UNIVERSITÀ DEGLI STUDI DI TRIESTE
Dipartimento di Ingegneria e Architettura

Laurea Triennale in Ingegneria Elettronica e Informatica

Stabilizzazione di un pendolo inverso con un volano

Laureando
Lorenzo D'Auria

Relatore
Professor Stefano Marsi

Anno accademico 2018/2019

Indice

Capitolo 1

Introduzione

L'obiettivo di questo lavoro è la progettazione di un sistema di stabilizzazione atto a mantenere in equilibrio un pendolo inverso, ovvero un corpo rigido imperniato al di sotto del proprio centro di massa, tramite l'accelerazione di un volano montato su di esso. Tale lavoro è stato svolto dal laureando presso l'ateneo e ha come fine, oltre allo studio teorico, la realizzazione di un prototipo funzionante.

Sebbene in letteratura scientifica siano presenti numerosi lavori di questo tipo, quello che verrà presentato qui ha come peculiarità l'utilizzo di accelerometro e giroscopio per la rilevazione dell'angolo di inclinazione, caratteristica che lo rende facilmente adattabile a svariati campi di impiego pratici come quello motociclistico e quello aerospaziale. Per ragioni di costo e praticità il prototipo è stato implementato con la piattaforma hardware Arduino, mentre per lo studio teorico mediante simulazione si è utilizzato MATLAB. I codici sono stati sviluppati interamente dal laureando ad eccezione di alcune librerie già pronte che verranno specificate in seguito.

La stabilizzazione di un pendolo inverso è tra i più classici problemi nell'ambito dei sistemi di automazione perché, oltre ad avere molti impieghi nel mondo reale, ha alla base una dinamica facilmente modellizzabile, ma una risoluzione tutt'altro che banale. Si tratta di un sistema con due punti di equilibrio: uno stabile, in cui il braccio è in verticale sotto al perno, e uno instabile, quello con il braccio verticale sopra il perno. Quest'ultimo si presta alla progettazione di un sistema di controllo atto a rendere stabile il sistema in quel punto. La realizzazione di un sistema di stabilizzazione di questo tipo vede numerose soluzioni possibili, ma ciò che le accomuna è che tutte puntano a generare un momento meccanico tale da opporsi a quello generato dalla gravità. Nel sistema studiato in questa tesi, il

momento meccanico è generato da un motore elettrico, con un disco attaccato all'asse di rotazione, montato sul pendolo. Infatti, per la Terza legge di Newton, la stessa coppia torcente che il motore fornisce al disco viene fornita, in direzione opposta, dal disco al motore e quindi all'intero pendolo.

Per quanto riguarda la progettazione del controllore, è stata utilizzata una soluzione semplice e versatile: un controllore Proporzionale-Integrale-Derivativo (PID) [1]. Un controllore PID ha un funzionamento semplice: riceve in input il target e lo stato attuale del sistema, ne calcola la differenza (o l'errore) e restituisce in output il valore della variabile che permette di controllare il sistema. L'uscita del controllore è ottenuta pesando, mediante dei parametri regolabili, il contributo proporzionale, quello integrale e quello derivativo calcolati a partire dall'errore. La realizzazione di un sistema di controllo basato su un'analisi accurata delle equazioni che governano il sistema, molto probabilmente, permetterebbe di ottenere prestazioni migliori, ma ottenere le migliori prestazioni possibili non è l'obiettivo di questa tesi.

Entrando più nello specifico, nel nostro caso il controllore PID calcola l'errore come l'angolo tra l'asse verticale e la posizione in cui si trova il pendolo in quel momento, ciò è possibile grazie all'accelerometro e al giroscopio montati sul pendolo. Calcolata l'uscita, essa andrà a modulare la corrente fornita al motore elettrico in grado di generare il momento meccanico necessario per la stabilizzazione.
La messa a punto dei parametri di un controllore PID non è un lavoro semplice perché viene effettuata a partire da prove sperimentali e graduali aggiustamenti dei parametri, inoltre dipende molto dalle caratteristiche fisiche del sistema, infatti piccole variazioni del setup sperimentale possono vanificare tutto il lavoro di messa a punto.

Un altro aspetto critico nella realizzazione del prototipo riguarda il calcolo dell'angolo di inclinazione del pendolo a partire dai dati grezzi che giroscopio e accelerometro forniscono al microcontrollore. La prima difficoltà consiste nel distinguere i disturbi dai dati utili per ognuno dei due sensori per un calcolo accurato dell'angolo di inclinazione. Il secondo problema è invece più di tipo meccanico e consiste nella riduzione delle vibrazioni che il motore trasmette al pendolo e quindi ai sensori per il calcolo dell'angolo. Riuscire a ridurre il più possibile le vibrazioni che l'accelerometro e il giroscopio ricevono e ottimizzare il filtraggio dei disturbi dal segnale ottenuto è sicuramente la chiave per ridurre al minimo l'errore nel calcolo dell'angolo di inclinazione del pendolo.

Riepilogando, il prototipo reale si compone di un pendolo (una struttura rigida

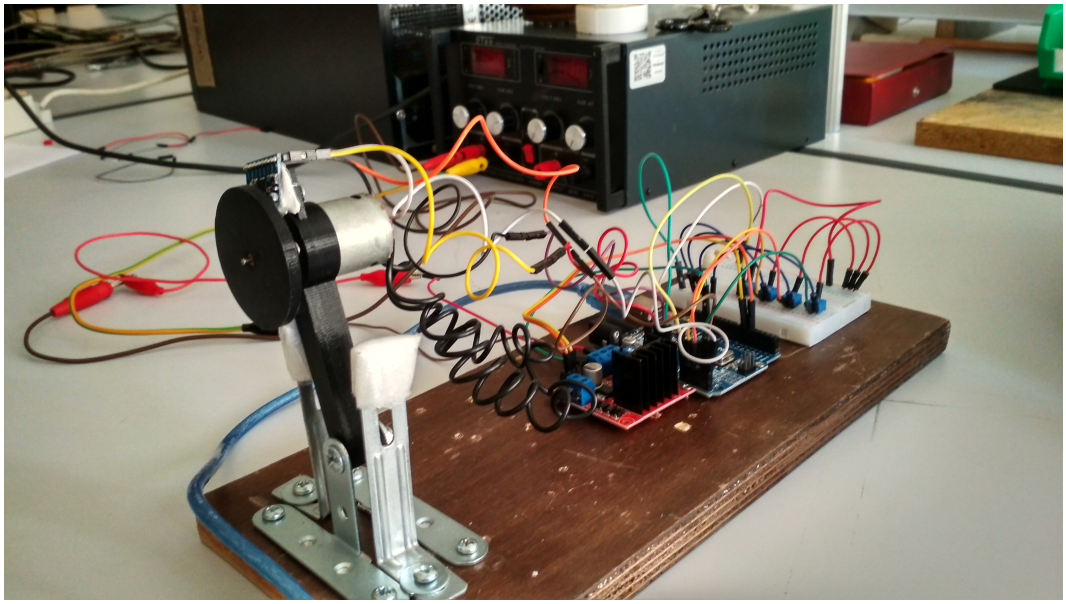

Figura 1.1: Prototipo finale funzionante.

in plastica di forma allungata) con attaccati ad un'estremità un motore elettrico, un accelerometro e un giroscopio, all'altra un perno che ne permette il movimento. All'asse del motore elettrico è attaccato un disco, mentre una staffa collega il perno alla struttura esterna. La struttura esterna è composta da una tavola di legno alla quale sono attaccati, oltre alla staffa che sostiene il pendolo, un Arduino Uno e un driver in corrente continua per controllare il motore. I segnali digitali provenienti da giroscopio e accelerometro vengono letti e interpretati dall'Arduino che calcola l'angolo di inclinazione del pendolo. Tramite il microcontrollore si implementa anche il sistema di controllo che, dall'angolo calcolato, comanda il driver del motore. Infine il driver, alimentato da una tensione di circa 12V e comandato dall'Arduino, fornisce al motore la giusta corrente.

La scelta dei componenti è stata guidata principalmente dal costo e dalla disponibilità immediata del materiale utilizzato, ma è anche stato necessario un lavoro sperimentale per selezionare fin da subito i componenti più adatti. In particolare, la prima scelta da fare era tra motori con le spazzole e senza spazzole. Come verrà approfondito nel Capitolo 3, la scelta è ricaduta su un motore con le spazzole perché permette di regolare con più precisione la coppia sviluppata in ogni momento. Scelto il motore, si è passati alla progettazione del pendolo vero e proprio tramite un

programma di CAD 3D e alla sua realizzazione con la stampa 3D. Anche in questo caso le dimensioni e il peso del pendolo sono state scelte sulla base di stime e prove sperimentali. Per quanto riguarda l'accelerometro e il giroscopio, si è optato per la soluzione più semplice e comune tra i sensori compatibili con Arduino: il modulo MPU-6050 (descritto nella Sezione 3.2).

Lo sviluppo del software Arduino ha richiesto diversi accorgimenti per ottenere le prestazioni desiderate. Il primo riguarda l'implementazione di filtri digitali che riducono i disturbi che l'accelerometro e il giroscopio rilevano. Il secondo è la velocità di esecuzione di ciclo, ovvero il numero di volte al secondo per cui viene calcolato l'angolo di inclinazione e quindi il nuovo valore di uscita del controllore. Ovviamente più il campionamento è rapido, migliore sarà la reattività del sistema di controllo. Si è riusciti ad ottenere una frequenza di 250Hz, sufficiente a svolgere tutte le operazioni con un buon margine di tempo.

Al fine di valutare come cambia il comportamento del sistema in base ai componenti utilizzati, è stato creato uno script MATLAB che simula le dinamiche del pendolo reale. Tale modello tiene conto dei momenti di inerzia di pendolo e disco, delle caratteristiche del motore e in generale di tutte le grandezze fisiche in gioco. Dopo aver accertato che il modello rispetti abbastanza fedelmente la realtà, sono state fatte considerazioni sulle prestazioni ottenibili con componenti diversi da quelli utilizzati. Ciò ha permesso di avere un'idea chiara di come si possa migliorare il prototipo per ottenere delle prestazioni tali da consentire un utilizzo commerciale di tale tecnologia.

Il presente sistema di stabilizzazione potrebbe ad esempio essere impiegato in campo motociclistico, infatti, semplificando molto il modello, il motociclo può essere visto come un pendolo inverso imperniato nel punto di appoggio delle ruote sull'asfalto. Integrando un giroscopio e un accelerometro su un motociclo e aggiungendo un volano alla struttura, si otterrebbe un sistema analogo a quello analizzato in questa tesi. Questa tecnologia permetterebbe di evitare cadute salvaguardando i viaggiatori fino a rendere possibile anche l'equilibrio del motociclo da fermo. Un altro vasto campo di applicazione è quello aerospaziale, infatti un velivolo può essere visto come un pendolo libero di ruotare intorno al proprio centro di massa. La stabilizzazione mediante dischi inerziali sui tre assi spaziali permetterebbe regolazioni precise nella modifica dell'assetto del velivolo anche fuori dall'atmosfera.

