POLITECNICO DI BARI

DEPARTMENT OF ELECTRICAL AND INFORMATION ENGINEERING

Master's Degree in Automation Engineering

DOCUMENTAZIONE BALL AND BEAM

Professor:
Dr. Eng. David Naso

Candidates:
D'Arcangelo Luca
Zinfollino Vincenzo

ACADEMIC YEAR 2023–2024

Contents

1 Introduzione **4**

2 Componenti **5**
2.1 Componenti Hardware 5
2.1.1 Componenti Strutturali 5
2.1.2 Motore 6
2.1.3 Encoder 7
2.1.4 Driver 7
2.1.5 Microcontroller 9
2.1.6 Sensore di Time of Flight 9
2.2 Componenti Softwere 11
2.2.1 Main 11
2.2.2 Implementation 12
2.2.3 PID 12
2.2.4 Ring buffer 14

3 Configurazioni **16**
3.0.1 Configurazione delle periferiche e del sistema 16
3.0.2 Configurazione STM32CubeMonitor 18

4 Schemi **25**
4.1 Schema Generale 25
4.2 Piedinatura 27

Abstract

Il sistema Ball and Beam è un sistema piuttosto semplice composto da pochi elementi il quale rappresernta un classico esempio didattico di controllo automatico. È composto da una trave orizzontale su cui è posizionata una sfera. L'obiettivo è mantenere la sfera in equilibrio su una posizione desiderata lungo la trave, regolando l'angolo di inclinazione della trave tramite un motore. Questo sistema è intrinsecamente instabile e richiede un controllo attivo per mantenere l'equilibrio della sfera. È utilizzato per illustrare concetti chiave come stabilità e risposta dinamica, ed è un banco di prova ideale per vari algoritmi di controllo.

1 Introduzione

Il sistema "ball and beam" è un classico problema di controllo utilizzato frequentemente come esempio didattico nei corsi di ingegneria meccanica, controllo automatico e robotica. Questo sistema è composto da una trave orizzontale (beam) sulla quale è posizionata una sfera (ball). L'obiettivo principale è mantenere la sfera in equilibrio su una posizione desiderata lungo la trave, nonostante le perturbazioni o le variazioni nella posizione iniziale.

La trave può ruotare attorno a un punto fisso, generalmente grazie a un motore che controlla l'angolo di inclinazione. Quando la trave è inclinata, la sfera rotola nella direzione della pendenza a causa della gravità. Il compito del sistema di controllo è regolare l'angolo della trave in modo tale da far sì che la sfera si sposti alla posizione desiderata e vi rimanga stabile.

Questo problema è particolarmente interessante dal punto di vista del controllo automatico perché è un sistema intrinsecamente instabile: senza un'azione di controllo, la sfera rotolerà via dalla trave. Inoltre, il sistema "ball and beam" è un ottimo banco di prova per vari tipi di algoritmi di controllo, come i regolatori proporzionali-integrali-derivati (PID), i controllori con retroazione di stato, e le tecniche di controllo robusto e adattivo.

Dal punto di vista educativo, il sistema "ball and beam" offre una chiara visualizzazione dei concetti di stabilità, risposta dinamica e comportamento in regime stazionario, rendendolo un esempio pratico ed efficace per l'apprendimento teorico e pratico dei sistemi di controllo.

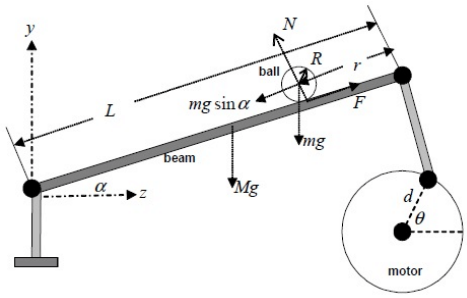


Figure 1: Schema Rappresentativo del sistema

2 Componenti

Poiché il sistema è composto da diversi elementi è necessario individuarli e classificarli al meglio per garantire un migliore comprensione delle loro funzionalità individuali e delle loro contributo al sistema, una volta interconnessi tra di loro. In questo capitolo saranno discusse tutte le parti che compongono il sistema dividendo la parte Hardware da quelle Software.

2.1 Componenti Hardware

In questa sezione saranno introdotte tutte le parti hardware da noi scelte cercando il più possibile di motivare le nostre decisioni progettuali.

2.1.1 Componenti Strutturali

Come accennato in precedenza il sistema è composto da una trave vincolata ai due estremi, dove un' estremità è lasciata libera di ruotare ,grazie all'utilizzo di un cuscinetto, invece la seconda è collegata ad un motore il quale tramite l'uso di vari link di connessione (entrambi dotati di cuscinetti) garantisce la mobilità della trave sulla quale scorre la pallina.

Nel sistema è possibile individuare tre componenti strutturali fondamentali:

- **Trave :** questa è la componente più grande del sistema, in un primo momento si era pensato di utilizzare delle aste di legno, sulle quali far poi scorre la pallina, questo però si è rivelato una scelta errata poiché il legno per la sua natura intrinseca di materiale vegetale, tendeva ad avere deformazioni ed a incurvarsi. Per far fronte a questa inconvenienza abbiamo deciso di utilizzare due estrusioni di alluminio larghe 2 mm montate in una configurazione a "V" tenuto insieme dalle due estremità stampate in 3D e fissate per interferenza.Questo garantisce una buona resistenza buona leggerezza e soprattutto gli estrusi di alluminio , essendo montati perpendicolarmente uno all'altro, creano un condotto perfetto per lo scorrere della pallina.

- **Terminale di fissaggio :** Questo terminale è realizzato lavorando dal pieno una barra di alluminio 6060. L'obiettivo di questa lavorazione era garantire un perfetta perpendicolarità rispetto al piano di montaggio.Nella sua sommità è stato inserito un cuscinetto questo per garantire il minimo attrito durante l'esercizio.A questo terminale è fissato il sensore TOF.

- **Terminale del motore :** Anche questo terminale è realizzato a partire da una barra di alluminio 6060. Oltre alle motivazioni precedentemente citate, i motori stepper tendono a riscaldarsi durante l'esercizio, ed essendo a contatto diretto con l'alluminio questo garantisce una gestione termica più efficiente. Se questo componente fosse stato stampato in 3D saremmo incappati in possibili deformazioni. Collegati al motore troviamo le due connessioni che trasformano il movimento rotatorio del motore in un displacement lineare della trave.

Le parti del sistema sono state realizzate considerando le possibili iterazioni future del dispositivo, dunque ogni componente è stato ideato in modo tale da essere il più essenziale possibile e di semplice sostituzione.
Le parti stampate in 3d, sono state realizzate in PETG.

2.1.2 Motore

Figure 2: Immagine del motore utilizzato

I motori passo-passo funzionano dividendo una rotazione completa in un gran numero di passi uguali. La posizione del motore può essere comandata per muoversi e mantenersi in uno di questi passi senza alcun sensore di retroazione (un sistema ad anello aperto), a condizione che il motore sia accuratamente dimensionato per l'applicazione. Il rotore del motore passo-passo è un magnete permanente, circondato da uno statore con avvolgimenti. Alimentando questi avvolgimenti in una sequenza specifica, si crea un campo magnetico rotante che trascina il rotore in allineamento con il campo magnetico dello statore. Questa sequenza di eccitazione degli avvolgimenti consente al motore di muoversi con incrementi precisi, rendendo i motori passo-passo ideali per le applicazioni che richiedono un posizionamento preciso. La scelta del motore NEMA 17, un'opzione popolare per molte piccole applicazioni di robotica e automazione, è stata motivata da diverse ragioni, in primo luogo perché offre un buon equilibrio tra prestazioni e costi. La prima ragione principale è che questo modello di motore è dotato di un encoder incrementale magnetico collegato al rotore del motore passo-passo, rendendo l'intero sistema motore più compatto. Questo aspetto era particolarmente vantaggioso in questo progetto in cui l'efficienza dello spazio era fondamentale. Uno dei motivi per cui è stato scelto questo motore è la sua forte coppia nonostante le dimensioni compatte e il controllo di precisione consentito dall'elevato numero di passi per giro. Ciò consente al motore di seguire una traiettoria precisa data in input, anche con un carico attaccato.

