**Abstract**

**Descizione del report**

Si vuole stabilizzare un modello hardware di un pendolo inverso tramite utilizzo di controllori PID.

**Setup**

Il pendolo inverso consiste in due unità hardware distinte:

* Controllore: si occupa dell’esecuzione degli algoritmi di controllo (PID) e di pilotaggio del motore passo passo;
* Lettore dell’angolo: montato sul carrello del pendolo, acquisisce dati sullo stato del pendolo e li trasmette al controllore.

Il controllore consiste in una scheda ESP8266 (modello Wemos d1), alimentato direttamente dal cavo USB C dal PC. A bordo di questo microcontrollore viene eseguito il codice che ha compito di ricevere i dati sullo stato del pendolo, calcolare l’accelerazione (input) da imprimere al carrello per la stabilizzazione, e comandare il motore passo con l’accelerazione indicata. Il comando generato dal ESP8266 consiste in un treno di impulsi che viene mandato in ingresso al piedino “STEP” del driver del motore. Il driver è un A4988, ed è capace di controllare un motore passo-passo bipolari a 4 fili con vari livelli di microstepping. L’alimentazione del motore è fornita da una batteria LiPo da 7.4v, connessa direttamente ai rispettivi piedini VMOT e GND del driver A4988. L’intero modulo è inscatolato in un case stampato in 3D, in cui alloggia la batteria e la basetta millefori su cui è stato realizzato il circuito. Dal driver escono i 4 fili delle due fasi del motore passo-passo, in cui il driver eroga la corrente necessaria per effettuare lo stepping. Il motore è fissato su un banco piano usando del nastro adesivo.

SCHEMA ELETTRICO 1

Sul rotore del motore è fissato con una vite il blocco “carrello”, stampato in 3D. In esso alloggiano due cuscinetti radiali in cui è inserita una barra libera di ruotare. Su un’estremità della barra è bullonato il braccio del pendolo; sull’altra estremità è incollato un magnete con magnetizzazione diametrale, che ruota in prossimità dell’encoder rotatorio AS5600, che rileva l’orientamento del magnete posto in sua prossimità. Il microcontrollore utilizzato è un'altra scheda ESP8266 (Wemos d1), collocato su una piccola breadboard. Sulla breadboard è anche inserito un potenziometro, il cui segnale analogico viene acquisito dall’unico piedino ADC presente sulla scheda. Il codice a bordo del ESP8266 si occupa dell’acquisizione (tramite i2c) dell’angolo rilevato dall’encoder, lieve post-processamento dei dati (per mitigare il rumore e aggiungere un offset, regolabile tramite potenziometro), e trasmissione dei dati alla scheda ricevente utilizzando il protocollo proprietario di espressif ESP-NOW.

SCHEMA ELETTRICO 2

**Strumenti software impiegati:**

Arduino IDE: per scrivere e caricare il codice sui microcontrollori, e per interagire tramite comandi su seriale. -> link <-

**Flusso di esecuzione del software embedded**

Un’apposita libreria permette al codice in esecuzione sul MCU di lettura di acquisire il valore dell'angolo dal sensore magnetico. Nel setup(), il sensore viene letto una prima volta per fissare l’angolo verticale (a testa in giù) del pendolo. Terminato il setup, il sensore viene letto a ogni iterazione del loop principale. Un filtro passa basso è applicato alla lettura dell'angolo per smussare il rumore e aumentare la risoluzione. All’angolo viene aggiunto un offset dettato dalla lettura del potenziometro presente sulla breadboard, in modo che eventuali imprecisioni nella fase di setup possano venire corrette manualmente durante l’operazione del pendolo. L' angolo processato è poi sottoposto a un algoritmo di differenziazione numerica, che ne calcola la derivata rispetto al tempo. I due valori vengono impacchettati in un oggetto e spediti tramite ESP-NOW al MCU di controllo, con una frequenza fissa prestabilita (generalmente poco superiore alla frequenza del loop di controllo).