Nei capitoli successivi verranno illustrati più nel dettaglio i vari passi che hanno permesso di realizzare un prototipo funzionante. In particolare, nel Capitolo 2

verrà descritta la fisica che governa il comportamento del pendolo e permette il funzionamento del sistema di stabilizzazione creato. Successivamente, nel Capitolo 3, verranno descritte le caratteristiche dei componenti utilizzati e le motivazioni che hanno portato alla loro scelta. Nel Capitolo 4 si analizzerà nel dettaglio la struttura e il funzionamento del software utilizzato su Arduino. A seguire, nel Capitolo 5, verrà presentata la simulazione effettuata con MATLAB focalizzandosi sulle limitazioni della sua attendibilità e sul confronto con il prototipo reale. Infine, tutte le considerazioni finali verranno esposte nel Capitolo 6.
In Appendice A è riportato il codice utilizzato dal prototipo Arduino e in Appendice B lo script sviluppato per la simulazione MATLAB.

Capitolo 2

Fisica di funzionamento

Per comprendere la fisica dietro alla stabilizzazione di un pendolo inverso, è bene partire dall'analisi della meccanica che governa un pendolo. Consideriamo un pendolo ideale, cioè un'asta di massa trascurabile imperniata ad un estremo con una massa m a distanza l dal perno. La forza di gravità che agisce sul pendolo genera un momento meccanico [2]

$$\tau = mgl\sin(\theta)$$

dove θ è l'angolo tra l'asse verticale e il pendolo. Dalla seconda legge di Newton sappiamo che tale momento meccanico produce sul pendolo un'accelerazione angolare

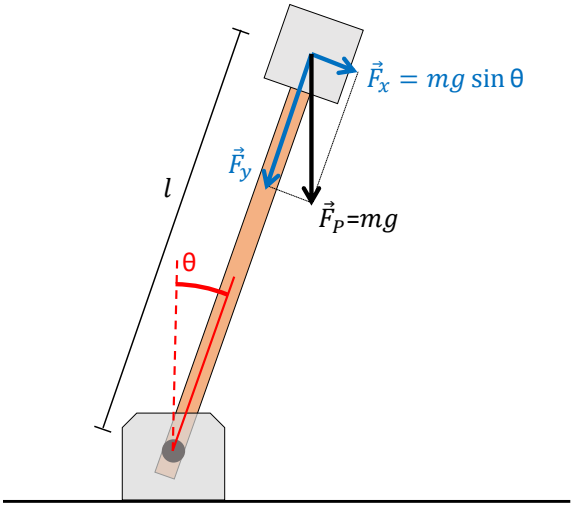

Figura 2.1: Schema delle forze applicate ad un pendolo ideale di massa m.

$$\alpha = \frac{\tau}{I}$$

dove I è il momento di inerzia del pendolo. Lo spostamento che tale accelerazione produce modifica l'angolo θ, da cui dipendono anche τ e α, ciò si traduce in un'oscillazione del pendolo attorno alla posizione di equilibrio $\theta = 180°$. Lo stesso ragionamento si può estendere anche per un pendolo non ideale, infatti si può considerare l'insieme dei contributi della forza di gravità sul pendolo come una singola

forza applicata nel centro di massa, perciò il momento meccanico esercitato è

$$\tau = mgl_c \sin(\theta)$$

dove l_c è la distanza tra perno e centro di massa del pendolo.

Per contrastare questa condizione di instabilità, è necessario applicare al pendolo un momento meccanico τ_m che si opponga a quello generato dalla gravità. Nel nostro sistema τ_m è generato da un motore elettrico montato sul pendolo. È da notare che il momento meccanico non dipende dal punto di applicazione, quindi applicare τ_m sul perno o in qualunque altro punto del pendolo produce lo stesso effetto. In altre parole il comportamento del sistema non dipende dalla posizione del motore sul pendolo, ma solo dal momento meccanico che è in grado di applicare.

Normalmente quando si pensa alla coppia esercitata da un motore si considera solo la coppia esercitata dal rotore, ma è bene ricordare che, per la terza legge di Newton, lo stesso momento meccanico che lo statore applica al rotore viene applicato dal rotore allo statore. Se lo statore è solidale ad un supporto esterno, il momento meccanico applicato allo statore viene annullato dalla reazione vincolare del supporto che quindi non subisce un'accelerazione, ma, se anche lo statore è libero di muoversi, il momento su di esso esercitato genera un'accelerazione dello statore $\alpha_s = \tau_m / I_s$, dove I_s è il momento di inerzia dello statore.

Quindi, quando il motore montato sul pendolo viene alimentato, il momento τ_m da esso generato fornisce un'accelerazione $\alpha_r = \tau_m / I_r$ al rotore e modifica l'accelerazione complessiva del pendolo in

$$\alpha = \frac{\tau + \tau_m}{I}$$

Occorre specificare che il momento di inerzia, così come l'accelerazione angolare, sono relative al punto attorno al quale il corpo gira. In questo caso, infatti, α_r e I_r si riferiscono all'asse del motore, mentre α e I si riferiscono al perno attorno al quale il pendolo è libero di ruotare.

Da quanto detto sopra si conclude che l'inclinazione del pendolo può essere modificata dall'accelerazione che il momento meccanico τ_m ha prodotto. Idealmente la dinamica del sistema non dipende dal momento di inerzia del rotore e dal suo movimento, ma in realtà, per come sono fatti i motori elettrici, la coppia τ_m diminuisce con l'aumentare della velocità angolare del rotore. Perciò, per evitare un calo della coppia fornita dal motore, è necessario minimizzare la velocità angolare del rotore. A parità di τ_m, aumentando il momento di inerzia del rotore I_r, l'accelerazione α_r

diminuisce, ciò si traduce in variazioni di velocità meno repentine e quindi un tempo maggiore per raggiungere velocità angolari tali da ridurre di molto τ_m. Bisogna prestare attenzione al fatto che aumentando la massa del rotore si aumenta I_r, ma, poiché il rotore è montato sopra il pendolo, va ad aumentare anche l'inerzia dell'intero pendolo e quindi il momento meccanico τ fornito dalla gravità. Per prestazioni ottimali è quindi necessario un accurato bilanciamento tra masse e momenti di inerzia di rotore e pendolo. Per quanto riguarda il motore elettrico, maggiore è la coppia massima generabile dal motore, migliori sono le prestazioni, in particolare la velocità di risposta del sistema.

Per comprendere meglio il comportamento complessivo del sistema, viene riportata in seguito la sequenza causa-effetto approssimata in tempo discreto di tutti i fenomeni descritti in questo capitolo. Tale schema si rivelerà utile anche per lo sviluppo della simulazione MATLAB approfondita nel Capitolo 5.

Sia θ_i l'angolo di inclinazione del pendolo all'istante di tempo t_i, siano m e I la massa e il momento di inerzia del pendolo ed l_c la distanza tra il suo centro di massa e il perno. Infine ipotizziamo che il rotore abbia momento di inerzia I_r e che il motore generi una coppia $\tau_{m,i}$ al tempo t_i. Per rendere più verosimile il modello consideriamo anche la coppia di attrito τ_a tra perno e pendolo che è proporzionale alla velocità angolare di quest'ultimo secondo la relazione $\tau_a = k_d\omega_i$. La coesistenza dei momenti meccanici dovuti a gravità, attrito e motore generano complessivamente un'accelerazione del pendolo

$$\alpha_i = \frac{mgl_c \sin(\theta_i) + \tau_{m,i} - k_d\omega_i}{I}$$

che modifica velocità e inclinazione del pendolo in

$$\omega_{i+1} = \omega_i + \alpha_i(t_{i+1} - t_i)$$

$$\theta_{i+1} = \theta_i + \omega_i(t_{i+1} - t_i) + \frac{1}{2}\alpha_i(t_{i+1} - t_i)^2$$

Come già anticipato, la coppia generata da un motore elettrico dipende anche dalla sua velocità di rotazione, quindi, per successive analisi, è bene evidenziare che la velocità angolare del rotore si modifica nel tempo secondo la formula

$$\omega_{r,i+1} = \omega_{r,i} + \frac{\tau_{m,i}}{I_r}(t_{i+1} - t_i)$$

Come approfondito nei capitoli successivi, con le formule appena descritte è possibile simulare abbastanza fedelmente il comportamento del sistema, in particolare si può descrivere come varia l'angolo di inclinazione del pendolo col passare del tempo. In Figura 2.2 è riportato un andamento tipico dell'angolo θ ottenuto da una simulazione in cui il motore non era alimentato ($\tau_{m,i} = 0 \quad \forall i$).

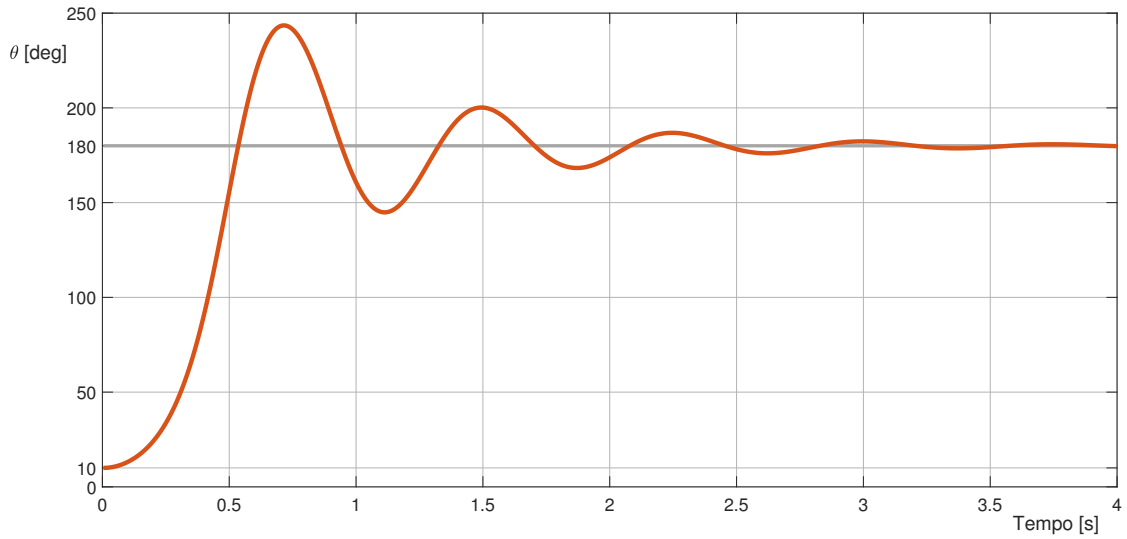

Figura 2.2: Angolo di inclinazione del pendolo in caduta a partire da 10° senza alcun sistema di controllo. Il sistema tende a stabilizzarsi nel punto di equilibrio $\theta = 180°$, ovvero quello in cui il pendolo è in verticale sotto al perno.

