delle figure	IV
Elenco dei ritagli di codice	IV
1 Premessa	1
2 Introduzione	2
3 Materiali e costruzione	3
3.1 Materiali	3
3.1.1 Prototipo	3
3.1.2 Modello definitivo	4
3.2 Costruzione	4
3.2.1 Prototipo	4
3.2.2 Modello definitivo	6
3.3 Schema elettrico	8
4 Sensore di accelerazione	10
4.1 Accelerometro	10
4.1.1 Princípio Capacitivo	10
4.1.2 Princípio piezoelettrico	11
4.1.3 Princípio piezoresistivo	12
4.2 Giroscopio	12
4.3 MPU-6050	12
5 PID	13
5.1 Introduzione	13
5.2 Parte proporzionale	14
5.3 Parte integrale	15
5.4 Parte derivativa	15
5.5 Adattamento per il progetto	16
6 Codice	17
6.1 Diagramma di flusso	17
6.2 Dichiarazioni iniziali	18
6.3 Setup e funzioni iniziali	19
6.4 Acquisizione dati dall'MPU-6050	22
6.5 Trasformazione dei dati in angoli	23

6.6 Trasformazione degli angoli tramite PID	24
6.7 Output finale	24
7 Risultati	29
8 Discussione	29
9 Conclusione	29
Riferimenti bibliografici	30
10 Appendice: Codici completi	31
10.1 Prototipo	31
10.2 Drone definitivo	36

Indice abbreviazioni e glossario

Acronimi

ESC Electronic Speed Control

IMU Inertia Measurement Unit

MEMS Micro Electro-Mechanical Systems

MPU Micro Processing Unit

PID Proportional Integral Derivative

PWM Pulse-Width-Modulation

Costanti tecniche

g	Accelerazione di gravità media terrestre	9.806 65 m/s ²
-----	--	---------------------------

Elenco delle figure

1	Prototipo da vista frontale	5
2	Prototipo dall'alto	5
3	Drone definitivo senza la parte superiore	6
4	Drone definitivo completo	7
5	Schema elettrico classico	8
6	Schema elettrico “visivo”	9
7	Funzionamento accelerometro capacitivo [2]	10
8	Schema di un condensatore [3]	11
9	MPU-6050 [6]	12
10	Flusso generale del sistema PID [9]	14

Elenco dei ritagli di codice

1	Librerie	18
2	Dichiarazione dei servomotori	18
3	Varie dichiarazioni	18
4	Somma degli errori del sensore	19
5	Divisione per ottenere il dato medio	20
6	Gli errori del sensore vengono stampati	20
7	Inizializzazione I^2C	21
8	Inizializzazione motori	21
9	Richiamo della funzione d'errore	21
10	Accensione tramite bottone	22
11	Richiesta dei registri di accelerazione	22
12	Ottenimento delle accelerazioni lineari sugli assi	22
13	Ottenimento dell'angolo attraverso le accelerazioni	23
14	Ottenimento degli angoli con le accelerazioni angolari	23
15	Ottenimento degli angoli finali unendo quelli trovati in precedenza .	23
16	Processo PID	24
17	Massimo e minimo del valore PID totale	24
18	Inizio del loop for contenente la x	24
19	Ottenimento dell'output finale	27
20	Valori massimi e minimi dell'output	28
21	L'output finale viene mandato ai motori	28
22	Prototipo	31
23	Drone definitivo	36

1 Premessa

Sin dalle elementari la materia che per me surclassava tutte le altre, senza discussioni, era la scienza. Questo comprende sia come funziona la natura intorno a noi, sia come operano gli oggetti creati dall'uomo. Per questi ultimi in particolare, sviluppai una passione che coltivai in particolar modo in un progetto di ated4Kids, un'associazione che organizza attività per giovani riguardanti la tecnologia. Quello a cui presi parte aveva come obiettivo l'ottenimento del Guinness World Record per il: "maggior numero di droni costruiti da minorenni e pilotati assieme per il maggiore tempo possibile". Riuscimmo nell'intento, e nel montaggio e settaggio del drone imparai molto sul modellismo e sul funzionamento di questi fantastici velivoli. Da qui mi appassionai al modellismo in generale, e presi spunto per quello che è diventato il mio progetto di LAM.

2 Introduzione

Alla base di questo progetto c'è l'analisi del funzionamento dei droni a 4 eliche, le loro modalità di volo e, in particolare, il modo in cui si stabilizzano autonomamente in aria. Sarà dunque necessario analizzare una tecnica in grado di stabilizzare un sistema, che in questo caso è il PID (Proportional Integral Derivative). Oltre a questo verrà anche visto il funzionamento di un accelerometro nel dettaglio, e il metodo di estrazione ed elaborazione dei dati presi dall'MPU 6050, un sensore contenente un accelerometro e un giroscopio.

Tutto questo al fine di costruire un drone quadricottero, in grado di ricevere dati dal sensore di movimento, analizzarli e trasformarli in output che andranno inviati ai motori grazie a un codice apposito su un Arduino UNO (una scheda open-source, quindi modificabile dall'utente), per infine restare il più stabile possibile a mezz'aria.

Per raggiungere questo obiettivo è prima necessario costruire un prototipo, una struttura più semplice su cui eseguire test di ogni tipo, sia per capire come sfruttare la teoria matematica e fisica integrata nel modo più efficiente possibile, che per evitare spiacevoli incidenti irreversibili (a meno di costose riparazioni). Queste conoscenze acquisite verranno poi applicate al modello definitivo, facilitandone il completamento.

Nei prossimi capitoli verranno approfonditi vari aspetti del progetto. Prima di tutto la parte prettamente "materiale", quindi i materiali utilizzati e la metodologia di costruzione. Si passerà in seguito alla parte più teorica, dapprima col funzionamento di un sensore di accelerazione lineare, poi col metodo dell'acquisizione ed elaborazione dati del sensore, e come questi vengano trasformati in output sotto forma di PWM (Pulse-Width-Modulation, un tipo di segnale che permette di variare la potenza dell'output) grazie a un controllo basato sul sistema PID. Un capitolo intero verrà dedicato al codice del drone, parte essenziale per il suo funzionamento. Infine verranno discussi i dati raccolti, dando spunti per possibili miglioramenti futuri e verranno date delle conclusioni sull'intero progetto.

3 Materiali e costruzione

3.1 Materiali

3.1.1 Prototipo

Per la costruzione del prototipo di un asse autostabilizzante il materiale utilizzato è stato:

- 1 asse di legno 50cm x 5cm x 3cm con un foro di trapano al centro del piano 50cm x 3cm
- 2 sostegni di legno 6cm x 8.5cm x 3cm con un “mezzo foro” al centro dell'estremità più corta del piano 6cm x 8.5cm
- 2 motori brushless con le relative eliche
- 2 ESC 420 Lite
- 1 sensore di accelerazione MPU-6050
- 1 scheda Arduino Uno
- 1 base con circuito elettrico integrato del drone Flame Wheel F450
- 1 batteria Lipo 3S 11.1V/3300
- 1 breadboard
- 1 vite di grosse dimensioni
- Cavetti maschio-maschio e femmina-maschio
- Nastro adesivo e biadesivo
- Fascette di plastica
- Spago

3.1.2 Modello definitivo

Per la costruzione del modello definitivo invece, mi sono avvalso principalmente del drone montabile F450, con l'aggiunta di altre componenti:

- Componenti del drone Flame Wheel F450:
 - 4 braccia di supporto motori
 - 4 motori brushless con le relative eliche
 - 4 ESC 420 Lite
 - Base con circuito elettrico integrato
 - Copertura della base
 - Viti di varie grandezze
- 1 scheda Arduino Uno
- 1 sensore di accelerazione MPU-6050
- 1 batteria a 9V
- 1 bottone che si può trovare nello Starter Kit di Arduino
- 4 supporti rialzanti per la struttura
- 1 batteria Lipo 3S 11.1V/3300
- Cavetti maschio-maschio e femmina-femmina
- Fascette di plastica, nastro biadesivo

3.2 Costruzione

3.2.1 Prototipo

L'ideazione del prototipo in sé non è durata molto, la sua costruzione un poco di più, divisa nell'arco di 2 giornate. Questo però per arrivare solo al prototipo iniziale, modificato continuamente durante i test per renderlo più stabile ed efficiente.

L'asse di legno del prototipo, come si può notare nella figura 1, tiene un asse di legno poggiato su 2 sostegni grazie a una vite che trapassa da parte a parte la barra. Agli estremi sono collocati i motori brushless, attaccati con del nastro biadesivo e fissati fermamente con dello spago che passa attraverso dei passaggi alla base dei motori. Diverse volte si sono staccati i motori per l'elevata propulsione prima

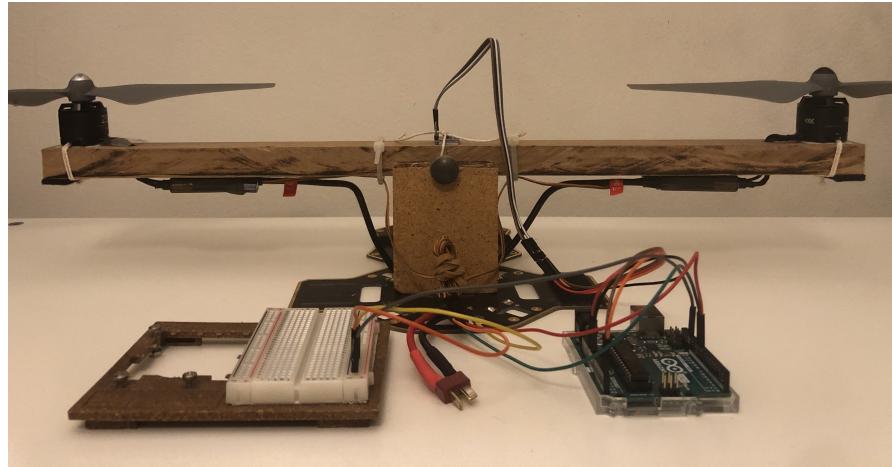


Figura 1: Prototipo da vista frontale

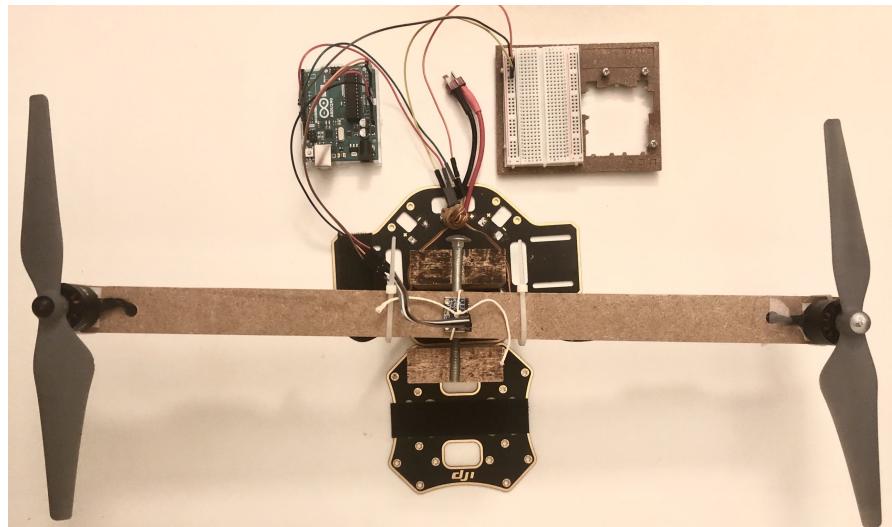


Figura 2: Prototipo dall'alto

di legarli con dello spago, non posso dire che non ci siano stati danni materiali (fili tagliati o parti strappate) o a persone (striature principalmente, ma le mie dita hanno rischiato più volte). Sotto le estremità dell'asse sono stati attaccati dei feltrini per ammorbidente i colpi subiti durante i test.