Figures

Sul esp8266 di controllo, la ricezione dei pacchetti ESP-NOW e il relativo aggiornamento delle variabili di stato del pendolo avviene in maniera asincrona: un interrupt è sollevato all'arrivo dei nuovi dati che blocca temporaneamente l'esecuzione del loop principale; i dati ricevuti vengono copiati in uno struct globale, che viene poi letto dal loop di controllo.

Callback code

All’interno del loop() è presente la porzione di codice che esegue l’algoritmo di controllo. Questa viene eseguita a una frequenza controllata fissa di 100Hz, e si occupa di calcolare, a partire dallo stato del pendolo attuale, l'ingresso (accelerazione) da applicare al pendolo. L’ingresso viene immagazzinato in una variabile globale u, che dunque rimane accessibile fuori da questa porzione di codice, anche per le successive iterazioni del loop(). Mentre l’angolo del pendolo è ricevuto come dato remoto, la posizione del carrello (posizione del rotore del motore) viene facilmente calcolata a bordo conoscendo il numero di passi compiuti dal motore (in ipotesi di non perdita di passi).

Codice del blocco a 100Hz semplificato

* **Algoritmo di controllo (PID):**

CODICE

Alla fine del loop() è presente la porzione di codice che si occupa del pilotaggio del motore. Questa viene eseguita ad ogni iterazione, per garantire la maggiore fluidità di rotazione possibile. La classe PendulumCart astrae tutta l'interazione con il driver in due principali metodi: driveAccel() e driveSpeed(). I parametri di accelerazione e velocità sono espressi in cm/s, e si riferiscono alla velocità/accelerazione tangenziale della base del pendolo (così come se fosse ancorato ad un carrello lineare). Il metodo effettivamente utilizzato dagli algoritmi di stabilizzazione è driveAccel(); driveSpeed() è principalmente utilizzato per motivi di debugging.

La generazione degli impulsi di step è realizzata utilizzando la libreria accellStepper. L’omonima classe fornita dalla libreria mette a disposizione il metodo runSpeed(), che consente di pilotare un motore passo passo alla velocità desiderata. Il principio di funzionamento di accellStepper è semplice: il metodo runSpeed() deve essere chiamato il più rapidamente possibile all’interno del loop del microcontrollore; se e solo se è ora di effettuare un passo, in base alla velocità selezionata, il metodo genera un brevissimo impulso sul piedino connesso al canale STEP del driver, e regola il piedino connesso a DIR in base alla direzione di rotazione del motore (orario o antiorario).

I piedini dell’esp8266 sono collegati direttamente ai canali MS del driver, consentendo il selezionamento del livello di microstepping tramite codice. La classe astratta MicrostepDriver astrae la logica di accensione/spegnimento dei rispettivi piedini MS in un solo metodo: setMicrostep(). Una sottoclasse chiamata A4988MicrostepSelector implementa la specifica logica richiesta dall’omonimo driver per selezionare i vari livelli di microstepping; il suo metodo di setMicrostep() accetta in ingresso 4 livelli di microstepping:

* 0 -> no microstep
* 1 -> ½ microstep
* 2 -> ¼ microstep
* 3 -> 1/8 microstep
* 4 -> 1/16 microstep

Un oggetto di qualunque classe derivata da MicrostepDriver è collegabile a un’istanza di PendulumCart, permettendo al carrello di scegliere in maniera adattiva il livello di microstepping, garantendo la migliore performance per la velocità di rotazione richiesta.