Capitolo 3

Componenti utilizzati

Dopo aver compreso le caratteristiche fisiche del sistema, si è passati alla realizzazione del prototipo funzionante. Come prima cosa è stato fondamentale decidere quali componenti fossero più opportuni per ottenere le prestazioni desiderate, tenendo anche conto del costo e della loro disponibilità. Chiaramente per realizzare un sistema ad alte prestazioni sarebbe necessario investire in componenti di alto livello, ma ciò non è prerogativa di questa tesi.

3.1 Motore elettrico

Per prima cosa ci si è concentrati sulla scelta del motore elettrico, in particolare tra motori con e senza spazzole. I motori senza spazzole (brushless) sono caratterizzati da magneti permanenti posti sul rotore e solitamente tre gruppi di bobine disposte sullo statore. Per controllare un motore brushless serve un apposito driver che fornisce con una temporizzazione specifica una tensione sinusoidale ad ogni bobina. Per aumentare la velocità di rotazione il driver modifica la tensione di ogni bobina e aggiusta di conseguenza la temporizzazione. L'inversione del senso di rotazione è l'operazione più problematica, in quanto viene totalmente gestita dal driver che si preoccupa di rallentare la velocità di rotazione per poi generare la sequenza di segnali adatta alla rotazione in senso opposto. Nello specifico il driver testato è controllato tramite un ingresso analogico per la velocità di rotazione e uno binario per il verso di rotazione. Dato che i segnali che controllano le bobine dipendono dal driver, è chiaro che i segnali forniti per controllare il driver non hanno una conseguenza immediata

e precisa sul comportamento effettivo del motore, ma dipendono dallo stato in cui si trova il rotore.

I motori con le spazzole, invece, sono composti da magneti permanenti sullo statore e una bobina sul rotore che cambia polarità automaticamente grazie alle spazzole che collegano la bobina all'alimentazione esterna. Per controllare un motore di questo tipo basta fornire una corrente continua tra i due poli del motore. La velocità di rotazione ω e la coppia del motore τ dipendono dalla corrente fornita I secondo la relazione [3]:

$$\tau = k_t I - \frac{{k_t}^2}{R}\omega \tag{3.1}$$

dove k_t è una la costante di coppia che dipende dalle caratteristiche fisiche del motore ed R è la resistenza elettrica della bobina interna al motore.

Poiché il sistema di controllo implementato dall'Arduino può intervenire direttamente sulla corrente, utilizzando un motore con le spazzole si è in grado di controllare facilmente quanta coppia viene generata dal motore, mentre utilizzando un motore brushless questa correlazione presenta non poche difficoltà. Al netto di questa considerazione, bisogna anche tenere conto che un motore brushless modifica o inverte la propria velocità di rotazione in molto più tempo di un motore con le spazzole, ciò si traduce in un ritardo tra l'uscita del controllore e la modifica dello stato del sistema.

Per le considerazioni fatte sopra, è logico concludere che la scelta migliore è il motore con le spazzole, ma restano da decidere le specifiche. A causa della disponibilità ridotta di driver in corrente continua e motori sufficientemente potenti, si è optato per un motore da 12V di piccole dimensioni. A seguito delle misurazioni effettuate, è stato possibile rilevare che tale motore ha una resistenza interna $R = 5\Omega$ e una costante di coppia $k_t = 0,002\text{Nm/A}$.

3.2 Accelerometro e giroscopio

Per conoscere l'angolo di inclinazione del pendolo sarebbe stato molto più semplice utilizzare un encoder rotazionale a basso attrito con un elevato numero di impulsi per rotazione, ma, a causa della mancata disponibilità di tale componente, si è optato per l'impiego di accelerometro e giroscopio. Una soluzione di questo tipo si adatta molto più facilmente ad utilizzi diversi da questo specifico prototipo e permette, con i giusti accorgimenti, delle misurazioni molto precise.

Un accelerometro è un sensore che misura le accelerazioni lungo un asse. Tipicamente i moduli disponibili in commercio sono composti da tre accelerometri disposti ortogonalmente sui tre assi spaziali, in questo modo un'accelerazione lungo una direzione qualsiasi viene rilevata attraverso la misura delle sue componenti cartesiane. Ipotizziamo, come nel nostro caso, che l'accelerazione da misurare si trovi sempre sullo stesso piano; posizionando due degli assi spaziali misurati dall'accelerometro su tale piano (Figura 3.1), con le note relazioni trigonometriche, si ottiene:

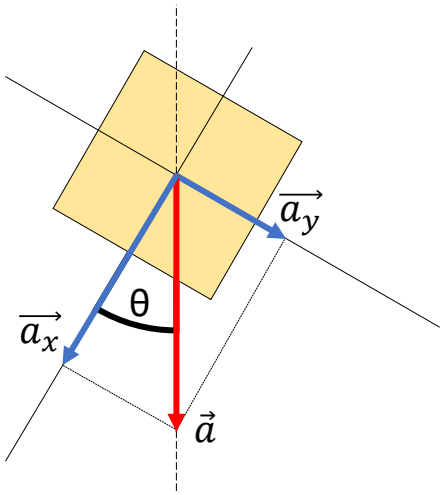

Figura 3.1: Scomposizione di una accelerazione lungo gli assi cartesiani.

$$\theta = \arccos\left(\frac{a_x}{a}\right) = \arcsin\left(\frac{a_y}{a}\right)$$

$$\Rightarrow \theta = \arctan\left(\frac{a_y}{a_x}\right) \tag{3.2}$$

Dato che la gravità fornisce costantemente un'accelerazione di $1g$ lungo l'asse verticale, posizionando i due assi sul piano verticale, si può applicare quanto detto sopra per calcolare l'angolo di inclinazione con l'asse verticale. Ovviamente l'angolo così ottenuto è attendibile solo se l'accelerazione rilevata dal sensore è dovuta esclusivamente alla gravità, cioè in assenza di disturbi.

Per misurare l'angolo di inclinazione si può anche utilizzare un giroscopio, ovvero un sensore in grado di misurare la velocità di rotazione intorno al proprio asse. Se la rotazione è a velocità costante, campionandone il valore ω_r tra intervalli di tempo noti Δt, è possibile calcolare l'angolo percorso durante ogni intervallo con la formula

$$\Delta\theta = \omega_r \Delta t \tag{3.3}$$

In questo modo, se al tempo di campionamento t_i è noto l'angolo di inclinazione $\theta(t_i)$, al successivo campionamento t_{i+1} si avrà

$$\theta(t_{i+1}) = \theta(t_i) + \omega_{r,t_{i+1}}(t_{i+1} - t_i) \tag{3.4}$$

Bisogna osservare che per misurare l'angolo di inclinazione con la procedura appena descritta si è assunto che il giroscopio si muova a velocità costante tra i due istanti di campionamento, cosa che in generale non è vera. Questa approssimazione col passare dei campioni porta inevitabilmente all'accumularsi di un errore tra l'angolo reale e quello misurato, ovvero l'angolo deriva col passare del tempo. Più si diminuisce l'intervallo di campionamento, più si riduce l'errore di approssimazione sull'angolo percorso tra due istanti di campionamento, ma va fatto notare che nello stesso arco di tempo si sommano gli errori di più misurazioni, perciò la frequenza di campionamento da sola non basta a risolvere il problema.

Ricapitolando quanto detto fino a qui, l'accelerometro fornisce l'angolo di inclinazione senza accumulare errore col passare del tempo, ma è molto sensibile ai disturbi, mentre il giroscopio, con un opportuno campionamento, rileva quasi senza disturbi le repentine variazioni dell'angolo, ma accumula errore col tempo. In altre parole, l'accelerometro è affidabile solo sulle basse frequenze di variazione dell'angolo, mentre il giroscopio solo sulle alte frequenze. Per utilizzare solo la parte di segnale utile proveniente da ognuno di questi sensori, una strategia possibile è quella di applicare un filtro passa-basso al segnale proveniente dall'accelerometro e uno passa-alto a quello del giroscopio. L'insieme di questi due filtri può essere ottimizzato tramite l'implementazione di un filtro complementare, che verrà illustrato nel Capitolo 4.

Accelerometro e giroscopio sono stati inseriti nel prototipo all'interno del modulo MPU-6050 [4] montato in cima al pendolo. Questo sensore è composto da un accelerometro a tre assi, un giroscopio a tre assi e un termometro. Il sensore è posizionato verticalmente in modo da avere due assi dell'accelerometro sul piano verticale per le ragioni spiegate prima. La comunicazione avviene tramite il protocollo I2C, che può essere gestito in modo semplice tramite la libreria `Wire.h` di Arduino.

3.3 Altri componenti

Come già anticipato, l'interazione tra tutti i componenti elettronici e l'implementazione del controllore PID avvengono attraverso una scheda Arduino Uno che si basa su un processore ATmega328. L'Arduino possiede 6 ingressi analogici e 14 pin digitali I/O, di cui 6 predisposti per output PWM. La scheda è alimentata con una tensione di 5V che nel nostro caso viene utilizzata anche per alimentare l'MPU-6050. La comunicazione I2C con quest'ultimo avviene tramite due ingressi dedicati, mentre

due output PWM sono utilizzati per comandare il motore elettrico.
Il codice è scritto e compilato tramite l'IDE Arduino che utilizza un linguaggio derivato da C, poi viene caricato sulla scheda che lo può eseguire anche senza essere collegata ad un calcolatore.
Il motore elettrico viene controllato tramite un driver in corrente continua alimentato con 12,5V. Due delle quattro uscite sono collegate ai due poli del motore e vengono controllate tramite i due segnali PWM provenienti dall'Arduino.

La struttura del pendolo è in plastica PLA ed è stata realizzata con la stampa 3D. Il modello è stato progettato da zero per adattarsi al meglio alla forma del motore elettrico e del perno su cui ruota. In maniera analoga è stato realizzato anche un disco da attaccare all'asse del motore, mentre gli altri dischi creati sono stati prodotti in legno con un laser cutter. Per finire staffe e viti in metallo tengono i vari componenti in posizione su un asse di legno che fa da supporto.

Capitolo 4

Sviluppo del software

Una volta selezionati i componenti, si è passati allo sviluppo del software (riportato integralmente in Appendice A). Poiché l'Arduino opera a tempo discreto, il calcolo dell'angolo di inclinazione e l'implementazione del sistema di controllo devono basarsi su campioni discreti delle grandezze misurate. La frequenza di campionamento, fissata a 250 Hz per motivi che saranno chiariti nella Sezione 4.3, viene rispettata assicurandosi che nel codice il tempo di ciclo sia esattamente di 4ms.
L'intero codice può essere riassunto con lo schema seguente.

