RELAZIONE PROGETTO: MOBILE PRINTER

Sistemi Embedded A.A 2020/2021

Matteo Kolyszko – 844526
Keivan Motavalli - 829598
Alessandra Deborah Oggioni - 835558

Indice

1. Introduzione
2. Componenti elettroniche
3. Realizzazione
    3.1 Soluzioni provate
4. Considerazioni Finali

Introduzione

Il progetto presentato in questa relazione nasce come prova finale del corso di Sistemi embedded e ha come obiettivo quello di testare le conoscenze acquisite nel corso.
Questo nostro lavoro è solo la prima parte del progetto finale che ha come obiettivo quello di avere una stampante mobile che sia in grado di girare per un piano dell'università, evitando ostacoli e reagendo a stimoli esterni variando la propria velocità.
Questa prima fase è consistita nel programmare una scheda Nucleo-F767ZI, attraverso il *Device Configuration Tool* dell'IDE, in modo da creare un controllo sulle ruote motrici di Robuter, piattaforma motorizzata messa a disposizione dall'università, attraverso il controllo di due ponti-H collegati agli encoder della piattaforma.

Componenti elettroniche

PONTE H:

- Roboponte V6: Il cuore del ponte H è il componente HIP4081AIBZ, che è un driver per ponti H. I due rami del ponte sono pilotati dai segnali ALI e BHI. Tutti gli ingressi accettano un segnale di massimo 12V. Gli ingressi riconoscono come livello logico alto una tensione superiore ai 2,7V mentre una tensione inferiore a 1V è riconosciuta come livello logico basso, sono quindi compatibili sia con logiche a 3,3V che con logiche a 5V.

NUCLEO-F767ZI:

- È una scheda STM32 Nucleo-144 con un microcontrollore STM32F767ZIT6. Il connettore ST Zio, un'estensione di Arduino Uno, fornisce l'accesso alle diverse periferiche e alle basette ST morpho. La scheda Nucleo-144 STM32 è dotata del programmatore/debugger ST-LINK/V2-1 ed è fornita insieme alla libreria software completa HAL STM32.
    - Microcontrollore STM32 in pacchetto LQFP144, alimentatore flessibile, 5V da ST-LINK/V2-1 USB VBUS
    - Alimentatori esterni da 3,3V & 7V-12V sui connettori ST Zio o ST morfo, 5V su connettore ST morpho
    - OTG USB o full speed con connettore Micro-AB (a seconda del supporto di STM32)

Robuter II:

- Il Robuter è un piattaforma mobile rettangolare non olonomo, sviluppato dalla compagnia ROBOSOFT. La locomozione del mezzo avviene tramite il controllo di due motori a corrente continua indipendenti, uno per ogni ruota. È lungo 102.5 cm, largo 68 cm, alto 44 cm, pesa 150 kg e può trasportare fino a 120 kg.

Realizzazione

Il progetto prevede di impiegare il robuter come piattaforma mobile e supporto dalla stampante, che verrebbe alimentata mediante un inverter 48V CC -> 230V AC tramite le batterie di trazione del Robuter stesso. Il controllo verrebbe affidato ad una scheda di sviluppo Nucleo-144 che leggendo attuerebbe i motori di trazione in corrente continua del mezzo tramite due ponti-H, leggendo i pochi sensori presenti sul robuter dopo le modifiche effettuate nel corso degli anni, ossia gli encoder di posizione posti sui due assi motore, con lo scopo di modulare la potenza fornita ad ogni singolo motore per compensare eventuali forzanti esterne incontrate sul percorso (piccoli ostacoli da superare verticalmente con una sola delle due ruote o entrambe, perdite di aderenza parziali).

Tale meccanismo di retroazione è realizzato mediante due anelli di controllo lineari e da un anello di controllo incrociato, posti nel ciclo principale del programma in funzione sulla Nucleo.

I due anelli di controllo lineari, uno per ruota, confrontano l'obiettivo in tick per millisecondo di una ruota, con la lettura di tick degli encoder occorsi nell'intervallo di tempo nel quale viene effettuata una iterazione dei cicli di controllo (1 millisecondo), e moltiplicano questa misura per una costante lineare che traduce la differenza (obiettivo - lettura) in un valore di duty cycle che determina la potenza fornita ai motori CC e dunque momento meccanico e velocità di rotazione dei motori.

