

Università degli Studi di Salerno

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE ED
ELETTRICA E MATEMATICA APPLICATA

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA



ROVER PROJECT WORK

Gruppo

Andrea Scala - 0622702346 - a.scala31@studenti.unisa.it

Giuseppe Squitieri - 0622702339 - g.squitieri8@studenti.unisa.it

Ermanno Troisi - 0622702288 - e.troisi10@studenti.unisa.it

Anno Accademico 2024/2025

Contents

Contents	2
1 Specifiche di Alto Livello	6
2 Diagrammi dei Casi d'Uso	8
2.1 UC01 - Diagramma di Interazione Utente-Rover	8
2.2 UC02 - Diagramma di Interazione Rover-FreeRTOS	9
3 Diagrammi delle Attività	10
3.1 AD01 - Rilevamento Ostacoli	10
3.2 AD02 - Gestione della Temperatura	11
3.3 AD03 - Rotazione del Rover	12
3.4 AD04 - Movimento del Rover	13
4 Diagramma di stato	14
5 Architettura	16
5.1 Sistema Hardware	16
5.2 Schema di cablaggio Fritzing	17
5.3 Foto del sistema	18
5.4 Configurazione GPIO Rover	19
5.5 Board Slave	20
5.6 Board Master	22
6 Progettazione e implementazione di un controllore PID	24
6.1 Progettazione del Controllore PID	24
6.2 Implementazione controllore PID	28
7 Controllo motori	31
7.1 Controllo dei Motori con PWM	31
7.1.1 Configurazione del PWM	31
7.1.2 Calcolo della Frequenza PWM	32

7.1.3	Calcolo del Duty Cycle	32
7.1.4	Motivazione per la Scelta della Frequenza PWM	32
7.1.5	Introduzione di Sabertooth per il controllo	34
7.1.6	Realizzazione del sistema	34
7.1.7	Schema Finale	35
7.2	Controllo dei motori tramite UART	36
7.2.1	Configurazione dell'UART	36
7.2.2	Introduzione di Sabertooth per il controllo	36
7.2.3	Realizzazione del sistema	37
7.2.4	Schema Finale	39
7.3	Confronto granularità	39
7.3.1	Analisi della Granularità nei Metodi di Controllo PWM e UART	39
7.3.2	Confronto tra PWM e UART	41
7.4	Pro e Contro	42
7.4.1	Pro e Contro del PWM	42
7.4.2	Pro e Contro dell'UART	42
7.4.3	Motivazione per la Scelta di UART	43
8	Strutture dati e task	45
8.1	Strutture dati	45
8.1.1	Struttura stato parziale della scheda	45
8.1.2	Struttura stato globale sistema	47
8.2	Descrizione task	48
8.2.1	Task di lettura	48
8.2.2	Task di comunicazione	48
8.2.3	Task di attuazione	49
8.2.4	Task di attuazione degradata	49
8.2.5	Task di emergenza	50
8.3	Stateflow	51
8.3.1	Task di lettura	52
8.3.2	Task di comunicazione	53
8.3.3	Task di attuazione	54
8.3.4	Task di attuazione degradata	54
8.3.5	Task di emergenza	55
9	Schedulazione task	56
9.1	Rilevazione tempi	56
9.2	Task di lettura	56
9.2.1	Board 1	56

9.2.2	Board 2	59
9.2.3	Task di comunicazione	62
9.2.4	Task di attuazione	62
9.2.5	Task di attuazione degradata	64
9.2.6	Task di emergenza	64
9.2.7	Tempi finali	64
9.3	Implementazione algoritmo di schedulazione	65
9.3.1	Impostazioni RTOS	68
10	Comunicazione	69
10.1	Introduzione	69
10.2	Motivazione della Scelta di SPI	69
10.3	Confronto dei Tempi di Trasmissione	70
10.4	Calcolo del Tempo di Trasmissione	71
10.4.1	UART	71
10.4.2	SPI	71
10.5	Risultati Finali	72
10.6	Implementazione codice comunicazione	72
10.6.1	Configurazione dei parametri SPI	72
10.6.2	Sincronizzazione trasmissione	74
10.7	Serialize e Deserialize	76
10.8	Flusso Completo di Comunicazione	77
11	Utilizzo dei relè per la trasmissione dei dati	79
11.1	Relè utilizzati	79
11.2	Trasmissione comandi	80
12	Specifiche	81
12.1	Temperatura	81
12.2	Sonar	83
12.3	Interfacciamento controller e calcolo set point	85
12.3.1	comunicazione STM-ESP32	85
12.3.2	calcolo set point e funzione PID	85
13	Montaggio e modalità d'uso del rover	87
13.1	Componenti	87
13.2	Montaggio	87
13.3	Modalità operative	88
14	Conclusioni e sviluppi futuri	89

List of Figures	92
------------------------	-----------

Chapter 1

Specifiche di Alto Livello

Il **progetto Rover** è stato sviluppato per soddisfare un insieme di specifiche di alto livello che definiscono le principali funzionalità e requisiti tecnici del sistema. Di seguito viene fornita una descrizione dettagliata delle specifiche selezionate, delle scelte progettuali adottate e delle loro implementazioni.

Il **Rover** è progettato per accettare comandi esterni da un **controller**. La comunicazione avviene tramite **Bluetooth** utilizzando il protocollo I2C, che garantisce affidabilità e bassa latenza. I comandi vengono inviati in **modalità analogica**, dove l'inclinazione degli stick rappresenta velocità e direzione: un angolo neutro corrisponde a velocità zero. Questa modalità è stata scelta per garantire un controllo fluido e preciso. Inoltre, i pulsanti del controller sono configurati per eseguire funzioni specifiche, come:

- Arresto di emergenza
- Arresto completo del veicolo
- Attivazione dei LED per fornire un feedback visivo

Il **Rover** può muoversi in linea retta in avanti, sincronizzando tutti e quattro i motori tramite un algoritmo che utilizza encoder per garantire movimenti stabili e precisi. Può anche ruotare attorno al proprio centro di gravità configurando i motori per muoversi in direzioni opposte, garantendo manovrabilità in spazi ristretti.

Per quanto riguarda la sicurezza, il **Rover** utilizza tre sensori ad ultrasuoni posizionati frontalmente e lateralmente per rilevare ostacoli fino a 3 metri di distanza, coprendo un angolo di 45° su ciascun lato. I dati dei sensori vengono elaborati in tempo reale per identificare eventuali ostacoli lungo il percorso. Se un ostacolo viene rilevato entro 70 cm, il sistema attiva un arresto di emergenza.

Il **Rover** è dotato di un **sistema di monitoraggio** per garantire affidabilità e sicurezza. In caso di malfunzionamenti nei componenti critici, come schede di controllo, encoder, motori o sensori, il sistema entra in uno stato sicuro con motori disattivati. I seguenti malfunzionamenti e soglie di allarme vengono monitorati:

1. Una delle due schede è in avaria.
2. Entrambe le schede sono in avaria.
3. Uno dei due driver dei motori non risponde ai comandi.
4. Entrambi i driver dei motori non rispondono ai comandi.
5. Almeno uno degli encoder non funziona.
6. Il sistema non accetta comandi.
7. La batteria del controller è scarica.
8. I motori non rispondono ai comandi.
9. La temperatura interna supera i **90°C** per più di **10 secondi**.

Allo stesso modo, vengono monitorate le condizioni operative, come la temperatura interna. Se una delle soglie di allarme viene superata, vengono attivati allarmi visivi e il **Rover** viene posto in uno stato sicuro.

Queste specifiche di alto livello definiscono le capacità essenziali del Rover e guidano le implementazioni tecniche del sistema, garantendo un equilibrio tra funzionalità, sicurezza e usabilità. Le scelte progettuali sono state orientate al soddisfacimento dei requisiti obbligatori e all'implementazione di alcune specifiche opzionali, come il controllo tramite pulsanti e le azioni di emergenza, per migliorare l'efficienza e la robustezza del sistema.

Chapter 2

Diagrammi dei Casi d’Uso

2.1 UC01 - Diagramma di Interazione Utente-Rover

Questo diagramma rappresenta l’interazione tra l’utente e il sistema Rover, delineando il flusso di dati e comandi scambiati. L’utente comunica con il Rover utilizzando un controller, che invia comandi tramite connessione Bluetooth. Il Rover interpreta questi comandi e decide quali azioni eseguire, come muoversi, ruotare o fermarsi. Inoltre, il sistema può fornire feedback all’utente, come segnali visivi tramite LED o messaggi di stato. Questo diagramma è cruciale per comprendere il ciclo di input-output tra l’utente e il sistema, evidenziando la sinergia tra hardware e software nel processo decisionale.

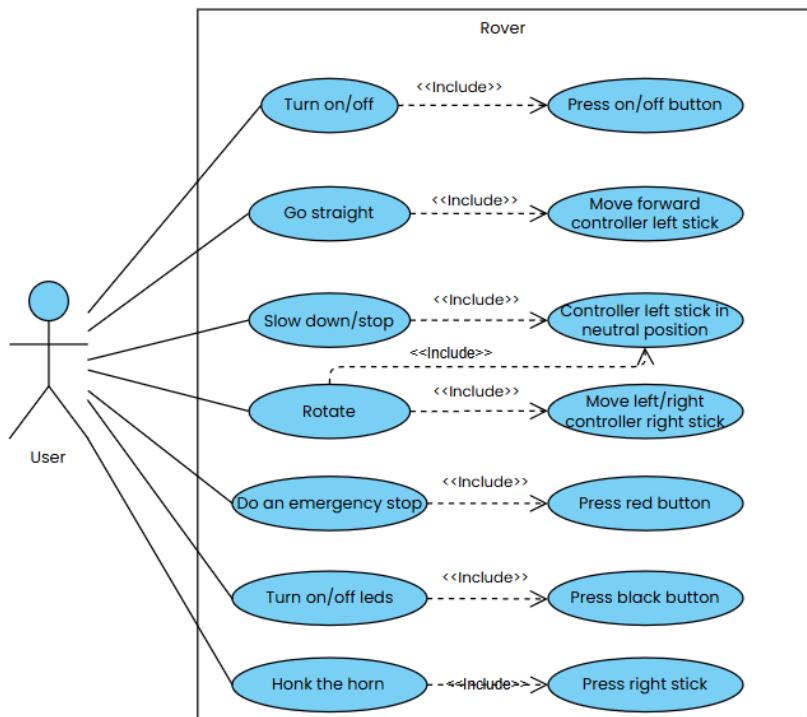


Figure 2.1: Diagramma dei casi d’uso per l’interazione utente-rover

2.2 UC02 - Diagramma di Interazione Rover-FreeRTOS

Questo diagramma rappresenta l'interazione tra il Rover e il sistema operativo in tempo reale FreeRTOS, enfatizzando la gestione dei task. FreeRTOS è responsabile della pianificazione e sincronizzazione dei processi, come il controllo dei motori, l'acquisizione dei dati dai sensori e l'elaborazione dei segnali. Ogni task ha una priorità, e il sistema garantisce che i task critici, come il rilevamento ostacoli, vengano eseguiti per primi. Il diagramma mostra anche come i task interagiscono tra loro, ad esempio tramite code o mutex, per mantenere l'integrità dei dati e la funzionalità in tempo reale.

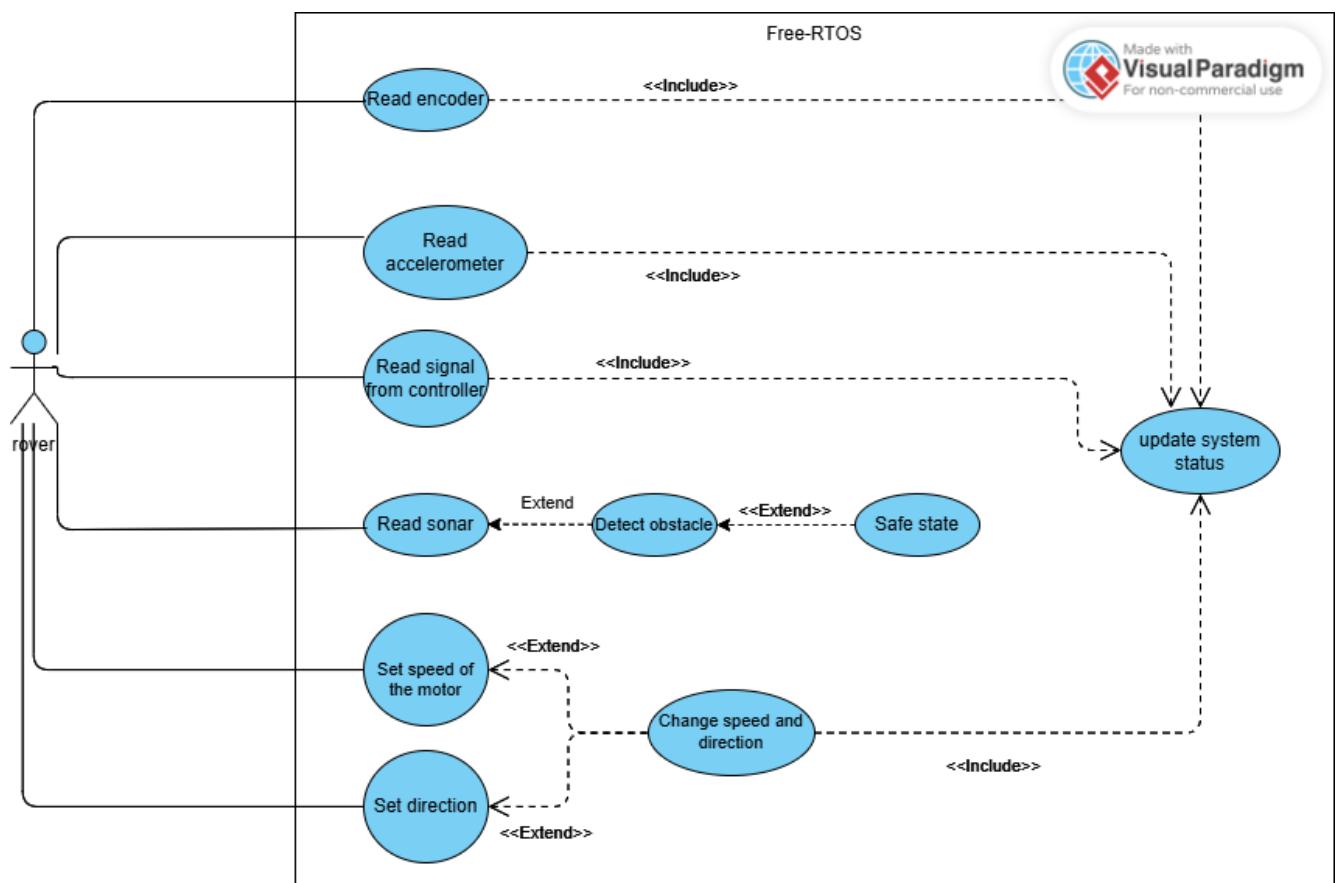


Figure 2.2: Diagramma dei casi d'uso per l'interazione Rover-FreeRTOS

Chapter 3

Diagrammi delle Attività

3.1 AD01 - Rilevamento Ostacoli

Questo diagramma illustra il processo mediante il quale il Rover rileva gli ostacoli. I sensori a ultrasuoni misurano la distanza dagli oggetti di fronte al Rover e inviano i dati a una scheda di controllo. Il sistema analizza i dati per determinare se gli ostacoli si trovano a una distanza critica. Se viene rilevato un ostacolo, il sistema decide l'azione appropriata, come fermarsi o cambiare direzione. Il processo prevede cicli continui di lettura dei dati, elaborazione e azioni correttive, garantendo una navigazione sicura anche in ambienti complessi.