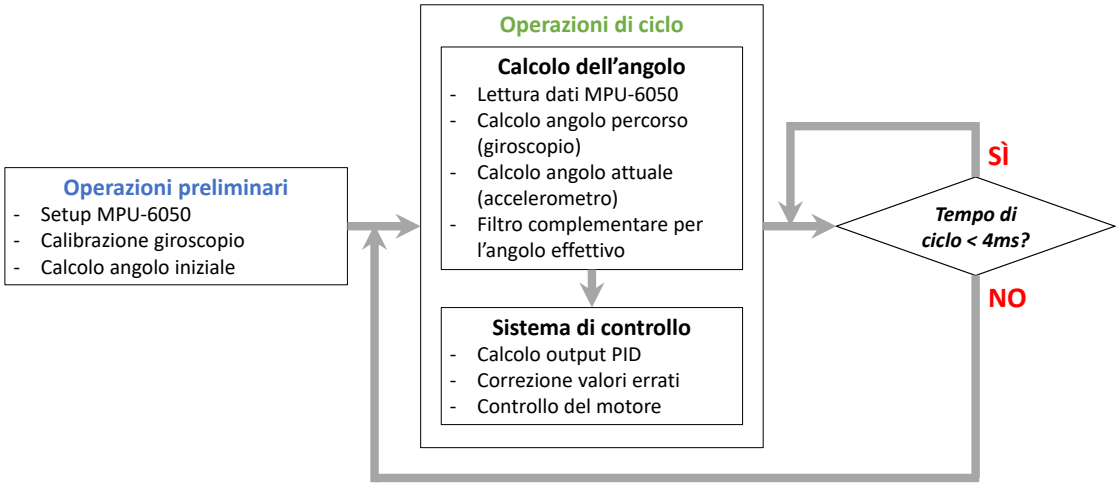

Figura 4.1: Schema riassuntivo del codice Arduino.

All'avvio del programma è necessario attivare l'MPU-6050 e impostare il fondo scala di accelerometro e giroscopio. Tali operazioni sono eseguite nella funzione `setup_mpu_6050_registers()` (righe 124-143) tramite le funzioni della libreria `Wire.h` per la comunicazione I2C. Si è verificato sperimentalmente che valori di fondo scala di $2g$ per l'accelerometro e di $500°/s$ per il giroscopio producono dati accurati e con una buona risoluzione.

Per la lettura dei dati grezzi si utilizza la funzione `read_mpu_6050_data()` (righe 105-122) che aggiorna le variabili globali `gyro_z`, `acc_x`, `acc_y` con gli ultimi valori registrati dall'MPU-6050. Come si vede in Figura 4.2, `gyro_z` è la velocità angolare rispetto all'asse z, mentre `acc_x` e `acc_y` sono rispettivamente le accelerazioni lungo l'asse x e lungo l'asse y.

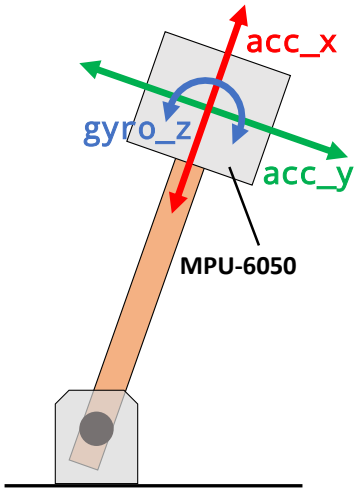

Figura 4.2: Rappresentazione fisica delle variabili lette dall'MPU-6050.

Per eliminare errori sistematici nella lettura dei dati forniti dal giroscopio, all'avvio viene eseguita anche una calibrazione del sensore. Con il pendolo fermo vengono letti per 1500 volte ogni 3 ms i valori rilevati dal giroscopio, se ne calcola la media e la si usa come offset da sottrarre ad ogni successiva lettura del sensore. Il ciclo di calibrazione viene utilizzato anche per calcolare il valor medio dei dati letti dai due accelerometri, in questo modo l'angolo iniziale ricavato successivamente è più accurato.

```
35  setup_mpu_6050_registers();     //Setup MPU-6050(+/-500 deg*s | +/-2g)
36
37  //Calcolo offeset medio
38  for (int i = 0; i < 1500 ; i ++){  //Calibrazione sensori
39    read_mpu_6050_data();            //Lettura dati grezzi da MPU-6050
40    gyro_z_cal += gyro_z;
41    val_x += acc_x;
42    val_y += acc_y;
43    delay(3);
44  }
45  gyro_z_cal /= 1500;                //Offset giroscopio
46  val_x /= 1500.0;                   //Valore medio iniziale acc
47  val_y /= 1500.0;
```

L'ultima operazione preliminare da svolgere a questo punto è il calcolo dell'angolo di inclinazione iniziale a partire dai soli dati forniti dall'accelerometro. Ciò è necessario perché, in ognuno dei successivi cicli, il calcolo dell'angolo di inclinazione dipende anche dal valore calcolato nel ciclo precedente. Per fare ciò si utilizza l'equazione 3.2 con l'accorgimento di trasformare il risultato fornito da `atan()` da radianti in gradi.

```
48   angle_out=-atan((float)val_y/val_x)* 57.29578;
```

A questo punto si può passare alle operazioni di ciclo, le quali sono divisibili in due blocchi ben distinti: il primo per il calcolo dell'angolo di inclinazione a partire dai dati provenienti dall'MPU-6050 e il secondo per il sistema di controllo, la cui uscita comanda il motore.

4.1 Calcolo dell'angolo

Come prima cosa i dati grezzi di accelerometri e giroscopio vengono letti tramite la funzione `read_mpu_6050_data()` citata precedentemente. Poi, dal valore di `gyro_z` corretto dall'offset, si calcola l'angolo percorso dall'ultimo campionamento tramite l'equazione 3.3. Dal datasheet dell'MPU-6050 [4] è noto che con il fondo scala di $500°/s$ la cifra meno significativa di `gyro_z` vale $\frac{1}{65.5}°/s$, inoltre, poiché la frequenza di ciclo è fissata e vale 250 Hz, si ha l'istruzione di riga 59.

```
55   read_mpu_6050_data();                //Lettura dati grezzi
56
57   //Angolo percorso dall'ultimo campionamento (giroscopio)
58   gyro_z -= gyro_z_cal;                //Sottrazione dell'offset
59   gyro_vel = gyro_z * (1/(250*65.5));   //Angolo percorso
```

Si passa dunque al calcolo dell'angolo attuale tramite `acc_x` e `acc_y`. Per ridurre fin da subito l'effetto del rumore sull'accelerometro, si filtrano i dati con un semplice filtro IIR del primo ordine [3] regolabile tramite il parametro β.

$$y[n] = \beta a[n] + (1 - \beta)y[n - 1]$$

Infine, a partire dai valori filtrati `val_x` e `val_y`, sempre tramite la formula 3.2, si calcola l'angolo d'inclinazione rilevato dagli accelerometri (riga 64).

```
61   //Angolo calcolato con l'accelerometro
62   val_y=(1-beta)*val_y+beta*acc_y;      //Filtro sui dati dell'acc
63   val_x=(1-beta)*val_x+beta*acc_x;
64   angle_acc = -atan((float)val_y/val_x)* 57.29578;   //Calcolo angolo acc
```

A questo punto l'angolo d'inclinazione effettivo `angle_out` viene calcolato mediante un filtro complementare del primo ordine [3][5] descritto dall'equazione

$$y[n] = \alpha\theta_a[n] + (1-\alpha)(y[n-1] + \Delta\theta_g[n]) \tag{4.1}$$

che nel codice è applicato nella riga 67.

```
66   //Angolo di output (filtro complementare)
67   angle_out= (1-alpha)*(angle_out+gyro_vel)+alpha*angle_acc;
```

Ottenuto il valore di `angle_out` si può passare al secondo blocco di operazioni di ciclo per l'implementazione del sistema di controllo.

4.2 Sistema di controllo

Il sistema di controllo PID [1][3] come prima cosa calcola l'errore come la differenza tra l'angolo target `set_a` e l'angolo `angle_out` calcolato nel segmento di ciclo precedente.

```
70   PID_error = set_a - angle_out;          //Errore
```

Poi viene calcolata l'approssimazione discreta dell'integrale dell'errore come la somma degli errori precedenti moltiplicati per il tempo di ciclo `enlapsedTime` che vale 0.004 s.

```
71   integral+= PID_error*elapsedTime;       //Calcolo dell'integrale
```

Va osservato che se il sistema si trovasse in una posizione lontana dall'equilibrio per tanto tempo, come ad esempio in seguito ad una caduta, il valore dell'integrale potrebbe raggiungere valori molto elevati. In tale situazione il sistema di controllo comincerebbe a comportarsi in modo indesiderato fino a quando l'integrale non si fosse scaricato, ovvero quando il suo valore ritornasse prossimo allo zero. Per aumentare la velocità di azzeramento dell'integrale e quindi evitare comportamenti anomali del sistema di controllo, si impone che il valore assoluto dell'integrale non superi il valore 100 dato che, come si è verificato dai test eseguiti, ciò non accade mai in assenza di una caduta.

```
72   if(integral>100) integral=100;          //Saturazione integrale
73   if(integral<-100) integral=-100;
```

A questo punto si può passare al calcolo dell'approssimazione discreta della derivata dell'errore tramite la formula

$$\mathrm{D}(e_a) = \frac{\Delta e_a}{\Delta t} = \frac{e_a[n] - e_a[n-1]}{\Delta t}$$

dove $e_a[n]$ è l'errore attuale ed $e_a[n-1]$ è l'errore del ciclo precedente. Nel codice tale operazione è svolta alla riga 74 riportata sotto.

```
74   derivate=(PID_error - previous_error)/elapsedTime;     //Derivata
```

Infine, tramite i parametri `kp`, `ki` e `kd` dichiarati all'inizio del codice, si può calcolare il valore di uscita del controllore PID.

```
75   PID_value= kp*PID_error + ki*integral + kd*derivate;  //Output
```

Per motivi di praticità, quando il sistema si trova a un'inclinazione eccessiva per essere riportato nella posizione di equilibrio, si inibisce l'azione del sistema di controllo azzerando il valore di `PID_value` calcolato prima.

```
78   //Se l'angolo e' troppo elevato si inibisce il controllo
79   if (angle_out>8 || angle_out<-8) {
80     PID_value=0;
81     integral=0;
82   }
83
84   controlDC(PID_value);            //Output controllore comanda il motore
```

Il motore è controllato tramite la funzione `controlDC(float vel)` che fa in modo di alimentarlo con una tensione proporzionale a `vel`. In particolare, per `vel=255` la tensione fornita è la massima applicabile dal driver, mentre per `vel=-255` verrà fornita la massima tensione ma con polarità opposta. Tutti i valori di `vel` compresi tra $+255$ e -255 sono accettati e corrispondono a tutti gli altri valori intermedi di tensione, mentre i valori eccedenti vengono saturati ad uno dei due estremi. Poiché la generazione di segnali PWM da parte dell'Arduino prevede valori interi compresi tra 0 e 255, il valore di `vel` viene trasformato in un una coppia di PWM da applicare ai due PIN `pa` e `pb` connessi al driver.

```
91 void controlDC( float vel){        //Controllo del motore
92
93   if (vel<-255) vel=-255;          //Valori eccedenti vengono saturati
94   if (vel>255) vel=255;
95   if (vel<0){
96     analogWrite(pb,0);
97     analogWrite(pa,(int) abs(vel));
98     }
99   else {
100    analogWrite(pa,0);
101    analogWrite(pb,(int) vel);
102    }
103 }
```

Come accennato all'inizio del capitolo, per garantire la frequenza di campionamento di 250 Hz, al termine delle operazioni di ciclo è presente un *while* che assicura un ritardo temporale di 4ms tra il ciclo attuale e quello precedente.

```
86   //Controllo del tempo di ciclo (freq di 250Hz)
87   while(micros() - loop_timer < 4000);   //Aspetta che passino 4000us
88   loop_timer = micros();           //Reset loop timer
89 }
```

4.3 Analisi delle prestazioni

Dopo svariate prove sperimentali è stato possibile individuare il valore più adeguato dei parametri del codice.