Siccome la correzione di velocità realizzata dagli anelli di controllo lineari non è immediata, una forzante che abbia alterato l'avanzamento di una sola delle due ruote produrrà comunque un errore rispetto alla traiettoria obiettivo del Robuter, errore che se non compensato andrà a diventare tanto più grande tanto più avanzerà il mezzo.

Per questa ragione esiste un terzo anello di controllo, detto controllo incrociato, che accumula in una variabile detta del "controllo integrale" l'errore tra due ruote rispetto all'obiettivo di avanzamento, secondo la formula

```
integral = integral + k_integral*(left_reading + right_bias - right_reading)
```

con k_integral la costante moltiplicativa impiegata per determinare l'aggressività della correzione proposta dal controllo incrociato.

Il right_bias è la traduzione in tick (passi di un encoder) dell'angolo di rotazione desiderato come obiettivo della traiettoria per la velocità di avanzamento corrente, il suo utilizzo nel controllo incrociato che permette di avere controllo direzionale nell'avanzamento del robuter.

Il risultato di questo calcolo viene poi impiegato all'interno degli anelli di controllo lineari in ognuna delle loro iterazioni, venendo sottratto, nel caso della ruota sinistra, alla differenza tra tick obiettivo sx e lettura corrente in tick dell'encoder sinistro, e viceversa venendo addizionato, nel caso della ruota destra, alla differenza tra tick obiettivo dx e lettura corrente in tick dell'encoder destro.

Dunque gli anelli di controllo lineari, una volta tenuto conto della correzione fornita dal controllo incrociato e moltiplicato il risultato per la costante proporzionale, assumono la seguente espressione:

```
float left_correction = k_proportional * ((float)(sx_ticks -
sx_read_ticks) - integral_error);

float right_correction = k_proportional * ((float)(dx_ticks -
dx_read_ticks) + integral_error);
```

La traiettoria da seguire, in obiettivo di avanzamento di metri al secondo, verso e direzione di avanzamento verrebbe calcolata da un computer posto a bordo del Robuter e collegato a diversi sensori ambientali (come misuratori di distanza ad ultrasuoni);

Tale computer verrebbe programmato grazie un insieme di librerie software atte a calcolare e modificare una traiettoria, come il framework ROS (Robot Operating System) e comunicherebbe l'obiettivo alla scheda Nucleo-144 mediante comunicazione seriale.

Allo scopo di semplificare l'architettura software e hardware del progetto, nella realizzazione del controllo motori PWM tramite ponti-H sono state sfruttate le periferiche Timer della scheda Nucleo - capaci non solo di compiti di timekeeping e di elevare interrupt alla scadenza di un timer, ma anche di analizzare segnali in ingresso per contare le occorrenze di un evento, come i tick di un encoder, rendendo molto semplice verificare la velocità di rotazione di un asse motore leggendo i tick registrati nell'apposito contatore della periferica timer in un prefissato intervallo di tempo, resettando successivamente tale contatore, senza dover scrivere più complicato e computazionalmente esigente codice di polling ed analisi dei segnali.
Similarmente, le periferiche timer della nucleo sono in grado di generare segnali PWM per il controllo motori venendo configurati in

inizializzazione con una delle modalità pwm utilizzabili, impostando in dei registri periferica la frequenza PWM, il rispettivo duty cycle ed un eventuale sfasamento del segnale e le uscite da utilizzare, semplificando di molto la realizzazione del software e rendendo non necessaria la realizzazione di schede aggiuntive atte, ad esempio, ad adattare il segnale pwm secondo le specifiche richieste dal modello di ponti-H impiegati (richiedono in input sia il segnale pwm, che il suo complementare: i timer della Nucleo sono in grado di generare questa coppia di segnali).

Nei vari tentativi che verranno descritti in questa sezione, vengono usati i timer general-purpose (come Tim9, Tim3, Tim2 e Tim4) o advanced-control (Tim1) per controllare i ponti-H attraverso la scheda nucleo.

Questi timer sono dei contatori auto-reload a 16-bit o 32-bit controllati da un prescaler programmabile, possono essere usati per vari scopi, tra cui misurare la lunghezza degli impulsi dei segnali d'onda dati in input o per generare onde come output (PWM). Sia il periodo delle lunghezze d'onda che delle onde possono essere modulati da pochi microsecondi fino a svariati millisecondi usando il il timer del prescaler e l'RCC clock controller prescalers.

I Timer sono stati settati in modalità generazione segnale PWM per controllare i motori tramite ponti H, questa modalità (Pulse Width Modulation mode) permette di generare un segnale con una frequenza determinata dal valore del TIMx_ARR register e un duty cycle determinato dal valore del TIMx_CCRx register.