Ogni motore ha un'elica ed è collegato a un ESC (scheda di controllo della propulsione dei motori), incollati sotto all'asse con del nastro biadesivo per comodità e sicurezza dei cavi (vedi figura 1). Gli ESC hanno due cavi per l'alimentazione e il Ground (si tratta di un cavo nero più spesso che si divide in 2 all'estremità), collegati al circuito elettrico che si collega a sua volta a una batteria per alimentar-

li. Altri 2 cavi degli ESC si collegano poi all'Arduino Uno tramite la breadboard (uno al Ground e l'altro all'input PWM). L'MPU-6050 è attaccato al centro dell'asse anch'esso con del nastro biadesivo e collegato con dei cavi maschio-femmina all'Arduino.

3.2.2 Modello definitivo

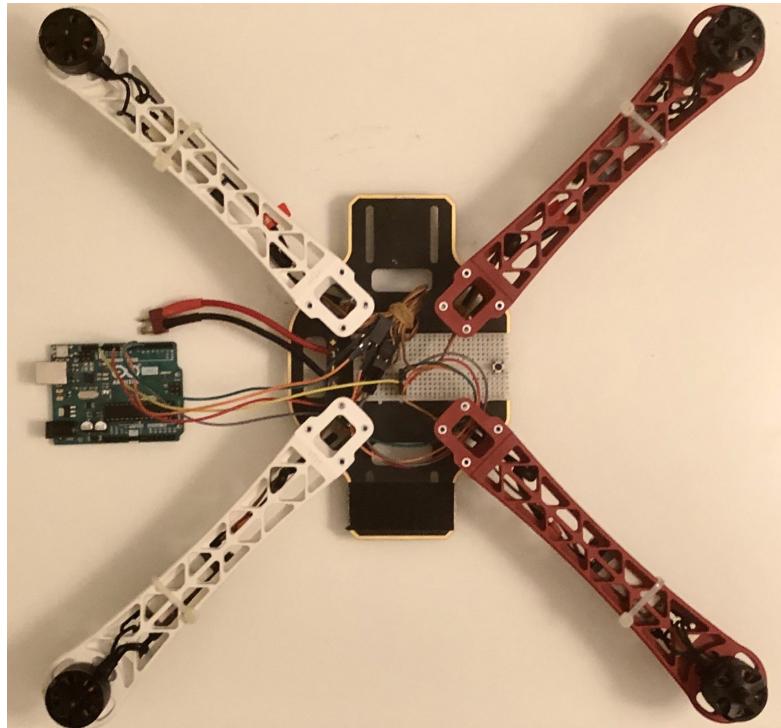


Figura 3: Drone definitivo senza la parte superiore

Per il modello definitivo la costruzione è stata molto più semplice, sia perché l'ho affrontata durante il progetto di ated4Kids (vedi sezione 1), sia perché dopotutto si tratta di una struttura fatta per essere montata in un certo modo, quindi non ci sono grandi migliorie da fare, se non magari riguardanti la protezione dei cavi.

Le 4 braccia si trovano tra la base, che funge anche da circuito elettrico per i motori, e la parte superiore del drone, fissate grazie a delle viti. Tra la base e la parte superiore si trova la breadboard; lo schema elettrico si può trovare alla sottosezione 3.3. Sulla breadboard si inseriscono diversi cavi e il bottone, che servirà ad attivare la struttura. Al piano superiore si trova l'Arduino UNO alimentato da una batteria 9V e l'MPU-6050, il primo attaccato con del velcro e il secondo fissato con delle viti. Posta sotto la parte inferiore si trova la batteria, fissata con del velcro così

da essere facilmente sostituibile con una carica. Alle estremità delle braccia sono fissati con delle viti i motori brushless, a cui vengono avvitate le eliche. I motori sono collegati alla scheda Arduino tramite degli ESC, riposti al di sotto di ogni braccio, in modo simile a come si è fatto col prototipo (vedi figura 1). Sotto alla struttura sono poi fissati 4 supporti rialzanti, in modo da non dover decollare e atterrare ogni volta con troppo poco distacco da terra, il che potrebbe rendere gli atterraggi più difficoltosi e rovinare le eliche. Tutti i fili lunghi, e che quindi possono in un qualche modo dare fastidio, sono infine legati assieme con degli elastici o delle fascette.



Figura 4: Drone definitivo completo

3.3 Schema elettrico

Per comprendere meglio come sono collegate tra loro le componenti del drone, qui di seguito si trovano 2 schemi elettrici del drone, uno di genere classico e uno più “visivo”.

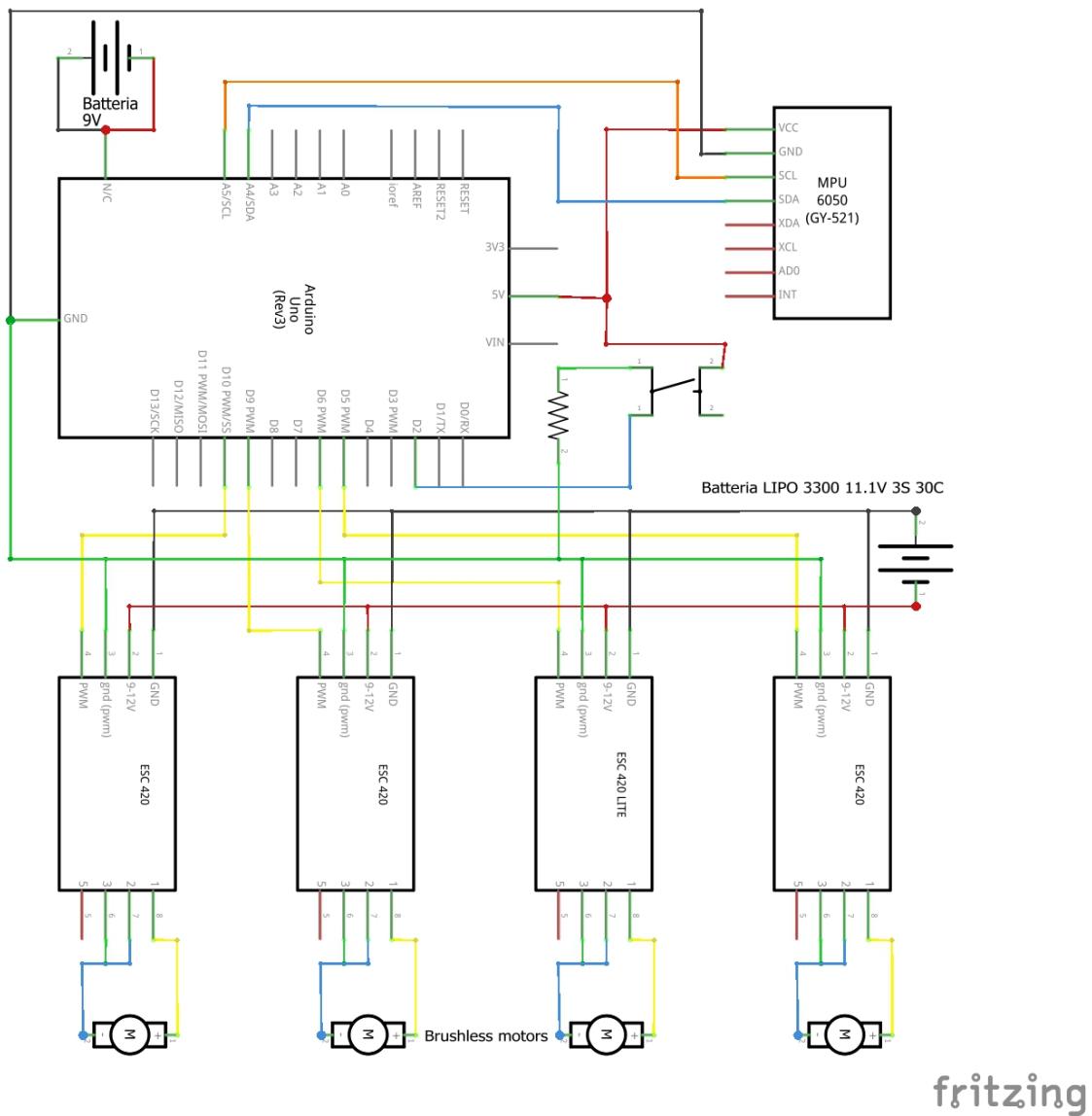


Figura 5: Schema elettrico classico

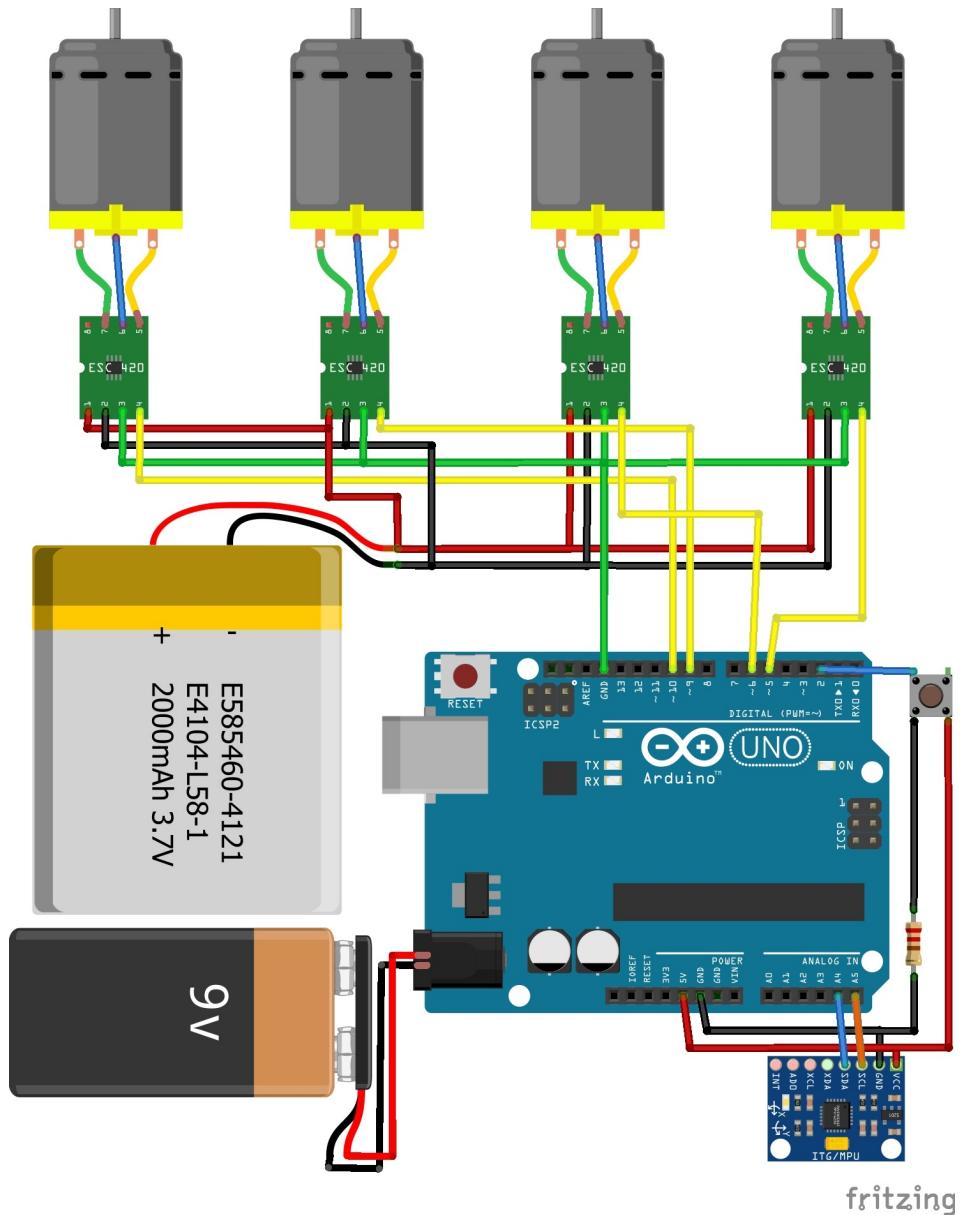


Figura 6: Schema elettrico “visivo”

4 Sensore di accelerazione

Il sensore di accelerazione è inteso come un componente in grado di rilevare le accelerazioni dei, o attorno, ai propri assi. Comprende quindi 2 sensori al suo interno: un accelerometro, in grado di rilevare l'accelerazione *lineare*, e un giroscopio, in grado di misurare l'accelerazione *angolare*.