Parametro	Valore	Ruolo
`set_a`	$-1.5 \div 1.5$	Target controllore PID
`beta`	0.2	Coefficiente filtro IIR accelerometro
`alpha`	0.001	Coefficiente filtro complementare
`kp`	121	Coefficiente proporzionale controllore PID
`ki`	0.4	Coefficiente integrale controllore PID
`kd`	5	Coefficiente derivativo controllore PID

Tabella 4.1: Valori ottimali dei parametri all'interno del codice.

Il valore di `set_a` dipende dal montaggio dell'MPU-6050 sul pendolo. Con un posizionamento ottimale il valore da impostare sarebbe 0, ma, essendo difficile da ottenere, risulta più pratico sistemare il pendolo in posizione verticale ed assegnare a `set_a` il valore letto dal sensore in quella posizione.

Il parametro `beta` permette di regolare la risposta del filtro IIR per la riduzione del rumore rilevato dall'accelerometro. Con valori prossimi a 1 il filtraggio diventa quasi inesistente e il rumore inquina i dati rendendoli inutilizzabili, mentre con valori prossimi allo 0 il dato in uscita dal filtro risente molto poco delle misurazioni recenti restando stabile attorno al suo valore precedente.

Per quanto riguardo il parametro `alpha`, responsabile della risposta del filtro complementare, con valori prossimi allo 0 l'angolo calcolato con il giroscopio ha più influenza sull'uscita rispetto a quello proveniente dall'accelerometro, mentre con valori prossimi a 1 avviene il contrario. Dato che le variazioni dell'angolo sono per lo più repentine, si cerca di utilizzare quasi esclusivamente l'angolo calcolato dal giroscopio correggendone la deriva temporale con quello ottenuto dell'accelerometro. In questo modo l'angolo di inclinazione è soggetto il meno possibile al rumore, risponde

in maniera precisa alle variazioni brusche e non subisce la deriva temporale causata dal giroscopio. Variazioni anche piccole di tale parametro modificando di molto l'accuratezza dell'angolo effettivo passato al sistema di controllo, per questo la ricerca del valore ottimale di `alpha` è stata la più problematica.

I tre coefficienti del controllore PID sono quelli da cui dipende il funzionamento del sistema di controllo. La ricerca della configurazione ottimale non può essere eseguita un parametro alla volta, bensì necessita di successivi aggiustamenti graduali di ognuno dei tre valori [3]. Il coefficiente `kp` è il primo da determinare per avere un ordine di grandezza della relazione tra errore e valore di uscita. Nel nostro caso il valore di tale parametro si aggira intorno a 100, ciò significa, in prima approssimazione, che con un angolo di inclinazione di $2,55°$ la tensione applicata al motore è la massima disponibile. Si è poi passati a incrementare progressivamente il parametro `kd` ottenendo già un effetto di stabilizzazione discreto. Va fatto notare che all'aumentare di tale parametro il sistema diventa più reattivo alle minime variazioni di inclinazione, ciò significa che eventuale rumore residuo nell'angolo fornito in input può creare reazioni brusche ed indesiderate del sistema di stabilizzazione. Infine, per rendere il sistema più robusto rispetto ad asimmetrie del pendolo e del posizionamento dell'MPU-6050, è stato regolato il parametro `ki` facendo attenzione al fatto che un impatto troppo elevato della componente integrale rallenta il tempo di assestamento del sistema.

Oltre ai parametri appena elencati, un altro elemento discriminante per ottenere delle buone prestazioni del sistema di controllo è il tempo di ciclo. Al diminuire del tempo di ciclo il campionamento dell'angolo di inclinazione ed il calcolo del nuovo valore di uscita dal sistema di controllo si fanno più frequenti, ciò permette di avere una stabilizzazione più precisa e reattiva. Per contro, maggiore è la frequenza di ciclo, minore è il numero di operazioni eseguibili al suo interno. Inoltre, a causa del rumore nei dati provenienti dai sensori, un campionamento particolarmente fitto non assicura necessariamente delle prestazioni migliori. Nel codice finale si è deciso di utilizzare una frequenza di campionamento di 250 Hz. Sebbene sia stato verificato che il tempo richiesto dalla scheda Arduino per svolgere tutte le operazioni di ciclo non supera mai i 3 ms, la frequenza non è stata aumentata ulteriormente perché i campioni dell'angolo d'inclinazione assumono già un andamento pressoché continuo.

Nonostante tutte le considerazioni fatte fino a qui, le prestazioni ottenute dal prototipo non sono particolarmente soddisfacenti e uno dei motivo di ciò è che il rumore residuo presente nell'angolo di inclinazione impedisce di aumentare ulteriormente il

coefficiente `kd`. Probabilmente un posizionamento diverso dell'MPU-6050 ed una struttura del pendolo meno rigida ridurrebbero l'effetto del rumore permettendo una regolazione più efficace dei parametri del controllore PID. Per ragioni che verranno discusse nel Capitolo 5, un modo per migliorare le prestazioni del sistema è quello che prevede l'utilizzo di un motore diverso, con più coppia e minor numero di giri. Tale soluzione porterebbe un notevole vantaggio anche in termini di riduzione delle vibrazioni, permettendo così di aumentare la componente derivativa del controllore.

Capitolo 5

Simulazione

Realizzato il prototipo funzionante si è tornati a studiare il problema in maniera più teorica tramite simulazioni eseguite con MATLAB. Le simulazioni hanno lo scopo di calcolare l'evoluzione del sistema nel tempo tenendo in considerazione in maniera più accurata possibile i fenomeni fisici coinvolti. Mediante l'analisi dei dati ottenuti è possibile capire i limiti e il margine di miglioramento del prototipo reale fino a comprendere sotto quali ipotesi questo sistema di stabilizzazione risulta maggiormente efficace.

5.1 Progettazione

Per effettuare le simulazioni è stato realizzato uno script MATLAB (riportato in Appendice B) che, a partire dallo stato iniziale del sistema, calcola i successivi stati del sistema come campioni a tempo discreto. Poiché la simulazione opera a tempo discreto, il suo risultato sarà sempre solo un'approssimazione della reale evoluzione del sistema, ma permette comunque di ricavare informazioni preziose per lo studio del caso reale. Inoltre, essendo troppo dispendioso tenere conto di tutti i fenomeni fisici che influenzano il sistema, si fa un'ulteriore approssimazione considerando solo quelli principali. In tabella 5.1 vengono riportate le variabili all'interno dello script che esprimono le grandezze fisiche di cui la simulazione tiene conto. Tali variabili vengono dichiarate nella parte iniziale del codice (righe 1-35).

Variabile	Grandezza fisica	Unità di misura
`th`	Angolo di inclinazione del pendolo	°
`vel`	Velocità angolare del pendolo	°/s
`vmot`	Velocità angolare del rotore	°/s
`mt`	Massa dell'intero pendolo	kg
`dc`	Distanza perno - centro di massa del pendolo	m
`It`	Momento di inerzia dell'intero pendolo	kg·m^2
`Ir`	Momento di inerzia del rotore	kg·m^2
`kd`	Coefficiente di attrito dinamico	Nm·s/°
`kt`	Costante di coppia del motore elettrico	Nm/A
`V`	Massima tensione applicabile al motore	V
`R`	Resistenza elettrica del motore	Ω

Tabella 5.1: Significato delle variabili all'interno della simulazione.

La simulazione vera e propria viene eseguita nella parte ciclica dello script (righe 37-68) in cui viene aggiornato lo stato del sistema a partire dello stato precedente e dalle equazioni dei fenomeni fisici presi in considerazione. Per prima cosa viene riprodotto il comportamento del controllore inserendo nel codice le stesse istruzioni che esegue l'Arduino, ovvero quelle precedentemente illustrate nella Sezione 4.2.

```
40      % Calcoli del controllore PID (identici a quelli su Arduino)
41      err=targ-th;                        % Errore
42      erri=erri+err*t;                    % Integrale
43      if abs(erri)>100                    % Saturazione integrale
44          erri=(erri/abs(erri))*100;
45      end
46      errd=(err-erro)/t;                  % Derivata
47      erro=err;                           % Aggiornamento errore precedente
48      pwm=round(P*err+D*errd+I*erri);     % Output controllore come valore di PWM
49
50      if abs(pwm)>255                     % Saturazione PWM non consentiti
51          pwm=(pwm/abs(pwm))*255;
52      end
```

A questo punto, tenendo conto di come l'uscita del controllore influenzi la coppia e la velocità del motore (equazione 3.1), si passa a calcolare la dinamica del pendolo tramite le equazioni illustrate nell'ultima parte del Capitolo 2.

```
54      % Corrente fornita al motore
55      corr=-(pwm/255)*(V/R);          % Corrente massima per pwm=255
56      taum=kt*corr-((kt^2)/R)*vmot;   % Coppia fornita dal motore
57
58      % Dinamica dell'intero pendolo
59      acc=(mt*dc*9.81*sind(th)-taum-kd*vel)/It;
60      th=th+0.5*(t^2)*acc+vel*t;
61      vel=vel+acc*t;
62
63      % Dinamica del rotore
64      vmot=vmot+(taum/Ir)*t;
```

Con lo script appena descritto, salvando all'interno di ogni ciclo il valore delle grandezze di interesse, è possibile rappresentare sotto forma di grafico l'evoluzione del sistema. Ad esempio, tenendo traccia dell'angolo `th`, si ottiene il grafico nella Figura 5.1.