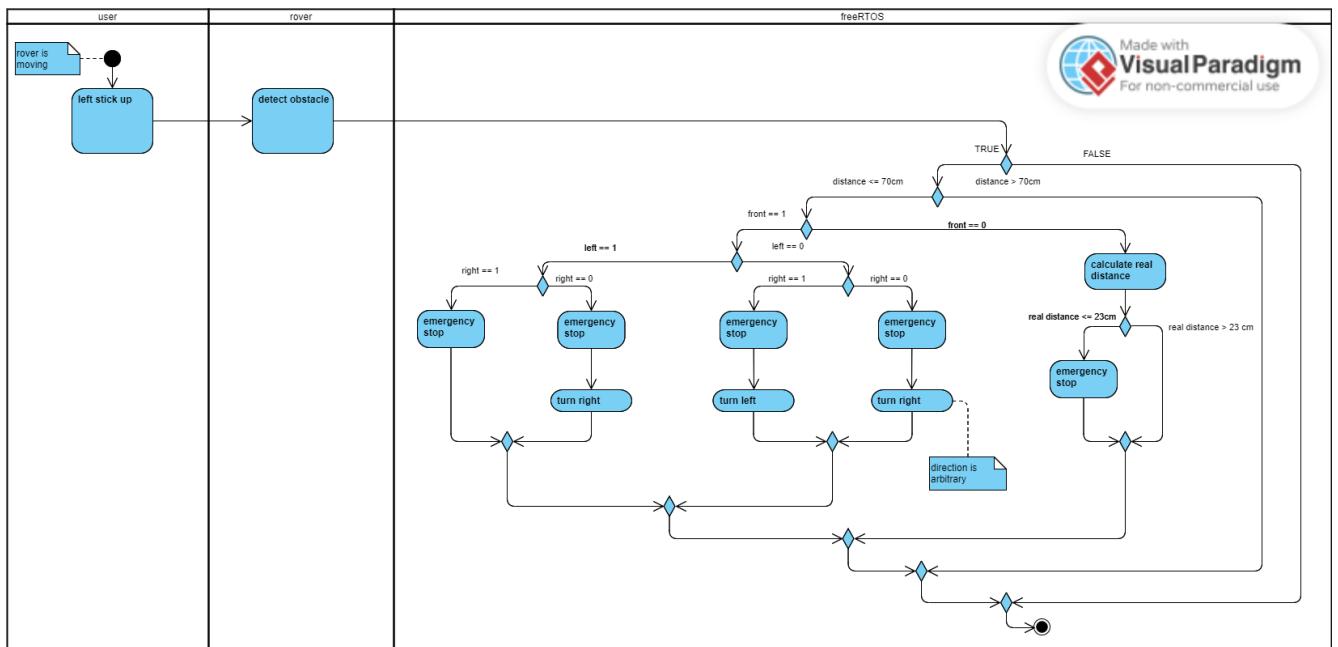


Figure 3.1: Diagramma delle attività per il rilevamento ostacoli

3.2 AD02 - Gestione della Temperatura

Questo diagramma descrive il sistema di monitoraggio e controllo della temperatura del Rover. I sensori leggono la temperatura e inviano i dati al sistema di controllo. Se la temperatura supera una soglia critica, il sistema attiva meccanismi di protezione, come la riduzione della velocità dei motori o lo spegnimento dei componenti non essenziali. In caso di surriscaldamento persistente, possono essere generati avvisi per notificare l'utente. Questo diagramma è fondamentale per garantire l'affidabilità e la sicurezza del Rover, proteggendo i componenti hardware da eventuali danni causati da temperature elevate.

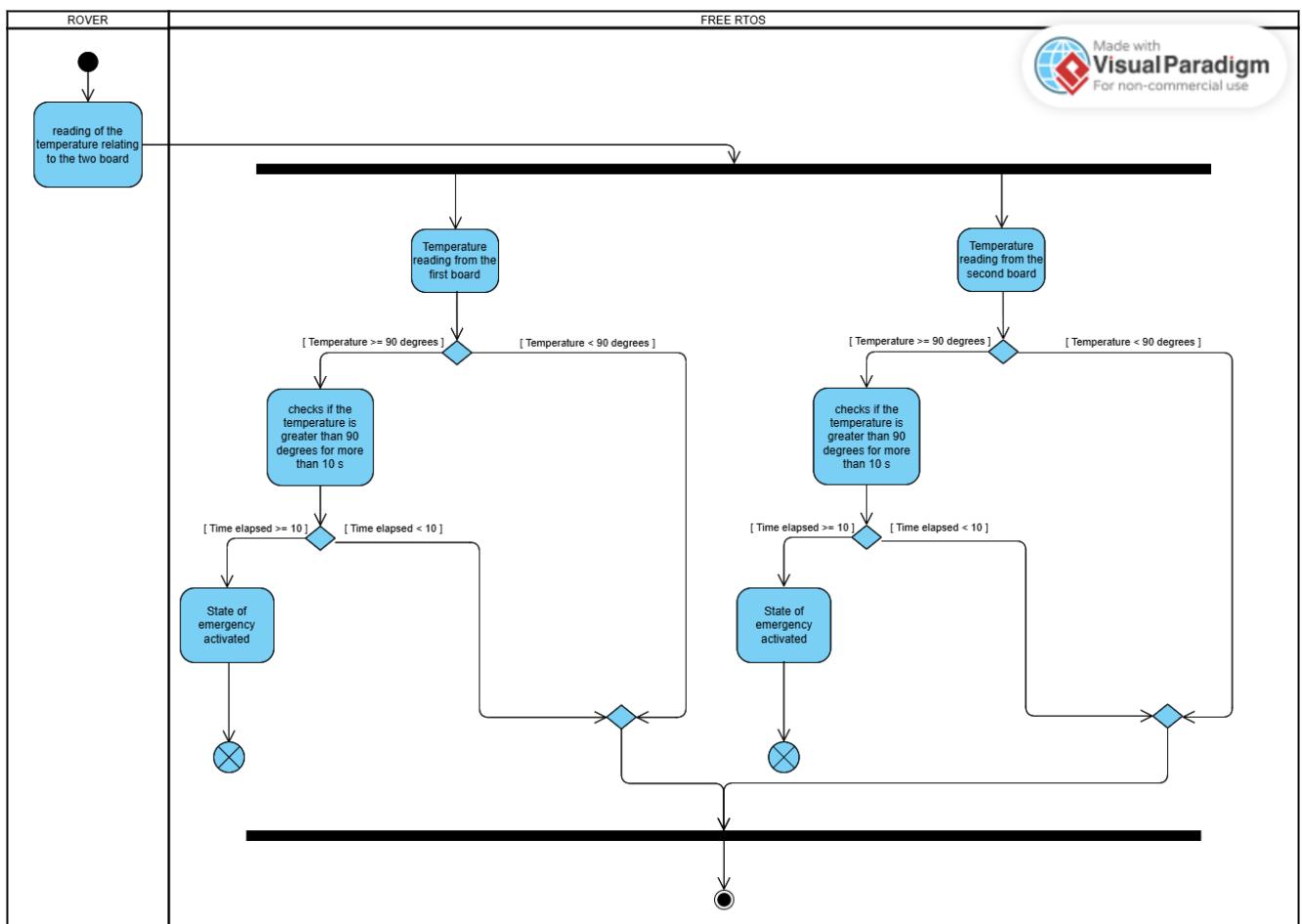


Figure 3.2: Diagramma delle attività per la gestione della temperatura

3.3 AD03 - Rotazione del Rover

Questo diagramma mostra il processo mediante il quale il Rover ruota per cambiare direzione. Il sistema calcola l'angolo di rotazione richiesto in base ai comandi dell'utente o agli ostacoli rilevati. I driver dei motori ricevono istruzioni specifiche per controllare i motori, facendoli muovere in direzioni opposte per facilitare la rotazione sul posto. Una volta completata la rotazione, il sistema verifica l'angolo effettivamente raggiunto tramite gli encoder ed esegue eventuali regolazioni necessarie. Questo diagramma evidenzia l'interazione tra sensori, motori e logica di controllo per ottenere movimenti precisi.

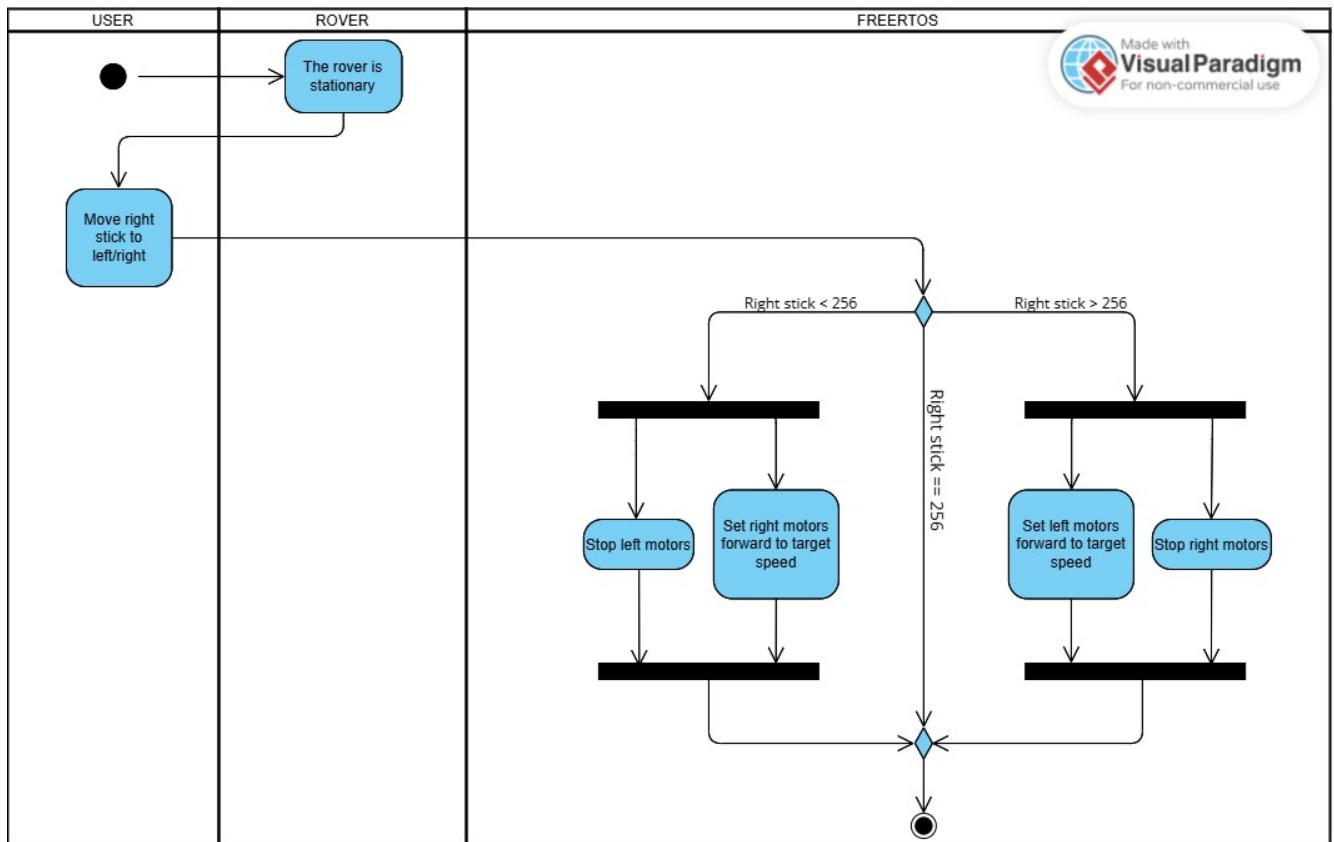


Figure 3.3: Diagramma delle attività per la rotazione del Rover

3.4 AD04 - Movimento del Rover

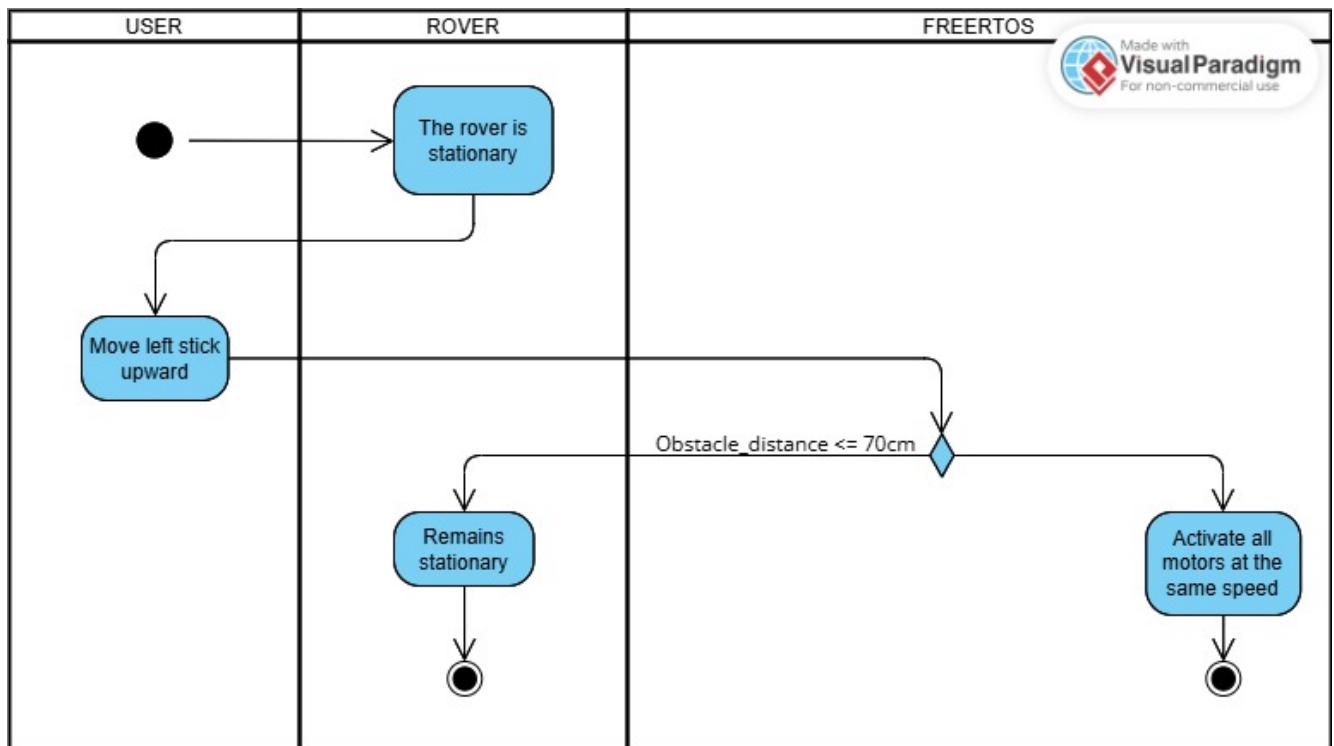


Figure 3.4: Diagramma delle attività per il movimento del Rover

Chapter 4

Diagramma di stato

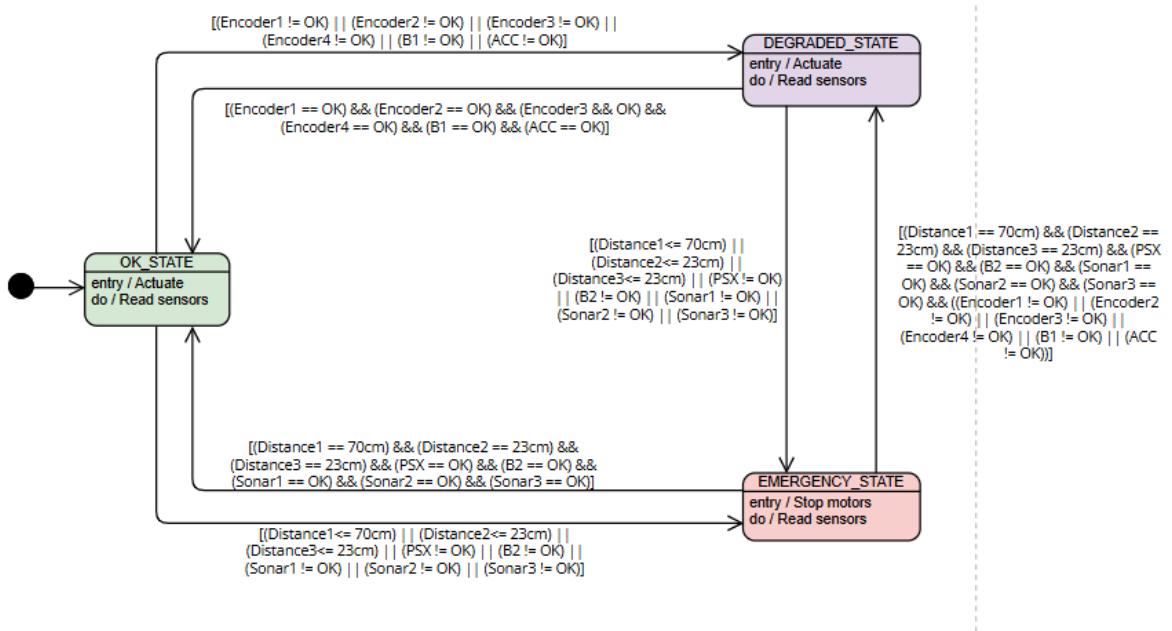


Figure 4.1: Diagramma di stato del sistema

Nella figura è rappresentato il diagramma di stato del sistema, che illustra i vari stati in cui il rover può trovarsi e le transizioni che consentono il passaggio da uno stato all'altro. Di seguito sono descritti i principali stati operativi:

1. OK_STATE

Questo è lo stato iniziale, in cui tutte le componenti del rover funzionano correttamente, consentendo il normale svolgimento delle attività. Quando il sistema entra in questo stato, il rover esegue i comandi impartiti dall'utente e successivamente acquisisce i dati dai sensori per aggiornarne lo stato.

2. DEGRADED_STATE

Il passaggio a questo stato si verifica in caso di malfunzionamenti che riducono le capacità operative del rover. Le cause principali includono:

- Malfunzionamento della board 1 (slave);
- Guasti in uno o più encoder dei motori;
- Malfunzionamento dell'accelerometro.