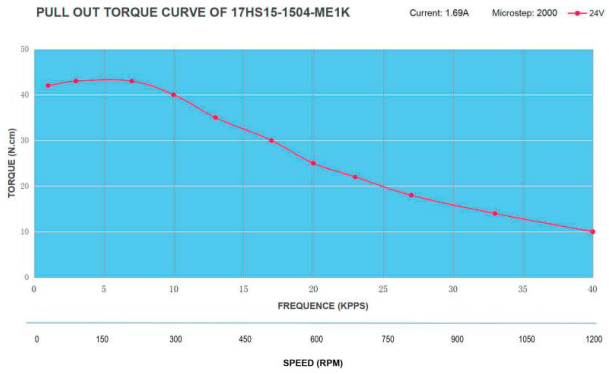


Figure 3: Curva della Coppia del Nema 17

2.1.3 Encoder

L'encoder magnetico utilizza un magnete permanente su un corpo rotante, come l'albero di un motore, per rilevare le variazioni del campo magnetico. È dotato di un magnete e di un sensore su una scheda PCB. Quando l'albero ruota, il sensore rileva le variazioni del campo magnetico, consentendo all'encoder di misurare la posizione e la velocità dell'albero. Il campo magnetico rotante proviene dal magnete sull'albero. Un elemento di Hall nel sensore rileva le variazioni di campo e le trasforma in segnali elettrici. Poiché rileva solo in una direzione, sono necessari due elementi di Hall. In effetti, l'encoder ha due uscite chiamate A e B in cui viene generata una forma d'onda quadratica. Se il fronte di salita del canale A precede il fronte di salita del canale B, l'encoder sta ruotando in senso orario, altrimenti in senso antiorario. In realtà, l'encoder utilizzato ha tre canali, di cui il terzo è per l'indice di zero, ma non è stato utilizzato perché non era necessario sapere quando il motore aveva superato il punto zero. Per questo tipo di lettura, l'STM utilizzato nel progetto dispone di un modulo hardware per la lettura degli encoder. Può essere impostato direttamente nella configurazione del timer, scegliendo la modalità Encoder. Questo modulo configura un registro con un contatore che aumenta o diminuisce in base alla direzione del fronte di salita. Possiamo accedere a questo registro per ottenere la lettura della posizione e per stimare la velocità e l'accelerazione.

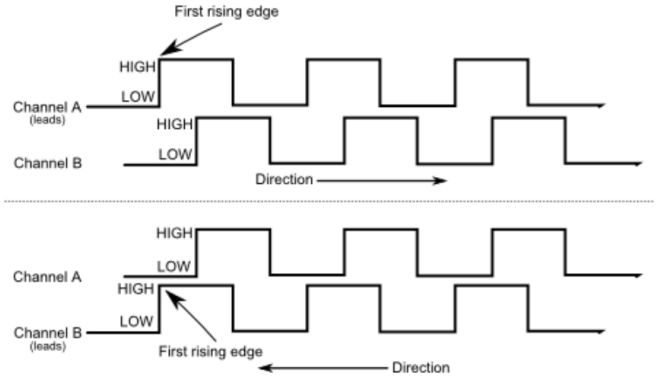


Figure 4: Uscita del segnale dell'encoder

2.1.4 Driver

Per interfacciarsi correttamente con il motore abbbiamo bisogno di un driver con le seguenti caratteristiche:

- Semplice interfaccia di controllo dello step e della direzione;
- Sei diverse risoluzioni di step: full-step, half-step, 1/4-step, 1/8-step, 1/16-step, and 1/32-step;
- Possibilità di interfacciarsi direttamente con sistemi a 3,3 V e 5 V;
- Il controllo della corrente regolabile consente di impostare la corrente massima in uscita con un potenziometro, che permette di utilizzare tensioni superiori alla tensione nominale del motore stepper per ottenere velocità di passo più elevate;

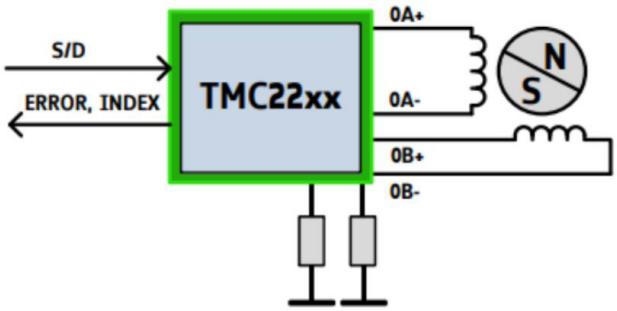


Figure 5: Driver TMC2209

Il driver scelto per controllare il motore è il TMC2209 grazie alle sue elevate prestazioni, alle protezioni integrate, alla facilità di configurazione e all'ampia compatibilità. È particolarmente adatto per le applicazioni che richiedono un controllo preciso e affidabile dei motori passo-passo. Inoltre, questo driver consente il microstepping, permettendo risoluzioni più elevate del numero di passi del motore grazie all'abilitazione di posizioni di passo intermedie. Queste posizioni intermedie si ottengono eccitando le bobine con livelli di corrente intermedi. Per selezionare una delle sei modalità di passo possibili, i tre pin di risoluzione dedicati devono essere collegati correttamente. Per ulteriori informazioni, consultare il datasheet del driver. Nel progetto è stata utilizzata la risoluzione micro-stepping a 1/16 di step, che consente di controllare la posizione del motore con maggiore precisione perché ci sono più passi disponibili e permette di avere un range sufficiente in termini di velocità massima raggiungibile dal motore. Infatti, il Nema 17 ha normalmente 200 step per giro, ma con i microstep si avranno 3200 microstep per giro.

$$Risolution = \frac{2\pi[rad]}{3200[microstep]} = 0,0019[rad/microstep] \tag{1}$$

Dal datasheet, è nota la massima frequenza di passo (alla massima risoluzione di risoluzione in microstep), quindi nel progetto è stata scelta una frequenza inferiore.

$$f_{step} = \frac{f_{clk}}{2} = \frac{12MHz}{2} = 6MHz \tag{2}$$

Per controllare il motore sono necessari solo due pin: uno per la direzione e uno per il passo. La direzione è semplice: il livello logico 0 corrisponde alla rotazione in senso orario, mentre il livello logico 1 corrisponde alla rotazione in senso antiorario (in base ai collegamenti dei fili del motore). Per attivare un passo sul pin del passo, è necessaria una transizione da 0V a 5V. Pertanto, se si desidera una rotazione continua, è necessario generare un segnale a onda quadra. Tuttavia, con questo driver non è possibile effettuare un controllo diretto della posizione ma solo un controllo della velocità, perché prende in ingresso solo la frequenza di quanti passi per secondo deve fare e la direzione. Quindi, la frequenza di questo segnale è proporzionale alla velocità desiderata. Per imporre la velocità, è stato necessario utilizzare un timer contatore per la generazione del PFM.