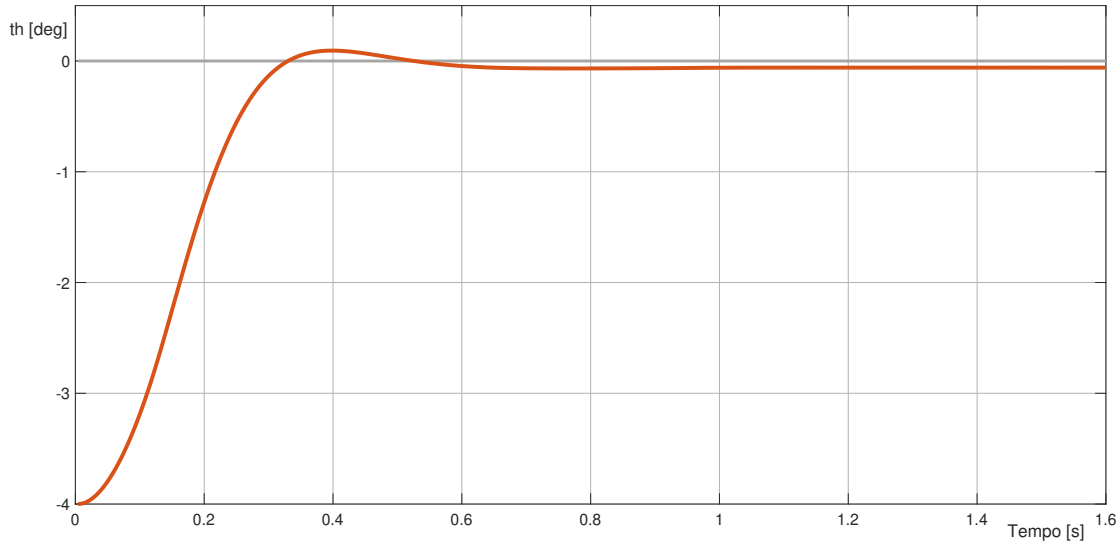

Figura 5.1: Andamento dell'angolo di inclinazione del pendolo stabilizzato a partire da $-4°$ di inclinazione. Simulazione basata sui parametri del prototipo reale (vedi Tabella 5.2).

5.2 Confronto

Realizzato lo script si è passati alla stima dei parametri fisici del prototipo reale. Per le grandezze che lo permettevano sono state effettuate misure dirette o indirette

utilizzando un calibro e una bilancia. Per le altre invece si è stimato grossolanamente il valore e lo si è corretto mediante confronto tra andamento reale e simulato del sistema. I valori ottenuti sono riportati in Tabella 5.2.

Variabile	**Valore**		**Ottenuto tramite**
`mt`	$0,07$	kg	Misura diretta
`dc`	0.06	m	Misura diretta
`It`	$1 \cdot 10^{-5}$	kg·m^2	Confronto
`Ir`	$6 \cdot 10^{-6}$	kg·m^2	Confronto
`kd`	$8 \cdot 10^{-5}$	Nm·s/°	Confronto
`kt`	$0,002$	Nm/A	Misura indiretta
`V`	$12,5$	V	Misura diretta
`R`	5	Ω	Misura diretta

Tabella 5.2: Valore dei parametri fisici del pendolo reale.

Come si vede dalla Figura 5.2, con i valori scelti i dati calcolati dalla simulazione sono ragionevolmente simili a quelli rilevati dal sistema reale.

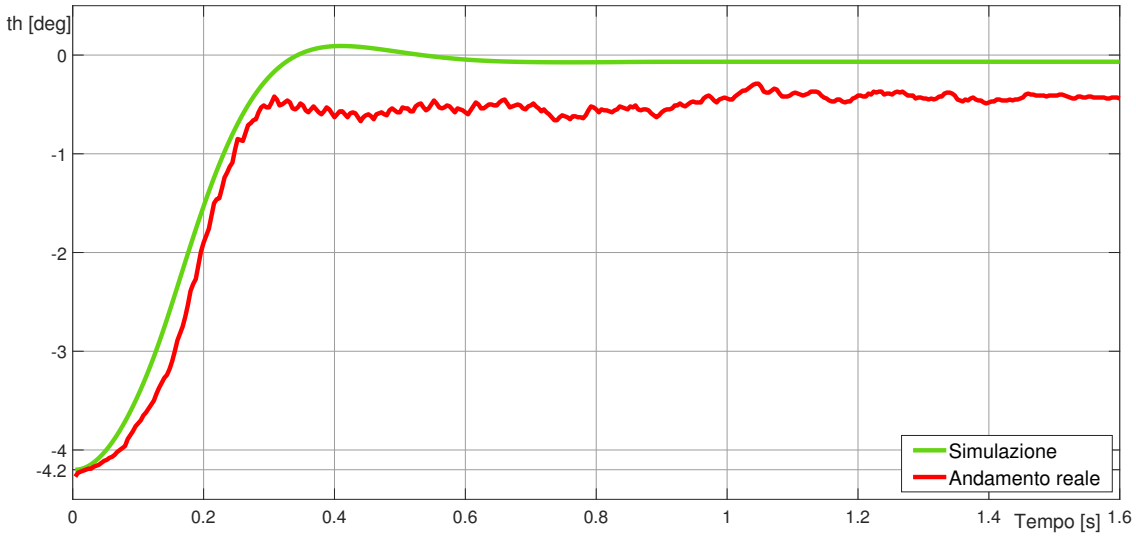

Figura 5.2: Confronto tra simulazione e andamento reale dell'angolo di inclinazione del pendolo stabilizzato a partire da $-4,2°$ di inclinazione.

Dal confronto si nota che la velocità di salita nella simulazione corrisponde a quella reale, ciò è sintomo del fatto che i parametri fisici scelti sono una buona approssimazione di quelli reali. Come previsto l'angolo di inclinazione rilevato dall'Arduino è affetto da rumore, lo si può notare facilmente confrontandolo con quello privo di rumore calcolato nella simulazione. Si può constatare, però, che il rumore ha un'ampiezza contenuta rispetto alle variazioni d'angolo a cui si è interessati, quindi non dovrebbe costituire un grande problema per il corretto funzionamento del sistema di controllo.

Una grossa discrepanza con la simulazione la si nota quando il pendolo si avvicina alla posizione verticale, quella in cui l'angolo di inclinazione vale zero. Avvicinandosi a questa posizione il sistema di controllo comincia ad alimentare il motore in senso opposto facendolo rallentare e cambiare senso di rotazione. In questa fase di transitorio l'equazione 3.1, usata nella simulazione per calcolare la coppia fornita dal motore, non rispetta più abbastanza fedelmente il comportamento reale [3]. Come si vede dal grafico, il pendolo reale si ferma prima di raggiungere gli $0°$ perché la coppia fornita dal motore durante il transitorio è maggiore di quella stimata dalla simulazione. Nel caso reale si vede inoltre che il sistema di controllo, dopo che il pendolo si è fermato intorno agli $-0,5°$, continua a provare a diminuire l'inclinazione pilotando il motore con un PWM piuttosto basso. I motori elettrici però non rispondono bene a tensioni modeste, ciò impedisce al pendolo di raggiungere la posizione verticale dalla sua attuale inclinazione nella quale rimane quindi bloccato.

5.3 Prospettive di miglioramento

Una delle principali limitazioni del sistema di stabilizzazione reale è che la massima coppia che il motore riesce a fornire è di troppo poco superiore alla coppia fornita dalla gravità, ciò significa che superati circa $6°$ di inclinazione il sistema non potrà mai riuscire a tornare in posizione verticale. Conseguenza ancora più grave di ciò è che disturbi anche piccoli applicati al pendolo, sommandosi alla gravità, generano facilmente una coppia superiore a quella che il motore può fornire, rendendo inevitabile la caduta del pendolo.

La soluzione più logica a questo problema è la scelta di un motore in grado di fornire più coppia, solo che in generale motori più potenti sono anche più pesanti e quindi, una volta montati sul pendolo, aumentano la coppia fornita dalla gravità. Per

risolvere questo problema conviene quindi utilizzare motori con motoriduttore che, a parità di peso, producono più coppia pagando con una diminuzione della velocità massima di rotazione. Nel nostro caso la rotazione del rotore genera vibrazioni che aumentano il rumore rilevato dai sensori, quindi il fatto che la velocità di rotazione sia inferiore è a sua volta un pregio. Motori simili a quello scelto nel prototipo ma dotati di motoriduttore hanno una costante di coppia k_t compresa tra $0,004$ e $0,01$ Nm/A in base al rapporto di riduzione utilizzato. Inserendo all'interno della simulazione `kt`$= 0,004$ otteniamo l'andamento di Figura 5.3.

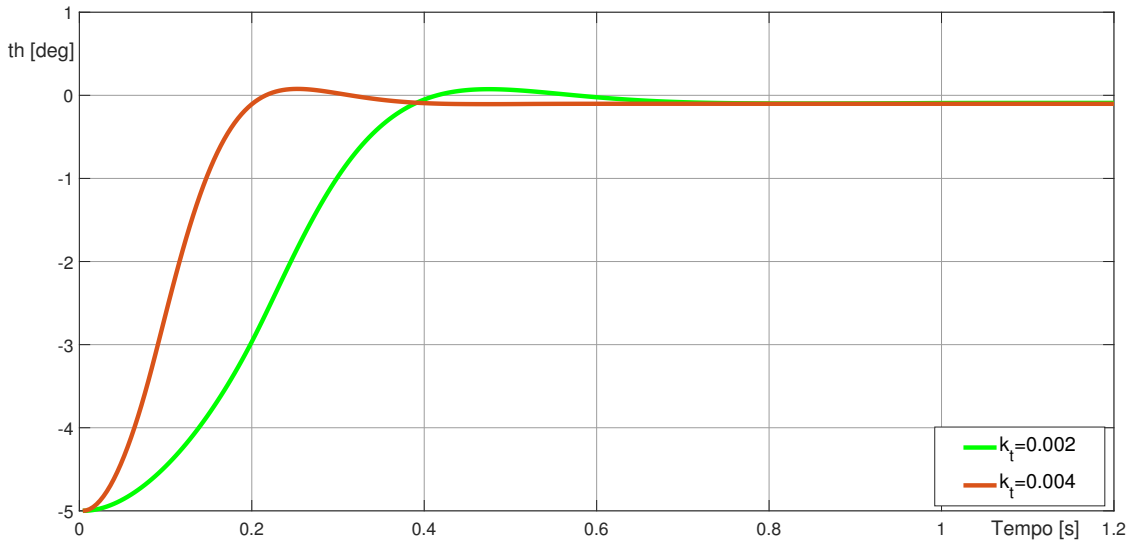

Figura 5.3: Confronto tra simulazioni con valori diversi di k_t.

Come previsto il sistema è molto più reattivo di prima, infatti si osserva che il tempo di salita si è dimezzato. Dopo aver osservato quanto questo singolo parametro faccia la differenza, possiamo dedurre che il principale requisito da soddisfare per rendere il sistema di stabilizzazione più efficace è utilizzare un motore leggero e in grado di fornire molta coppia.

Chiaramente, al variare dei componenti utilizzati, sarebbe bene aggiustare i parametri del controllore PID per ottenere prestazioni ottimali seguendo le indicazioni illustrate nella Sezione 4.3. Regolare tali parametri sulla base delle simulazioni non ha molto senso poiché il sistema di controllo reale è influenzato da non idealità di cui la simulazione non tiene conto. Dato che lo scopo di questo capitolo è confrontare le prestazioni al variare dei parametri fisici del sistema, si è lasciato inalterato il funzio-

namento del sistema di controllo in tutte le simulazioni così da potersi concentrare esclusivamente sull'effetto che ha una variazione di tali parametri.