4.1 Soluzioni provate

Di seguito vengono descritti i tentativi effettuati con lo scopo di controllare i ponti-H con la periferica Timer della Nucleo.

Nel primo tentativo è stato usato il *Timer9* per generare i segnali PWM di controllo di entrambi i motori alla medesima frequenza ma con duty cycle impostati rispettivamente dal registro *CCR1* e dal registro *CCR2*.

Tale approccio effettivamente permetteva di generare i segnali di controllo, ma questi non erano direttamente impiegabili sui ponti-H forniti (tipo *lock antiphase drive*), in quanto ognuno di questi ponti-H richiedeva un doppio segnale per essere pilotato: uno attivo per controllare il potenziale del motore in un verso di rotazione con l'altro contestualmente non attivo; e l'altro segnale per controllare il potenziale del motore nel verso di rotazione opposto.

Ciò si può dunque ottenere con la configurazione della nucleo provata unicamente prendendo il segnale di controllo destinato al ponte-H di un singolo motore e derivandone, in parallelo, un altro segnale che sia la sua negazione (passando attraverso un gate NOT).

La scelta di utilizzare solo configurazioni software della scheda nucleo senza ricorrere a soluzioni hardware, ha portato a sperimentare diverse modalità d'impiego dei timer per ottenere per ogni motore un segnale ed il suo complementare.

Tra le varie modalità di impiego dei timer general purpose descritte nella documentazione la scelta è ricaduta sulla modalità chiamata *Asymmetric PWM mode* che permette su una periferica Timer della nucleo di generare due segnali PWM alla stessa frequenza, con duty cycle distinti, rispettivamente controllati dai registri *CCR1* e *CCR3,* e ritardo (fase) controllati dai registri *CCR2* e *CCR4*.

L'idea era di impostare nel registro *CCR1* il duty cycle del primo segnale e nel *CCR2* il ritardo a 0, nel registro *CCR3* il complementare del duty cycle del primo segnale e come ritardo nel registro *CCR4* la durata stessa per il quale il primo segnale restava attivo, in modo da attivare il secondo appena il primo passasse a valore basso.

Sono stati riscontrati due problemi: il primo causato dalla dimenticanza di attivare effettivamente la generazione non chiamando la funzione *HAL_TIM_Encoder_Start*. Il secondo dovuto alla cattiva comprensione della documentazione fornita: infatti uno dei due segnali, il secondo, era sempre costante.

Sperimentando è stato notato che il registro *CCR0*, impostato con valore del ritardo '0', in realtà impostava il duty cycle del secondo segnale: assegnando un valore diverso da 0 o il valore complementare del primo duty cycle, questo veniva effettivamente generato, ma non è stata trovata una soluzione in grado di controllare il ritardo agendo sugli altri due registri *CCR* della periferica timer.

Nei successivi tentativi è stata provata la modalità di *complementary output* dei dispositivi timer, generando dunque un solo segnale ed impostando una maschera di bit nel registro *CCER* della periferica Timer per attivare la generazione automatica di un segnale complementare a quello del primo canale sul secondo canale.

Non era chiaro se servisse abilitare la modalità *complementary output* solo per il secondo canale dove doveva essere generato, o anche sul primo dove venivano impostati manualmente i parametri della generazione PWM.

Per abilitarla solo sul primo canale è stata provata la seguente impostazione nel registro *CCER*:

```
bit0: 1
bit1: 0 /active high
bit2: 1 //enable complementary output
bit3: 0 //active high
bit4-21: 0
```

Scrivendo le seguenti maschere:

Maschera degli uni: 10100000000000000000000000000000 in *OR* sul registro *CCER*

Maschera degli zeri: 10101111111111111111111111111111 in *AND* sul registro *CCER*

Non riuscendo ad ottenere output sul secondo canale, che era comunque stato configurato per la generazione PWM dal *Device Configuration Tool* dell'IDE, è stato fatto un tentativo impostando per la generazione complementare anche il primo canale mediante la seguente maschera di bit del registro *CCER* della periferica Timer in uso:

```
bit0: 1
bit1: 0 /active high
bit2: 0 //enable complementary output
bit3: 0 //active high
bit4: 1
bit5: 0
bit6-21: 0
```

Maschera degli uni, secondo tentativo: 10001000000000000000000000000000 in OR sul registro