4.1 Accelerometro

Un accelerometro è uno strumento misuratore delle **accelerazioni lineari**, in forma statica, come la gravità, o dinamica, come le vibrazioni. Il principio di base si fonda sul rilevare le reazioni di una massa (nel sensore è di dimensioni molto minute) quando viene accelerata, quindi si tratta di trovare la sua inerzia in base al suo spostamento rispetto alla posizione di riposo. Ciò vuol dire che è sensibile ai cambiamenti di velocità, come anche alla forza applicata. Infatti:

$$\vec{a} = \frac{\vec{V}_f - \vec{V}_i}{\Delta t} \quad \vec{a} = \frac{\vec{F}}{m}$$

La prima equazione è la formula generale per ottenere l'accelerazione media \vec{a} , sottraendo la velocità iniziale \vec{V}_i a quella finale \vec{V}_f e dividendo il tutto per il tempo trascorso Δt . La seconda invece rappresenta il secondo principio della dinamica, che afferma che la forza totale risultante delle forze agenti su un corpo imprime su di esso un'accelerazione nella direzione e nel verso della forza applicata (per questo motivo \vec{a} e \vec{F} , oltre alle velocità, sono vettori).

Gli accelerometri sul mercato possono utilizzare diversi principi o metodi di funzionamento, alcuni più utili per alcune situazioni di altri. Qui di seguito sono elencati i 3 che solitamente vengono ritenuti come i più importanti, ma ce ne sono anche altri, che utilizzano per esempio la luce, delle bobine e altro ancora. [1]

4.1.1 Principio Capacitivo

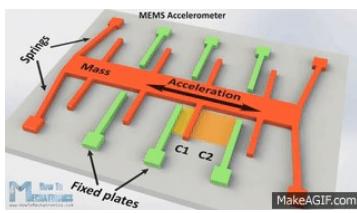


Figura 7: Funzionamento accelerometro capacitivo
[2]

Qui la massa ha la forma di pettine, in pratica un'unione di armature singole (vedi figura 7). È mobile, sospesa su una membrana (poiché deve potersi spostare, seppur in modo limitato) a una certa distanza da un altro pettine che però è fisso sul sensore. I 2 pettini sono conduttori, e questo li rende anche condensatori, quindi soggetti ai cambiamenti di capacità, in quanto questi mutamenti sono proporzionali alla variazione di distanza tra i condensatori [1]. Infatti:

$$C = \frac{Q}{\Delta V}$$

La capacità C è uguale alla carica Q fratto la differenza di potenziale ΔV . Dato che ΔV è anche uguale all'intensità del campo elettrico E moltiplicato per la distanza d tra le piastre, la formula diventa:

$$C = \frac{Q}{Ed}$$

Dunque si deduce che la capacità tra le piastre (o armature) è inversamente proporzionale alla loro distanza. Quello che fa un accelerometro capacitivo è quindi misurare la capacità delle sue armature e mandare un output che ne è direttamente proporzionale (minore è la distanza, maggiore è la capacità e maggiore è l'accelerazione lineare).

Gli accelerometri che usano questo principio sono per esempio i sensori MEMS, come anche l'MPU-6050 utilizzato in questo progetto, e sono i più diffusi tra tutti i tipi di accelerometro, questo è dovuto alla loro alta sensibilità a poco prezzo. Li troviamo infatti in: macchine, smartphones, aerei, ecc. Senza di loro il nostro stile di vita cambierebbe radicalmente.

4.1.2 Principio piezoelettrico

Questo metodo si basa sul fatto che alcuni cristalli, sottoposti a una pressione, generano un segnale elettrico proporzionale alla forza applicata: questo si chiama effetto piezoelettrico diretto (l'effetto piezoelettrico inverso avviene invece quando una carica passa attraverso il cristallo, deformandolo [4]). Dunque viene posta una o più masse, che in base all'accelerazione subita imprimono una forza sul cristallo, che a sua volta genererà un segnale elettrico che il sensore andrà a percepire, e, in modo proporzionale al segnale elettrico, il sensore invierà un output di accelerazione.

Dato che a venir compresso è un cristallo, quindi un materiale rigido e con una costante elastica elevata, l'accelerometro piezoelettrico ha poca sensibilità, e soprattutto ha dei problemi a rilevare accelerazioni costanti (dopo non molto tempo questa non verrà più rilevata), ma è utile a rilevare grandi accelerazioni, poiché si deforma molto meno facilmente rispetto ad altri accelerometri, ad esempio di tipo capacitivo. Infatti, gli accelerometri di tipo piezoelettrico sono utili a misurare accelerazioni dinamiche, che possiamo trovare negli shock e nelle vibrazioni, che per l'appunto sono accelerazioni elevate e di breve durata. [1]

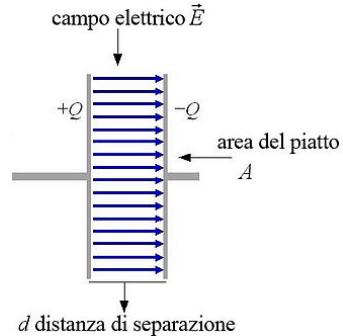


Figura 8: Schema di un condensatore [3]

4.1.3 Principio piezoresistivo

Il principio utilizzato negli accelerometri piezoresistivi riguarda il fatto che alcuni elementi cambiano di resistività elettrica in base alla loro deformazione. Questo accorciamento/allungamento varia la resistività del materiale sensibile, e collegando a questo materiale un circuito in grado di misurare le differenze di resistenza, si può risalire alla lunghezza della deformazione e quindi all'accelerazione.

Questo principio non è da confondersi con quello degli estensimetri, che funzionano grazie a un principio simile ma per l'appunto non uguale. Questi variano la propria resistenza solo perché varia la lunghezza della sezione del conduttore. [5]

4.2 Giroscopio

Un giroscopio è invece uno strumento misuratore delle **accelerazioni angolari**. Esso utilizza l'effetto Coriolis. A una massa m viene impresso un movimento angolare, che può essere anche di tipo vibratorio. Una velocità angolare ω perpendicolare a questo movimento causerà una forza proporzionale F_{cor} sia alla velocità angolare e perpendicolare a entrambi sia al movimento di vibrazione. La forza di Coriolis vale:

$$F_{cor} = 2m\omega v$$

Questa viene rilevata attraverso i metodi descritti nel punto 4.1, dove però i sensori non sono posti nel centro del sensore, ma leggermente spostati, così da poter rilevare la F_{cor} di rotazione.

4.3 MPU-6050

Il sensore di accelerazione lineare e angolare MPU-6050 è un sensore IMU (Inertia Measurement Unit), che comprende al suo interno un accelerometro e un giroscopio MEMS (Micro Electro-Mechanical Systems), entrambi a 3 assi. Ci sarebbe anche un sensore di temperatura, ma è inutile per questo progetto.

È particolarmente utile a questo progetto poiché ci permette di avere sempre a disposizione i 3 angoli che indicano l'inclinazione del dispositivo, quindi del drone, e questo con l'utilizzo di un solo componente. [7]

L'MPU-6050 è uno dei pochi sensori ad avere questa doppia funzionalità di accelerometro e giroscopio, e per questo viene ampiamente usato in diversi ambiti. Il suo funzionamento è più preciso di altri sensori simili sul mercato; questo è dovuto al fatto che contiene per cia-

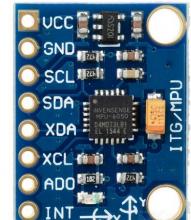


Figura 9: MPU-6050 [6]

scun canale un convertitore digitale/analogico a 16-bit. Riesce dunque a catturare contemporaneamente i valori degli assi X, Y e Z.

Per i collegamenti fisici del sensore MPU si veda lo schema elettrico al punto 3.3, mentre per i passaggi del codice al riguardo si vedano i punti 6.3 e 6.4.

5 PID

5.1 Introduzione

Il PID (Proportional-Integral-Derivative) è una tipologia di controllo che si basa sulla retroregolazione negativa, questo gli dà la capacità di regolare il sistema costantemente in base a degli input (o errori) ricevuti, tramite degli output (in questo caso il valore di potenza trasmesso ai motori); è molto comodo nei sistemi di stabilizzazione di un automa. Come suggerisce il nome, è composto da 3 parti: una proporzionale, che chiamerò P , una integrale I e una derivativa D . La parte più importante del sistema è l'output che si ricava alla fine unendo queste 3 componenti, poiché sarà quello a decidere se il sistema resterà stabile o meno. Il segnale d'uscita verrà determinato dall'errore (e in questo caso, dato che l'angolo desiderato è 0, l'errore equivale al valore dell'angolo d'inclinazione). Più precisamente l'output viene regolato dal valore dell'errore attuale (parte proporzionale), i valori precedenti dell'angolo (parte integrale) e la velocità di variazione del segnale (parte derivativa). In pratica è come se la parte proporzionale si occupasse del presente, cosa sta succedendo ora (punto 5.2), quella integrale del passato, utilizzando dunque anche il valore precedente (punto 5.3), e la parte derivativa guarda alla differenza tra presente e passato, in un certo senso per prevedere il segnale futuro (punto 5.4). A ognuna delle 3 azioni viene assegnato un peso che equivale alle costanti K , e infine questi valori vengono sommati algebricamente e trasformati nel valore finale, quindi output necessario. [8]

$$\sum = K_P \cdot e(t) + K_I \int e(t) dt + K_D \frac{de(t)}{dt}$$

Dove \sum è l'output “grezzo”, non ancora trasformato nel modo desiderato, ed $e(t)$ è l'errore al tempo t , dunque l'input.

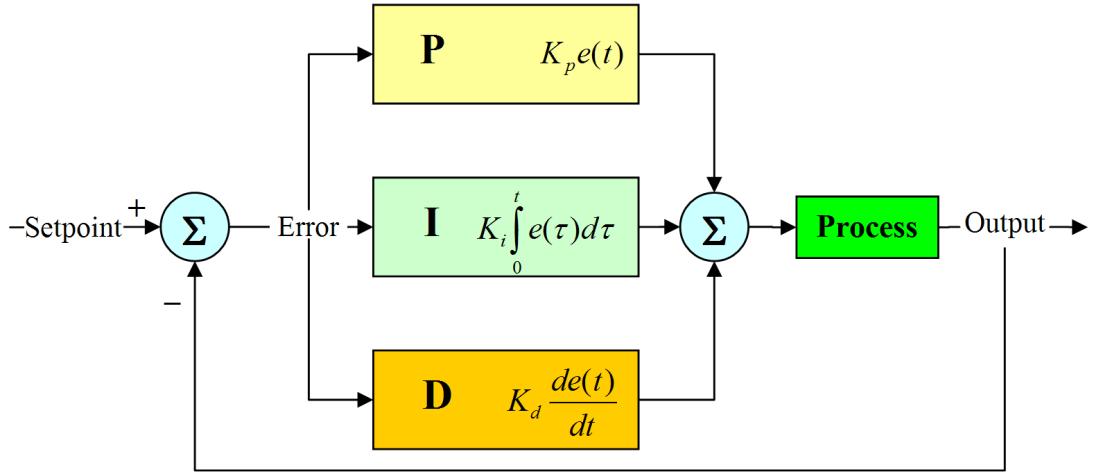


Figura 10: Flusso generale del sistema PID [9]

5.2 Parte proporzionale

$$P = K_P \cdot e(t)$$

L’azione proporzionale P segue l’andamento dell’errore e moltiplicato per la costante K_P . Qui viene preso in considerazione solo il puro valore di e nel preciso istante in cui viene letto. La costante K_P è la più grande tra le 3 costanti, quindi la parte principale della stabilizzazione verrà da qui, anche se si tratta, se presa da sola, di una stabilizzazione grossolana. Senza le altre 2 parti, infatti, è quasi impossibile arrivare a un’oscillazione nulla, e questo per un semplice motivo. Quando si arriva all’angolo desiderato (prendo come esempio il mio progetto), cioè 0, l’output sarà altrettanto nullo, ma l’asse avrà ancora tenuto un certo momento angolare dovuto alla sua inerzia e alla sua velocità precedente, (dato che la costante è abbastanza grande, verrà data una propulsione elevata anche con angoli minimi, dunque già con una leggera differenza dall’angolo desiderato l’asse comincerà ad oscillare). Quindi questa componente di controllo da sola non raggiunge uno stato di equilibrio, o almeno non lo fa in tempo efficiente, è quindi necessario aggiungere le parti che seguono. [10]