Dopo aver ipotizzato di utilizzare un motore diverso, proviamo ad ipotizzare di voler stabilizzare un pendolo più lungo e più pesante. All'aumentare di lunghezza e massa del pendolo aumenta la coppia fornita dalla gravità e il momento di inerzia del pendolo, quindi, a parità di motore utilizzato, la stabilizzazione diventa meno efficace. In Figura 5.4 possiamo osservare come si modifica la risposta al gradino al variare di `mt` e `dc`. Va sottolineato che per semplicità nelle simulazioni prese in esame il valore di `It` è rimasto lo stesso di Tabella 5.2. Ovviamente nella realtà un aumento di massa e lunghezza del pendolo provocano anche un aumento del momento di inerzia, rendendo la stabilizzazione ancora meno efficace di quella ottenuta nelle simulazioni in figura.

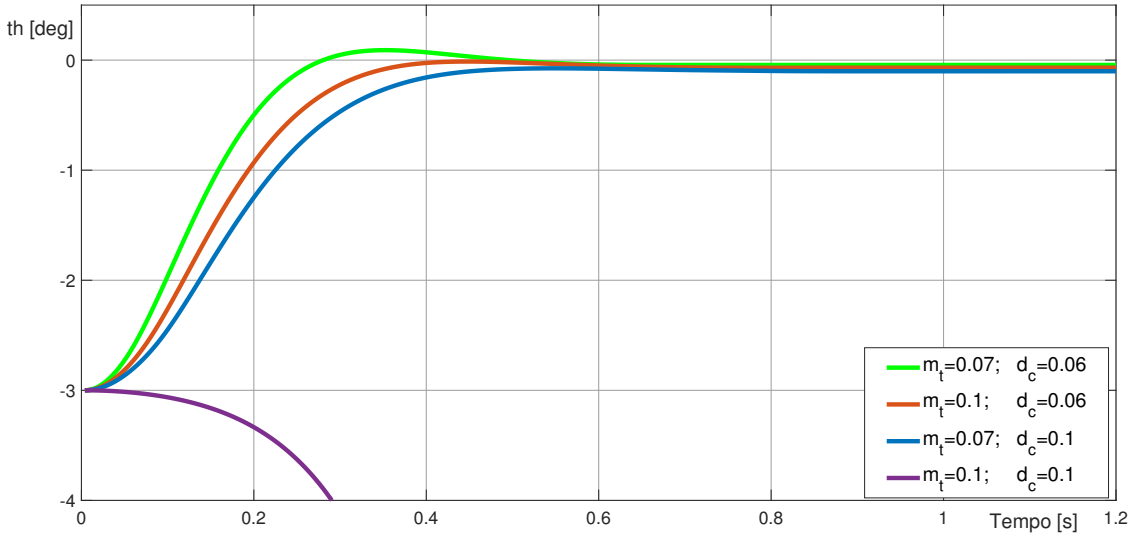

Figura 5.4: Confronto tra simulazioni al variare di massa e lunghezza del pendolo.

Analizzando gli andamenti di Figura 5.4 notiamo che, all'aumentare di massa e lunghezza del pendolo, la coppia generata dal motore riesce a opporsi sempre meno all'effetto della gravità, che finisce quindi per avere la meglio. Infatti nell'ultimo caso, quello che vede un amento del 43% della massa e del 68% della lunghezza, il pendolo cade istantaneamente anche partendo da solo 3° di inclinazione. Risulta subito chiaro che, essendoci un limite alla coppia che i motori elettrici possono fornire, la massa, la lunghezza e l'inerzia del pendolo devono essere in generale piuttosto contenute

per un funzionamento efficace, ciò pone dei grossi limiti sull'utilizzo in larga scala di questo sistema di stabilizzazione.

In base al massimo angolo di inclinazione dal quale si vuole essere in grado di stabilizzare il pendolo va scelto il motore con la k_t più adatta, in particolare il massimo angolo di inclinazione recuperabile $\theta_{\max}$ dipende dalle caratteristiche del motore secondo l'equazione

$$m_t d_c g \sin \theta_{\max} = \tau_{\max} = k_t I$$

dove $\tau_{\max}$ è la massima coppia che il motore è in grado di fornire, ottenuta dall'equazione 3.1. Chiaramente $\theta_{\max}$ è il valore limite oltre il quale il sistema di stabilizzazione non può riuscire a contrastare la gravità, ma da quanto visto nei test reali effettuati, per un buon funzionamento del sistema di controllo sarebbe bene non superare nemmeno l'inclinazione di $\frac{1}{2}\theta_{\max}$.

Dato che motori con $\tau_{\max}$ molto elevata richiedono correnti notevoli e quindi anche fonti di alimentazione adeguate, la stabilizzazione di oggetti grandi come motocicli o velivoli può avvenire solo tramite l'utilizzo di diversi motori e volani, che però rendono l'intero sistema piuttosto complesso. Possiamo quindi concludere che, considerate le prestazioni ottenibili da un motore elettrico, questo sistema di stabilizzazione non è adatto a gestire sistemi con massa del pendolo elevata e/o sistemi da equilibrare a partire da angoli di inclinazione notevoli.

In ultimo analizziamo il momento di inerzia del rotore, un fattore che, anche se in modo meno significativo di quelli visti fino a qui, può modificare il comportamento del sistema. Il vantaggio di avere un rotore con momento di inerzia maggiore è che, a parità di coppia fornita dal motore, l'accelerazione del disco è minore. Come già anticipato, un rotore che gira più lento produce meno vibrazioni, inoltre dall'equazione 3.1 sappiamo che la coppia fornita dal motore è maggiore tanto più piccola è la velocità di rotazione. Il momento di inerzia del rotore può essere incrementato aumentando la massa del disco, ma ciò porta ad un aumento della massa totale del pendolo riportandoci alle considerazioni fatte nei paragrafi precedenti. Dato che un aumento della massa totale del pendolo porta molti più svantaggi di quanti vantaggi porta un maggiore momento di inerzia del rotore, è preferibile agire principalmente sulla forma del disco, ad esempio passando da un disco pieno ad un disco a razze. In questo modo si ottengono i benefici di una maggiore resistenza all'accelerazione del disco senza aumentare la massa complessiva del sistema. Per comprendere meglio quanto appena detto, in Figura 5.5 è riportato il confronto tra gli andamenti di

angolo di inclinazione del pendolo e velocità di rotazione del rotore ottenuti da due simulazioni con valori diversi di `Ir` ma stessa massa del rotore.

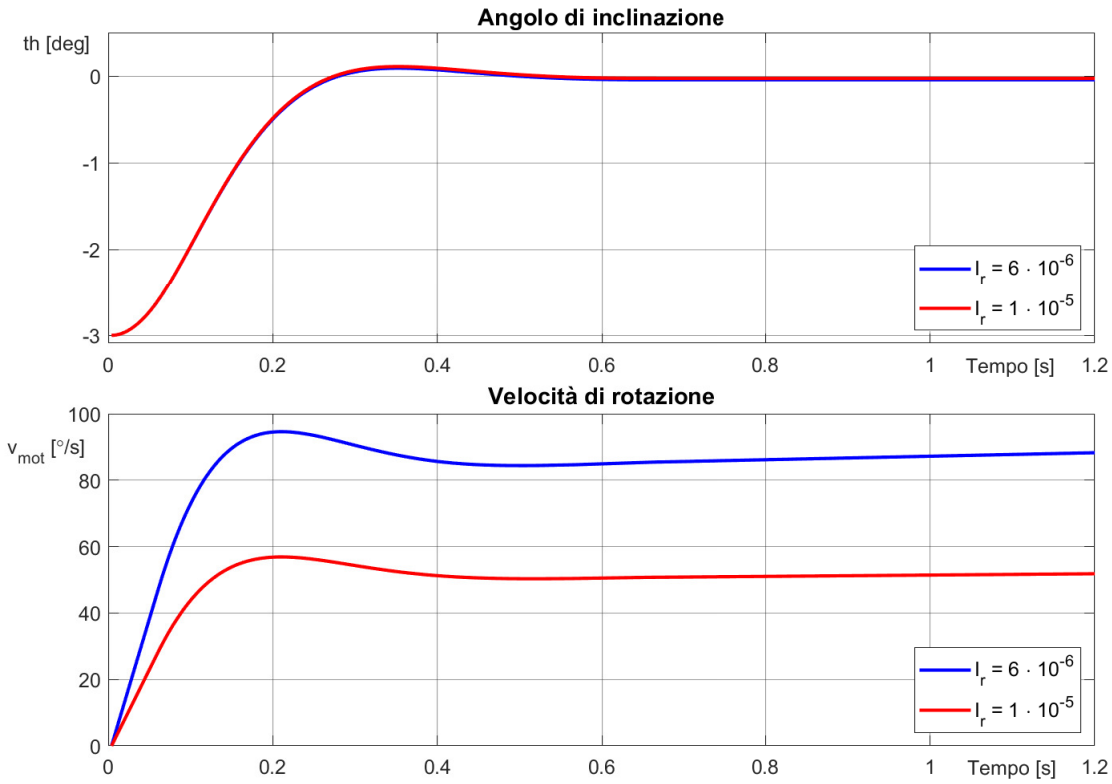

Figura 5.5: Confronto tra simulazioni al variare del momento di inerzia del rotore.

Si può osservare che l'andamento dell'angolo di inclinazione è pressoché invariato, ciò significa che il sistema di controllo non è stato particolarmente influenzato dall'aumento di `Ir`. Invece, osservando il secondo grafico, si nota come l'incremento di `Ir` abbia ridotto complessivamente la velocità di rotazione del rotore portando al sistema tutti i benefici prima citati.

Capitolo 6

Conclusioni

Nel corso di questa tesi sono stati analizzati tutti gli aspetti necessari per realizzare un sistema di stabilizzazione in grado di tenere in equilibrio un pendolo tramite un volano. L'obiettivo di creare un prototipo funzionante è stato raggiunto solo parzialmente in quanto quello costruito, sprovvisto di alcuni componenti chiave per un funzionamento ottimale, opera in maniera piuttosto discontinua poiché accusa eccessivamente rumori e disturbi esterni ed è dotato di un motore che fornisce una coppia troppo modesta. Ad ogni modo, il successivo studio che il prototipo ha permesso si è rivelato cruciale per comprendere quali fattori possono rendere più efficace questo sistema di controllo.

Grazie all'analisi fisica svolta, è stato possibile comprendere i principi che rendono funzionante un sistema di controllo di questo tipo. In particolare si è visto come, grazie alla terza legge di Newton, un motore elettrico riesca a fornire un momento meccanico al pendolo tale da opporsi a quello generato dalla gravità. Per quanto riguarda la lettura dell'angolo di inclinazione del pendolo, la scelta del modulo MPU-6050 ha reso questo sistema di stabilizzazione adattabile più facilmente ai campi di impiego più disparati, ma ha anche richiesto lo studio di alcuni accorgimenti per interpretare e utilizzare correttamente i dati grezzi affetti da rumore.
L'impiego del motore elettrico più adatto si è rivelata la chiave per raggiungere delle buone prestazioni, infatti sono stati evidenziati i benefici che porta averne uno leggero e con molta coppia, magari dotato di motoriduttore.