In questo stato, il rover continua a funzionare ma con potenza ridotta, limitando la velocità massima a 20 g/min. Questa riduzione è necessaria per garantire un minimo livello di controllo nonostante la perdita di alcune funzionalità.

3. EMERGENCY STATE

Questo stato viene attivato in caso di condizioni critiche che compromettono la sicurezza o il funzionamento del sistema. Le principali cause sono:

- Malfunzionamento della board 2 (master);
- Malfunzionamento della board 1 (slave);
- Guasto di uno o più sonar;
- Malfunzionamento del controller;
- Guasto di uno o più encoder;

In EMERGENCY_STATE, il rover si blocca completamente fino alla risoluzione del problema. Durante questo stato, i motori vengono disattivati per evitare ulteriori danni o situazioni pericolose, e il sistema rimane in stand-by fino a quando non si torna a una condizione stabile.

Chapter 5

Architettura

5.1 Sistema Hardware

L'intero sistema è costituito da due schede STM32G474RE che comunicano tra loro tramite il protocollo USART. La scheda 1 è collegata ai quattro encoder dei motori e a due LED indicatori, mentre la scheda 2 è collegata a tre sensori a ultrasuoni per il rilevamento degli ostacoli davanti al Rover, un accelerometro e un controller che comunica via Bluetooth utilizzando il protocollo I2C. Entrambe le schede sono collegate a due driver Sabertooth per il controllo dei motori.

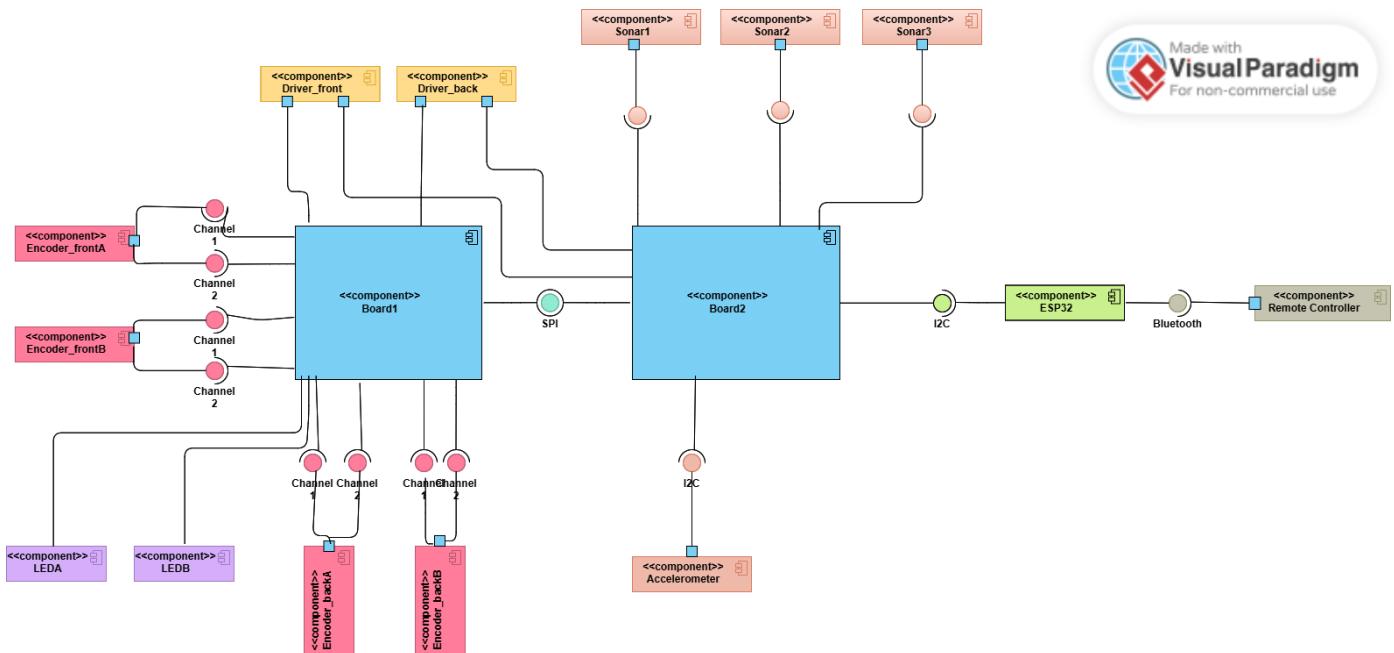


Figure 5.1: Diagramma dei componenti

5.2 Schema di cablaggio Fritzing

In Figura è visibile lo schema di cablaggio, sviluppato con Fritzing. Sono presenti tutti i dispositivi del rover, inclusi relè e convertitore buck.