Una volta nota la velocità desiderata, si dovranno eseguire alcune operazioni per impostare correttamente i registri:

- **Auto-Reload Register (ARR):** responsabile del periodo della forma d'onda;
- **CCR :**responsabile del duty-cycle. Il duty-cycle è sempre stato impostato al 50% (metà del valore di ARR). Viene impostato a 0% solo per arrestare il motore.

2.1.5 Microcontroller

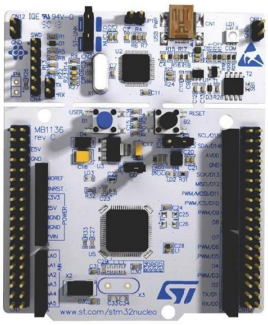


Figure 6: STM32F411RE Microcontroller

È stata utilizzata la scheda STM32F411RET6, che offre tutto il necessario per il corretto svolgimento del progetto. Per ulteriori dettagli è possibile consultare la scheda tecnica della scheda. Per scrivere il codice e flasharlo è stato utilizzato l'IDE del produttore, STM32CubeIDE. L'IDE dispone dello strumento STM32CubeMX, che consente di regolare la scheda a un livello superiore.

2.1.6 Sensore di Time of Flight

Figure 7: Sensore di distanza VL53L0X

Il telemetro laser VL53L0X è un sensore di distanza ad alta precisione basato sul Time of Flight (ToF). A differenza della tecnologia tradizionale, il VL53L0X fornisce una misurazione accurata della distanza, indipendentemente dal riflesso del bersaglio. Può misurare distanze assolute fino a 2 m.

Scelta del sensore e problematiche :

Durante il progetto, sono stati testati diversi sensori VL53L0X di fascia economica. Tuttavia, questi sensori hanno mostrato problemi significativi nelle letture, risultando rumorosi e poco precisi.

Secondo il datasheet di ST per il VL53L0X, è consigliabile utilizzare un sensore con due circuiti di alimentazione separati: uno per il fascio di luce e un altro per la gestione della lettura. Questa caratteristica non è presente nei sensori di fascia bassa. Per il progetto, è stato scelto il sensore di

DFRobot che possiede questa configurazione.I vantaggi di questo sensore includono:

- Fascio di luce migliorato
- Maggiore resistenza alla luce ambientale
- Letture più precise ad alte velocità

Le problematiche riscontrate nel sensore sono state diverse, nonostante l'upgrade di sensore. La velocità di lettura, impostata al massimo, non è risultata sufficiente per un controllo ottimale. Questo ha impedito l'implementazione efficace di un filtro per il rumore, poiché avrebbe introdotto un ritardo nelle letture, compromettendo ulteriormente le prestazioni. Inoltre, sono stati riscontrati picchi (spike) nelle letture quando la pallina urtava il supporto del sensore. Infine, il sensore è risultato ancora sensibile alla luce ambientale, rendendo necessaria l'operazione in condizioni di scarsa illuminazione per ottenere migliori performance, come suggerito dal datasheet.

Librerie :
ST fornisce librerie software che semplificano l'uso del sensore e permettono di impostare diverse modalità di funzionamento. Nel progetto è stata scelta la modalità High Speed, regolando i parametri seguendo le istruzioni del datasheet. Per la temporizzazione, è stato adottato un approccio basato su letture in modalità continua (Continous Timed Ranging), effettuando una misurazione ogni 25ms. Il dato viene poi richiamato ogni volta che il main lo richiede.
Le librerie sono presenti all'interno della cartella $Drivers \rightarrow Librerie \rightarrow VL53L0X$

Anche se il sensore è molto preformate risulta poco adatto a questo scenario applicativo.

2.2 Componenti Softwere

Come nella parte Hardware anche la parte software è stata scomposta in più componenti per garantire una buona modularità e soprattutto garantire un iterazione più dinamica.

Il codice del progetto può essere scomposto nelle seguenti parti:

- *Main :* Rappresenta effettivamente l'entry point del progetto qui si può trovare il loop principale del micro controllore nel quale è effettuata la richiesta di lettura del sensore TOF.
- *Ring Buffer :* costituisce una coda FIFO, l'uscita viene mantenuto l'ultimo valore in ingresso.
- *PID controller :* Implementazione del controllore derivato utilizzando le formule di Tustin.
- Implementation : In questo file vengono inserite tutte le implementazioni, per evitare che in caso di generazione automatica del codice vengano perse.In questo file è anche presente la dichiarazione del typedef che aggrega tutte e variabili salienti del sistema.
- *Lidar :* file wrapper utilizzato per implementare correttamente la lettura del sensore sfruttando le primitive offerte dalle librerie del produttore del sensore.

Di seguito saranno trattati singolarmente i componenti software e descritti nel loro dettaglio.

2.2.1 Main

Come detto in precedenza il componente main rappresenta, entry point del progetto. Una volta programmato il microcontrollore il primo metodo eseguito sarà proprio il main. All'interno del metodo mai troviamo il loop infinito nel quale andremo ad interconnettere tutte le funzioni necessarie per il corretto funzionamento del controllo.
Scorrendo il codice del metodo main possiamo individuare i seguenti metodi:

- *sysem_init*() Questo metodo funge da "costruttore", all'interno viene popolato il typedef del sistema, il quale tiene traccia di tutte le periferiche utilizzate e gli ingressi e le uscite del sistema. In questo metodo viene chiamato anche il metodo *cont_lidar_init*() il quale effettua il setup del sensore TOF.
- *setupReadingTime*() Questo metodo configura e avvia il timer nel quale saranno effettuate le letture e le azioni di controllo. Risulta necessario utilizzare un timer per garantire un elevata precisione temporale nelle letture e nel esecuzione delle azioni di controllo. Abbiamo effettuato questa scelta poiché l'esecuzione di un ciclo di while loop del main, non è deterministico e potrebbe introdurre dei ritardi non trascurabili.
- *startMeasurement*() Comunica al sensore l'inizio delle misurazioni, per come viene configurato il sensore le letture saranno effettuate in modalità continua, va richiesto l'inizio delle letture.
- *PID_init*()) Questo metodo inizializza il controllore PID con i valori di gain di default da noi trovati.
- *set_limit*()) Questo metodo imposta i limiti delle azioni del controllo. Nello specifico i primi due valori si riferiscono al massimo ed al minimo dell'uscita del controllore, gli altri due definiscono i limiti del integratore.

All'interno del loop troviamo fondamentalmente due metodi:

- *getrangeData()* il quale richiede tramite i2c l'ultimo dato misurato dal sensore, è necessario quest'approccio è necessario poiché qualsiasi tentativo di acquisizione temporizzata non porta a nessun esito positivo, quindi per garantire una corretta ricezione dei dati è stato deciso di utilizzare il sensore in modalità misurazione continua, il nuovo dato è letto ad ogni nuova iterazione del while loop, nel metodo main. Poiché la lettura del dato non è sincrona con le operazioni di controllo, il dato è memorizzato in un buffer circolare in modo che possa essere consumato al più presto.
- *set_parameters()* Da la possibilità di modificare i gain del controllo durante l'esecuzione del codice. questo permette di tarare i parametri in tempo reale senza dover nuovamente ricompilare.

Inoltre nel metodo main sono inserite tutta una serie di funzioni auto generate dal software CubeMX. Queste funzioni sono indispensabili poiché sfruttando l'HAL semplificano di molto il setup delle periferiche necessarie al progetto. Poiché ogni volta che si apporta una modifica alle periferiche, il codice viene ri-generato, per evitare delle spiacevoli perdite di codice, è stato creato un file *implementation.c* nel quale sono state inserite tutte le funzioni utili al progetto.