5.3 Parte integrale

$$I = K_I \int e(t) dt$$

L'azione integrale I è proporzionale all'integrale nel tempo dell'errore e , calcola dunque l'area sotto la curva dell'andamento dell'input, moltiplicandolo per una costante K_I . Questa variabile è da subito molto più interessante di quella proporzionale. Si può notare immediatamente che l'output non è per forza nullo se lo è l'input. Quello che fa l'integrale infatti è contrastare l'errore prendendo come riferimento i valori passati, e risulta quindi molto più semplice raggiungere un sistema di equilibrio. Molti sistemi di auto-pilotaggio utilizzano infatti un sistema di stabilizzazione esclusivamente PI, che quindi prende in considerazione solo la parte proporzionale e integrale, e pur tralasciando la parte derivativa funzionano discretamente bene (ovviamente questo dipende dalla mansione del sistema). La costante K_I però deve rimanere abbastanza piccola, altrimenti influenzerebbe in maniera troppo aggressiva il sistema. [10]

5.4 Parte derivativa

$$D = K_D \frac{de(t)}{dt}$$

Quest'ultima, l'azione derivativa D , ha un ruolo più di “previsione” del futuro segnale d'errore $e(t + \Delta t)$, dove Δt rappresenta la differenza di tempo tra una misurazione e l'altra. Per arrivare a prevedere questo valore, questa componente del PID sfrutta la velocità di variazione dell'errore e . Infatti non attende che l'input persista per un certo lasso di tempo (variabile integrale) o che aumenti in modo rilevante (variabile proporzionale). Calcola invece la pendenza tra il valore attuale e quello precedente e agisce di conseguenza. Ovviamente non si può raggiungere la derivata esatta del punto in questione, vorrebbe dire prevedere il futuro, questa è solo un'approssimazione. Con questa descrizione ci si rende anche conto che la parte derivativa di questo sistema di controllo non prende in considerazione il valore dell'errore, ma la sua tangente (considerando $e(t)$ come funzione). È per questo motivo che questa costante non può venir utilizzata da sola (senza quindi il supporto di una parte proporzionale o integrale) in un qualsiasi sistema di autostabilizzazione (in base alla sua mansione specifica poi, può venir usata da sola in alcuni sistemi, ma raramente). Un valore standard per la costante K_D è solitamente piccola rispetto alla costante proporzionale K_P , in questo progetto però il suo peso (anche se non ancora esatto) è di $\frac{3}{4}$ la costante K_P . [10]

5.5 Adattamento per il progetto

Dopo aver compreso bene i principi del PID, esso risulta non troppo complicato da applicare a un progetto simile al mio. Basta convertire il linguaggio puramente matematico in codice decifrabile dalla scheda Arduino. L'azione proporzionale ovviamente resta uguale, quindi:

$$pid_P = K_P \cdot e(t)$$

Quella integrale diventa:

$$pid_I = pid_I + (K_I \cdot e(t))$$

Quella derivativa invece muta in:

$$pid_D = K_D \frac{e(t) - e(t - \Delta t)}{\Delta t}$$

Infine si sommano i valori per ottenere il PID finale:

$$PID = pid_P + pid_I + pid_D$$

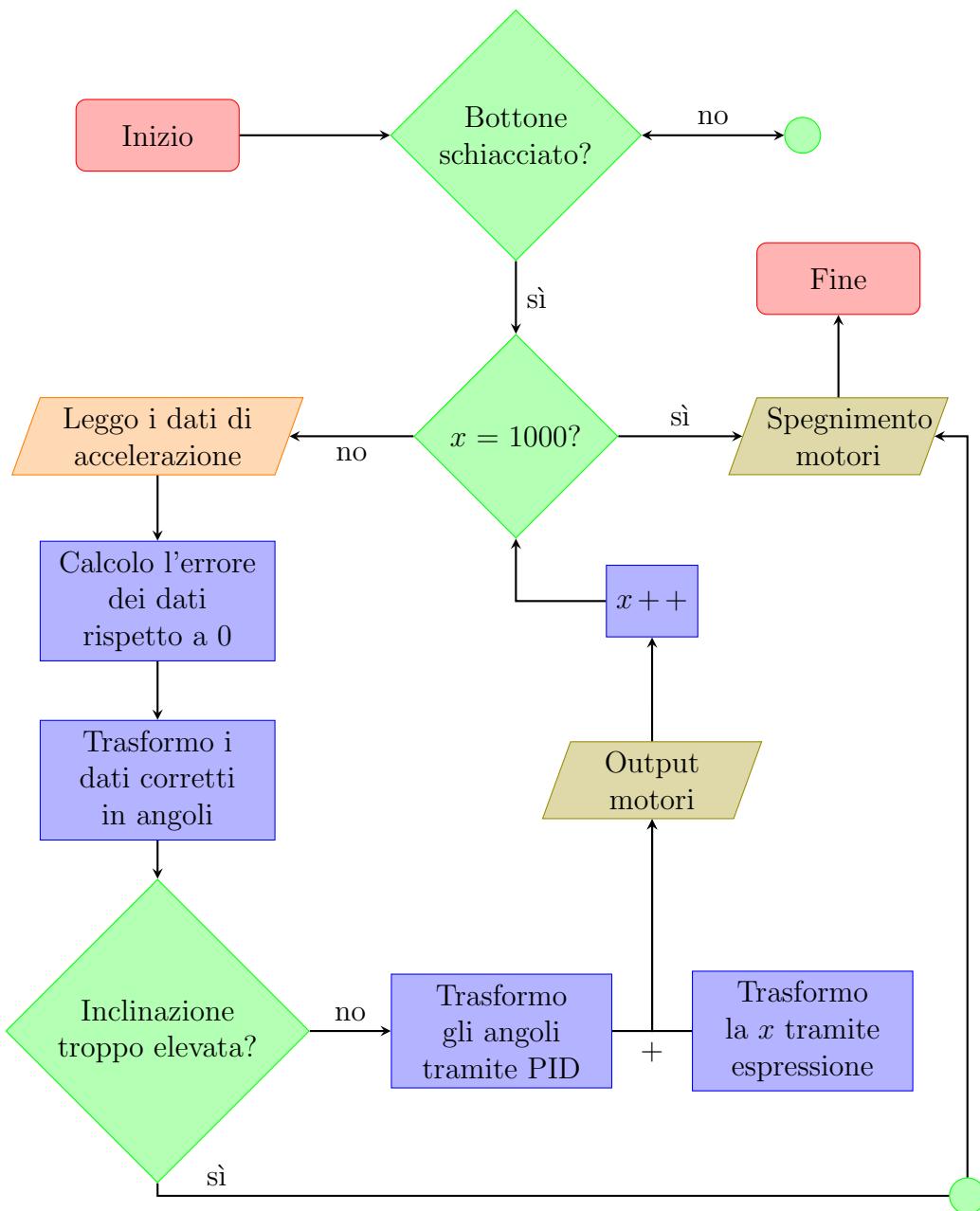
Dove $e(t - \Delta t)$ è il valore antecedente all'errore attuale $e(t)$ e Δt è il tempo passato dal loop precedente.

Nel codice finale del quadricottero ci sono 3 valori pid per ogni asse, e la somma finale, per questo motivo, è un po' più complicata di quella mostrata sopra (la si può ritrovare nel ritaglio di codice numero 19). Infine a questo loop si aggiunge un valore che cambia nel tempo e che fa alzare il drone da terra (si possono trovare più dettagli nella sottosezione 6.7).

Nel prototipo invece, dato che si ha solo un angolo da monitorare, si ha soltanto un valore PID finale, il che rende in buona parte più semplice il codice da applicare. A questo viene aggiunto un valore per tenere le eliche accese, e che però in questo caso non ha bisogno di cambiare nel tempo.

6 Codice

6.1 Diagramma di flusso



6.2 Dichiarazioni iniziali

```
1 #include <Wire.h>
2 #include <Servo.h>
```

Ritaglio di codice 1: Librerie

Per far funzionare il codice sarà necessario includere la libreria `Wire.h`, che permette di utilizzare l' I^2C , cioè un tipo di comunicazione seriale utile a trasmettere informazioni tra due circuiti integrati, e che in questo caso serve a far comunicare l'MPU-6050 con l'Arduino UNO. Oltre a questa, includo anche la libreria `Servo.h`, dato che sarà necessario l'utilizzo di servomotori.

```
1 Servo motAd;
2 Servo motAs;
3 Servo motDd;
4 Servo motDs;
```

Ritaglio di codice 2: Dichiarazione dei servomotori

Ora vengono dichiarati i servomotori. Riguardo alla loro nomenclatura: “mot” indica il fatto che sono dei motori, “A” vuol dire “Avanti”, quindi che si trova nella parte anteriore del drone, “D” significa “Dietro”, mentre “d”, questa volta non maiuscola, indica che si trova sulla destra, e “s” sulla sinistra.

```
1 // Acc = accelerazione lineare
2 // Gir = accelerazione angolare
3 float AccX, AccY, AccZ, GirX, GirY, GirZ;
4 float AccErroreX, AccErroreY, AccErroreZ, GirErroreX, GirErroreY,
     GirErroreZ;
5
6 // Vari valori di tempo
7 double tpassato, tempo, tloop;
8
9 // Variabili legate agli angoli
10 double girAngolo[2];
11 double accAngolo[2];
12 double angTotale[3];
13
14 double errore[3], errorePrec[3];
15
16 // Per la funzione d'errore iniziale
17 int c = 0;
18
19 // Costanti e variabili delle parti PID
20 float kp[3];
21 float ki[3];
22 float kd[3];
23
24 double PID[3];
```

```

25 double pidP[3], pidI[3], pidD[3];
26
27 double pwmAd, pwmAs, pwmDd, pwmDs;
28
29 // Variabili rimanenti
30 bool capovolto = false;
31 bool bottone = false;
32 int x = -350;

```

Ritaglio di codice 3: Varie dichiarazioni

Subito dopo vengono dichiarate le variabili e le costanti che verranno utilizzate nel codice, come i vari valori di accelerazione, di tempo, per gli angoli di ogni asse ecc. Da notare che diverse variabili sono vettori a 3 componenti, questo poiché il loro uso verrà ripetuto per ogni asse (componente 0 = asse X = movimento avanti/indietro; 1 = asse Y = spostamento destra/sinistra; 2 = asse Z = rotazione su sé stesso). Si sarebbe potuta utilizzare questa nomenclatura anche per alcune delle variabili iniziali (ad esempio AccX, AccY ecc, righe 7-9), ma ho deciso di non farlo per rendere il codice più leggibile.