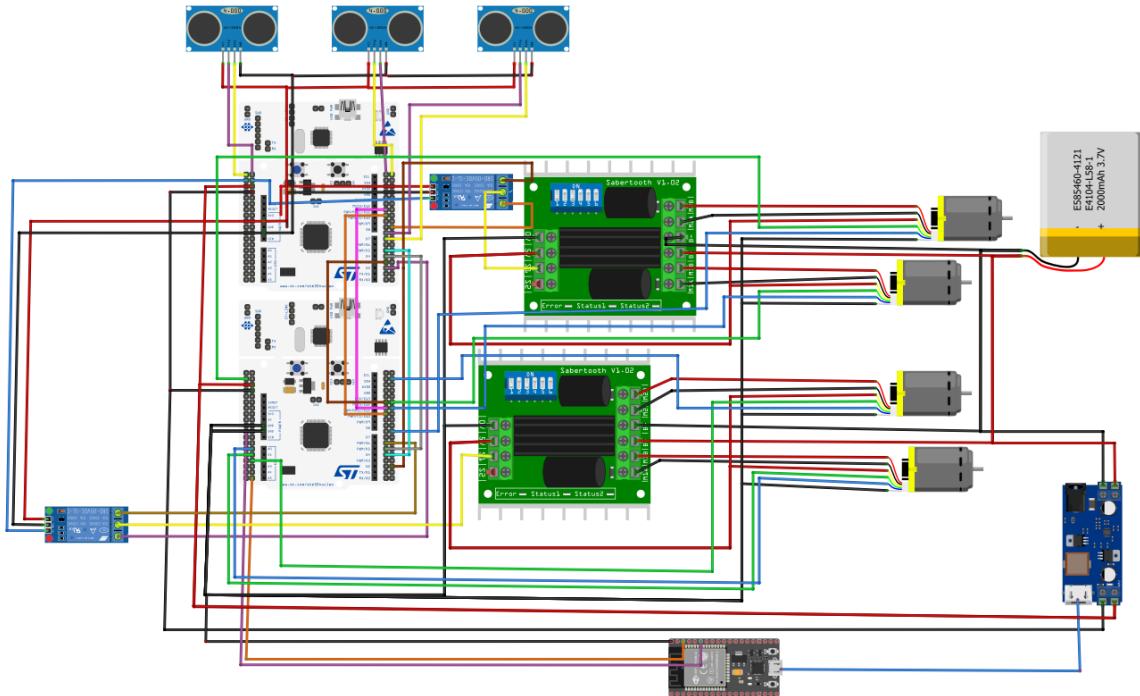


Figure 5.2: Schema di cablaggio Fritzing

5.3 Foto del sistema

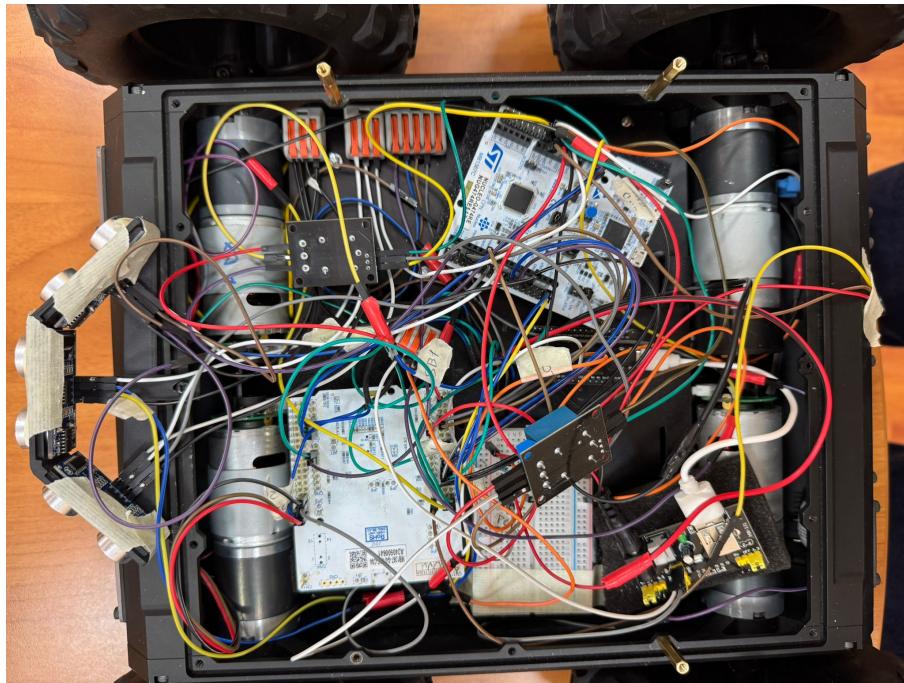


Figure 5.3: Parte superiore

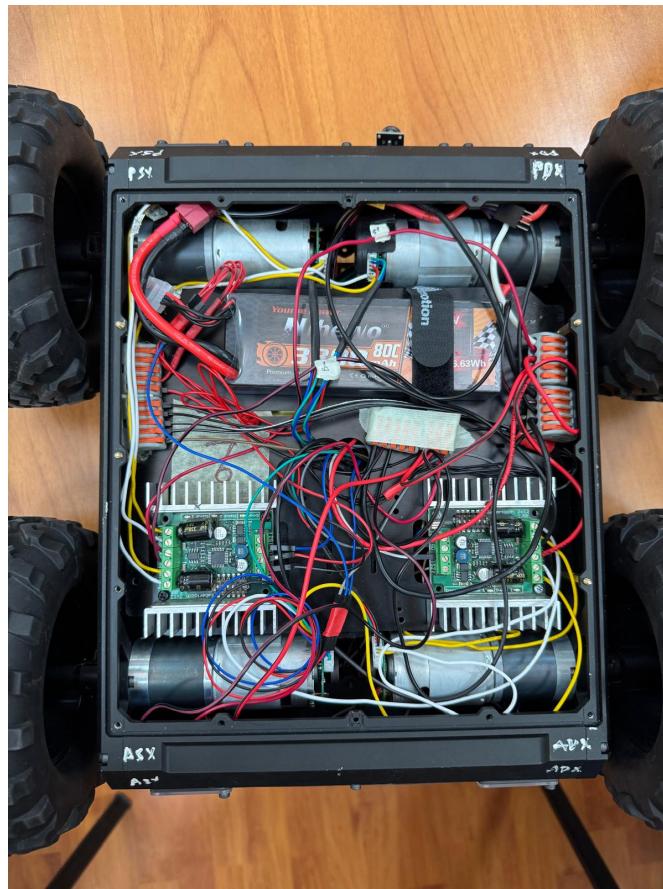


Figure 5.4: Parte inferiore

5.4 Configurazione GPIO Rover

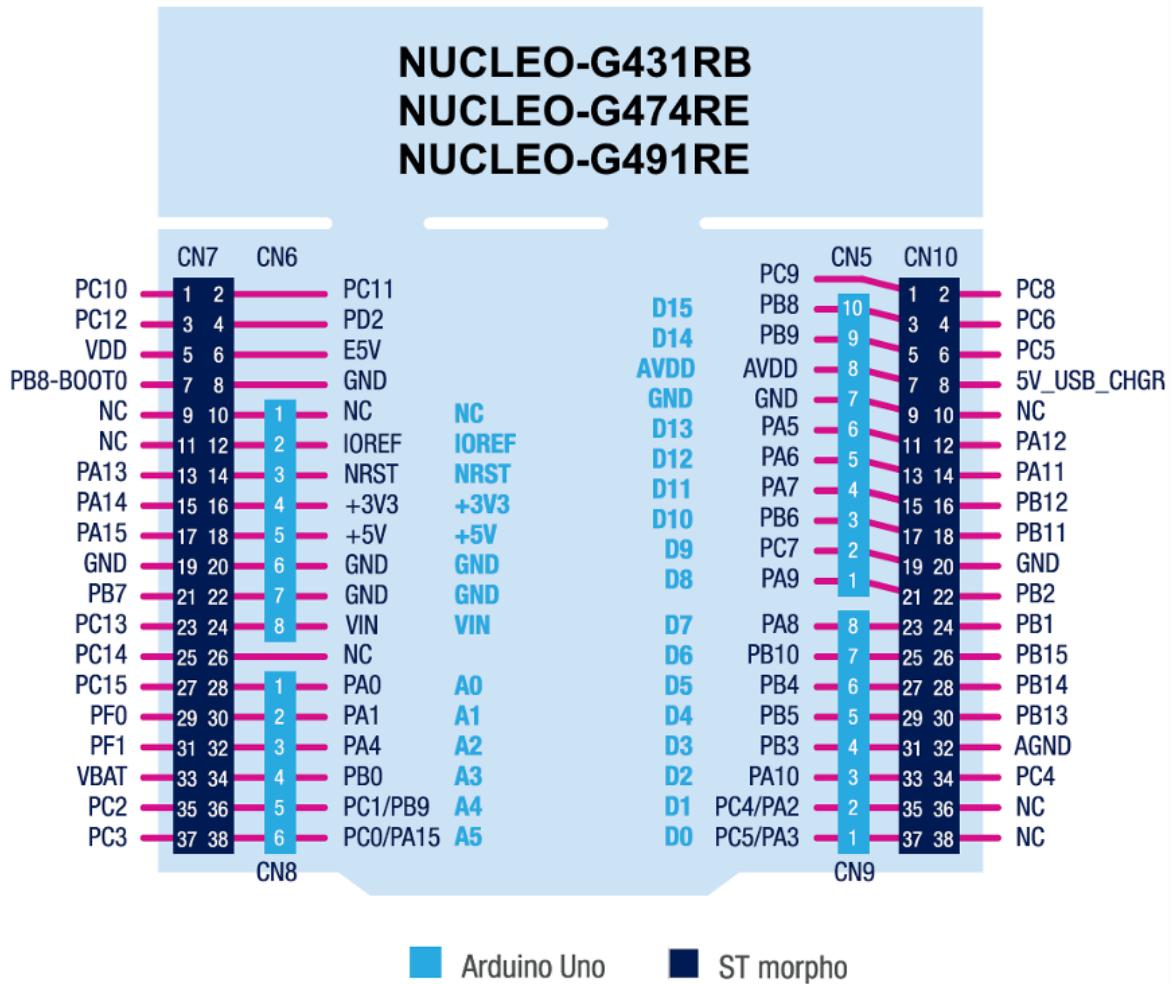


Figure 5.5: pinout della scheda STM32G474RE

5.5 Board Slave

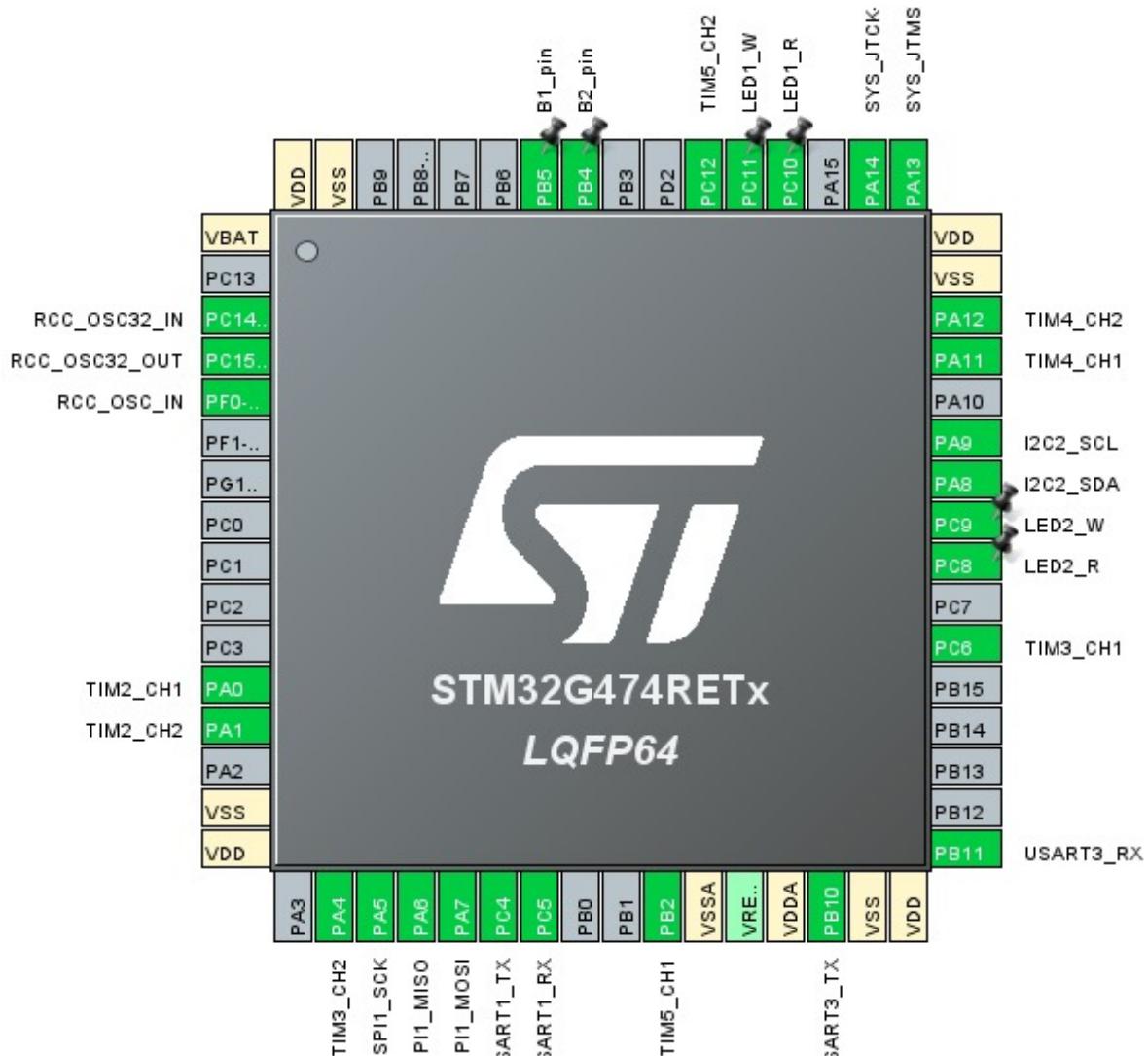


Figure 5.6: Pin B1 slave

Pin	Periferica	Utilizzo
PC14	RCC_OSC32_IN	-
PC15	RCC_OSC32_OUT	-
PF0	RCC_OSC_IN	-
PA13	SYS_JTMS-SWDIO	-
PA14	SYS_JTCK- SWCLK	-
PA8	I2C2_SCL	-
PA9	I2C2_SDA	-
PA0	TIM2_CH1	ENCODER
PA1	TIM2_CH1	ENCODER
PC6	TIM3_CH1	ENCODER
PA4	TIM3_CH2	ENCODER
PA11	TIM4_CH1	ENCODER
PA12	TIM4_CH2	ENCODER
PB2	TIM5_CH1	ENCODER
PC12	TIM5_CH2	ENCODER
PC4	USART1_TX	Controllo motori
PC5	USART1_RX	-
PB10	USART3_TX	Controllo motori
PB11	USART3_RX	-
PB4	B2_pin	Sincronizzazione comunicazione
PB5	B1_pin	Sincronizzazione comunicazione
PA5	SPI1_SCK	Comunicazione
PA6	SPI1_MISO	Comunicazione
PA7	SPI1_MOSI	Comunicazione
PC8	LED2_R	Controllo led 2 rosso
PC9	LED2_W	Controllo led 2 bianco
PC10	LED1_R	Controllo led 1 rosso
PC11	LED1_W	Controllo led 2 rosso

Table 5.1: Mappatura dei pin, periferiche e utilizzi Slave

5.6 Board Master

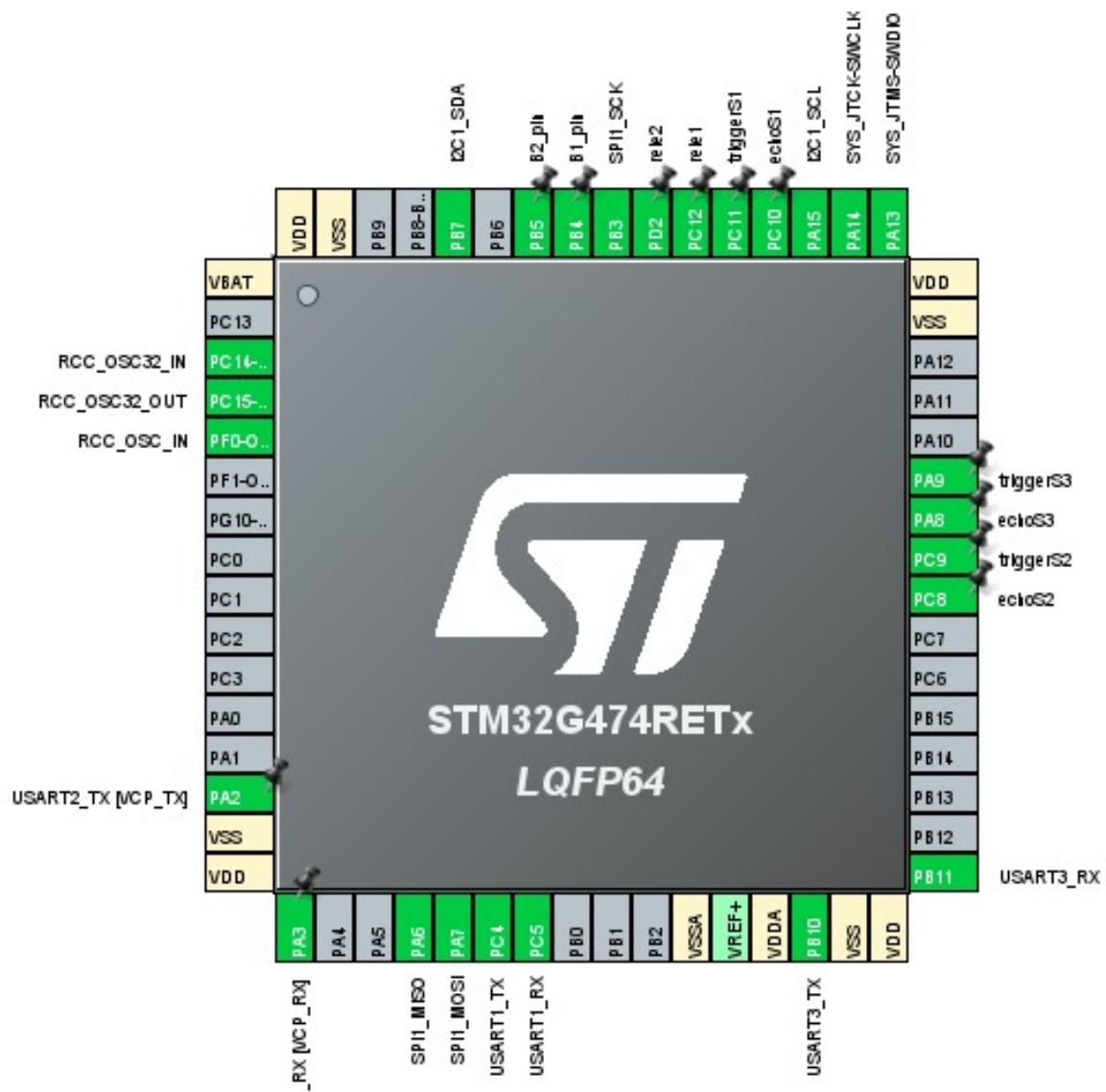


Figure 5.7: Pin B2 master

Pin	Periferica	Utilizzo
PC14	RCC_OSC32_IN	-
PC15	RCC_OSC32_OUT	-
PF0	RCC_OSC_IN	-
PC4	USART1_TX	Controllo motori
PC5	USART1_RX	-
PA2	USART2_TX	-
PA3	USART2_RX	-
PB10	USART3_TX	Controllo motori
PB11	USART3_RX	-
PA6	SPI1_MISO	Comunicazione
PA7	SPI1_MOSI	Comunicazione
PB3	SPI1_SCK	Comunicazione
PC10	echoS1	Echo sonar1
PC11	triggerS1	Trigger sonar1
PC8	echoS2	Echo sonar2
PC9	triggerS2	Trigger sonar2
PA8	echoS3	Echo sonar3
PA9	triggerS3	Trigger sonar3
PA15	I2C1_SCL	Controller
PB7	I2C1_SDA	Controller
PB4	B2_pin	Sincronizzazione comunicazione
PB5	B1_pin	Sincronizzazione comunicazione
PC12	rele1	Rele1 stato degradato
PD2	rele2	Rele2 stato degradato
PA13	SYS_JTMS-SWDIO	-
PA14	SYS_JTCK-SWCLK	-

Table 5.2: Mappatura dei pin, periferiche e utilizzi Master

I pin che non sono stati riportati nelle tabelle precedenti non sono stati utilizzati.

Chapter 6

Progettazione e implementazione di un controllore PID

In questo capitolo si analizzerà la progettazione di un PID, esaminando in modo generale i passaggi eseguiti, senza entrare nel dettaglio di ogni singolo passo effettuato, per poi arrivare a una conclusione finale.

6.1 Progettazione del Controllore PID

Nella progettazione del controllore PID, il punto di partenza è stata la funzione di trasferimento empirica del sistema, data da:

$$G(s) = \frac{650}{s + 45}$$

Questa funzione ha rappresentato il comportamento dinamico del sistema in esame, permettendo di effettuare diverse prove per ottimizzare i parametri del controllore PID.

Risposta a ciclo aperto

L'analisi della funzione di trasferimento ha evidenziato che il sistema presenta una dinamica relativamente lenta, con un guadagno statico elevato. La risposta a ciclo aperto ha mostrato un comportamento stabile, ma con un errore a regime elevatissimo.

K_p = 1

Il sistema è stato inizialmente testato con un semplice controllore proporzionale con $K_p = 1$. Questo ha migliorato la risposta del sistema, riducendo leggermente il tempo di risposta. Tuttavia, il principale svantaggio riscontrato è stato il mantenimento di un errore a regime significativo, rendendo necessario l'introduzione di un termine integrale.

K_p = 1, K_i = 1

Introducendo il termine integrale con $K_i = 1$, la risposta ha mostrato una maggiore velocità di convergenza, con una riduzione dell'errore a regime. Tuttavia, si è notato un leggero incremento dell'overshoot, che avrebbe potuto causare instabilità in condizioni di variazione del carico.

K_p = 0.1, K_i = 1

Riducendo K_p a 0.1 e mantenendo $K_i = 1$, il sistema ha evidenziato un comportamento più stabile, riducendo le oscillazioni indesiderate. Sebbene l'errore a regime fosse nullo, il tempo di risposta risultava più lungo rispetto alla configurazione precedente.

K_p = 0.001, K_i = 1

Successivamente, con $K_p = 0.001$ e $K_i = 1$, si è ottenuta una risposta più lenta, ma con errore a regime eliminato. Il vantaggio principale di questa configurazione è stata la maggiore stabilità e l'assenza di overshoot, sebbene il tempo di assestamento fosse più elevato.

K_p = 0.001, K_i = 0.5

Riducendo K_i a 0.5, si è osservato un ulteriore aumento del tempo di risposta, ma il sistema ha mantenuto stabilità senza overshoot eccessivo. Tuttavia, l'errore a regime risultava più sensibile alle perturbazioni, suggerendo che un valore di K_i leggermente più alto potesse essere una scelta migliore.

K_p = 0.001, K_i = 0.9

Con $K_i = 0.9$, si è trovato un buon compromesso tra stabilità e tempo di assestamento. La risposta del sistema ha mostrato una riduzione significativa dell'errore a regime senza introdurre oscillazioni indesiderate. Questa configurazione si è rivelata la più efficace nel mantenere l'equilibrio tra prestazioni e robustezza del sistema.

Introduzione del termine derivativo (K_d)

È stato testato l'aggiunta del termine derivativo K_d per migliorare ulteriormente la risposta del sistema. Tuttavia, i risultati hanno mostrato che K_d non apportava benefici significativi, poiché introduceva sensibilità al rumore e non migliorava in modo rilevante le prestazioni transitorie. Sebbene il sistema risultasse più reattivo, appariva anche più instabile in presenza di disturbi. Nell'immagine successiva (7.1) si può verificare, tramite un grafico, la risposta a gradino a ciclo chiuso di tutti i parametri testati per lo sviluppo.

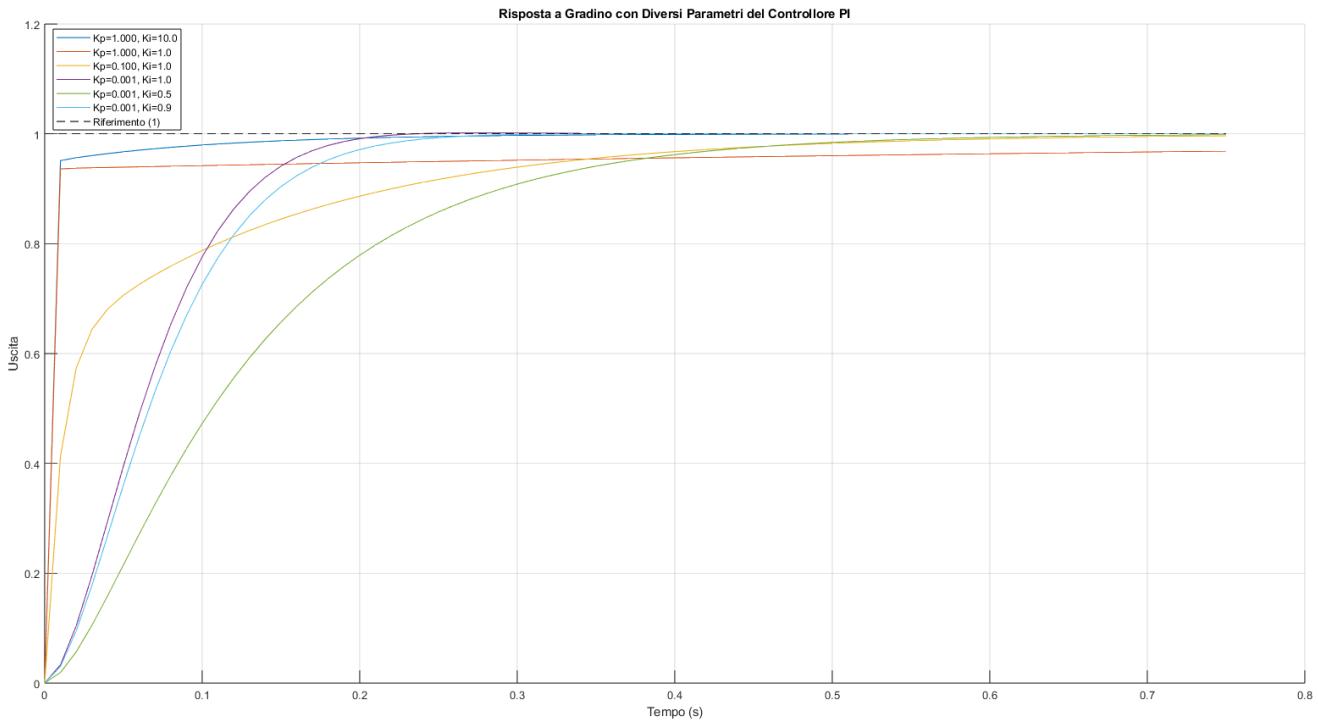


Figure 6.1: Risposta a gardino complessiva

Scelta finale: $K_p = 0.001$, $K_i = 0.9$

Dopo aver analizzato le diverse prove, è stata scelta la configurazione finale di $K_p = 0.001$ e $K_i = 0.9$, che ha garantito:

- Un errore a regime nullo.
- Una risposta stabile e senza sovraelongazioni eccessive.
- Un buon compromesso tra tempo di assestamento e stabilità.