2.2.2 Implementation

Nel file implementation abbiamo inserito diverse funzioni.Quelle più importanti sono:

- $HAL\_TIM\_PeriodElapsedCallback()$ In questa callback di sistema, ogni volta che il **timer 10** fa sorgere un interrupt vengono eseguite la lettura dell'encoder e viene eseguita l'azione di controllo.
- $PID\_controller\_position()$ In questo metodo avviene il controllo vero e proprio del dispositivo, una volta settato il set-point, vengono richiamate dai buffer la misura più recente dell'encoder, subito dopo viene richiamata la funzione di update dei valori del PID utilizzando il metodo relativo $PID\_update()$. Nella funzione viene calcolata l'uscita del controllore al passo successivo. Una volta calcolata la nuova uscita, viene selezionata la direzione di rotazione del motore in base al segno dell'output del PID. L'output del PID viene utilizzato come set-point per la nuova traiettoria.
- $read\_encoder()$ Converte l'output dell'encoder incrementale in radianti.

2.2.3 PID

Per gestire l'anello di controllo del Ball And Beam planare, nel progetto è stato utilizzato il controllore proporzionale-integrale-derivativo (controllore PID o controllore a tre termini), una delle tecniche di controllo più utilizzate nei campi dell'automazione e del controllo dei sistemi dinamici. Questa tecnica è fondamentale per regolare e mantenere le prestazioni di un'ampia gamma di processi, sistemi e dispositivi industriali. Un controllore PID calcola continuamente un valore di errore e(t) come differenza tra un setpoint desiderato (SP) e una variabile di processo misurata (PV) e applica una correzione basata su tre termini:

- **Proporzionale (P):** Risponde all'errore corrente applicando un'azione correttiva proporzionale all'entità dell'errore, riducendolo rapidamente;

- **Integrale (I):**Accumula gli errori nel tempo, aumentando gradualmente l'azione di controllo per eliminare gli errori persistenti, migliorando l'accuratezza a lungo termine;
- **Derivativa (D):** considera il tasso di variazione dell'errore, aiutando ad anticipare i cambiamenti futuri e consentendo al controllore di regolarsi preventivamente per migliorare la stabilità.

L'uso appropriato del PID richiede la regolazione dei parametri: Kp, Ki e Kd. Per la regolazione è stata utilizzata come base di partenza i valori restituiti da PID Tuner di Matlab, inserendo la funzione di trasferimento del sistema in tempo discreto. Tuttavia, questi valori sono risultati non troppo vicini a quelli poi più usati nella realtà, trovati attraverso la sperimentazione in tempo reale grazie all'uso STM32CubeMonitor.

Per il progetto, il PID a tempo continuo è stato discretizzato utilizzando Tustin. I dati di un controllore PID, che può essere suddiviso in quattro blocchi, si trovano all'interno di una struttura chiamata pid_controller_t. Il primo blocco contiene i parametri di guadagno descritti in precedenza. Il secondo contiene i valori dell'errore e della misura precedenti, oltre ai valori dell'azione proporzionale e derivativa. Il terzo è dedicato ai limiti dell'uscita del controllore e dell'elemento integratore. L'ultimo è l'uscita, che rappresenta l'azione di controllo del PID.

```
typedef struct{

                int type;

                /* GAINS */
                float Kp;
                float Ki;
                float Kd;
                float tau;


                /* MEMORY */
                float prev_err;
                float prev_meas;
                float integrator;
                float derivative;

                /* LIMITS */
                float lim_out_min;
                float lim_out_max;

                float lim_integ_min;
                float lim_integ_max;

                /* OUTPUT */
                float out;

} pid_controller_t;
```

Figure 8: Typedef relativo al PID

Si notino i membri lim_out_min, lim_out_max, lim_integ_min e lim_integ_max della struct: questi valori sono utilizzati per una semplice implementazione di un meccanismo anti-windup, che si limita a impostare un valore limite massimo e minimo sia per il valore di uscita del controllore PID che per il suo valore integrale, evitando così la saturazione sia dell'integratore che dell'attuatore che riceverà l'uscita PID. Per l'inizializzazione del controllore, sono state definite due funzioni che vengono richiamate all'interno del main, prima del ciclo principale. La prima è $PID_init(pid\_controller_t * pid, floatKP, floatKI, floatTD, floattau)$ per i valori del controllore e $set_limit(pid\_controller\_t*pid, floatlim\_out\_min, floatlim_out_max, floatlim\_integ\_min, floatlim_integ_max)$ per inizializzare i limiti massimo e minimo dell'uscita e dell'integratore. Dopo l'inizializzazione,

viene utilizzato il metodo $PID\_update(pid\_controller\_t*pid, floatset\_point, floatmeasure, floatT\_C)$, che aggiorna il valore del controllore rispetto al valore di errore, calcolato come differenza tra il setpoint e la misura dell'uscita.
Per settare i parametri del PID è stata utilizzata una funzione $set_parameters(pid\_controller\_t * pid, floatKp, floatKi, floatKd, floattau)$, la quale viene richiamata ogni volta nel main.

Il metodo $PID\_controller\_position(system\_t * sys)$ viene utilizzato per calcolare l'uscita di controllo della velocità dei motori durante l'esecuzione delle traiettorie. Anche se si tratta di un controllo di posizione, è necessario inserire la velocità necessaria ai motori per raggiungere la posizione desiderata, poiché il driver del motore accetta un ingresso STEP/DIR, che rappresenta un comando di velocità. Il primo passo del metodo consiste nel calcolare la distanza tra la posizione di riferimento e la posizione effettiva misurata; quindi, per convertire l'ingresso di posizione in ingresso di velocità è stato utilizzato un metodo di derivazione discreta (metodo di Eulero) utilizzando come periodo di tempo il tempo equivalente che una traiettoria cicloidale avrebbe impiegato per raggiungere la posizione desiderata. Questo approccio consente di ottenere una traiettoria più lenta ma più sicura, che non presenta discontinuità, con un'implementazione del codice relativamente semplice:

```
if (fabs(u0-encoder_measure)<0.01){
    tc0= 1000000;
}else{
//tc0 = sqrtf(2*M_PI*fabs(u0-encoder_measure)/0.9); // 0.9 è l'accellerazione massima
  tc0 = sqrtf(2*M_PI*fabs(u0-encoder_measure)/1.25);
}

u0=(fabs(u0-encoder_measure))/tc0;
```

Figure 9: Calcolo del tempo relativo alla cicloidale

se il movimento risultante del manipolatore è troppo piccolo (cioè scende sotto la soglia del singolo microstep), invece di utilizzare la relazione tra l'accelerazione cicloidale e quella massima, il periodo di tempo viene impostato su un valore costante, sufficientemente grande da far avvicinare la velocità a zero, evitando qualsiasi movimento del motore. La velocità di uscita è determinata dal rapporto tra distanza e tempo.

Va sottolineato che in quest'implementazione del PID è stato scelto di utilizzare l'azione derivativa sulla misura piuttosto che sull'errore.Inoltre,l'azione derivativa è stata preceduta da un filtro passa basso il cui è possibile variare la pulsazione di taglio variando il parametro TAU. Questo serve a ridurre gli effetti del rumore sulle letture.