* **Logica di selezionamento dinamico del microstepping**

Il livello di microstepping viene scelto in maniera automatica in funzione della velocità angolare desiderata, affinché l’operazione del motore sia fluida a basse velocità ma reattiva qualora venissero richiesti movimenti rapidi. L’algoritmo mira a selezionare sempre il più alto livello di microstepping che mantenga la frequenza dei passi sotto una certa soglia massima (la massima frequenza a cui il treno di impulsi può essere generato dal MCU). L’equazione utilizzata è :

EQUAZIONE

**Funzionamento del driver A4898**

<https://www.alldatasheet.com/datasheet-pdf/download/455036/ALLEGRO/A4988.html>

**Interazione tramite seriale**

Nel loop() del controllore, la funzione processSerialCommands() viene chiamata ad ogni iterazione e si occupa di ricevere ed eseguire eventuali comandi testuali inviati alla scheda tramite seriale. Ecco alcuni dei comandi più utili per interfacciarsi con il pendolo:

**Tecniche di controllo in dettaglio**

Per stabilizzare il pendolo, è stato scelto di utilizzare due controllori PID, che agiscono parallelamente sull’ingresso u da imprimere al sistema.

DIAGRAMMA A BLOCCHI

Il primo PID, a cui ci si riferirà con il nome PID primario, è il PID di stabilizzazione vero e proprio. Esso reagisce direttamente sull’angolo del pendolo e tenta di portarlo a zero. I suoi gain sono calibrati in maniera molto aggressiva in modo da garantire sempre una risposta rapida e decisa. Tuttavia, avere un unico PID di controllo comporta alcuni problemi. In primis, il controllore è molto sensibile a errori di offset costanti della lettura dell’angolo: se la verticale virtuale è spostata rispetto a quella reale, l’unico modo in cui il pendolo può rimanere in “equilibrio” (raggiungere un angolo pari a zero) sarebbe imprimendo sempre un’accelerazione costante. Questo, oltre a non essere il goal da raggiungere, è chiaramente irrealizzabile in un modello hardware, a causa dei limiti di velocità del motore. In secondo luogo, anche considerando il caso ideale in cui la misura dell’angolo non sia affetta da errori di offset costanti, un unico PID non garantisce che, effettuata la stabilizzazione, il carrello abbia velocità nulla. Questo perché l’ingresso al sistema u corrisponde esattamente l’accelerazione del carrello, e poiché la velocità è l’integrale dell’accelerazione nel tempo, non è garantito il fatto che

E’ dunque necessario aggiungere un secondo controllore che agisce sulla posizione del carrello (la prende in input), a cui ci si riferirà come PID secondario. In particolare, questo PID tenta di minimizzare la differenza fra la posizione attuale del carrello e una posizione di target, consentendo il piazzamento del carrello in una posizione arbitraria. Poiché i due controllori PID agiscono in parallelo su due varabili di stato diverse, il tuning dei parametri deve essere effettuato con cautela, onde evitare che i due controllori interferiscano troppo fra di loro. Per evitare che la calibrazione dei PID risulti troppo difficoltosa, si è cercato di separare il più possibile i due controllori in termini di “velocità” (rapidità di convergenza); in questo modo, i controllori lavorano su scale temporali diverse e interferiscono fra di loro in maniera molto più contenuta. Il PID secondario è infatti calibrato in modo da avere una risposta apprezzabilmente più lenta e meno aggressiva rispetto al PID primario; così facendo, i due PID diventano calibrabili in maniera quasi del tutto indipendente fra loro, velocizzando notevolmente il processo.

E’ da notare che, con l’aggiunta del PID secondario, il problema del possibile disallineamento fra la verticale virtuale e reale viene risolto; infatti, qualora ci fosse un disallineamento, l’azione delle componenti PI del controllore secondario porterebbe comunque il carrello a frenare e stabilizzarsi su una posizione fissa (Nota: l’errore di posizione può essere facilmente calcolato con la formula P1 \* thetaerr / P2, come si evince dal diagramma \ref). Incorporando anche la componente integrale nel controllore, l’errore steady state sulla posizione del carrello che si genera viene gradualmente compensato fino al raggiungimento dello stato desiderato.

La stabilizzazione della posizione avviene in maniera molto più lenta rispetto alla stabilizzazione dell’angolo,

**RISULTATI SPERIMENTALI**

**Vari settaggi dei gain dei PID**

GRAFICI