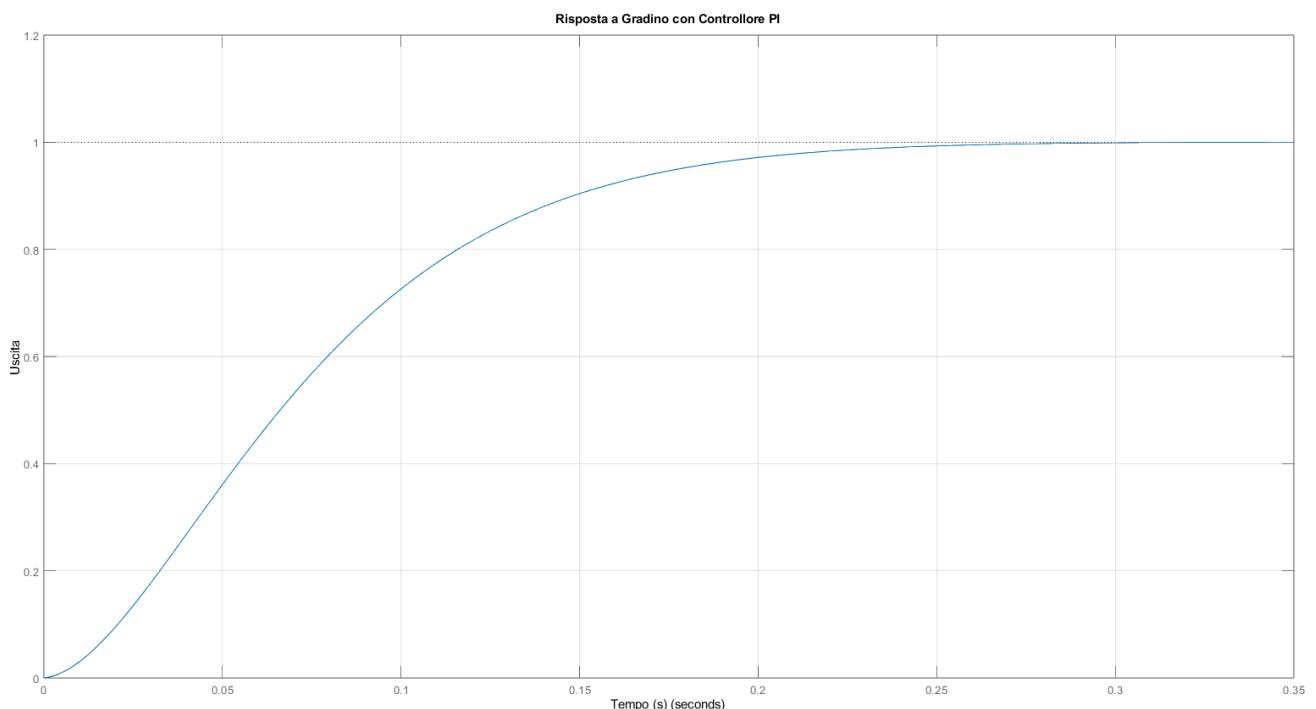


Figure 6.2: Risposta a ciclo chiuso con $k_p=0.001$ e $ki=0.9$

La funzione di trasferimento complessiva del sistema finale è:

$$T(s) = \frac{0.65s + 585}{s^2 + 45.65s + 585}$$

6.2 Implementazione controllore PID

La funzione `PID_Init` serve per inizializzare la struttura `PID_Controller`. Essa accetta i seguenti parametri: K_p , K_i , K_d che rappresentano rispettivamente i guadagni del controller PID, e inizializza i valori di altri parametri:

- **ITerm**: È il termine integrale del PID, che viene inizializzato a zero.
- **lastInput**: Memorizza l'ultimo valore di input (in questo caso la velocità angolare) ed è impostato inizialmente a zero.
- **lastTick**: Indica l'ultimo tempo di riferimento (in ticks), anch'esso inizializzato a zero.
- **lastAngle**: Memorizza l'ultimo angolo calcolato, inizializzato anch'esso a zero.

Questa inizializzazione è fondamentale per il corretto funzionamento del controller PID durante i cicli successivi.

Calcolo del PID

La funzione `PID_Compute` calcola l'uscita del controller PID in base a variabili come il valore del contatore `counterValue`, il tempo `ticks`, il setpoint desiderato `setpoint`, la risoluzione del sensore `resolution` e il motore. I passaggi principali del calcolo sono i seguenti:

1. Calcolo dell'Angolo

L'angolo attuale del motore viene calcolato in base al valore accumulato nel contatore. La formula per calcolare l'angolo è la seguente:

$$\text{angle} = \frac{360}{\text{resolution}} \times \text{counterValue}$$

Dove:

- `counterValue` è il valore accumulato nel contatore per il motore, e rappresenta il numero di ticks registrati dal timer associato al motore.
- `resolution` è il numero di passi per giro del sensore.

Nel nostro caso, la risoluzione è impostata a 2448.

2. Calcolo del DeltaTime

La variabile `deltaTime` calcola il tempo trascorso dall'ultimo ciclo PID, espresso in secondi. La formula utilizzata è:

$$\text{deltaTime} = \frac{\text{ticks} - \text{pid-}\zeta\text{lastTick}}{1000.0}$$

Dove ticks rappresenta il tempo corrente in millisecondi, e pid- ζ lastTick è il tempo di riferimento memorizzato nel ciclo precedente.

3. Calcolo della Velocità Angolare

La velocità angolare viene calcolata come la variazione dell'angolo divisa per il tempo trascorso, espressa in giri al minuto (RPM). La formula per calcolarla è la seguente:

$$\text{angularSpeed} = \frac{60 \times (\text{angle} - \text{pid-}\zeta\text{lastAngle})}{\text{deltaTime} \times 360.0}$$

Questa formula converte la variazione dell'angolo (in gradi) in velocità angolare (in RPM).

Caso Setpoint Negativo

Quando il setpoint è negativo, il motore deve girare nella direzione opposta. In questo caso, viene calcolato l'errore tra il setpoint e la velocità angolare:

$$\text{error} = \text{setpoint} - \text{angularSpeed}$$

Successivamente, viene aggiornato il termine integrale $ITerm$, che viene accumulato nel tempo moltiplicando l'errore per il guadagno integrale K_i :

$$ITerm+ = K_i \times \text{error}$$

Il termine integrale viene limitato ai valori di potenza minima (powMin) e massima (powMax), che definiscono i limiti di potenza del motore. Se il valore di $ITerm$ supera powMin o powMax, viene "clippato" al limite:

$$\text{if } ITerm > \text{powMin} \quad ITerm = \text{powMin}$$

$$\text{else if } ITerm < \text{powMax} \quad ITerm = \text{powMax}$$

Successivamente, viene calcolato il termine derivativo $dInput$ come la differenza tra la velocità angolare attuale e quella precedente:

$$dInput = \text{angularSpeed} - \text{pid-}\zeta\text{lastInput}$$

L'uscita del PID viene quindi calcolata come la somma dei tre termini:

$$\text{output} = K_p \times \text{error} + ITerm - K_d \times dInput$$

Infine, l'output viene limitato ai valori di potenza minima e massima.

Caso Setpoint Positivo o Uguale a Zero

Quando il setpoint è positivo o uguale a zero (motore deve girare nella direzione desiderata), la logica è simile a quella del caso precedente. In questo caso, i limiti di potenza sono definiti da valori differenti:

$$\text{powMin} = 64.0 \quad \text{e} \quad \text{powMax} = 127.0$$

I calcoli per l'errore, il termine integrale e il termine derivativo sono gli stessi, ma l'output viene limitato ai nuovi valori di potenza powMin e powMax.

Aggiornamento dello Stato

Al termine del ciclo PID, vengono aggiornati i seguenti valori:

- **lastTick**: Memorizza il tempo corrente in ticks.
- **lastAngle**: Memorizza l'angolo calcolato.
- **lastInput**: Memorizza la velocità angolare corrente.

Questi aggiornamenti sono necessari per il calcolo dell'errore nel ciclo successivo e per garantire un controllo preciso.

Meccanismi di Controllo

Gestione dell'Anti-Windup nel Codice

L'anti-windup è una tecnica per prevenire l'accumulo eccessivo dell'errore integrale quando il setpoint è lontano dal valore reale. Nel nostro codice, l'anti-windup viene gestito limitando il valore del termine integrale *ITerm* a un intervallo definito da powMin e powMax.

- Se *ITerm* supera powMin, viene "clippato" a powMin.
- Se *ITerm* è inferiore a powMax, viene "clippato" a powMax.

Questa tecnica impedisce che il termine integrale cresca indefinitamente, evitando il fenomeno del "windup" che potrebbe causare un comportamento instabile o un overshoot eccessivo.

Chapter 7

Controllo motori

Per il controllo dei motori sono state sperimentate due tecniche:

- comunicazione seriale **UART**.
- modulazione **PWM**.

Nel seguito verranno analizzate le scelte effettuate per il controllo dei motori e verranno spiegate le motivazioni che hanno portato alla preferenza di una soluzione rispetto all'altra.

7.1 Controllo dei Motori con PWM

Nel nostro progetto, una delle tecniche sperimentate per il controllo dei motori è la *Modulazione a Larghezza di Impulso* (PWM). Questa tecnica consente di regolare la velocità dei motori variando il ciclo di lavoro (duty cycle) del segnale PWM, ovvero la durata degli impulsi rispetto ai periodi di pausa. Modificando il duty cycle, è possibile regolare con precisione la potenza fornita al motore, aumentando o diminuendo la velocità di rotazione senza generare sprechi significativi di energia. Un ulteriore vantaggio del PWM è la sua granularità nella regolazione della velocità, che permette un controllo molto fine.

7.1.1 Configurazione del PWM

Per implementare il controllo PWM, abbiamo utilizzato il Timer 1 nell'ambiente di sviluppo STM32. Questo timer dispone di 6 canali, dei quali i primi 4 sono stati configurati per generare i segnali PWM necessari per controllare ciascun motore.

Il Timer 1 è collegato alla linea APB1 del bus delle periferiche, e per la generazione del segnale PWM è stato scelto l'oscillatore interno ad alta velocità (HSI), che fornisce un clock di base a 16 MHz. Questo valore rimane invariato durante l'uso del PWM, mentre la frequenza del segnale può essere regolata modificando il *counter period* e il *prescaler*.

7.1.2 Calcolo della Frequenza PWM

La frequenza del segnale PWM è determinata dalla seguente formula:

$$f_{PWM} = \frac{f_{clk}}{(ARR + 1) \times (PSC + 1)}$$

Dove:

- f_{PWM} è la frequenza del segnale PWM,
- f_{clk} è la frequenza del clock di base (16 MHz),
- ARR è il valore del **counter period**,
- PSC è il valore del **prescaler**.

Nel nostro caso, con $f_{clk} = 16$ MHz, $ARR = 799$, e $PSC = 0$, la frequenza PWM è calcolata come:

$$f_{PWM} = \frac{16 \times 10^6}{(799 + 1) \times (0 + 1)} = \frac{16 \times 10^6}{800} = 20,000 \text{ Hz}$$

Pertanto, la frequenza PWM impostata è 20 kHz.

7.1.3 Calcolo del Duty Cycle

Successivamente, il valore del **CCR** per ottenere un duty cycle specifico può essere calcolato usando la seguente formula:

$$CCR = \frac{\text{Duty Cycle}(\%) \times (ARR + 1)}{100}$$

Per un duty cycle del 50%, sostituendo i valori si ottiene:

$$CCR = \frac{50 \times (799 + 1)}{100} = \frac{50 \times 800}{100} = 400$$

Pertanto, impostando $CCR = 400$, otteniamo un duty cycle del 50% con una frequenza PWM di 20 kHz. Questi valori sono ottenuti senza l'uso della Sabertooth, che sarà discussa nei paragrafi successivi. La Sabertooth, infatti, prevede un range di tensione che va da 1.875 V a 3.125 V quando configurata per ricevere segnali pwm (modalità analogica con precisione 4x). Di conseguenza, è stata effettuata un'analisi per adeguare questi valori alle specifiche esigenze di controllo, che sarà trattata più nel dettaglio in seguito.

7.1.4 Motivazione per la Scelta della Frequenza PWM

La frequenza PWM è stata scelta tenendo conto della necessità di ridurre al minimo la componente ad alta frequenza nel segnale. Per raggiungere questo obiettivo, è stato utilizzato un filtro

passa basso per attenuare le alte frequenze del segnale PWM e ottenere una tensione continua (DC) stabile.

Abbiamo utilizzato una resistenza da $10\text{ k}\Omega$ e un condensatore da $0,1\mu\text{F}$ per costruire il filtro.

Calcolo della Frequenza di Taglio del Filtro Passa Bassa

La frequenza di taglio del filtro passa basso RC è determinata dalla seguente formula:

$$f_c = \frac{1}{2\pi RC}$$

Dove:

- f_c è la frequenza di taglio in Hz,
- R è la resistenza in ohm (Ω),
- C è la capacità in farad (F).

Nel nostro caso, con $R = 10\text{ k}\Omega = 10,000\Omega$ e $C = 0,1\mu\text{F} = 0,1 \times 10^{-6}\text{F}$, calcoliamo la frequenza di taglio:

$$f_c = \frac{1}{2\pi \times 10,000 \times 0,1 \times 10^{-6}} = \frac{1}{2\pi \times 10^{-2}} = \frac{1}{0.0628} \approx 15,9\text{ Hz}$$

La frequenza di taglio del filtro risulta quindi essere circa 15,9 Hz. Questo significa che il filtro attenuerà tutte le frequenze superiori a 15,9 Hz, mentre lascerà passare quelle inferiori.

Comportamento del Filtro Passa Bassa

Il filtro passa basso ha la funzione di attenuare le frequenze alte e lasciare passare quelle basse. Nel nostro circuito:

- Le frequenze inferiori a 15,9 Hz passeranno senza essere attenuate.
- Le frequenze superiori a 15,9 Hz verranno attenuate.

Poiché il segnale PWM ha una frequenza molto più alta (20 kHz), il filtro ridurrà quasi completamente la componente ad alta frequenza, lasciando passare la media della modulazione, che si tradurrà in una tensione continua stabile.

Scelta della Frequenza PWM

Per ottenere un buon effetto di smussatura con il filtro passa basso, è fondamentale che la frequenza del segnale PWM sia molto più alta della frequenza di taglio del filtro. Poiché la frequenza di taglio è di 15,9 Hz, abbiamo scelto una frequenza PWM di 20 kHz, che è circa

1.250 volte maggiore della frequenza di taglio. Questa scelta garantisce che il filtro passi solo la componente continua del segnale PWM e attenui efficacemente la parte ad alta frequenza.

7.1.5 Introduzione di Sabertooth per il controllo

Per il controllo dei motori è stato utilizzato anche il driver per motori duale Sabertooth 12A, che permette di collegare due motori e controllarli tramite i suoi segnali di ingresso S1 e S2. Questi segnali vengono inviati rispettivamente ai pin M1A e M1B per il motore 1, e M2A e M2B per il motore 2. La corrente in ingresso viene alimentata ai pin B+ e B-.

Configurazione modalità Sabertooth

La Sabertooth dispone di 6 switch per la configurazione del sistema. Nel caso del controllo tramite PWM, lo switch 4 e 6 era impostato su "OFF" mentre gli altri erano "ON".

Lo switch 6 era impostato su "OFF", il che significa che il range del segnale di ingresso andava da 1.875V a 3.125V, con 2.5V come punto di zero.

Ora, con l'introduzione della *Sabertooth*, abbiamo verificato i valori di *CCR* (Counter Compare Register) necessari per ottenere segnali compresi tra 1.875V e 3.125V. Questa analisi è stata effettuata utilizzando un oscilloscopio Hantek a 4 canali. Dai risultati, abbiamo osservato che:

- Con un valore di *CCR* pari a 450, il segnale risultante è di 1.875V, corrispondente alla massima velocità del motore in senso antiorario.
- Con un valore di *CCR* pari a 612, il segnale è di 2.5V, indicando che i motori sono fermi.
- Con un valore di *CCR* pari a 750, il segnale è di 3.125V, corrispondente alla massima velocità dei motori in senso orario.

7.1.6 Realizzazione del sistema

Dopo aver completato tutte le analisi necessarie, sono state implementate le considerazioni precedentemente discusse. Per il calcolo del *CCR* è stata utilizzata la funzione `pid_compute`, descritta in dettaglio nel paragrafo 3.2, con i valori misurati tramite oscilloscopio.

Come possiamo osservare, i valori minimi e massimi sono stati impostati sulla base delle riflessioni e delle analisi effettuate in precedenza. Tale funzione restituirà un valore di *CCR* che successivamente sarà assegnato al registro con il comando: `TIM1->CCR1 = power(VALORE)`

DEL CCR);

Di seguito sono riportati i valori che sono stati impostati nella funzione PID_COMPUTE.

```
if(setpoint < 0.0) {
    double powMin = 612.0, powMax = 450.0;

}
else if(setpoint>0){
    double powMin = 600, powMax = 750;
}
else{
    double powMin = 600, powMax = 612;
}
```

7.1.7 Schema Finale

L'immagine riportata di seguito illustra lo schema fisico implementato per **PWM**, con i relativi collegamenti tra la scheda STM32, il filtro RC, il driver Sabertooth, i motori e la batteria.