2.2.4 Ring buffer

Per memorizzare i dati, è stata utilizzata una semplice struttura dati a buffer circolare, chiamata Ringbuffer, che segue il metodo di accesso FIFO: due indici puntano alla coda e alla testa della struttura, relativamente agli indici di ingresso e di uscita; questi indici vengono incrementati quando i dati vengono inseriti o rimossi, rendendo questo ringbuffer una struttura buffer circolare mobile.
Per accedere ai dati memorizzati all'interno della struttura, sono stati implementati i metodi *rbpush*() e *rbpop*(), che consentono all'utente di spingere e di estrarre i dati dal buffer. Il metodo rbpop restituisce l'elemento meno recente inserito nel buffer e lo elimina dalla struttura dati, ma in alcuni casi l'elemento potrebbe essere ancora necessario altrove: Per evitare la perdita di dati, sono stati implementati i metodi *rbpeek*() , *rblast*() e *rbget*(), che permettono di accedere ai dati

senza cancellare l'elemento a cui si accede. Infine, il metodo *rbclear*() serve a resettare completamente il ringbuffer, inizializzandolo a uno stato noto. La suddetta struttura dati è stata utilizzata per memorizzare i dati del set-point e i dati ricevuti dai sensori (posizione della palla sulla beam e le posizioni degli alberi dei motori), questi ultimi utilizzati per calcolare la velocità dei motori anch'essi memorizzati all'interno di un ringbuffer. Tutti questi dati relativi al Ball and Beam sono stati memorizzati all'interno di una struttura dati chiamata system, contenente diversi ringbuffer.

```
typedef struct ringbuffer {
    uint8_t tail;                           /* where data will be pushed */
    uint8_t head;                           /* where data will be popped */
    uint8_t length;                         /* number of elements */
    rbelement_t buffer[RBUF_SZ];            /* actual buffer */
} ringbuffer_t;
```

Figure 10: Typedef relativo al PID

3 Configurazioni

In questo capitolo saranno descritte in maniera il più possibile esaustiva tutte le configurazioni da noi effettuate.Sottolineando sopratutto le motivazioni che hanno portato a tali conclusioni.

3.0.1 Configurazione delle periferiche e del sistema

Per quanto riguarda le periferiche e la configurazione del sistema, la maggior parte dei dettagli è contenuta nel rapporto di configurazione: nei paragrafi che seguono verrà descritto brevemente come sono stati configurati il sistema e le sue periferiche.

CubeMX

Per variare le configurazioni delle periferiche e i pin ai quali sono associati viene utilizzato il software CubeMx, gia all interno dell'STMCubeIDE. Questa software dispone di un'interfaccia grafica rende piuttosto agevole,la quale permette l'assegnazione e una prima configurazione delle periferiche. Questa procedura guidata risulta particolarmente conveniente poiché se le stesse operazioni fossero effettuate a mano, richiamando uno ad uno i relativi comandi dell'HAL, richiederebbero parecchio tempo,il quale risulterebbe sottratto all'implementazione vera e propria dell'applicazione.Una volta terminata la configurazione delle periferiche, tramite una finestra pop-up, si viene avvertiti della generazione automatica del codice relativo alle impostazioni scelte. In questa fase bisogna stare molto attenti ad inserire il proprio codice nelle zone delimitate dagli appositi commenti, poiché il codice al di fuori delle opportune aree, se si andranno a modificare nuovamente le impostazioni dal file *.ioc*, alla nuova generazione di codice, l'intero corpo del file verrà riscritto e dunque si perderanno le righe di codice poste al di fuori aree delimitate dall'IDE.

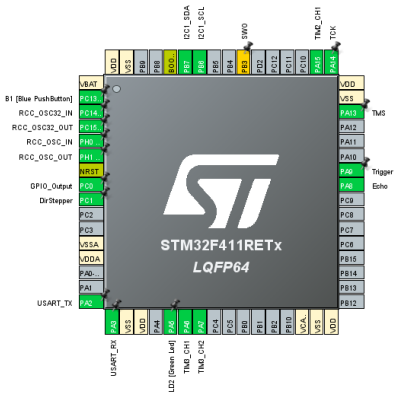


Figure 11: Configurazione dei pin del microcontollore

Timers

I timer sono componenti fondamentali nei sistemi embedded, utilizzati per gestire operazioni temporizzate in modo preciso e affidabile. Questi dispositivi consentono di eseguire azioni a intervalli di tempo regolari o dopo un certo periodo, indipendentemente dal processore principale. I timer possono generare interruzioni, misurare intervalli di tempo, controllare eventi periodici, gestire protocolli di comunicazione temporizzati e realizzare funzioni di controllo in tempo reale. In sintesi,

i timer sono strumenti essenziali per garantire la sincronizzazione e il corretto funzionamento di molte applicazioni embedded, contribuendo a migliorare l'efficienza e la reattività del sistema.

Nel nostro sistema sono stati utilizzati una serie di timer per gestire diverse operazioni:

- **TIM2 :**utilizzato per generare i segnali PWM per controllare il motore che muove il braccio della Beam. Per controllare i segnali PWM, i registri ARR, CCR e PSC vengono impostati programmaticamente. Il Prescaler è impostato su un valore fisso di 0 per garantire che la frequenza massima del segnale PWM inviato al driver TMC2209 non superi il valore massimo di 6MHZ, come indicato nella documentazione ufficiale. Il registro ARR è impostato rispetto al punto di regolazione della velocità:

$$ARR = \frac{RESOLUTION * f_{clk}}{|v|} \tag{3}$$

- **TIM3 :**timer utilizzato per leggere e decodificare i segnali emessi dall'encoder collegato al motore. Il timer è impostato in modalità Encoder, dunque utilizza due canali.Il registro **ARR** è impostato su un valore fisso: rispettivamente 4000, che rappresenta il numero di tick dell'encoder necessari per una rotazione completa

- **TIM10:** questo timer viene utilizzato per cronometrare correttamente l'istante in cui verrà letto l'encoder: i registri **ARR** e **PSC** vengono fissati in base al tempo di campionamento scelto:

$$ARR = T_s * \frac{f_{clk}}{PSC} \tag{4}$$

  con Ts è il tempo di campionamento, fclk è la frequenza di clock e il prescaler è impostato a 2. Questa configurazione ci garantisce un tempo di campionamento di circa 1.3ms.

I2C

La periferica di comunicazione i2c, è essenziale per la comunicazione con il sensore TOF,poiché le librerie fornite dal costruttore sono specifiche per questa interfaccia. Nella comunicazione i2c poiché più dispositivi potrebbero essere connessi su uno stesso bus di comunicazione è necessario specificare l'indirizzo del dispositivo con il quale bisogna comunicare, a differenza di quanto riportato da diversi datasheet, il corretto indirizzo per il sensore è **0x52**.
Per quanto riguarda la comunicazione vera e propria è gestita all interno dell libreria del sensore. Quando si inizializza il sensore va solo fornito l'handler dell'interfaccia i2c selezionata durante la fase di configurazione dell MCU. Dopo aver selezionato l' interfaccia va selezionato il rate di comunicazione,nel nostro caso è stato scelto un rate di 100 KHz.

3.0.2 Configurazione STM32CubeMonitor

Il flow necessario per poter far l'acquisizione dati del Ball And Beam è presente nel cartella RangeTest ed è un file di tipo .json dal nome "*tuning_BallAndBeam_Cubemonitor.json*", di conseguenza bisogna importarlo attraverso la sezione "Import" presente nel menu che si apre cliccando sull'icona con tre le barrette orizzontali in alto a destra. Una volta importato avremo il flow come nella seguente figura.