Maschera degli zeri, secondo tentativo: 10001011111111111111111111111111 in AND sul registro

Non cambiando il risultato, è stata provata la maschera di bit trovata su un tutorial per un altro dispositivo della famiglia STM32, che comunque avrebbe abilitato l'output complementare su entrambi i canali in uso:

```
TIM1->CCER |= 4; //enable ch1 complimentary output
TIM1->BDTR |= 0xff; // specify the maximum amount of deadtime
TIM1->CCER |= 64; //enable ch2 complimentary output
TIM1->BDTR |= 0xff; // specify the maximum amount of deadtime
```

Anche questo tentativo, verificato con un diverso modello di oscilloscopio, ha prodotto un solo segnale in output.

É stata infine adottata una soluzione che si è verificata semplice da implementare: per ottenere, su un encoder dedicato al controllo motore, un segnale sul primo canale ed il suo complementare sul secondo canale, la configurazione del progetto software richiedeva semplicemente di abilitare entrambi i canali impiegati in modalità generazione PWM, impostando la polarità (*CH Polarity*) di un canale ad *High* mentre la polarità del secondo canale a *Low*.
Nonostante il nome del parametro possa indurre a ritenere che serva a controllare quale verso segue la corrente tra il pin segnale ed il ground, in realtà il parametro porta a generare il complementare di un segnale, impiegando il medesimo clock ed i medesimi registri (senza bisogno di inizializzare dei diversi registri per il complementare del primo segnale).

Realizzata questa semplice modifica alla configurazione del progetto è stato finalmente possibile controllare i due motori del Robuter.

Il gruppo si è quindi dedicato a trovare parametri adeguati per le costanti di feedback proporzionale ed integrale.

Valori consistenti della costante integrale, utilizzata nel controllo incrociato per compensare nel tempo su una ruota disallineamenti nella velocità di avanzamento della ruota opposta rispetto all'obiettivo dato, portavano a blocchi o a rotazione a scatti delle ruote.

Ciò era dovuto al verificarsi di condizioni di integer overflow sui registri che controllano il duty cycle generato dalle periferiche timer: è stato tamponato ponendo dei controlli di limite superiore ed inferiore sul duty cycle da ottenere una volta calcolate le correzioni mediante i loop di controllo proporzionale ed integrale per le due ruote.

Insieme alla risoluzione di altri errori di programmazione, queste modifiche e correzioni non hanno comunque portato a realizzare il funzionamento desiderato da progetto. Infatti una ruota tendeva a rallentare progressivamente, l'altra ad accelerare.

Riducendo, in diverse esperienze, il valore della costante integrale usata nel controllo incrociato tra le due ruote, questo effetto non era più riscontrabile.

Individuate delle costanti che ponevano il sistema in condizioni operative stabili, si è iniziato a pensare a come comandare il microcontrollore con un obiettivo (velocità obiettivo di avanzamento in e bias aggiunto/sottratto all'obiettivo della ruota destra per determinare la rotazione del robuter).
É stato suggerito di adottare un approccio incrementale: scrivere prima un task sul microcontrollore stesso che generasse localmente degli

obiettivi, solo successivamente generare gli obiettivi per il microcontrollore su un computer esterno, connesso a sensori, come quelli ad ultrasuoni, capaci di ottenere informazioni sull'ambiente circostante e la presenza di ostacoli.

Preparandosi alle prove dinamiche e, di conseguenza, ad individuare nuove costanti per i loop di controllo integrale e proporzionale, è stato osservato che i valori scelti per la costante integrale erano inefficaci a realizzare una effettiva correzione integrale in caso di scostamenti nella velocità di avanzamento di una ruota.
Anche un leggero incremento portava al riverificarsi della condizione di rallentamento di una ruota e progressiva accelerazione dell'altra, nonostante le correzioni software apportate.
Si è quindi provato a misurare la velocità di rotazione delle singole ruote anche eliminando il controllo incrociato, impostando il medesimo obiettivo di avanzamento per entrambe, apponendovi una striscia di nastro colorato e misurando visivamente le rotazioni in un dato intervallo di tempo.
Le velocità di rotazione effettive delle due ruote, posto un uguale obiettivo ed in assenza di controllo incrociato, differivano.
Analizzando con il debugger della suite STM32CUBE i valori PWM scritti dal loop di controllo proporzionale si sono potuti verificare valori differenti;
Si è quindi passato a verificare le letture dagli encoder posti sugli assi motore delle ruote.
Fissato un indicatore su ognuna delle ruote, è stata fatta realizzare una rivoluzione completa ad ognuna delle ruote, con riferimento un elemento della carrozzeria del robuter presente simmetricamente sulle due opposte fiancate.
L'encoder di una ruota leggeva circa 20 mila tick completata una rotazione, l'altro circa 43 mila tick completata una rotazione della ruota associata.