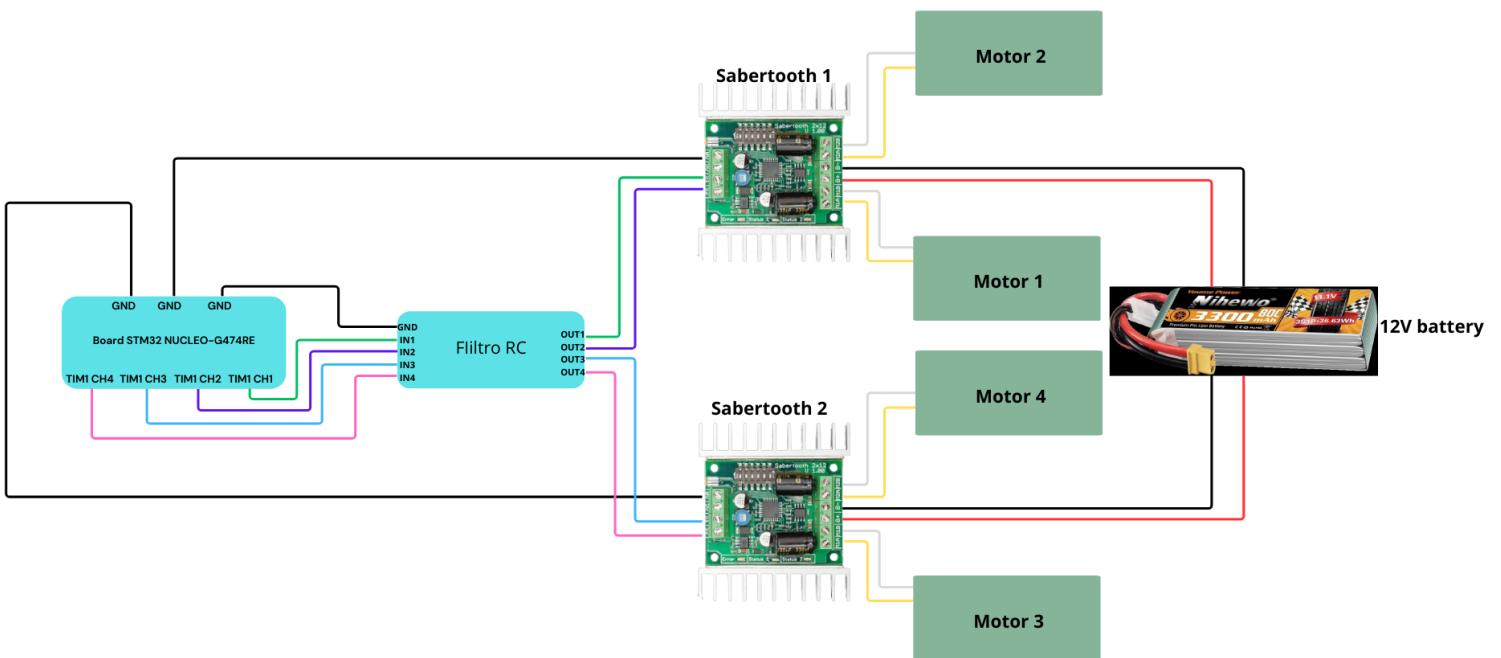


Figure 7.1: Schema finale PWM

7.2 Controllo dei motori tramite UART

Nel progetto, la comunicazione tra la scheda STM32 e altri dispositivi è stata gestita utilizzando l'interfaccia *UART* (*Universal Asynchronous Receiver-Transmitter*). La scelta di UART è stata dettata dalla sua semplicità di implementazione e dall'affidabilità per lo scambio di dati a bassa latenza.

7.2.1 Configurazione dell'UART

Per la configurazione della comunicazione UART, è stato utilizzato l'ambiente di sviluppo *STM32CubeMX*, che ha permesso di impostare rapidamente i parametri principali. I clock delle periferiche UART sono stati configurati come segue:

- **USART1:** Clock derivato da *PCLK2*, con frequenza impostata a 170 MHz.
- **USART3:** Clock derivato da *PCLK1*, con frequenza impostata a 170 MHz.

Il *baud rate* per entrambe le UART è stato configurato a 9600 bps direttamente dall'interfaccia di configurazione del file `.ioc` di STM32CubeMX. Non sono stati eseguiti calcoli manuali per il baud rate, poiché il tool si è occupato automaticamente di gestire i divisorì e i registri necessari.

Le impostazioni principali, comuni a entrambe le UART, sono:

- **Bit di dati:** 8.
- **Bit di stop:** 1.
- **Parità:** Nessuna.
- **Modalità:** Transmitter e Receiver attivi.

I pin utilizzati per la comunicazione sono stati mappati sui pin fisici TX e RX della scheda STM32 tramite il tool di configurazione CubeMX.

7.2.2 Introduzione di Sabertooth per il controllo

Per il controllo dei motori è stato utilizzato anche il driver per motori duale Sabertooth 12A, che permette di collegare due motori e controllarli tramite uart tramite un unico segnale di ingresso S1. Questo segnale viene inviato rispettivamente ai pin M1A e M1B per il motore 1, e M2A e M2B per il motore 2. La corrente in ingresso viene alimentata ai pin B+ e B-.

La Sabertooth dispone di 6 switch per la configurazione del sistema. Nel caso del controllo tramite UART, gli switch 1,3,5 e 6 era impostato su "ON" mentre gli altri erano "OFF".

Impostando la Sabertooth in questa configurazione, entrambi i motori possono essere controllati con il segnale S1. In particolare, il range di controllo dei motori è il seguente:

- Da 1 a 63: il motore gira in senso antiorario (decrescente), dove:
 - 1 rappresenta il massimo antiorario,
 - 63 rappresenta il minimo antiorario.
- 64: i motori sono fermi.
- Da 65 a 127: il motore gira in senso orario (crescente), dove:
 - 65 rappresenta il minimo orario,
 - 127 rappresenta il massimo orario.

Ora, controllando con un solo segnale S1 entrambi i motori, dal range 1-127 si controlla il primo motore e dal range 128-254 si controlla il secondo motore.

La Sabertooth fornisce una tensione in uscita che varia da 0V a 5V, con un punto di zero di 2.5V. In pratica, quando il segnale di controllo è pari a 64 (il valore di "zero" che ferma i motori), la tensione in uscita è di 2.5V, corrispondente al punto centrale del range di controllo. Se il valore del segnale aumenta sopra 64 (range da 65 a 127), la tensione in uscita aumenta progressivamente fino a 5V, indicando che i motori stanno girando in senso orario. Al contrario, se il valore del segnale scende sotto 64 (range da 1 a 63), la tensione diminuisce, arrivando a 0V, indicando il movimento in senso antiorario. Questo intervallo di tensione permette di regolare in modo preciso la velocità e la direzione dei motori, con 2.5V come punto di neutralità, che assicura che i motori siano fermi quando il segnale è a 64.

7.2.3 Realizzazione del sistema

Discusso il range di valori che la Sabertooth prende in ingresso, andiamo ora a vedere come è stato realizzato il meccanismo di comunicazione tra la scheda STM32 e la Sabertooth, e come è stato utilizzato il risultato restituito dalla funzione `PID_Compute`, discussa in dettaglio nel paragrafo 3.2.

Quello che si è fatto è calcolare il PID separatamente per ciascun motore e, per il primo motore, inviare direttamente tramite una trasmissione USART a S1 il valore restituito dalla funzione `PID_Compute`. Per il secondo motore, invece, si prende il valore restituito dalla funzione `PID_Compute` e gli si somma 127, in modo da adattarlo al range di controllo del secondo motore, che viene poi inviato anch'esso a S1 tramite una trasmissione USART. Questo procedimento è stato applicato anche sull'altra Sabertooth per controllare gli altri due motori.

```
power1 = PID_Compute(&pid1, counterValue1, tick1,
                      speeds.right_speed, 2448.0, 1);
HAL_UART_Transmit(&huart1, &power1, 1, HAL_MAX_DELAY);
```

```
power2 = PID_Compute(&pid2, counterValue2, tick2,
                      speeds.left_speed, 2448.0, 2);
INpower2 = power2 + 127;
HAL_UART_Transmit(&huart1, &INpower2, 1, HAL_MAX_DELAY);
```

Di seguito sono riportati i valori che sono stati impostati nella funzione PID_COMPUTE.

```
if(setpoint < 0.0){
    double powMin = 63, powMax = 1;

}
else if(setpoint>0){
    double powMin = 64, powMax = 127;
}
else{
    double powMin = 64, powMax = 64;
}
```

7.2.4 Schema Finale

L'immagine riportata di seguito illustra lo schema fisico implementato per **USART**, con i relativi collegamenti tra la scheda STM32, il driver Sabertooth, i motori e la batteria.

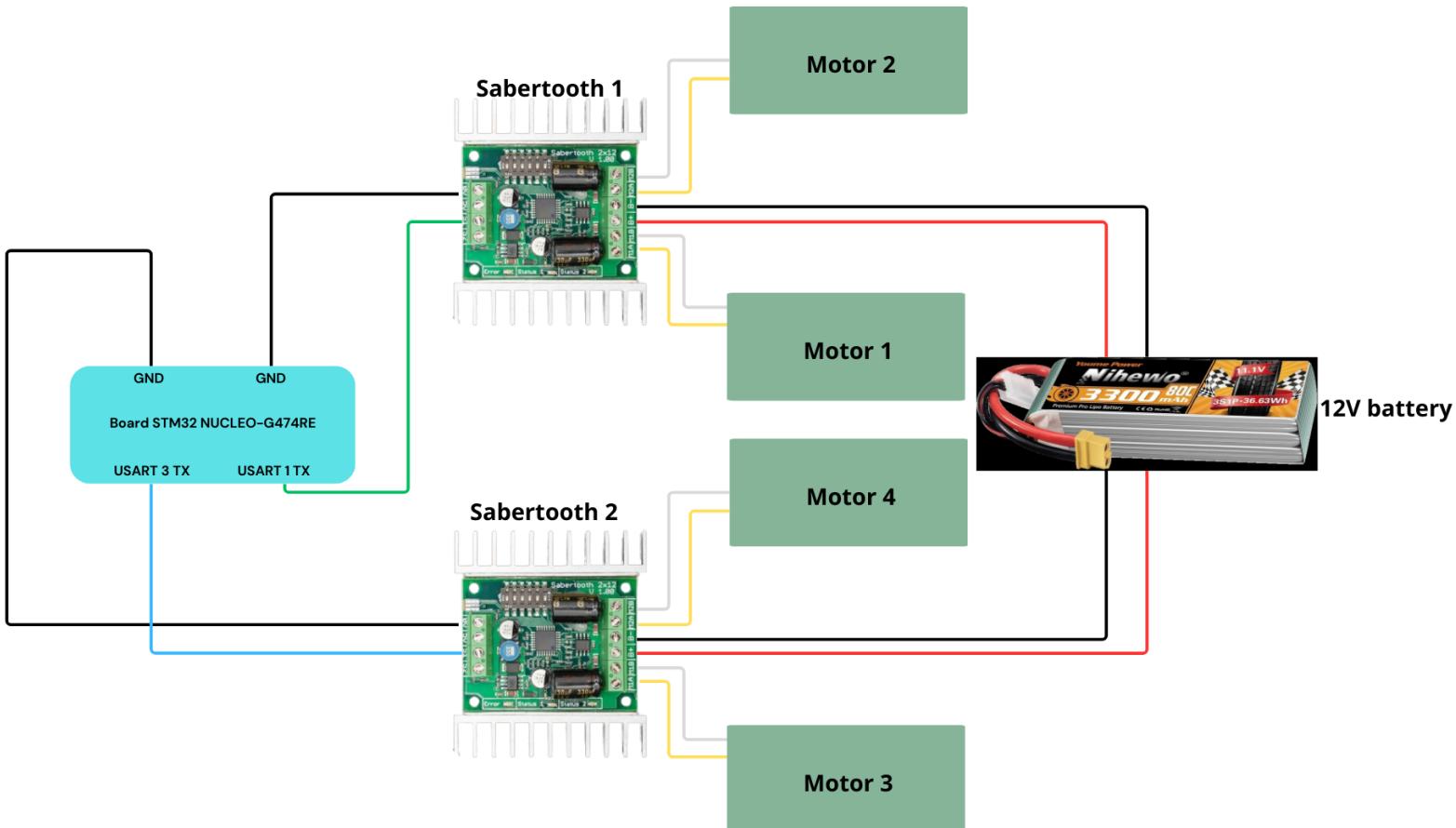


Figure 7.2: Schema finale USART

7.3 Confronto granularità

Nel nostro progetto, abbiamo esplorato due tecniche principali per il controllo dei motori: la modulazione a larghezza di impulso (PWM) e la comunicazione tramite UART. Entrambe le soluzioni presentano vantaggi e svantaggi, e la scelta tra le due dipende dalle specifiche necessità del sistema.

7.3.1 Analisi della Granularità nei Metodi di Controllo PWM e UART

Durante l'analisi dei metodi di controllo per il rover, è stata valutata la granularità offerta da due meccanismi distinti: il controllo PWM (Pulse Width Modulation) e il controllo UART (Universal Asynchronous Receiver-Transmitter). Entrambi i metodi permettono di regolare la velocità del rover in base a un intervallo di valori, ma con modalità e caratteristiche differenti. La

granularità del controllo è fondamentale per determinare quanto fine possa essere la regolazione della velocità e la precisione con cui il rover può essere manovrato.

Granularità PWM

Il segnale PWM controlla la velocità del motore variando il ciclo di lavoro, con un intervallo che va da 450 a 750. Questo intervallo è suddiviso in due regioni:

- **Movimento antiorario:** Il range va da 450 a 600, dove 450 corrisponde alla velocità massima antioraria e 600 al punto di fermo (0 rpm).
- **Movimento orario:** Il range va da 600 a 750, con 600 come punto di fermo e 750 come velocità massima oraria.

Ogni regione ha una suddivisione di 150 passi, ovvero un intervallo di 150 valori per ciascuna direzione di movimento.

Calcolo dell'incremento di velocità PWM

La velocità massima del rover è di 170 rpm. La variazione della velocità viene distribuita su 150 passi, per cui l'incremento di velocità per ogni passo del PWM è:

$$\text{Incremento di velocità per passo (antiorario)} = \frac{170 \text{ rpm}}{150} = 1.13 \text{ rpm per passo}$$

Questo incremento di 1.13 rpm si applica sia per la direzione antioraria che per quella oraria, consentendo un controllo fine della velocità in entrambe le direzioni.

Granularità UART

Il controllo tramite UART, che si basa su segnali seriali, segue un intervallo di valori che va da 0 a 127, suddiviso anch'esso in due sezioni distinte:

- **Movimento antiorario:** Il range va da 0 a 63, con 0 che rappresenta la velocità massima antioraria e 63 il punto di fermo (0 rpm).
- **Movimento orario:** Il range va da 64 a 127, con 64 come punto di fermo e 127 come velocità massima oraria.

Ogni regione ha una suddivisione di 64 passi, ovvero un intervallo di 64 valori per ciascuna direzione di movimento.

Calcolo dell'incremento di velocità UART

Analogamente al caso del PWM, la velocità massima del rover è di 170 rpm. La variazione della velocità viene distribuita su 64 passi, per cui l'incremento di velocità per ogni passo dell'UART è:

$$\text{Incremento di velocità per passo (antiorario)} = \frac{170 \text{ rpm}}{64} = 2.66 \text{ rpm per passo}$$

Anche in questo caso, l'incremento di 2.66 rpm si applica sia per la direzione antioraria che per quella oraria, consentendo un controllo meno fine rispetto al PWM, ma comunque preciso.

7.3.2 Confronto tra PWM e UART

Metodo	Range Antiorario	Range Orario	Incremento per passo	Totale valori	Velocità Massima
PWM	450-600	600-750	1.13 rpm per passo	300	170 rpm
UART	0-63	64-127	2.66 rpm per passo	128	170 rpm

Table 7.1: Confronto tra PWM e UART

Analisi del Confronto

- Granularità:** Il PWM offre una granularità più fine, con un incremento di 1.13 rpm per ogni passo, mentre l'UART ha un incremento di 2.66 rpm per passo. Questo significa che il controllo tramite PWM è più preciso rispetto a UART, consentendo regolazioni più piccole della velocità. L'incremento di 1.13 rpm (PWM) rispetto a 2.66 rpm (UART) è meno della metà e quindi la percentuale di precisione va calcolata rispetto al valore più grande (2.66) come rapporto di quanto più piccolo è l'incremento di PWM.

Per calcolare quanto è più preciso l'incremento di PWM rispetto a quello di UART, possiamo usare la formula inversa:

$$\text{Precisione percentuale} = \left(\frac{\text{Incremento UART} - \text{Incremento PWM}}{\text{Incremento UART}} \right) \times 100$$

In questo caso:

$$\text{Precisione percentuale} = \left(\frac{2.66 - 1.13}{2.66} \right) \times 100 = \left(\frac{1.53}{2.66} \right) \times 100 \approx 57.52\%$$

Questo significa che l'incremento di PWM è circa il 57.52% più preciso rispetto all'incremento di UART. In altre parole, l'incremento di PWM rappresenta una frazione più piccola della

velocità totale per ogni passo, rendendo il controllo più dettagliato rispetto all'UART.