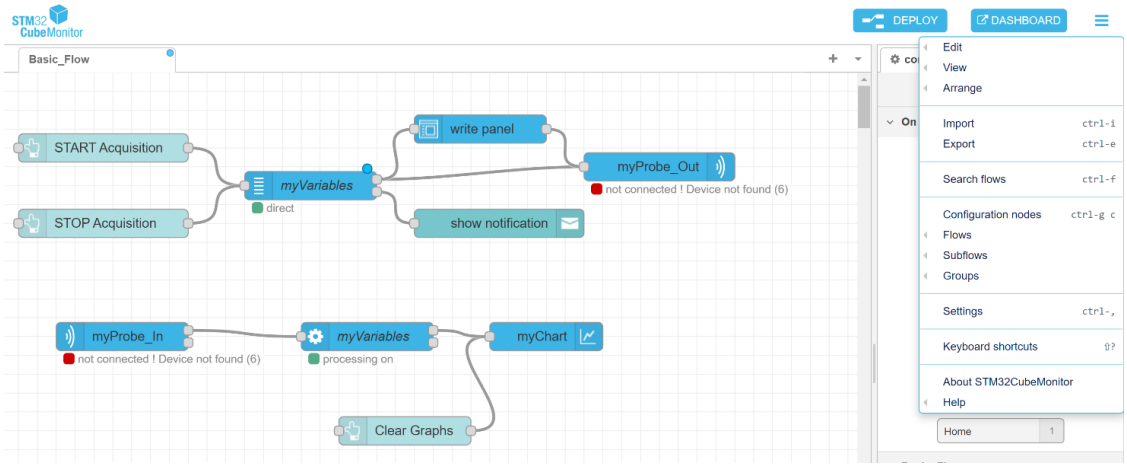


Figure 12: Diagramma del Flow

Il flow si divide in due parti, ognuna costituita da "nodi":

1. La parte in alto è necessaria per controllare l'acquisizione dati e modificare real-time i parametri del PID.
2. La parte inferiore permette di mostrare su un unico grafico le variabili dai noi selezionate.

Configurazione Variabili

Il primo step richiede la configurazione della variabili, ed è necessario cliccare "myVariables" cerchiato in rosso.

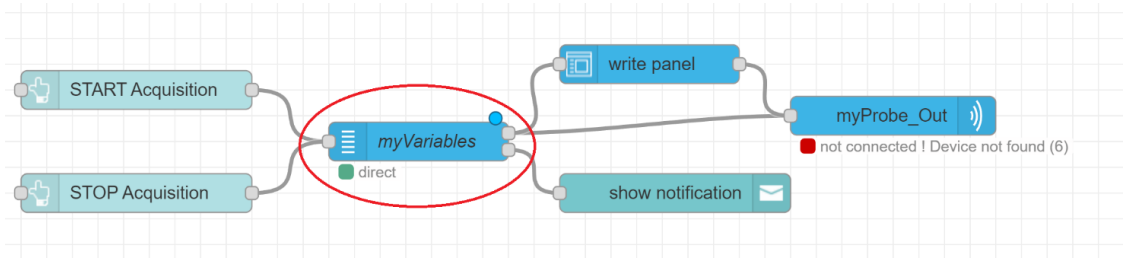


Figure 13: Configurazione delle Variabili

Si aprirà una finestra per modificare il nodo. Da qui è necessario selezionare il file .elf dal nome "RangeTest.elf", un file eseguibile creatosi con la compilazione del codice nel STM32CubeIDE.

Quindi, prima di fare questo passaggio, è necessario assicurarsi che sia stato compilato il codice. Come da immagine, appena si apre il menù la prima volta che è stato aperto il flow sul proprio computer, ci sembrerà che sia già configurato perché vediamo che è già selezionato il file **.elf**. Selezionare l'icona a forma di matita.

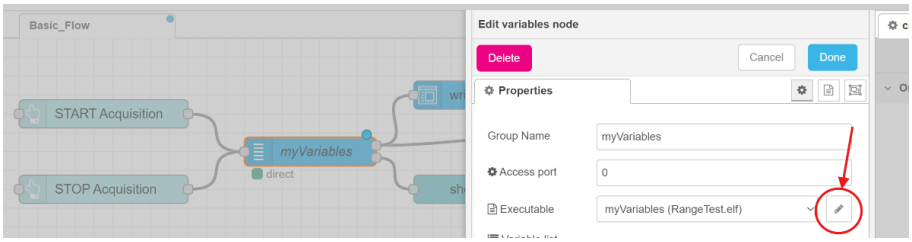
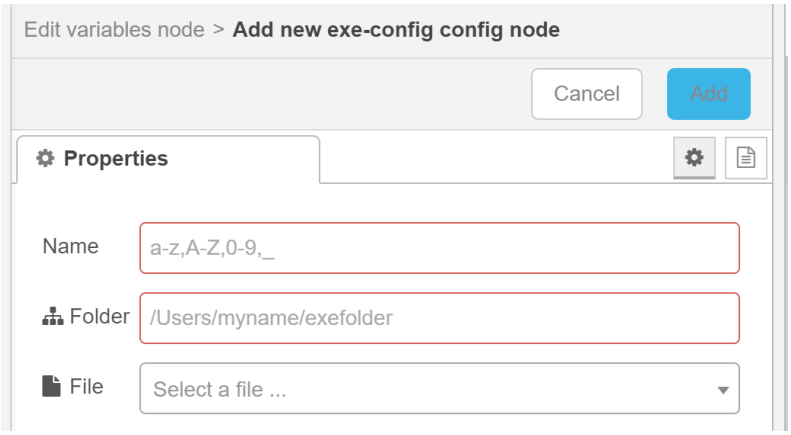
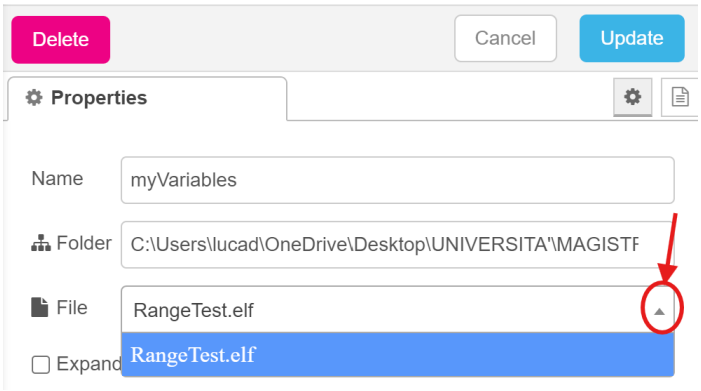


Figure 14: Selezione file .elf

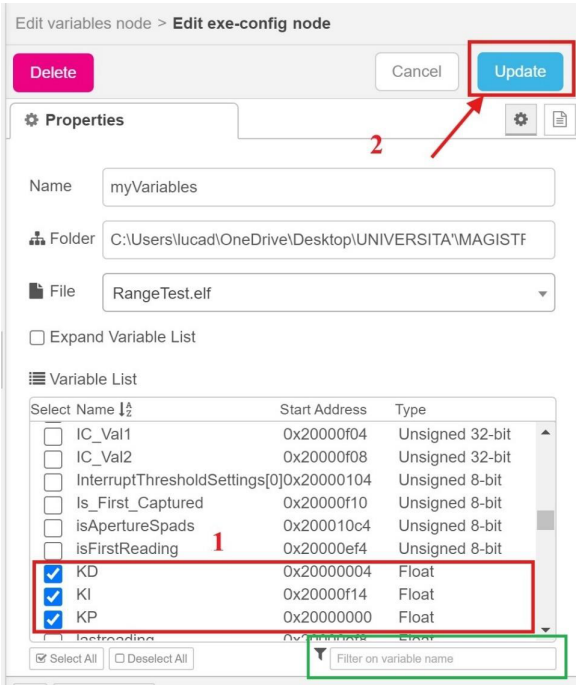
Ci si troverà in questa finestra. Nella casella "Folder" sarà presente già un percorso, il quale però andrà eliminato per inserire il proprio percorso. È molto importante inserire correttamente il percorso, altrimenti CubeMonitor non troverà il file elf. Il percorso da inserire dovrà avere questo inizio e fine: `C:\Users\.....\Debug`. Nel campo "Name" se già presente un nome lasciare quello, altrimenti inserire il nome del nodo "myVariables".