Il codice Arduino progettato, anche se di appena 143 righe, vede implementato al suo interno un controllore PID che opera a tempo discreto ad una frequenza di 250Hz, un insieme di filtri digitali per ridurre il rumore sull'angolo di inclinazione

e le istruzioni necessarie per generare i corretti segnali elettrici che comandano il motore. Infine, con l'aiuto di diverse simulazioni su MATLAB, è stato possibile analizzare singolarmente gli effetti che ogni parametro fisico del sistema ha sulle prestazioni finali, permettendo di delineare quali caratteristiche sono essenziali per un funzionamento ottimale.

Sebbene poco soddisfacente per il candidato, il lavoro svolto si dimostrerà eventualmente utile per la progettazione di altri sistemi di stabilizzazione basati su un volano. Grazie a questa tesi è stato possibile concludere che un utilizzo su larga scala di questo sistema di controllo è improbabile. Si tratta infatti di una tecnologia adatta solo a equilibrare oggetti di massa e dimensioni piuttosto ridotte come piccoli droni o bicicli, che comunque vedono già disponibili altri tipi di sistemi di controllo ormai affermati da tempo.

Appendice A

Codice Arduino

```
#include <Wire.h>
int pa=5;                          //PIN + Hbridge (motore)
int pb=6;                          //PIN - Hbridge (motore)

//Variabili controllore PID
float set_a=0.0;                   //Angolo target
float PID_error, PID_value;        //INPUT e OUTPUT
float previous_error;              //Errore precedente
float derivate;                    //Derivata e integrale
float integral=0.0;

//Parametri PID
const float elapsedTime = 0.004;  //Tempo tra due campionamenti
float kp=121;
float ki=0.4;
float kd=5;

//Variabili globali per l'angolo
int gyro_z;                        //Valore grezzo del giroscopio
long acc_x, acc_y;                 //Valore grezzo dell'accelerometro
float val_x=0;                     //Valori accelerometro dopo filtraggio
float val_y=0;
long gyro_z_cal=0;                 //Offset giroscopio
long loop_timer;                   //Per gestire il tempo di loop
float angle_acc, gyro_vel, angle_out;
float alpha=0.001;                 //Parametri per i filtri digitali
float beta=0.2;

void setup() {

  Wire.begin();
  pinMode(pa, OUTPUT);
  pinMode(pb, OUTPUT);

  setup_mpu_6050_registers();      //Setup MPU-6050(+/-500 deg*s | +/-2g)
```

```
36
37   //Calcolo offeset medio
38   for (int i = 0; i < 1500 ; i ++){  //Calibrazione sensori
39     read_mpu_6050_data();            //Lettura dati grezzi da MPU-6050
40     gyro_z_cal += gyro_z;
41     val_x += acc_x;
42     val_y += acc_y;
43     delay(3);
44   }
45   gyro_z_cal /= 1500;                //Offset giroscopio
46   val_x /= 1500.0;                   //Valore medio iniziale acc
47   val_y /= 1500.0;
48   angle_out=-atan((float)val_y/val_x)* 57.29578;
49   previous_error=set_a-angle_out;    //Necessario per il controllore
50   loop_timer = micros();             //Reset del timer
51 }
52
53 void loop(){
54
55   read_mpu_6050_data();                 //Lettura dati grezzi
56
57   //Angolo percorso dall'ultimo campionamento (giroscopio)
58   gyro_z -= gyro_z_cal;                 //Sottrazione dell'offset
59   gyro_vel = gyro_z * (1/(250*65.5));   //Angolo percorso
60
61   //Angolo calcolato con l'accelerometro
62   val_y=(1-beta)*val_y+beta*acc_y;      //Filtro sui dati dell'acc
63   val_x=(1-beta)*val_x+beta*acc_x;
64   angle_acc = -atan((float)val_y/val_x)* 57.29578;   //Calcolo angolo acc
65
66   //Angolo di output (filtro complementare)
67   angle_out= (1-alpha)*(angle_out+gyro_vel)+alpha*angle_acc;
68
69   //Calcolo input motore con controllore PID
70   PID_error = set_a - angle_out;         //Errore
71   integral+= PID_error*elapsedTime;      //Calcolo dell'integrale
72   if(integral>100) integral=100;         //Saturazione integrale
73   if(integral<-100) integral=-100;
74   derivate=(PID_error - previous_error)/elapsedTime;    //Derivata
75   PID_value= kp*PID_error + ki*integral + kd*derivate;  //Output
76   previous_error = PID_error;            //Aggiornamento errore precedente
77
78   //Se l'angolo e' troppo elevato si inibisce il controllo
79   if (angle_out>8 || angle_out<-8) {
80     PID_value=0;
81     integral=0;
82   }
83
84   controlDC(PID_value);            //Output controllore comanda il motore
85
86   //Controllo del tempo di ciclo (freq di 250Hz)
87   while(micros() - loop_timer < 4000);   //Aspetta che passino 4000us
88   loop_timer = micros();           //Reset loop timer
89 }
```

```
90
91 void controlDC( float vel){        //Controllo del motore
92
93   if (vel<-255) vel=-255;          //Valori eccedenti vengono saturati
94   if (vel>255) vel=255;
95   if (vel<0){
96     analogWrite(pb,0);
97     analogWrite(pa,(int) abs(vel));
98     }
99   else {
100    analogWrite(pa,0);
101    analogWrite(pb,(int) vel);
102    }
103 }
104
105 void read_mpu_6050_data(){              //Lettura dei dati grezzi
106
107   int temp, gyro_x, gyro_y, acc_z;      //Variabili non utilizzate esternamente
108
109   Wire.beginTransmission(0x68);               //Inizio comunicazione
110   Wire.write(0x3B);                           //Richiesta dei registri
111   Wire.endTransmission();                     //Fine comunicazione
112
113   Wire.requestFrom(0x68,14);                  //Lettura di 14 byte
114   while(Wire.available() < 14);
115   acc_x = Wire.read()<<8|Wire.read();         //Salvataggio di acc_x
116   acc_y = Wire.read()<<8|Wire.read();         //Salvataggio di acc_y
117   acc_z = Wire.read()<<8|Wire.read();         //Salvataggio di acc_z
118   temp  = Wire.read()<<8|Wire.read();         //Salvataggio di temp
119   gyro_x = Wire.read()<<8|Wire.read();        //Salvataggio di gyro_x
120   gyro_y = Wire.read()<<8|Wire.read();        //Salvataggio di gyro_y
121   gyro_z = Wire.read()<<8|Wire.read();        //Salvataggio di gyro_z
122 }
123
124 void setup_mpu_6050_registers(){            //Setup MPU-6050
125
126   //Attivazione MPU-6050
127   Wire.beginTransmission(0x68);
128   Wire.write(0x6B);
129   Wire.write(0x00);
130   Wire.endTransmission();
131
132   //Configurazione giroscopio
133   Wire.beginTransmission(0x68);
134   Wire.write(0x1B);
135   Wire.write(0b00001000);                     //500deg*s fondo scala
136   Wire.endTransmission();
137
138   //Configurazione accelerometro
139   Wire.beginTransmission(0x68);
140   Wire.write(0x1C);
141   Wire.write(0b00000000);                     //2g fondo scala
142   Wire.endTransmission();
143 }
```

Appendice B

Script MATLAB

```
% Condizioni iniziali
th=-3;                  % Angolo iniziale pendolo [deg]
vel=0;                  % Velocità iniziale [deg/s]
vmot=0;                 % Velocità iniziale del motore [deg/s]

% Parametri fisici pendolo
mt=0.07;                % Massa totale pendolo [kg]
dc=0.06;                % Distanza perno-centro di massa pendolo [m]
It=0.00001;             % Momento di inerzia totale del pendolo rispetto al perno
                        % [kg*m^2] (ottenuto sperimentalmente)
Ir=0.000006;            % Momento di inerzia del rotore [kg*m^2]
                        %(ottenuto sperimentalmente)

kd=0.00008;             % Coef. di attrito dinamico pendolo-perno
                        % [Nms/deg] (ottenuto sperimentalmente)

% Parametri fisici del motore
kt=0.002;               % Costante di coppia del motore [Nm/A]
V=12.5;                 % Voltaggio di alimentazione [V]
R=5;                    % Resistenza interna del motore [Ohm]

% Variabili per controllore PID
targ=0;                 % Target
erri=0;                 % Errore integrale
erro=targ-th;           % Errore precedente
t=1/250;                % Tempo tra due campioni [s] (250Hz di frequenza)

P=121;
D=5;
I=0.4;

% Variabili per la simulazione e la visualizzazione
x=1000;
par1=1:x;
par1(1)=th;

% Ciclo di simulazione
for i=2:x
```

```
39
40      % Calcoli del controllore PID (identici a quelli su Arduino)
41      err=targ-th;                        % Errore
42      erri=erri+err*t;                    % Integrale
43      if abs(erri)>100                    % Saturazione integrale
44          erri=(erri/abs(erri))*100;
45      end
46      errd=(err-erro)/t;                  % Derivata
47      erro=err;                           % Aggiornamento errore precedente
48      pwm=round(P*err+D*errd+I*erri);     % Output controllore come valore di PWM
49
50      if abs(pwm)>255                     % Saturazione PWM non consentiti
51          pwm=(pwm/abs(pwm))*255;
52      end
53
54      % Corrente fornita al motore
55      corr=-(pwm/255)*(V/R);            % Corrente massima per pwm=255
56      taum=kt*corr-((kt^2)/R)*vmot;     % Coppia fornita dal motore
57
58      % Dinamica dell'intero pendolo
59      acc=(mt*dc*9.81*sind(th)-taum-kd*vel)/It;
60      th=th+0.5*(t^2)*acc+vel*t;
61      vel=vel+acc*t;
62
63      % Dinamica del rotore
64      vmot=vmot+(taum/Ir)*t;
65
66      % Salvataggio del nuovo stato del sistema
67      par1(i)=th;
68  end
69
70  % Visualizzazione
71  w=t:t:x*t;
72  plot(w,par1);
```

Bibliografia

[1] Kiam Heong Ang, G. Chong and Yun Li, *PID control system analysis, design, and technology*, IEEE Transactions on Control Systems Technology, vol. 13, no. 4, pp. 559-576, July 2005

[2] Gianni Vannini, W. Edward Gettys, *Gettys fisica 1*, Edizione 5, McGraw-Hill Education (Italy), Milano, 2015

[3] Kevin M. Lynch, Nicholas Marchuk, Matthew L. Elwin, *Embedded Computing and Mechatronics with the PIC32 Microcontroller*, Newnes, Oxford, 2015

[4] InvenSense Inc., *MPU-6000 and MPU-6050 Product Specification*, Revision 3.4, 2013

[5] J. Wu, Z. Zhou, J. Chen, H. Fourati and R. Li, *Fast Complementary Filter for Attitude Estimation Using Low-Cost MARG Sensors*, IEEE Sensors Journal, vol. 16, no. 18, pp. 6997-7007, Sept.15, 2016