6.3 Setup e funzioni iniziali

```

1 void erroreMPU()
2 {
3 while (c < 500)
4 {
5     // Errore accelerometro
6     Wire.beginTransmission(0x68);
7     Wire.write(0x3B);
8     Wire.endTransmission(false);
9     Wire.requestFrom(0x68, 6, true);
10
11    AccX = (Wire.read() << 8 | Wire.read()) / 16384.0 ;
12    AccY = (Wire.read() << 8 | Wire.read()) / 16384.0 ;
13    AccZ = (Wire.read() << 8 | Wire.read()) / 16384.0 ;
14
15    // Sommo gli errori
16    AccErroreX += (atan((AccY) / sqrt(pow((AccX), 2)) * 180 / PI));
17    AccErroreY += (atan(-(AccX) / sqrt(pow((AccY), 2)) * 180 / PI));
18    AccErroreZ += (AccZ-1);
19
20    // Errore del giroscopio
21    Wire.beginTransmission(0x68);
22    Wire.write(0x43);
23    Wire.endTransmission(false);
24    Wire.requestFrom(0x68, 6, true);
25

```

```

26     GirX = Wire.read() << 8 | Wire.read();
27     GirY = Wire.read() << 8 | Wire.read();
28     GirZ = Wire.read() << 8 | Wire.read();
29
30     // Sommo gli errori
31     GirErroreX += (GirX / 131.0);
32     GirErroreY += (Giry / 131.0);
33     GirErroreZ += (GirZ / 131.0);
34     c++;
35 }
```

Ritaglio di codice 4: Somma degli errori del sensore

Dopo la dichiarazione delle variabili che verranno usate, ho elaborato una funzione per calcolare l'errore medio dei dati di accelerazione letti dal sensore (il metodo per ricavare i dati viene approfondito nel sottocapitolo 6.4.). In pratica vengono sommati i dati di ogni parametro per un certo numero di volte (qui 500). Visto che il dato desiderato è quasi sempre 0 per un sensore fermo (tranne per l'accelerazione in Z, che vuole essere 1 poiché la forza di gravità che agisce su di esso è equivalente per l'appunto a $1g$), non bisogna modificare la somma in nessun modo.

```

1 AccErroreX /= 500;
2 AccErroreY /= 500;
3 AccErroreZ /= 500;
4
5 GirErroreX /= 500;
6 GirErroreY /= 500;
7 GirErroreZ /= 500;
```

Ritaglio di codice 5: Divisione per ottenere il dato medio

Questa somma viene poi divisa per il numero di volte che vengono letti i parametri e si ottengono gli errori medi di ogni accelerazione.

```

1 Serial.print("AccErroreX: ");
2 Serial.print(AccErroreX);
3 Serial.print("  AccErroreY: ");
4 Serial.print(AccErroreY);
5 Serial.print("  GirErroreX: ");
6 Serial.print(GirErroreX);
7 Serial.print("  GirErroreY: ");
8 Serial.print(GirErroreY);
9 Serial.print("  GirErroreZ: ");
10 Serial.println(GirErroreZ);
11 }
```

Ritaglio di codice 6: Gli errori del sensore vengono stampati

Vengono poi stampati per poterli controllare durante i test.

```

1 void setup()
2 {
3   Serial.begin(19200);
4
5   pinMode(2, INPUT);
6
7   Wire.begin();
8   Wire.beginTransmission(0x68);
9   Wire.write(0x6B);
10  Wire.write(0x00);
11  Wire.endTransmission(true);

```

Ritaglio di codice 7: Inizializzazione I^2C

void setup: Si arriva quindi al setup, dove comincio col `Serial.begin` per poter leggere i dati dall'Arduino durante i test, e subito dopo dichiaro il pin numero 2, che servirà per verificare la premuta del pulsante. Avvio poi la comunicazione I^2C tramite `Wire.begin()`, specificando che voglio averla con l'MPU-6050 immettendo nelle parentesi della funzione `Wire.beginTransmission()` il registro per richiamare il sensore, cioè lo 0x68. Viene utilizzato poi il comando `Wire.write(0x6B)`, la cui funzione è di “mettere in fila” i dati richiesti dall'Arduino all'MPU. Uso quindi la funzione `Wire.endTransmission(true)` per riferire che la fine della trasmissione è “vera”, e quindi deve smettere di mantenerla (verrà poi ripresa nella parte 6.4). [11]

```

1 motAd.attach(9);
2 motAs.attach(10);
3 motDd.attach(6);
4 motDs.attach(5);
5
6 motAd.writeMicroseconds(1150);
7 motAs.writeMicroseconds(1150);
8 motDd.writeMicroseconds(1150);
9 motDs.writeMicroseconds(1150);

```

Ritaglio di codice 8: Inizializzazione motori

Nel setup devono venire assegnati i pin da cui partiranno gli output per ogni motore attraverso in comando `attach`. Assegno quindi il primo valore ai motori con la funzione `writeMicroseconds`, che servirà per inizializzarli. Il valore assegnato infatti non attiva ancora ai motori, 1150 è il valore PWM minimo che si può assegnare al tipo di motori brushless utilizzati in questo progetto, il massimo si dovrebbe aggirare attorno a 2000, ma non è rilevante per questo progetto.

```
1 erroreMPU();
```

Ritaglio di codice 9: Richiamo della funzione d'errore

Richiamo in seguito la funzione di cui si parlava a inizio sezione per calcolare l'errore medio del sensore.

```

1 while (bottone == false)
2 {
3     bottone = digitalRead(2);
4 }
```

Ritaglio di codice 10: Accensione tramite bottone

Utilizzo un bottone per far chiudere il setup, leggo il valore che mi da il pin 2 a cui è collegato con `analogRead`, se positivo è stato schiacciato e posso andare avanti. Lo scopo del bottone è di far partire la fase `void loop`.

6.4 Acquisizione dati dall'MPU-6050

```

1 Wire.beginTransmission(0x68);
2 Wire.write(0x3B);
3 Wire.endTransmission(false);
4 Wire.requestFrom(0x68, 6, true);
```

Ritaglio di codice 11: Richiesta dei registri di accelerazione

Comunico con l'MPU come spiegato nel punto 6.3, ma cambiando il registro richiesto. Questa volta il registro dovrà essere quello da cui parte a leggere i dati di accelerazione lineare che serviranno, con i dati di accelerazione angolare, a calcolare gli angoli. Nel caso dell'accelerazione lineare, per l'MPU-6050, il primo registro è 0x3B [12]. Aggiungo poi soltanto un comando per chiedere i registri al sensore, cioè `Wire.requestFrom()`, dove nelle parentesi immetto il registro che richiama l'MPU, quindi di nuovo lo 0x68, e poi la quantità di registri che voglio partendo dal 0x3B, che sono 6. Leggo dunque i dati con la funzione `Wire.read()`, con cui posso leggere un registro alla volta, che corrisponde a 8 dei 16 bit di un parametro, quindi la metà di esso.

```

1 AccX = (Wire.read() << 8 | Wire.read()) / 16384.0;
2 AccY = (Wire.read() << 8 | Wire.read()) / 16384.0;
3 AccZ = (Wire.read() << 8 | Wire.read()) / 16384.0;
```

Ritaglio di codice 12: Ottenimento delle accelerazioni lineari sugli assi

Ho bisogno quindi di unirli a 2 a 2 in una variabile `float` da 16 bit, e dato che i primi 8 bit di ogni parametro si trovano nei registri dispari, posso unirli a coppie. Utilizzo per questo compito il *bitshift left* “`<<i`”, che permette di spostare un certo numero di bit a sinistra di un numero *i* di posizioni, lasciando i restanti alla seconda parte del parametro, che aggiungo col *bitwise or* `|`, che permette di unire assieme le 2 parti. Divido infine per 16384, il valore che riferisce il datasheet dell'MPU-6050 [12] per ottenere il valore in $\frac{m}{s^2}$. Qui è mostrato il procedimento per la parte di accelerazione lineare, che è identico per l'accelerazione angolare tranne

per la divisione finale, che viene eseguita stavolta per 131, per ottenere i gradi al secondo.

6.5 Trasformazione dei dati in angoli

```
1 accAngolo[0] = (atan(AccY / sqrt(pow(AccX, 2))) * 180 / PI);
2 accAngolo[1] = (atan(-AccX / sqrt(pow(AccY, 2))) * 180 / PI);
```

Ritaglio di codice 13: Ottenimento dell'angolo attraverso le accelerazioni

Per calcolare l'angolo con le accelerazioni (in X e Y) utilizzo un calcolo trigonometrico:

$$\theta = \arctan\left(\frac{Acc}{Acc_{\perp}}\right) \frac{180^o}{\pi}$$

Dove θ è l'angolo cercato, Acc è una delle 3 accelerazioni lineari dei 3 assi, e Acc_{\perp} è una delle due rimanenti, che per forza di cose è perpendicolare alla prima. Converto poi il risultato in radanti per poterlo avere in gradi, e tolgo il valore di errore trovato in precedenza. Si può notare che questa formula viene utilizzata infatti anche nella funzione di errore in 6.3, questo per semplici motivi di semplificazione, per non dover andare a rifare tutto il calcolo nel loop ogni volta.

```
1 girAngolo[0] += GirX * tpassato;
2 girAngolo[1] += GirY * tpassato;
```

Ritaglio di codice 14: Ottenimento degli angoli con le accelerazioni angolari

Per la parte riguardante il giroscopio il calcolo è più semplice: abbiamo già l'angolo per secondo, quindi basta moltiplicare per il tempo passato dal loop precedente e ricaviamo l'angolo.

```
1 angTotale[0] = 0.98 * girAngolo[0] + 0.02 * accAngolo[0];
2 angTotale[1] = 0.98 * girAngolo[1] + 0.02 * accAngolo[1];
3 angTotale[2] += GirZ * tpassato;
```

Ritaglio di codice 15: Ottenimento degli angoli finali unendo quelli trovati in precedenza

Combiniamo infine questi 2 per ottenere l'angolo più preciso possibile, moltiplicando la parte del giroscopio per 0.98 e aggiungendo la parte dell'accelerometro moltiplicata per 0.02 (sono arrivato a fare in questo modo tramite una serie di test, dove risultano molto più precisi i dati ricevuti dal giroscopio) per l'asse X e Y, per Z ho utilizzato soltanto i valori del giroscopio (è risultato problematico acquisire l'angolo in Z con i valori di accelerazione lineare).

6.6 Trasformazione degli angoli tramite PID

```

1      for (int i=0; i<3; i++)
2      {
3          // Parte Proporzionale
4          pidP[i] = kp[i]*errore[i];
5
6          // Parte Integrale
7          pidI[i] += (ki[i]*errore[i]);
8
9          // Parte Derivativa
10         pidD[i] = kd[i]*((errore[i] - errorePrec[i])/tpassato);
11
12         // Somma algebrica per ottenere il PID finale
13         PID[i] = pidP[i] + pidI[i] + pidD[i];

```

Ritaglio di codice 16: Processo PID

Trasformo i vari errori attraverso il PID come spiegato nel punto 5.5, utilizzo poi un `for` per ripetere il processo 3 volte, una per ogni asse. Sommo infine le varie trasformazioni per ottenere un valore `PID[i]` totale.

```

1      if(PID[i] < -300)
2      {
3          PID[i] = -300;
4      }
5      if(PID[i] > 300)
6      {
7          PID[i] = 300;
8      }
9      PID[i] = map(PID[i], -300, 300, -250, 250);

```

Ritaglio di codice 17: Massimo e minimo del valore PID totale

Do anche un valore minimo e massimo al PID, così da mantenere dei valori di sicurezza per non ottenere un output troppo alto o troppo basso, e utilizzo la funzione `map()` per distribuire questi valori su un intervallo che va da -250 a 250.

6.7 Output finale

```

1 while (x<1200)
2 {

```

Ritaglio di codice 18: Inizio del loop for contenente la `x`

All'inizio del `void loop()` faccio partire un ciclo `while` con una variabile `x` che va da -350 a 1200. Il suo scopo è di, tramite un'espressione, trasformare la `x` in un valore che possa far alzare e scendere il drone. Il valore iniziale è negativo

perché durante i test veniva cambiato continuamente in base alle mie necessità, e alla fine quello più comodo ho trovato fosse -350 , che mi concede un lasso di tempo con i motori spenti per potermi preparare al volo del drone. In modo simile sono arrivato al valore finale 1200 . Ho cominciato con 1000 e poi ho cambiato man mano fino ad arrivare al numero finale. Spiegando qui di seguito però come ho trovato i valori utilizzerò il valore iniziale di 1000 con cui ho trovato le equazioni necessarie al progetto.