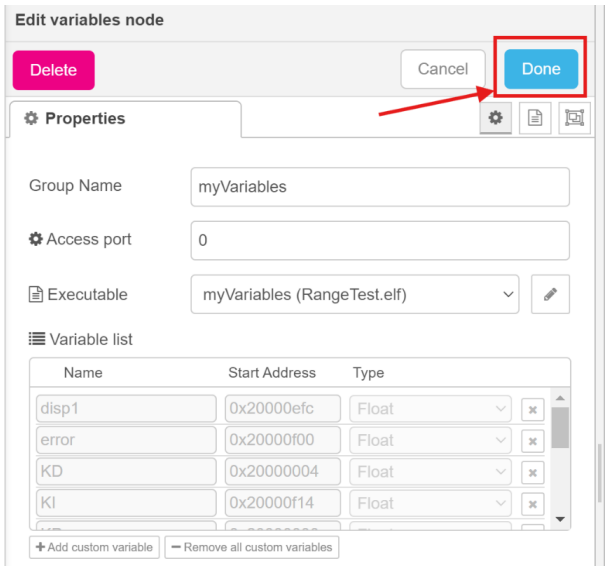
Inserito il percorso giusto, il software sarà in grado di trovare il file che andrà selezionato. Per selezionarlo è necessario cliccare sul triangolino della sezione "File" e selezionare "RangeTest.elf".



Selezionato il file elf , avremo la possibilità di accedere alle variabili desiderate per mostrale sul grafico o per il tuning del PID. Per trovarle è possibile sfruttare la barra di ricerca in basso a destra. Una volta selezionate, bisognerà cliccare su "Update".



Una volta fatto l'update, si potranno notare le variabili scelte nella "Variable list" è si potrà cliccare su "Done".


Edit variables node
Delete
Cancel
Done
Properties
Group Name
myVariables
Access port
0
Executable
myVariables (RangeTest.elf)
Variable list
Name
Start Address
Type
disp1
0x20000efc
Float
error
0x20000f00
Float
KD
0x20000004
Float
KI
0x20000f14
Float
Add custom variable
Remove all custom variables

Log dei dati

Per andare a selezionare, la tipologia di salvataggio dei dati e la modalità con cui catturare i valori da inserire, bisogna cliccare sul blocco “myVariables” in basso. Una volta aperto, è possibile

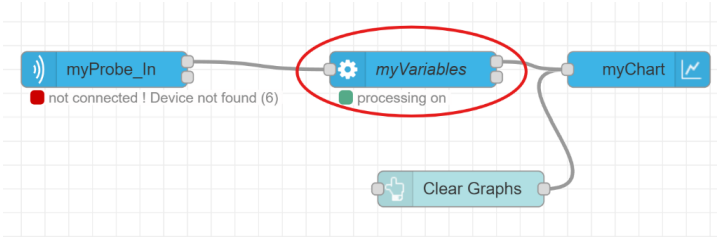
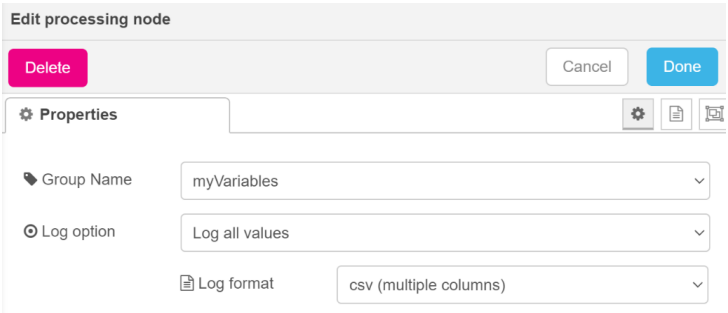
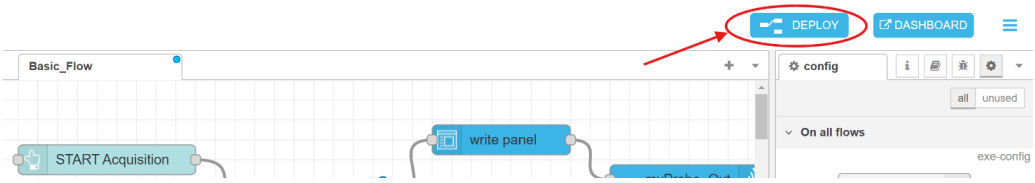


Figure 15: Diagramma del Flow

selezionare il tipo di formato per salvare i dati e le modalità. Nel progetto è stato scelto di usare le modalità come mostrato nella figura successiva.

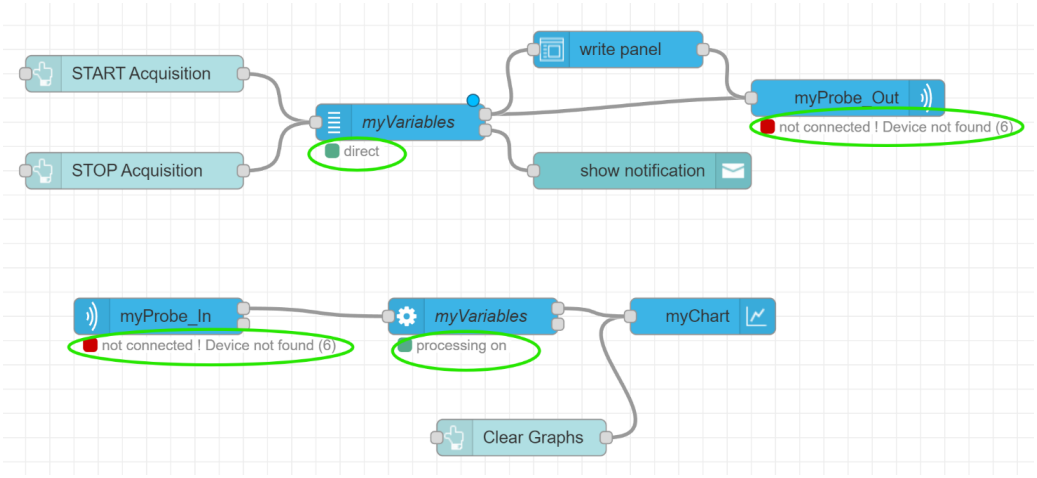


Per salvare tutte le modifiche fatte sul flow, è necessario cliccare su “Deploy”

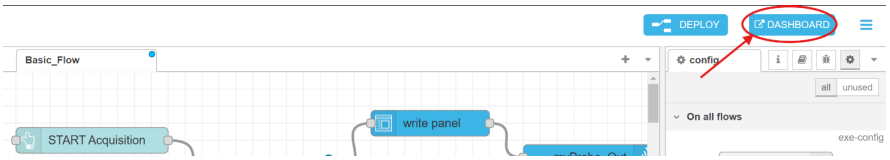


Acquisizione dati e tuning

Prima di poter effettuare l'acquisizione dati e il tuning, è necessario che i quadrati cerchiati siano tutti verdi. Questi lo sono se la board è connessa e se il codice NON non è in run su STM32CubeIDE.



Una volta controllato che sono tutti verdi, è possibile cliccare sul tasto "Dashboard".



Il Dashboard porta alla sezione di controllo. Se tutto sarà stato configurato bene, avremo questa dashboard.

- **Start acquisition:** avvia il codice e l'acquisizione dati.
- **Stop Acquisition:** ferma solamente l'acquisizione dati ma non il codice. Una volta fatto lo stop, le variabili in memoria si resetteranno di default, perciò sarà necessario reinserirli.
- **Variabili:** per modificare i valori real-time, dovremo scrivere il numero desiderato nella casella "float" e per salvare la modifica andrà cliccato sul tasto "Write".