- **Numero di passi:** Il PWM ha un intervallo di 300 valori (150 per ciascuna direzione), mentre l'UART ha solo 128 valori totali, suddivisi in 64 per ciascuna direzione. Ciò implica che, mentre il controllo tramite PWM consente una maggiore quantità di passi e quindi una regolazione più dettagliata della velocità, l'UART offre una risoluzione inferiore.
- **Controllo della velocità:** Sebbene entrambi i metodi permettano di ottenere la stessa velocità massima di 170 rpm, il controllo tramite PWM consente una regolazione più fine e graduale, soprattutto nelle velocità basse. L'UART, pur essendo meno preciso, potrebbe risultare più adatto per applicazioni in cui non è richiesta una precisione estrema, ma solo una velocità generale di movimento.

7.4 Pro e Contro

7.4.1 Pro e Contro del PWM

La modulazione a larghezza di impulso (PWM) è una tecnica molto utilizzata per il controllo preciso della velocità dei motori. Grazie alla sua capacità di modulare il ciclo di lavoro del segnale, permette di ottenere un controllo fine della velocità senza generare sprechi significativi di energia. Uno dei suoi principali vantaggi è la sua efficienza, poiché permette di regolare la potenza fornita al motore senza conversioni complesse di segnale, riducendo al minimo le perdite energetiche. Inoltre, la regolazione del duty cycle è semplice da implementare e consente un controllo fluido e immediato.

Tuttavia, il PWM presenta anche alcune difficoltà. La gestione della frequenza del segnale e del filtro passa-basso per ottenere una tensione continua stabile può essere complessa, richiedendo attenzione a dettagli come la selezione dei componenti e la gestione delle interferenze ad alta frequenza. Inoltre, l'utilizzo di PWM può comportare la necessità di un'accurata calibrazione dei parametri per adattarsi correttamente ai driver dei motori, come nel caso della Sabertooth.

7.4.2 Pro e Contro dell'UART

La **comunicazione UART**, al contrario, è una soluzione molto più semplice da implementare, specialmente in un sistema come il nostro, dove si intende controllare più motori contemporaneamente con un solo segnale. UART è una comunicazione seriale che permette l'invio di dati tra dispositivi in modo affidabile e senza la necessità di operazioni matematiche complesse, come nel caso del PWM. Inoltre, offre la possibilità di inviare comandi precisi ai motori in modo

digitale, riducendo i rischi di errore dovuti a interferenze o rumori elettrici.

Un vantaggio significativo dell'UART è la sua **facilità di configurazione e gestione** rispetto al PWM, che richiede una gestione più complessa del segnale e dell'hardware. Il sistema UART, infatti, consente una configurazione semplice attraverso la programmazione, senza la necessità di implementare filtri hardware o gestire variabili come il duty cycle e la frequenza PWM. Inoltre, il controllo tramite UART è particolarmente vantaggioso quando si desidera inviare comandi di motori a distanza o in un ambiente di comunicazione più ampio.

7.4.3 Motivazione per la Scelta di UART

Per il progetto è stato scelto il protocollo di comunicazione UART per il controllo dei motori, per i seguenti motivi:

- **Semplicità di implementazione:** A differenza del PWM, l'UART non richiede l'utilizzo di un filtro RC né fili aggiuntivi. Inoltre, il filtro RC disponibile non era di qualità ottimale, il che, in rare occasioni, poteva generare segnali imprecisi, causando l'invio di valori errati ai motori.
- **Precisione adeguata alle esigenze del progetto:** Sebbene il PWM offra una granularità superiore (con incrementi o decrementi di velocità di 1,13 RPM rispetto ai 2,66 RPM dell'UART), tale livello di precisione non era necessario per i requisiti del progetto. Pertanto, è stata preferita la soluzione più semplice ed efficace offerta dall'UART.
- **Risorse scarse:** Inizialmente, il rover era funzionante con un sistema PWM, supportato da 4 filtri RC che, come chiarito nei punti precedenti, causavano problemi di instabilità. Successivamente, è stata introdotta un'altra scheda STM32, che poteva controllare i motori nel caso in cui la prima si rompesse. Tuttavia, questo ha comportato la necessità di costruire altri 4 filtri RC, portando il totale a 8 filtri RC su una breadboard. È evidente che questo rappresentava un problema per la stabilità del sistema.
- **Spazio:** Non avendo la possibilità di saldare questi filtri RC su una millefori, lo spazio occupato dalla breadboard e dai collegamenti era notevole, rendendo difficile l'installazione di componenti essenziali, come sensori a ultrasuoni, giroscopio, ESP32, le due schede STM32.

La scelta di UART per il controllo dei motori è stata motivata dalla necessità di semplificare l'implementazione del sistema, considerando le risorse limitate e il tempo a disposizione. **Sebbene si sia consapevoli che il PWM offre una maggiore precisione e granularità nel controllo della velocità**, i costi e le difficoltà di implementazione (filtr RC, spazio e risorse

limitate) hanno reso UART una scelta più adatta al progetto. In generale, UART ha offerto un buon compromesso tra semplicità, affidabilità e performance, soddisfacendo le esigenze specifiche del sistema senza introdurre eccessiva complessità.

Chapter 8

Strutture dati e task

In questo capitolo verranno descritte le strutture dati utilizzate per memorizzare gli stati del sistema e, successivamente, sarà illustrato il funzionamento di ciascun task, accompagnato dalla rispettiva implementazione.

8.1 Strutture dati

8.1.1 Struttura stato parziale della scheda

Questa struttura contiene le informazioni relative esclusivamente alla scheda in cui è memorizzata. Pertanto, ogni scheda dispone di una propria struttura di stato parziale, che differisce da quella dell'altra.

Board 1 (slave)

In questa struttura sono memorizzati i dati raccolti dai sensori collegati alla scheda 1 durante il task di lettura. Le informazioni salvate includono:

```
typedef struct partial_slave_s{
    int16_t temperatural;
    uint16_t led1;
    uint16_t led2;
    uint16_t encoder1;
    uint16_t encoder2;
    uint16_t encoder3;
    uint16_t encoder4;
    uint8_t B1_state;

}partial_slave_t;
```

Figure 8.1: Stato parziale B1

- temperatura1: registra la temperatura rilevata dalla scheda 1.
- led1: rappresenta lo stato del primo LED.
- led2: rappresenta lo stato del secondo LED.
- encoder1: contiene i valori dell'encoder della ruota anteriore sinistra.
- encoder2: contiene i valori dell'encoder della ruota anteriore destra.
- encoder3: memorizza i dati dell'encoder della ruota posteriore sinistra.
- encoder4: memorizza i dati dell'encoder della ruota posteriore destra.
- B1_state: indica lo stato della scheda e dei suoi sensori, assumendo il valore 1 se tutto funziona correttamente e 0 in caso di errori. Questa variabile è utilizzata per determinare l'attuazione da effettuare.

Board 2 (master)

```
typedef struct partial_master_s {
    controller_t controller;
    accelerometer_t accelerometer;
    int16_t temperatura2;
    uint16_t sonar1;
    uint16_t sonar2;
    uint16_t sonar3;
    uint8_t B2_state;

} partial_master_t;
```

Figure 8.2: Stato parziale B2

In questa struttura vengono memorizzati i dati raccolti dai sensori collegati alla scheda 1 durante il task di lettura. Le informazioni salvate comprendono:

- controller: struct contenente le informazioni inviate dall'utente tramite il controller.

```
typedef struct controller_s {
    uint16_t ax;
    uint16_t ay;
    uint8_t a_btn;

    uint16_t bx;
    uint16_t by;
    uint8_t b_btn;

    uint8_t btn1;
    uint8_t btn2;
} controller_t;
```

Figure 8.3: Struct controller

- accelerometer: anch'essa una struct che raccoglie i dati relativi all'accelerometro.

```
typedef struct{
    int16_t acc_x;
    int16_t acc_y;
    int16_t acc_z;
    int16_t gyro_x;
    int16_t gyro_y;
    int16_t gyro_z;
} accelerometer_t;
```

Figure 8.4: Struct accelerometro

- sonar1: che memorizza le informazioni rilevate dal sonar orientato verso sinistra.
- sonar2: che memorizza le informazioni rilevate dal sonar orientato verso avanti.
- sonar3: che memorizza le informazioni rilevate dal sonar orientato verso destra.
- B2_state: che indica lo stato della scheda e dei suoi sensori, assumendo il valore 1 se tutto funziona correttamente e 0 in caso di errori. Questa variabile viene utilizzata per determinare l'attuazione da eseguire.

8.1.2 Struttura stato globale sistema

Questa struttura raccoglie le informazioni relative all'intero sistema, inclusi i componenti non direttamente connessi alla scheda. In una fase iniziale, prima della comunicazione tra le schede e dello scambio dei dati, la struttura dello stato globale contiene esclusivamente le informazioni della scheda locale. Una volta completato lo scambio di informazioni, la struttura viene aggiornata per rappresentare lo stato complessivo del sistema e viene utilizzata per determinare l'attuazione da eseguire. Include quindi tutte le informazioni già viste nelle sezioni precedenti.

```
typedef struct global_s{
    controller_t controller;
    accelerometer_t accelerometer;
    int16_t temperatural;
    int16_t temperatura2;
    uint16_t sonar1;
    uint16_t sonar2;
    uint16_t sonar3;
    uint16_t led1;
    uint16_t led2;
    uint16_t encoder1;
    uint16_t encoder2;
    uint16_t encoder3;
    uint16_t encoder4;
    uint8_t B1_state;
    uint8_t B2_state;
}global_t;
```

Figure 8.5: Struttura stato globale

8.2 Descrizione task

Sono stati implementati cinque task distinti, di cui quattro sono presenti su ogni scheda. Questi task sono i seguenti:

- Task di lettura: in questo task vengono letti i dati dai sensori e viene determinato lo stato della scheda.
- Task di comunicazione: in questo task le informazioni vengono scambiate tra le due schede e viene creato lo stato parziale. Successivamente, viene presa la decisione sull’attuazione da effettuare.
- Task di attuazione (presente solo nella scheda slave): in questo task i valori vengono inviati agli attuatori per il controllo del sistema.
- Task di attuazione degradata (presente solo nella scheda master): in questo task vengono inviati i valori agli attuatori con funzionalità limitate, in caso di malfunzionamento parziale.
- Task di emergenza: in questo task i motori vengono bloccati per prevenire danni in caso di emergenza.

8.2.1 Task di lettura

Task in cui vengono letti i dati dai sensori e viene determinato lo stato della scheda, è presente su entrambe le schede, ma la sua durata varia a causa delle diverse tipologie di sensori collegati a ciascuna. La scheda 1, oltre a rilevare la temperatura, è incaricata di leggere i dati provenienti dai quattro encoder. La scheda 2, oltre alla temperatura, è responsabile della lettura dei dati del controller, dell’accelerometro e dei tre sonar. In particolare, il sonar richiede un tempo maggiore di esecuzione, come vedremo nei capitoli successivi, poiché il sistema deve attendere che l’onda acustica colpisca l’ostacolo e poi ritorni indietro.

8.2.2 Task di comunicazione

Questo task gestisce lo scambio di informazioni tra le due schede e l’aggiornamento dello stato globale, che viene utilizzato per determinare l’attuazione da eseguire. Come verrà spiegato nei prossimi capitoli, il task di lettura del master richiede più tempo rispetto a quello dello slave. Per questo motivo, vengono utilizzati due pin, B1_pin e B2_pin, per la sincronizzazione delle schede. Il processo di comunicazione si suddivide in tre fasi:

1. La scheda 1 riceve e la scheda 2 trasmette: una volta completato il processo di lettura, il master imposterà il pin B2_pin a 1, sbloccando così lo slave che imposterà B1_pin a 1 per avviare la comunicazione. Se la trasmissione non inizia entro un determinato lasso

di tempo, significa che il master ha superato la sua deadline, causando il passaggio del sistema allo stato di emergenza. La scheda 1, ricevuti i parametri, calcola la propria decisione per la prossima attuazione e la trasmette insieme allo stato parziale.

2. La scheda 1 trasmette e la scheda 2 riceve: dopo che la prima trasmissione è stata completata, la scheda 2 attende che la scheda 1 imposti a 1 il pin B1_pin. A quel punto, la scheda 2 imposterà B2_pin a 1 per indicare che è pronta a ricevere i dati, avviando così la comunicazione. Una volta ricevuti i parametri, la scheda 2 elabora la propria decisione e la confronta con quella della scheda 1. In base al confronto, assegna un valore alla variabile final_decision: 1 se le due decisioni coincidono, 0 in caso contrario.
3. Una volta calcolata, la decisione finale viene trasmessa dalla scheda 2 alla scheda 1. Se il valore è 1, la decisione presa viene attuata direttamente. Se invece è 0, indicando una discordanza tra le due schede, l'attuazione viene delegata al master. In questo caso, se il master è funzionante, l'attuazione avviene in modalità degradata; in caso contrario, si attiva la modalità di emergenza.

La progettazione e l'implementazione della comunicazione tra le schede sono descritte in dettaglio in un capitolo dedicato.

8.2.3 Task di attuazione

Questo task, presente esclusivamente sulla scheda 1 (slave), viene eseguito solo quando i dati della struttura di stato globale indicano che il sistema è pienamente operativo. In altre parole, entrambe le variabili, B1_state e B2_state, descritte nella sezione precedente, devono avere valore pari a 1. Durante l'esecuzione del task, i valori vengono inviati agli attuatori per consentirne il corretto funzionamento, in particolare ai LED e ai motori. Per quanto riguarda i motori, viene prima calcolato il setpoint, ovvero la velocità desiderata di rotazione del motore. Successivamente, viene eseguito il calcolo del PID e infine la potenza calcolata viene trasmessa ai motori tramite il protocollo di comunicazione UART.

8.2.4 Task di attuazione degradata

Questo task, presente esclusivamente sulla scheda 2 (master), viene eseguito solo nel caso in cui uno dei componenti deputati alla rilevazione della velocità del rover non funzioni correttamente. Ciò include il malfunzionamento di uno o più encoder, dell'accelerometro o della scheda B1 stessa. Pertanto, il task viene attivato quando la variabile B1_state è pari a 0 e la variabile B2_state è pari a 1. Durante l'esecuzione di questo task, i dati vengono prelevati dalla struttura di stato globale e il rover continua a muoversi in base ai comandi ricevuti dal controller. Tuttavia, la velocità massima sarà limitata a 20 g/min.

8.2.5 Task di emergenza

Questo task è presente su entrambe le schede, ma viene avviato in circostanze specifiche: si attiva solo quando la scheda 2 non funziona correttamente o quando uno dei sensori, come i sonar o il controller, presenta un malfunzionamento. Se la scheda 1 è ancora operativa, il task viene eseguito su di essa; diversamente, se anche la scheda 1 è guasta, il task si avvia sulla scheda 2. All'ingresso in questo task, i motori vengono immediatamente bloccati, poiché il suo avvio indica che la scheda B2 non è funzionante, rendendo impossibile rilevare ostacoli tramite i sonar o ricevere correttamente i comandi dal controller.

8.3 Stateflow

In figura è rappresentato lo stateflow del sistema, che ne illustra il funzionamento. È presente uno stato principale che contiene due sottostati, ognuno dei quali rappresenta una scheda del sistema che opera in parallelo. All'interno di ciascun sottostato sono definiti quattro stati, corrispondenti ai task, progettati per lavorare in maniera sequenziale.

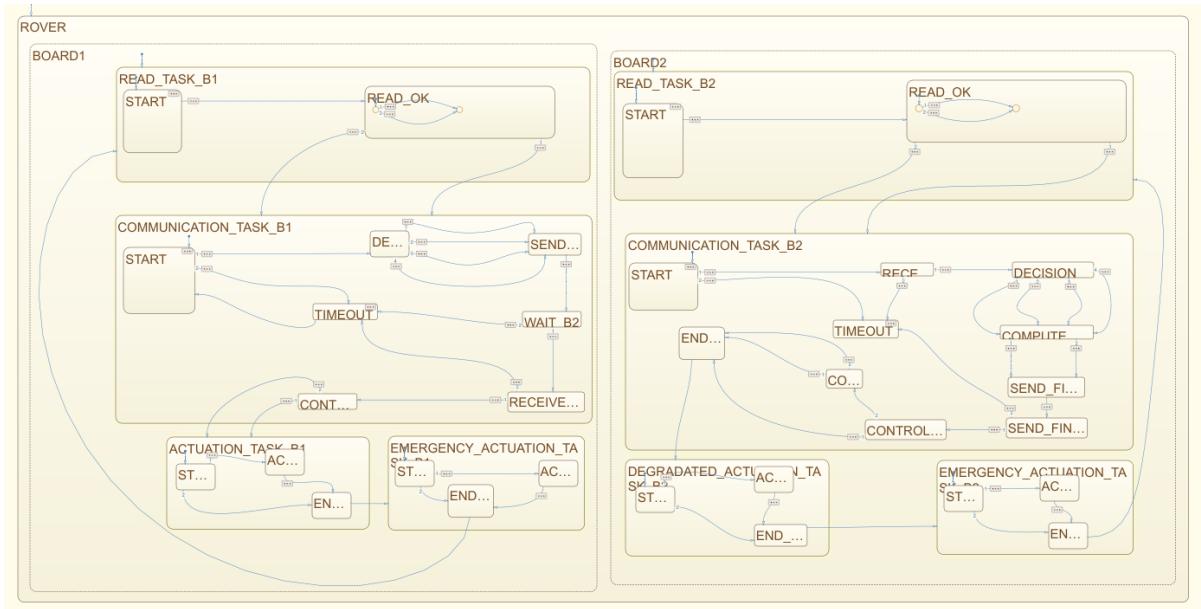


Figure 8.6: Stateflow del sistema

8.3.1 Task di lettura

In figura 2.9 è rappresentato il task di lettura della scheda 2 (master), che differisce da quello della scheda 1 solo in alcuni dettagli.