Per cambiare i valori della x c'è bisogno in questo caso di una funzione di secondo grado, quindi di una parabola (visto che il valore deve salire e poi scendere) fino ad un certo punto, poi, dato che l'atterraggio con i valori finali di una parabola è troppo brusco (l'accelerazione è troppo alta), continuo con un'iperbole, o almeno con la parte della funzione dove il valore scende in modo non troppo violento. L'equazione della parabola è della forma:

$$f(x) = ax^2 + bx + c$$

Mentre quella dell'iperbole sarà invece del tipo:

$$g(x) = \frac{d}{x + e} + f$$

Per trovare i parametri a , b e c dell'iperbole ho proceduto nel seguente modo: so che il valore minimo nel punto $x = 0$ che voglio dare è 1350 , valore per cui i motori girano con una potenza discreta, dato che l'accensione dei motori è già cominciata (il valore iniziale è negativo). Questo valore è la c , che aggiungerò alla fine. Ora, so che voglio un valore massimo di 1570 , che è uguale a $1350 + 220$, e visto che ancora non prendo in considerazione la c , il valore massimo diventa 220 , che è quindi il vertice V della parabola. Sapendo che $V = \frac{-b^2+4a}{4a}$, allora diventa:

$$220 = \frac{-b^2 + 4a}{4a} \quad (1)$$

Ho inserito alla fine della funzione `while` un `delay` di 15 millisecondi, in questo modo l'output ai motori durerà (non prendendo in considerazione la parte negativa dove ancora non si alza il drone): $1000 \cdot 0.015 = 15$ secondi. La seconda equazione del sistema posso ricavarla proprio da questa informazione, poiché $f(x) = 0$ quando $x = 0$ o $x = 1000$. Quindi:

$$0 = 1000000a + 1000b \quad (2)$$

Risolvendo il sistema:
$$\begin{cases} a \cong \frac{-219}{250000} = -0.00088 \\ b \cong \frac{219}{250} = 0.88 \end{cases}$$

La parabola finale è quindi:

$$f(x) = -0.00088x^2 + 0.88x + 1350$$

L'iperbole invece ho dovuto trovarla in base alla parabola appena trovata. Per non avere cambiamenti bruschi di velocità, il modo più logico per eseguire questo "cambio" è di prendere il valore della x della seconda funzione da un punto dove le due funzioni hanno uguale valore e la stessa pendenza. Quindi so che devono avere la stessa tangente in quel punto. Derivando le equazioni, quella della parabola e quella dell'iperbole, otteniamo:

$$\begin{cases} f'(x) = -0.00176x + 0.88 \\ g'(x) = -\frac{d}{(x+e)^2} \end{cases}$$

Voglio che il punto di tangenza sia (837; 1470), valore che in realtà ho trovato inizialmente quando ancora non mi serviva un valore esatto, e ho trovato più veloce (anche se ovviamente più impreciso) provare manualmente a modificare un'iperbole che si avvicinasse alla parabola che avevo trovato. Questo per poter provare subito se funzionava l'idea che ho avuto di suddividere la funzione finale in due per ottenere un atterraggio più morbido, e in caso di insuccesso di scartare immediatamente l'idea. Tornando alla spiegazione, ora basta inserire 837 nella prima derivata per ottenere la pendenza della tangente, che eguallo alla seconda derivata e risolvo rispetto a d :

$$f'(x) = -0.00176 \cdot 837 + 0.88 = -0.593$$

$$-0.593 = -\frac{d}{(837+e)^2}$$

$$d = 0.593 \cdot (837+e)^2$$

Ora, sapendo che l'iperbole passa per il punto $y = 1470$, e che voglio che il parametro f sia uguale a c (così da far tendere l'iperbole a 1350), ottengo un sistema di equazioni:

$$\begin{cases} d = 0.593 \cdot (837+e)^2 \\ d = 120 \cdot (837+e) \end{cases}$$

Eguagliando le 2 posso ottenere il parametro e :

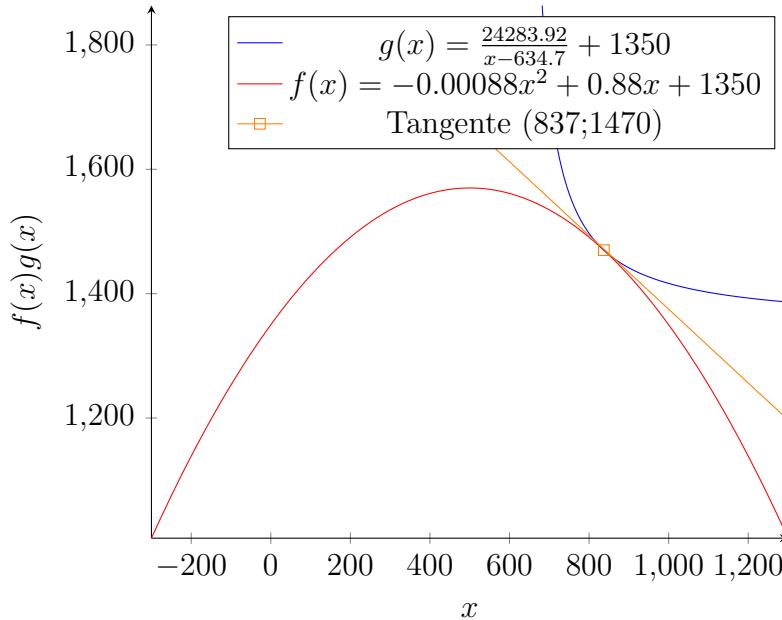
$$496.34 + 0.593e = 120$$

$$e \cong 634.7$$

E ora manca solo d :

$$d = 120 \cdot (837 + e) \cong 24283.92$$

Grafico delle funzioni $f(x)$ e $g(x)$



```

1      if  ((x>837)  && (x<1198))
2      {
3          X = (24283.92/(x-634.739)) + 1350;
4      }
5
6      else
7      {
8          X = (-0.00088*x*x)+(0.88*x)+1350;
9      }
10
11     pwmAd = X - (2*PID[0]+PID[2])/4 -13;
12     pwmDs = X + (2*PID[0]-PID[2])/4;
13     pwmAs = X - (2*PID[1]-PID[2])/4;
14     pwmDd = X + (2*PID[1]+PID[2])/4 - 17;

```

Ritaglio di codice 19: Ottenimento dell'output finale

Ora inserisco queste funzioni come espressioni per cambiare continuamente l'output che verrà assegnato ai motori in base al valore di x . A questo aggiungo ovviamente il valore PID per la stabilizzazione. Dato che l'MPU è posto in diagonale, avendo in faccia il motore Ad (avanti destra), i motori a cui assegno i valori PID sono 2 per l'asse X (Ad e Ds), 2 per l'asse Y (As e Dd) e 2 coppie per l'asse

Z (quindi tutti e 4). Per diminuire la rotazione nell'asse Z, quindi in un certo senso su se stesso, il drone ha bisogno di poter controllare l'output di 2 motori in diagonale, che hanno la rotazione delle eliche nella medesima direzione e con cui posso controllare il momento angolare del drone.

```
1 if(pwmAd < 1120)
2 {
3     pwmAd = 1120;
4 }
5 if(pwmAd > 2000)
6 {
7     pwmAd = 2000;
8 }
```

Ritaglio di codice 20: Valori massimi e minimi dell'output

Devo poi dare un valore massimo a tutti gli output, così da non superare una certa soglia che può a sua volta portare rischi di sicurezza (il procedimento qui descritto viene ripetuto per tutti gli output).

```
1 motAd.writeMicroseconds(pwmAd);
2 motAs.writeMicroseconds(pwmAs);
3 motDd.writeMicroseconds(pwmDd);
4 motDs.writeMicroseconds(pwmDs);
```

Ritaglio di codice 21: L'output finale viene mandato ai motori

Infine mando gli output corretti a ogni motore con `writeMicroseconds`.

7 Risultati

Al momento il prototipo funziona abbastanza bene, il modello definitivo un po' meno. A volte capitano degli sbalzi di propulsione e per questo ho paura a lasciarlo andare completamente, e preferisco tenerlo leggermente con le mani e vedere come reagisce ai miei impulsi di inclinazione. Sostenendolo in parte, percepisco che i motori reagiscono mediamente bene e velocemente ai cambi di pendenza che induco artificialmente. Senza un sostegno però il problema è dato che la stabilizzazione non è ancora perfetta, quando il drone sta a mezz'aria e s'inclina incomincia a muoversi orizzontalmente. Per poterlo testare senza però sostenerlo avrei bisogno di un ampio spazio vuoto, possibilmente all'aria aperta. Appena raggiungerò un'autostabilizzazione che riterrò abbastanza buona, eseguirò un test simile a quello appena descritto.

8 Discussione

Come dicevo nella sezione riguardante i risultati, il modello definitivo non è ancora capace di volare in autonomia completamente. Il problema maggiore è trovare le varie costanti della parte PID; visto che sono 9, è difficile capire quali bisogna cambiare ogni volta che le si testa e di quanto. Una possibilità sarebbe stata di utilizzare inizialmente un comando pilotato che ha anche la possibilità di cambiare le costanti ricercate durante il volo (per esempio usando i trim presenti in diversi radiocomandi). In questo modo trovare le varie costanti sarebbe risultato molto più semplice, ma includerebbe ovviamente del lavoro aggiuntivo per inserire la radio e i comandi in tempo reale.

9 Conclusione

Tramite questo progetto ho potuto apprendere molto riguardo ai linguaggi di programmazione, ovviamente parlo principalmente del linguaggio che viene utilizzato su Arduino, cioè il linguaggio C++. Oltre a questo ho potuto approfondire le mie conoscenze riguardanti i droni e il loro funzionamento: come si stabilizzano autonomamente in aria, quindi i sensori che utilizzano e come questi funzionino precisamente, il sistema di comunicazione che sfruttano, il programma che viene installato su di essi, ecc. Non sono però riuscito a raggiungere le mie aspettative iniziali, che erano di progettare un drone che superasse un labirinto 3D autonomamente, anche se ora mi accorgo, dopo aver approfondito abbastanza l'argomento, che quelle aspettative erano in realtà abbastanza pretenziose.

Riferimenti bibliografici

- [1] Wikipedia. *Accelerometro*. URL: <https://it.wikipedia.org/wiki/Accelerometro>. (accessed: 22.12.2019, 02.40).
- [2] Simone. *Sensori, la scelta migliore*. URL: <https://www.makeritalia.org/elettronica/2017/07/07/sensori-la-scelta-migliore/>. (accessed: 23.12.2019, 23.40).
- [3] Wikipedia. *Condensatore (elettrotecnica)*. URL: [https://it.wikipedia.org/wiki/Condensatore_\(elettrotecnica\)](https://it.wikipedia.org/wiki/Condensatore_(elettrotecnica)). (accessed: 23.12.2019, 23.40).
- [4] Treccani. *Piezoelettricità*. URL: <http://www.treccani.it/enciclopedia/piezoelettricità/>. (accessed: 21.12.2019, 01.25).
- [5] Wikipedia. *Sensore Piezoelettrico*. URL: https://it.wikipedia.org/wiki/Sensore_piezoresistivo. (accessed: 21.12.2019, 1.40).
- [6] Arduiner. *GY-521 MPU-6050 Modulo triassiale Digitale*. URL: [https://www.arduiner.com/it/gy-series-axis-accelerometers/424-gy-521-mpu-6050-modulo-triassiale-digitale-giroscopio-6dof-3-assi-3809200622084.html](http://www.arduiner.com/it/gy-series-axis-accelerometers/424-gy-521-mpu-6050-modulo-triassiale-digitale-giroscopio-6dof-3-assi-3809200622084.html). (accessed: 2.12.2019, 10.20).
- [7] InvenSense. *MPU-6050 TDK*. URL: <https://www.invensense.com/products/motion-tracking/6-axis/mpu-6050/>. (accessed: 10.10.2019, 19.30).
- [8] Wikipedia. *Controllo PID*. URL: https://it.wikipedia.org/wiki/Controllo_PID. (accessed: 21.12.2019, 01.34).
- [9] Dizionario automazione. *Controllo PID*. URL: <https://dizionarioautomazione.com/glossario/controllo-pid/>. (accessed: 24.12.2019, 01.20).
- [10] Ingegneria elettrotecnica.com. *Guida automazione : Cosa sono i controllori PID, il significato dell'azione proporzionale , integrale , derivativa*. URL: <http://www.ingegneria-elettronica.com/appunti1/pid/regolatori-pid.html>. (accessed: 20.09.2019, 01.00).
- [11] Giuseppe Caccavale. *MPU-6050 (GY-521) ARDUINO TUTORIAL*. URL: <http://www.giuseppecaccavale.it/arduino/mpu-6050-gy-521-arduino-tutorial/>. (accessed: 01.09.2019, 20.10).
- [12] InvenSense. *MPU-6000-Datasheet1*. URL: <https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf>. (accessed: 12.09.2019, 22.40).