Una volta fermata l'acquisizione dei dati, si creerà un file dal nome "log_myVariables_data", contenente i valori assunti nel tempo delle variabili. Il file è presente all'interno di una cartella di nome "log" che si creerà con l'installazione di CubeMonitor.

Sul grafico saranno presente tutte le variabili selezionate ma per visualizzare solo quelle di interesse, ad esempio disp1 ed error, basta cliccare sulle variabili, riportate in basso all'asse dei tempi accompagnate da un cerchio di un colore, per rimuoverle.

Accorgimenti

Se il Ball and Beam non sta funzionando bene durante l'esecuzione di CubeMonitor, non è per forza necessario fermare l'acquisizione dati ma basta provare ad effettuare reset con il bottone sulla board o se il sensore si è bloccato (lo si può notare sia da disp1 se assume un valore costante o dal comportamento della sistema reale) basta scollegare il filo rosso dell'alimentazione del sensore e ricollegarlo.

Un ulteriore problema riscontrato, è la mancata risposta da parte del sensore, la causa del problema al momento della stesura del documento è ignota. Per evitare questa inconvenienza, ad ogni nuovo re-set del sistema rimuoviamo per qualche secondo l' alimentazione del sensore TOF questo fa si che al nuovo avvio, il microprocessore all'interno del sensore risulta resettato e pronto ad accettare una nuova configurazione.

4 Schemi

4.1 Schema Generale

Il Seguente schema concettuale rappresenta in maniera coincisa la struttura delle connessioni e la disposizione dei componenti del sistema. Nello schema sono stati riportati i piedini utilizzati per connettere il microcontrollore ai relativi dispositivi.

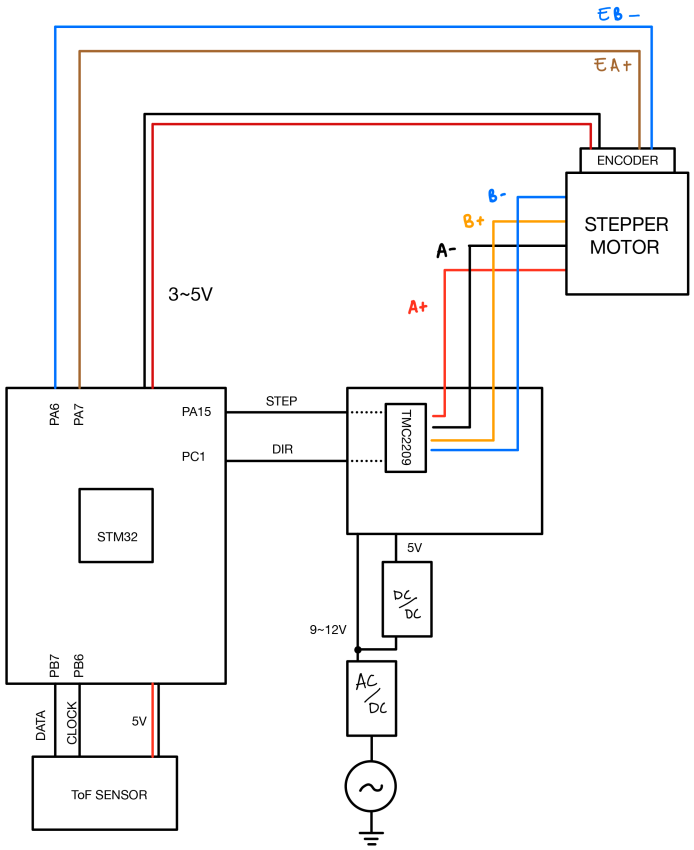


Figure 16: Schema riassuntivo delle connessioni del sistema.

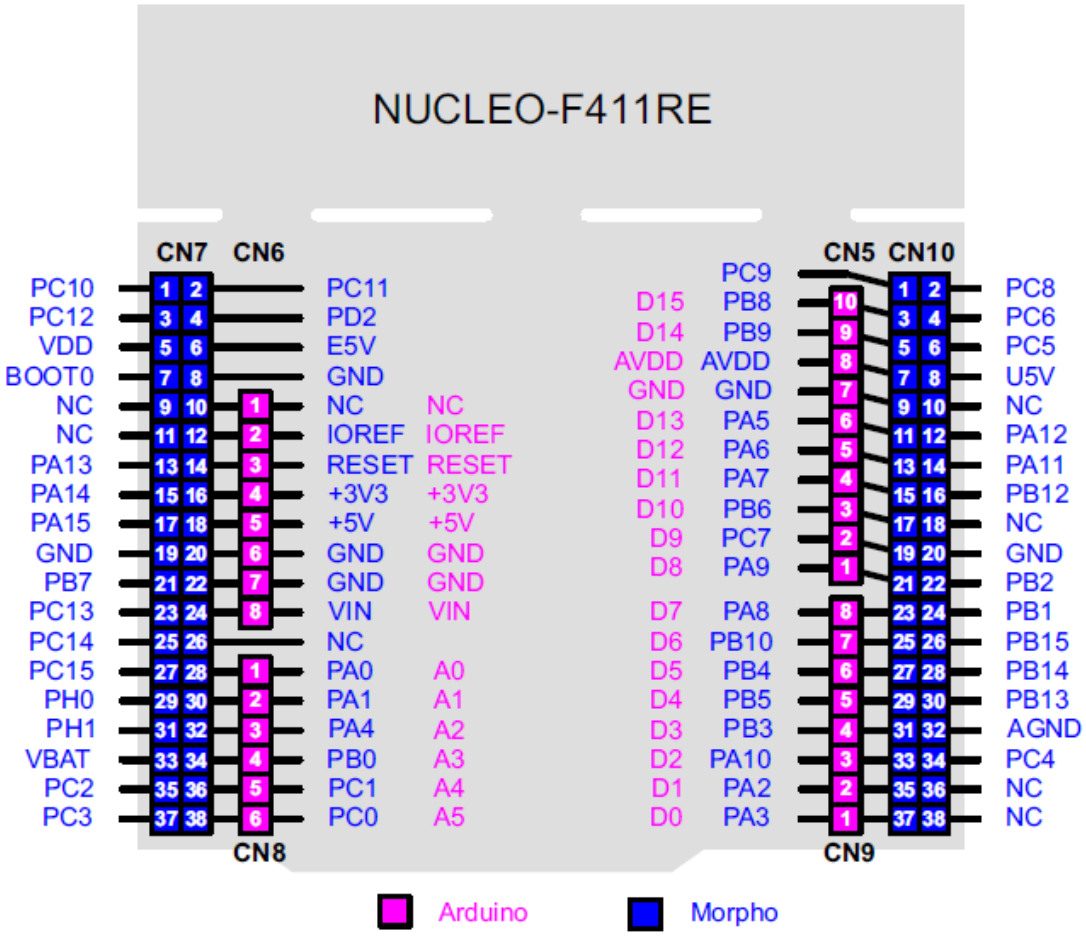


Figure 17: Disposizione dei Pin della Board del microcontrollore .

4.2 Piedinatura

Nella seguente sezione è descritta in maniera sintetica la piedinatura da noi configurata per il microcontrollore.I restanti pin definiti nel file .ioc derivano dalle configurazioni standard delle board Nucleo, e sono necessari per il corretto funzionamento del dispositivo.

PIN	Dispositivo
PA7	Canale A dell'Encoder
PA6	Canale B dell'Encoder
PA15	Output del segnale PWM (Step)
PC1	Output del segnale per la direzione (Dir)
PB7	Segnale dati dell'interfaccia i2c
PB6	Segnale clock dell'interfaccia i2c