Per escludere che tali differenti letture fossero causate da un problema software, di configurazione del microcontrollore, o di collegamenti e realizzazione delle breadboard poste tra il microcontrollore e le uscite di segnale degli encoder, sono state invertite direttamente le uscite degli encoder destro e sinistro sulla breadboard.

Le diverse letture per-rivoluzione si sono manifestate nuovamente e con numeri congrui, permettendo di escludere fossero causate da problemi software, di configurazione o di collegamento tra il microcontrollore, la breadboard, nonché nella realizzazione di quest'ultima.

Il problema potrebbe essere stato causato da un difetto nei collegamenti in uscita degli encoder, da un difetto in uno degli encoder, da una diversa tipologia di encoder utilizzata oppure da un diverso rapporto di riduzione tra albero motore-encoder e ruote.

Prima di smontare il robuter per verificare rapporti di riduzione, collegamenti interni e modello dei due encoder, è stato provato a verificare se nelle sezioni di collegamenti elettrici esterne al tubo contenente motori, meccanica ed encoder fossero presenti interruzioni o distorsioni in uno dei segnali della coppia di segnali utilizzati per leggere la posizione di ognuno degli encoder, osservando con un oscilloscopio la forma d'onda rilevabile su ognuno dei cavi segnali in diversi punti a partire dall'estremità che viene collegata alla breadboard.

Collegato il ground di una sonda dell'oscilloscopio al ground della scheda Nucleo-F767ZI e prima di poter avvicinare la sonda all'estremità del cavo segnale, si è verificato un evento termico incontrollato nel SOC contenente il microcontrollore della scheda Nucleo, che ha portato alla sua completa operatività. Contestualmente, uno dei due ponti-H ha visto interrompersi il proprio fusibile di protezione (20A), tentativi di alimentare nuovamente quel ponte-H in assenza di input e segnale di enable, con un nuovo fusibile, hanno portato quest'ultimo a bruciarsi immediatamente.

CONSIDERAZIONI FINALI

Alla luce degli eventi tecnici che si sono verificati durante l'ultima esperienza di laboratorio, come anche delle diverse lezioni che erano state destinate a correzioni software minori sui loop di controllo e sulle costanti in questi impiegate, da individuare empiricamente in assenza di un modello che approssimasse la dinamica del sistema robuter, abbiamo valutato che sarebbe stato prioritario dedicarci nelle fasi iniziali delle sperimentazioni, dopo aver realizzato i primi collegamenti e prima di dedicarsi a verificarne il funzionamento via software, a inscatolare la parte elettronica in una configurazione che garantisse stabilità ai collegamenti e protezione dalle scariche elettrostatiche ai componenti, dato che queste ultime possono danneggiare l'elettronica in maniera non immediatamente evidente, sino al verificarsi di un guasto, così come avviene nel caso di urti meccanici ed erosione delle saldature.
Una mancanza è stata anche misurare il numero di tick per giro di ruote ed il numero di tick per metro di avanzamento del robuter su uno solo dei due encoder.
Se avessimo ripetuto la misura su entrambi gli encoder, se non altro per rilevare differenze minori di cui tenere conto nella scelta delle costanti proporzionali, avremmo potuto riconoscere, indagare e forse risolvere il problema di lettura nelle fasi iniziali del progetto.

In conclusione quindi avremmo dovuto dare maggiore priorità allo studio del sistema ed alla preparazione fisica dell'integrazione dell'elettronica del robuter.

Sviluppi futuri

Un upgrade inevitabile sarebbe quello di inserire dei sensori esterocettivi, come ad esempio un leather, così da avere per ogni direzione l'angolo e la distanza a cui si trova l'ostacolo più vicino e sapendo com'è montato avrò anche la possibilità di creare una mappa cartesiana con il sistema di riferimento del robot. Con il tempo acquisendo informazioni su ostacoli statici e dinamici si andrebbe a creare una base dati contenente la mappatura dei percorsi frequenti così da poter ottenere una occupancy grid, in cui l'ambiente viene rappresentato come una griglia bidimensionale e in cui ogni oggetto è associato un valore che indica lo stato della cella (libera/occupata).