10 Appendice: Codici completi

10.1 Prototipo

```

1 #include <Wire.h>
2 #include <Servo.h>
3
4 Servo motD; // DichiaraZione dei servomotori
5 Servo motS;
6
7 //
8 float AccX, AccY, AccZ, GirX, GirY, GirZ;
9
10 double tpassato, tempo, tloop; // Vari valori di tempo
11
12 double girAngolo[2];
13 double accAngolo[2];
14 double angTotale[3]; // angTotale[0] = asse X, angTotale[1] = asse Y,
15     angTotale[2] = asse Z
16
17 int c = 0; // usato nella funzione per calcolare l'errore medio
18     iniziale
19
20 // Costanti delle parti PID
21 float kp = 4;
22 float ki = 0.04;
23 float kd = 2.5;
24
25 // Valori finali del PID totale, delle 3 parti che lo compongono
26 // e dei pwm che andranno a dare una certa propulsione ai motori
27 double PID;
28 double pidP = 0;
29 double pidI = 0;
30 double pidD = 0;
31 double pwmD, pwmS;
32
33 // Valori di errore rispetto all'angolo voluto (quindi l'angolo
34 // totale),
35 // il secondo sarebbe l'errore del loop precedente che viene usato
36 // nella parte derivativa del pid
37 double errore, errorePrec;
38
39 // Qui, dato che l'asse non parte da posizione orizzontale, uso i
40 // dati trovati in precedenza
41 // Gli errori di misurazione delle varie accelerazioni per l'asse
42 // nella posizione in cui dovrebbe essere
43 float AccErroreX = -0.37;
44 float AccErroreY = -2.66;

```

```
39 float AccErroreZ = -0.01;
40
41 float GirErroreX = 1.26;
42 float GirErroreY = -2.34;
43 float GirErroreZ = -1.04;
44
45 void setup()
46 {
47 Serial.begin(19200);
48 Wire.begin(); // Inizializzo la comunicazione
    I2C
49 Wire.beginTransmission(0x68); // Chiamo il registro che si
    riferisce all'MPU
50 Wire.write(0x6B); // Chiamo il registro 0x6B
51 Wire.write(0x00);
52 Wire.endTransmission(true); // Qui sto dicendo che in pratica
    la fine della trasmissione risulta "vera", quindi finisce e non
    prende dati
53 motD.attach(9); // Attacca il motore di destra al
    pin digitale 9
54 motS.attach(10); // Attacca il motore di sinistra
    al pin digitale 10
55
56 // Do il valore minimo ai motori per attivarli (non girano ancora
    le eliche)
57 motD.writeMicroseconds(1000);
58 motS.writeMicroseconds(1000);
59
60 delay(15000);
61 }
62
63 //-----
64
65 void loop()
66 {
67 tloop = tempo;
68 tempo = millis();
69 tpassato = (tempo-tloop) / 1000; // con questo calcolo ci rimangono i
    secondi passati da un loop all'altro
70
71 // Leggo i dati di ACCELERAZIONE (accelerometro)
72
73 Wire.beginTransmission(0x68); // Chiedo i registri partendo dal 0
    x3B
74 Wire.write(0x3B); // Chiedo i registri partendo dal 0
    x3B
75 Wire.endTransmission(false); // La fine della trasmissione
    risulterebbe "falsa", quindi continua
76 Wire.requestFrom(0x68, 6, true); // Richiedo all'MPU (0x68) 6
    registri
```

```

77
78 // Ogni registro contiene 8 dei 16 bit di ogni parametro (es. accX)
79 // i primi 8 bit si trovano nei registri 1, 3 e 5 e quelli finali
    // nei restanti
80 AccX = (Wire.read() << 8|Wire.read()) / 16384.0; // Qui vengono uniti
    // 2 regitri alla volta per ottenere il parametro
81 AccY = (Wire.read() << 8|Wire.read()) / 16384.0; // Per una portata
    // di +/- 2G bisogna dividere per 16384,
82 AccZ = (Wire.read() << 8|Wire.read()) / 16384.0;
83
84 // Uso la formula di Eulero per trovare l'angolo con le
    // accelerazioni
85 accAngolo[0] = (atan(AccY / sqrt(pow(AccX, 2) + pow((AccZ-AccErroreZ),
        , 2))) * 180 / 3.141592654) - AccErroreX;
86 accAngolo[1] = (atan(-1 * AccX / sqrt(pow(AccY, 2) + pow((AccZ-
        AccErroreZ), 2))) * 180 / 3.141592654) - AccErroreY;
87
88 // Leggo i dati di ACCELERAZIONE ANGOLARE (giroscopio)
89
90 Wire.beginTransmission(0x68);
91 Wire.write(0x43); // Chiedo i registri questa volta
    // partendo dal 0x43
92 Wire.endTransmission(false);
93 Wire.requestFrom(0x68, 6, true);
94
95 // Per una portata di +/-250 gradi/secondo devo dividere per 131
96 GirX = (Wire.read() << 8|Wire.read()) / 131.0;
97 GirY = (Wire.read() << 8|Wire.read()) / 131.0;
98 GirZ = (Wire.read() << 8|Wire.read()) / 131.0;
99
100 // Correggo con l'errore trovato in precedenza
101 GirX = GirX - GirErroreX;
102 GirY = GirY - GirErroreY;
103 GirZ = GirZ - GirErroreZ;
104
105 // Moltiplico i gradi/secondo per i secondi passati
106 girAngolo[0] = girAngolo[0] + GirX * tpassato;
107 girAngolo[1] = girAngolo[1] + GirY * tpassato;
108
109 // Combino i valori del giroscopio con quelli dell'accelerometro
110 angTotale[0] = 0.98 * girAngolo[0] + 0.02 * accAngolo[0];
111 angTotale[1] = 0.98 * girAngolo[1] + 0.02 * accAngolo[1];
112 angTotale[2] = angTotale[2] + GirZ * tpassato;
113
114 errore = angTotale[1]; // L'errore sarebbe l'angolo in Y
115
116
117

```

```
118 ///////////////
119 //      PID      //
120 ///////////////
121
122 // Parte Proporzionale
123 pidP = kp*errore;
124
125 // Parte Integrale
126 pidI = pidI + (ki*errore);
127
128 // Parte Derivativa
129 pidD = kd*((errore - errorePrec)/tpassato);
130
131 // Somma algebrica per ottenere il PID finale
132 PID = pidP + pidI + pidD;
133
134 // Do dei valori massimi e minimi al PID
135 if(PID < -1000)
136 {
137     PID = -1000;
138 }
139 if(PID > 1000)
140 {
141     PID = 1000;
142 }
143
144 // Mando un segnale arbitrario di 1200 per attivare le eliche
145 pwmD = 1200 - PID;
146 pwmS = 1200 + PID;
147
148 //Do dei valori massimi e minimi alle variabili pwm
149 if(pwmD < 1000) //Per la parte destra
150 {
151     pwmD = 1000;
152 }
153 if(pwmD > 2000)
154 {
155     pwmD = 2000;
156 }
157
158 if(pwmS < 1000) // Per la parte sinistra
159 {
160     pwmS = 1000;
161 }
162 if(pwmS > 2000)
163 {
164     pwmS = 2000;
165 }
```

```
167 // Do ai motori il valore pwm che ho trovato
168 motD.writeMicroseconds(pwmD);
169 motS.writeMicroseconds(pwmS);
170
171 errorePrec = errore; // Preparo la variabile di errore precedente
    usata dal pid derivativo per il prossimo loop
172
173 Serial.print("pwmD: ");
174 Serial.print(pwmD);
175 Serial.print("  pwmS: ");
176 Serial.print(pwmS);
177 Serial.print("  angolo Y: ");
178 Serial.print(angTotale[1]);
179 Serial.print("  angolo X: ");
180 Serial.print(angTotale[0]);
181 Serial.print("  angolo Z: ");
182 Serial.println(angTotale[2]);
183
184 delay(15);
185 }
```

Codice Completo 22: Prototipo

10.2 Drone definitivo

```

1 #include <Wire.h>
2 #include <Servo.h>
3
4 Servo motAd; // Dichiarazione dei servomotori
5 Servo motAs; // mot = servomotore;
6 Servo motDd; // A = avanti; D = dietro; d = destra e s = sinistra
7 Servo motDs;
8
9 // 0 --> Asse X --> Avanti / Dietro
10 // 1 --> Asse Y --> Destra / Sinistra
11 // 2 --> Asse Z --> Rotazione su se stesso in orizzontale
12
13 // Acc = accelerazione lineare
14 // Gir = accelerazione angolare
15 float AccX, AccY, AccZ, GirX, GirY, GirZ;
16 float AccErroreX, AccErroreY, AccErroreZ, GirErroreX, GirErroreY,
     GirErroreZ;
17
18 double tpassato, tempo, tloop; // Vari valori di tempo
19
20 double girAngolo[2];
21 double accAngolo[2];
22 double angTotale[3];
23
24 int c = 0; // Usato nella funzione "erroreMPU" per calcolare l'errore
             // medio iniziale
25
26 // Costanti delle parti PID
27 float kp[3];
28 float ki[3];
29 float kd[3];
30
31 // Valori finali del PID totale, delle 3 parti che lo compongono
32 // (P = proporzionale, I = integrale, D = derivativa)
33 // e dei pwm che andranno a dare una certa propulsione ai motori
34 double PID[3];
35 double pidP[3], pidI[3], pidD[3];
36
37 // Ho posto l'MPU in diagonale, quindi agli assi X e Y corrispondono
     // 2 motori che stanno ai vertici opposti,
38 // per l'asse Z invece bisogna modificare i motori a coppie,
39 // unendo i valori PID di ogni asse ad un valore che possa tenere i
     // motori stabili si ottiene il valore PWM finale di ogni motore
40 double pwmAd, pwmAs, pwmDd, pwmDs;
41
42 // Il secondo sarebbe l'errore del loop precedente, usato nella
     // parte derivativa del pid

```

```

43 double errore[3], errorePrec[3];
44
45 bool capovolto = false;
46 bool bottone = false;
47
48 // Serve per far alzare e scendere il drone
49 int x = -350;
50
51 // Funzione per calcolare l'errore dell'MPU6050 (calibrazione)
52 void erroreMPU()
53 {
54 while (c < 500)
55 {
56 // Questi passaggi vengono spiegati nella parte "void setup/loop"
57
58 // Errore accelerometro
59 Wire.beginTransmission(0x68);
60 Wire.write(0x3B);
61 Wire.endTransmission(false);
62 Wire.requestFrom(0x68, 6, true);
63
64 AccX = (Wire.read() << 8 | Wire.read()) / 16384.0 ;
65 AccY = (Wire.read() << 8 | Wire.read()) / 16384.0 ;
66 AccZ = (Wire.read() << 8 | Wire.read()) / 16384.0 ;
67
68 // Sommo gli errori
69 AccErroreX += (atan((AccY) / sqrt(pow((AccX), 2)) * 180 / PI));
70 AccErroreY += (atan(-1 * (AccX) / sqrt(pow((AccY), 2)) * 180 / PI))
71 ;
72 AccErroreZ += (AccZ-1);
73
74 // Errore del giroscopio
75 Wire.beginTransmission(0x68);
76 Wire.write(0x43);
77 Wire.endTransmission(false);
78 Wire.requestFrom(0x68, 6, true);
79
80 GirX = Wire.read() << 8 | Wire.read();
81 GirY = Wire.read() << 8 | Wire.read();
82 GirZ = Wire.read() << 8 | Wire.read();
83
84 // Sommo gli errori
85 GirErroreX += (GirX / 131.0);
86 GirErroreY += (Giry / 131.0);
87 GirErroreZ += (GirZ / 131.0);
88 c++;
89 }
90 // Divido gli errori per ottenere l'errore medio

```

```
91 AccErroreX /= 500;
92 AccErroreY /= 500;
93 AccErroreZ /= 500;
94
95 GirErroreX /= 500;
96 GirErroreY /= 500;
97 GirErroreZ /= 500;
98
99 // Stampa i dati ottenuti, per controllarli durante i test
100 Serial.print("AccErroreX: ");
101 Serial.print(AccErroreX);
102 Serial.print(" AccErroreY: ");
103 Serial.print(AccErroreY);
104 Serial.print(" GirErroreX: ");
105 Serial.print(GirErroreX);
106 Serial.print(" GirErroreY: ");
107 Serial.print(GirErroreY);
108 Serial.print(" GirErroreZ: ");
109 Serial.println(GirErroreZ);
110 Serial.println("-----");
111 }
112
113 //-----
114
115 void setup()
116 {
117 Serial.begin(19200);
118
119 pinMode(2, INPUT);
120
121 Wire.begin(); // Inizializzo la comunicazione I2C
122 Wire.beginTransmission(0x68); // Chiamo il registro che si
123 // riferisce all'MPU
124 Wire.write(0x6B); // Chiamo il registro 0x6B
125 Wire.write(0x00);
126 Wire.endTransmission(true); // Qui sto dicendo che in pratica la
127 // fine della trasmissione sarebbe vera, quindi finisce
128
129 motAd.attach(9); // Attacca il motore di destra al pin
130 // digitale 9
131 motAs.attach(10); // Attacca il motore di sinistra al
132 // pin digitale 10
133 motDd.attach(6);
134 motDs.attach(5);
135
136 // Do il valore minimo ai motori per attivarli (non girano ancora le
137 // eliche)
138 motAd.writeMicroseconds(1000);
139 motAs.writeMicroseconds(1000);
```

```

135 motDd.writeMicroseconds(1000);
136 motDs.writeMicroseconds(1000);

137
138 // Richiamo la funzione per trovare l'errore medio, quindi per
     calibrare il sensore
139 erroreMPU();

140
141 // Il tutto inizia premendo un pulsante e quindi leggendo il pin che
     riceve il suo segnale
142 while (bottone == false)
143 {
144     bottone = digitalRead(2);
145 }
146 delay(20);
147 }

148
149 //-----
150
151 void loop()
152 {
153 while (x<1200)
154 {
155     tloop = tempo;
156     tempo = millis();
157     tpassato = (tempo-tloop) / 1000; // Con questo calcolo ci
         rimangono i secondi passati da un loop all'altro
158
159     // Leggo i dati di ACCELERAZIONE LINEARE (accelerometro)
160
161     Wire.beginTransmission(0x68);
162     Wire.write(0x3B);           // Chiedo i registri partendo dal
         0x3B (nel datasheet --> ACCEL_XOUT_H)
163     Wire.endTransmission(false); // La fine della trasmissione
         sarebbe falsa, quindi continua
164     Wire.requestFrom(0x68, 6, true); // Richiedo all'MPU (0x68) 6
         registri
165
166     // Ogni registro contiene 8 dei 16 bit di ogni parametro (per
         esempio l'accelerazione lineare in X),
167     // i primi 8 bit si trovano nei registri 1, 3 e 5 e quelli finali
         nei restanti
168     AccX = (Wire.read() << 8|Wire.read()) / 16384.0; // Qui vengono
         uniti 2 registri alla volta per ottenere il parametro
169     AccY = (Wire.read() << 8|Wire.read()) / 16384.0; // Per una
         portata di +/- 2G bisogna dividere per 16384,
170     AccZ = (Wire.read() << 8|Wire.read()) / 16384.0;
171
172     // Uso la formula di Eulero per trovare l'angolo con le
         accelerazioni

```

```

173 accAngolo[0] = (atan(AccY / sqrt(pow(AccX, 2))) * 180 / PI);
174 accAngolo[1] = (atan(-AccX / sqrt(pow(AccY, 2))) * 180 / PI);
175
176 // Leggo i dati di ACCELERAZIONE ANGOLARE (giroscopio)
177
178 Wire.beginTransmission(0x68);
179 Wire.write(0x43); // Chiedo i registri questa volta
180     partendo dal 0x43, dove si trovano dunque quelli del giroscopio
181 Wire.endTransmission(false);
182 Wire.requestFrom(0x68, 6, true);
183
184 // Per una portata di +/-250 gradi/secondo devo dividere per 131
185 GirX = (Wire.read() << 8|Wire.read()) / 131.0;
186 GirY = (Wire.read() << 8|Wire.read()) / 131.0;
187 GirZ = (Wire.read() << 8|Wire.read()) / 131.0;
188
189 // Correggo con l'errore trovato in precedenza
190 GirX -= GirErroreX;
191 GirY -= GirErroreY;
192 GirZ -= GirErroreZ;
193
194 // Bisogna moltiplicare i gradi/secondo per i secondi passati per
195 // ottenere i gradi
196 girAngolo[0] += GirX * tpassato;
197 girAngolo[1] += GirY * tpassato;
198
199 // Combino i valori del giroscopio con quelli dell'accelerometro
200 angTotale[0] = 0.98 * girAngolo[0] + 0.02 * accAngolo[0];
201 angTotale[1] = 0.98 * girAngolo[1] + 0.02 * accAngolo[1];
202 angTotale[2] = GirZ * tpassato;
203 // Se motAd si solleva --> angTotale[0] aumenta
204 // Se motAs si solleva --> angTotale[1] aumenta
205 // Se il drone ruota in senso antiorario (guardando dall'alto)
206 // --> angTotale[2] aumenta */
207
208 // Nel caso il drone si inclinasse troppo o si capovolgesse,
209 // fermo i motori con questa funzione
210 if((angTotale[0] < -20) | (angTotale[0] > 20) | (angTotale[1] < -20) | (
211 angTotale[1] > 20))
212 {
213     motAs.writeMicroseconds(1000);
214     motAd.writeMicroseconds(1000);
215     motDs.writeMicroseconds(1000);
216     motDd.writeMicroseconds(1000);
217     capovolto = true;
218 }
219
220 // Potrebbe poi continuare a roteare in aria e raggiungere un
221 // angolo che normalmente andrebbe bene (magari addirittura

```

```

    atterrando in piedi),
216   // anche se dovesse succedere, appena supera i parametri limite
217   // dettati sopra, mantengo fermi i motori in questo modo:
218   if (capovolto == true)
219   {
220     motAs.writeMicroseconds(1000);
221     motAd.writeMicroseconds(1000);
222     motDs.writeMicroseconds(1000);
223     motDd.writeMicroseconds(1000);
224   }
225
226   else
227   {
228     // Visto che l'angolo desiderato sarebbe 0, l'errore da
229     // controbilanciare sarebbe quindi l'angolo stesso,
230     // qui viene cambiato il nome per fare chiarezza nei passaggi
231     errore[0] = angTotale[0]; // Errore in X
232     errore[1] = angTotale[1]; // In Y
233     errore[2] = angTotale[2]; // E in Z
234
235 //-----
236
237   /////////////
238   // PID //
239   ///////////
240   kp[0]= 5;  ki[0]= 3;      kd[0]= 2;
241   kp[1]= 5;  ki[1]= 2.5;   kd[1]= 4;
242   kp[2]= 5;  ki[2]= 3;      kd[2]= 2;
243
244   double X;
245   if (x >= 1198)
246   {
247     X = 0;
248   }
249
250   if ((x>837) && (x<1198))
251   {
252     X = (24283.92/(x-634.739)) + 1350;
253   }
254
255   else
256   {
257     X = (-0.00088*x*x)+(0.88*x)+1350;
258   }
259
260   if (X>1450)
261   {
262     for (int i=0; i<3; i++)
263     {

```

```

262         // Parte Proporzionale
263         pidP[i] = kp[i]*errore[i];
264
265         // Parte Integrale
266         pidI[i] += (ki[i]*errore[i]);
267
268         // Parte Derivativa
269         pidD[i] = kd[i]*((errore[i] - errorePrec[i])/tpassato);
270
271         // Somma algebrica per ottenere il PID finale
272         PID[i] = pidP[i] + pidI[i] + pidD[i];
273
274         // Il valore minimo del pwm sarebbe 1000 e il massimo
275         // 2000, quindi il valore massimo che posso aggiungere sarebbe 1000,
276         // e al contrario il massimo che posso togliere sarebbe
277         // anche 1000 (che diventa -1000)
278         if(PID[i] < -300)
279         {
280             PID[i] = -300;
281         }
282         if(PID[i] > 300)
283         {
284             PID[i] = 300;
285         }
286         PID[i] = map(PID[i], -300, 300, -250, 250);
287     }
288
289     // Per far attivare le eliche non basta il valore minimo di
290     // 1150 che viene dato nel setup,
291     // e ho trovato, per questo prototipo, 1200 come valore
292     // ideale per questa funzione
293
294
295     // Come dicevo prima, il valore pwm minimo sarebbe 1100 e il
296     // massimo 2000 circa
297     // Il valore minimo per cui si attivano le eliche risulta
298     // 1120, dunque l'ho impostato come minimo
299
300
301         if(pwmAd < 1120)
302         {
303             pwmAd = 1120;
304         }
305         if(pwmAd > 2000)
306         {

```

```

305         pwmAd = 2000;
306     }
307
308     if(pwmDs < 1120)
309     {
310         pwmDs = 1120;
311     }
312     if(pwmDs > 2000)
313     {
314         pwmDs = 2000;
315     }
316
317     if(pwmAs < 1120)
318     {
319         pwmAs = 1120;
320     }
321     if(pwmAs > 2000)
322     {
323         pwmAs = 2000;
324     }
325
326     if(pwmDd < 1120)
327     {
328         pwmDd = 1120;
329     }
330     if(pwmDd > 2000)
331     {
332         pwmDd = 2000;
333     }
334 //-----
335
336     // Mando l'output ai motori col valore pwm che ho trovato
337     motAd.writeMicroseconds(pwmAd);
338     motAs.writeMicroseconds(pwmAs);
339     motDd.writeMicroseconds(pwmDd);
340     motDs.writeMicroseconds(pwmDs);
341
342     // Preparo la variabile di errore precedente usata dal pid
343     // derivativo per il prossimo loop
344     for(int i=0; i<=2; i++)
345     {
346         errorePrec[i] = errore[i];
347     }
348
349     // Inserisco un delay abbastanza corto per poter dare segnali
350     // ai motori abbastanza velocemente
351     delay(20);
352     x++;

```

```

352     } // Chiudo la funzione "else" che controllava che il drone non
353     // fosse troppo inclinato / capovolto
354
355     // STAMPA VALORI DURANTE I TEST:
356
357     // Valore di X
358     Serial.print(" X: ");
359     Serial.print(X);
360
361     // Angolo di ciascun asse
362     Serial.print(" angX: ");
363     Serial.print(angTotale[0]);
364     Serial.print(" angY: ");
365     Serial.print(angTotale[1]);
366     Serial.print(" angZ: ");
367     Serial.print(angTotale[2]);
368
369     //Valori PID di ogni motore
370     Serial.print(" Ad: ");
371     Serial.print(-(2*PID[0]+PID[2])/4);
372     Serial.print(" Ds: ");
373     Serial.print((2*PID[0]-PID[2])/4);
374     Serial.print(" As: ");
375     Serial.print(-(2*PID[1]-PID[2])/2);
376     Serial.print(" Dd: ");
377     Serial.println((2*PID[1]+PID[2])/2);
378
379     // Valori di Output per ogni motore
380     Serial.print("pwmAd = ");
381     Serial.print(pwmAd);
382     Serial.print(" pwmAs = ");
383     Serial.print(pwmAs);
384     Serial.print(" pwmDd = ");
385     Serial.print(pwmDd);
386     Serial.print(" pwmDs = ");
387     Serial.print(pwmDs);
388 }
389 } // Chiudo il for iniziale

```

Codice Completo 23: Drone definitivo