Appena inizia il task, vengono acquisiti i valori dai sensori collegati alla scheda. Nel caso della scheda 2, i dati raccolti includono:

- *readSonar*: lettura dei valori restituiti dai tre sonar.
- *readController*: lettura dei dati inviati dal controller.
- *readAcc*: lettura dei dati forniti dall'accelerometro.
- *TemperatureSensor_ReadB2*: lettura della temperatura della scheda 2.

Nella scheda 1, invece, vengono rilevati esclusivamente i dati relativi alla temperatura e ai quattro encoder.

Dopo la raccolta, viene effettuato un controllo sulla temperatura: se questa supera i 90°C per più di 200 misurazioni consecutive (circa 10 secondi), la scheda è considerata surriscaldata e quindi inutilizzabile.

Infine, viene eseguito un controllo sui dati raccolti. In base all'esito, viene assegnato un valore alla variabile *B2State*, che rappresenta lo stato della scheda. Se uno dei valori è fuori dai limiti accettabili, la variabile assume il valore 0; in caso contrario, il valore assegnato è 1.

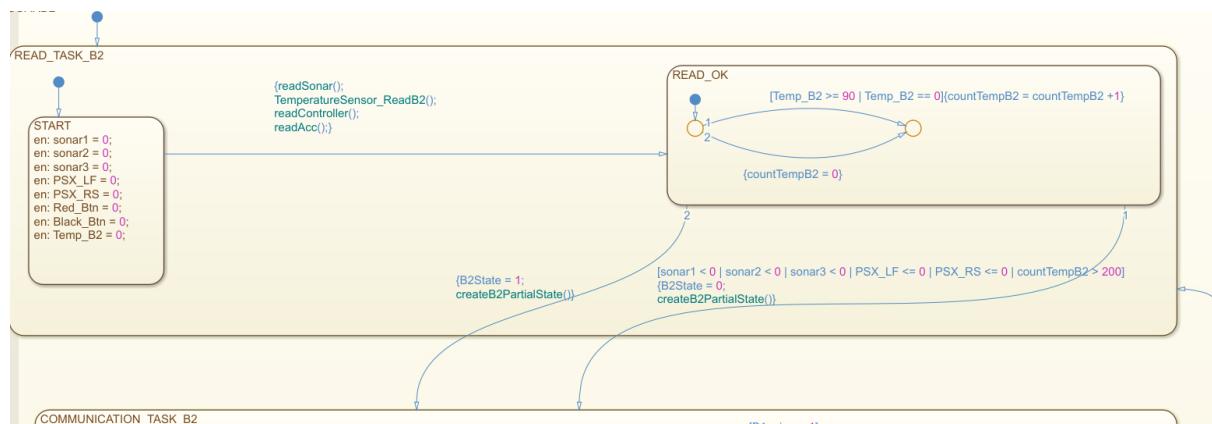


Figure 8.7: Task di lettura

8.3.2 Task di comunicazione

In figura è rappresentato il task di comunicazione della scheda 2 (master), che ha il compito di scambiare i dati rilevati dai sensori tra le due schede. Sebbene il task di comunicazione sia presente su entrambe le schede, il loro funzionamento differisce poiché mentre una scheda invia, l'altra riceve, e viceversa. Per realizzare questa sincronizzazione, vengono utilizzati due pin (*B1_pin* e *B2_pin*). La comunicazione inizia con la scheda master: al termine del suo task di lettura, imposta il pin *B2_pin* a 1 per segnalare che è pronta a trasmettere. La scheda slave, a sua volta, imposta il pin *B1_pin* a 1 per indicare che è pronta a ricevere, avviando così il processo di comunicazione. Una volta completata questa prima trasmissione, le due schede si sincronizzano nuovamente utilizzando gli stessi pin per consentire la trasmissione dei parametri e della decisione della scheda slave alla master. Il master quindi procede al calcolo della propria decisione e la confronta con quella della scheda slave. In base al confronto, assegna un valore alla variabile final decision: 1 se le due decisioni coincidono, 0 in caso contrario. Calcolata la decisione finale viene trasmessa dalla scheda 2 alla scheda 1. Se il valore è 1, la decisione presa viene attuata direttamente. Se invece è 0, indicando una discordanza tra le due schede, l'attuazione viene delegata al master. In questo caso, se il master è funzionante, l'attuazione avviene in modalità degradata; in caso contrario, si attiva la modalità di emergenza.

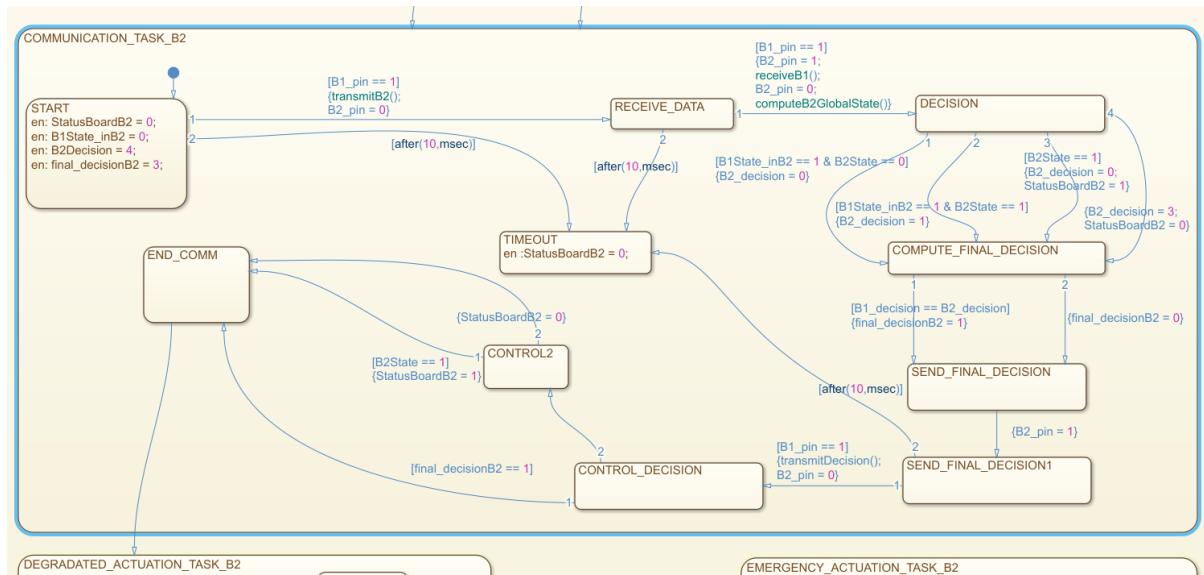


Figure 8.8: task di comunicazione

8.3.3 Task di attuazione

In figura è illustrato il task di attuazione, responsabile dell'esecuzione dei comandi inviati dall'utente tramite il controller. Questo task è presente esclusivamente sulla scheda 1 e viene eseguito solo quando tutti i componenti del rover sono in stato di piena funzionalità.

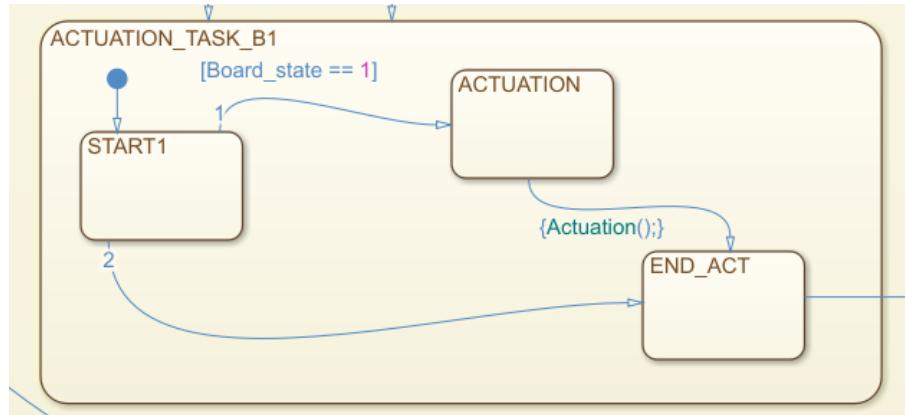


Figure 8.9: Task di attuazione

8.3.4 Task di attuazione degradata

In figura è illustrato il task di attuazione degradata, che consente al sistema di eseguire i comandi inviati dall'utente tramite il controller. In questa modalità, tuttavia, il rover opera con capacità motorie ridotte, limitando la velocità massima a 20 giri al minuto.

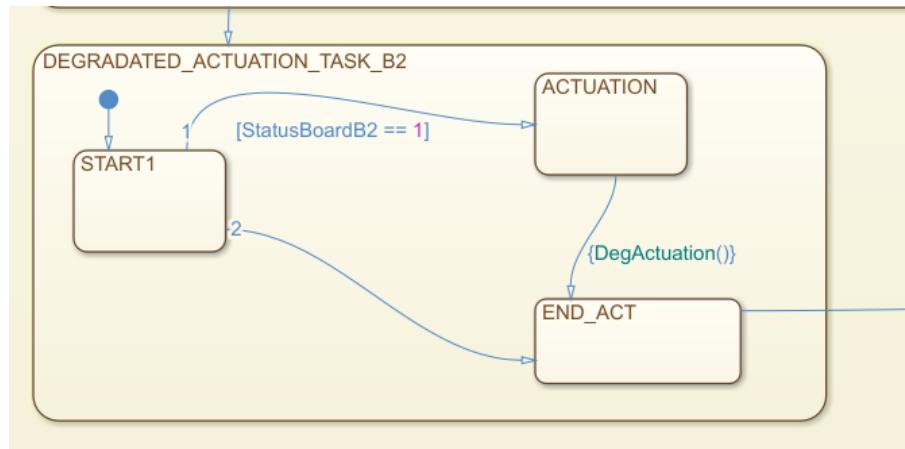


Figure 8.10: Task di attuazione degradata

8.3.5 Task di emergenza

In figura è rappresentato il task di emergenza, il cui scopo è bloccare tutti i motori per portare il rover in uno stato sicuro, evitando possibili danni. Questo task è presente su entrambe le schede e si attivano in circostanze differenti.

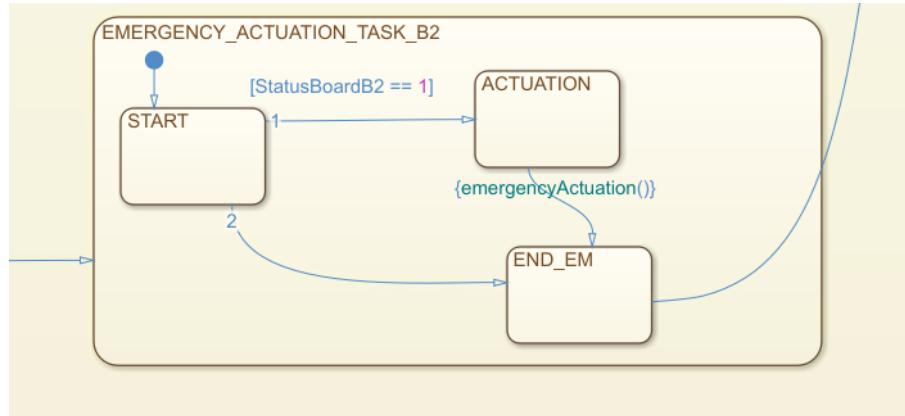


Figure 8.11: Task di emergenza

Chapter 9

Schedulazione task

In questo capitolo vengono implementati i vari task utilizzando FreeRTOS. Prima di procedere, è necessario definire i valori di WCET (Worst Case Execution Time).

9.1 Rilevazione tempi

Per misurare il tempo impiegato da ciascun task per eseguire le proprie operazioni, è stato utilizzato l'oscilloscopio "Hantek 6074BC". La misurazione è stata effettuata mediante l'uso di un pin che veniva attivato (set) all'inizio di una funzione e disattivato (reset) al termine della stessa. Questo processo è stato applicato a ogni funzione del task, consentendo di determinare il tempo complessivo tra l'attivazione e la disattivazione del pin attraverso l'oscilloscopio. Tutti i tempi rilevati sono stati arrotondati per eccesso. Poiché alcuni tempi risultano nell'ordine dei microsecondi e altri nell'ordine dei millisecondi, i valori inferiori a 1 ms sono stati sovrastimati a 1 ms, per uniformare le misurazioni su un'unica scala di grandezza.

9.2 Task di lettura

I task di lettura delle due schede differiscono tra loro in quanto utilizzano sensori diversi. Di conseguenza, il tempo necessario per effettuare le misurazioni varia, determinando WCET (Worst Case Execution Time) differenti.

9.2.1 Board 1

Con la board 1(slave), il task di lettura si occupa, oltre a monitorare la propria temperatura, di acquisire i dati provenienti dagli encoder dei quattro motori. Di seguito sono riportate le varie tempistiche associate:

Lettura temperatura scheda

La lettura della temperatura della scheda richiede un tempo di circa 11.5 μ s

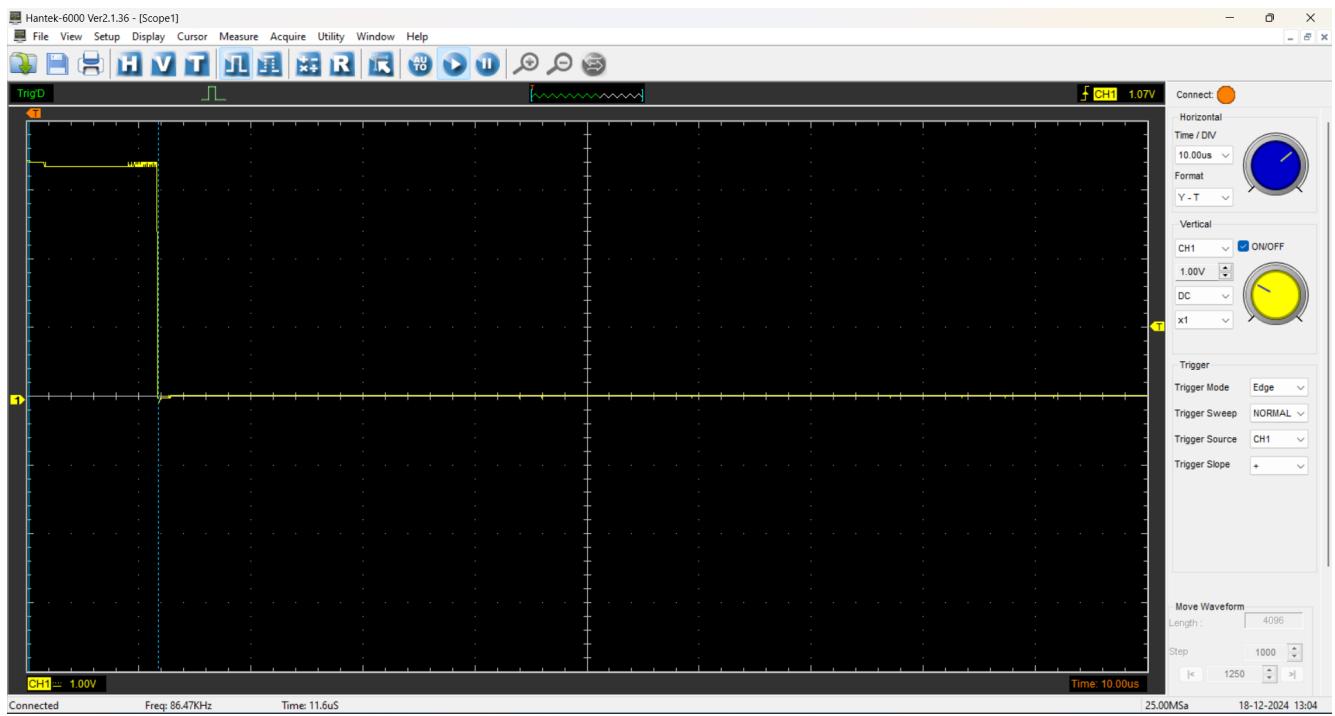


Figure 9.1: Tempo impegno per il controllo della temperatura della scheda

Lettura encoder

Il tempo necessario per leggere un singolo encoder è di circa $2 \mu\text{s}$, portando a un totale di $8 \mu\text{s}$ per i quattro encoder.

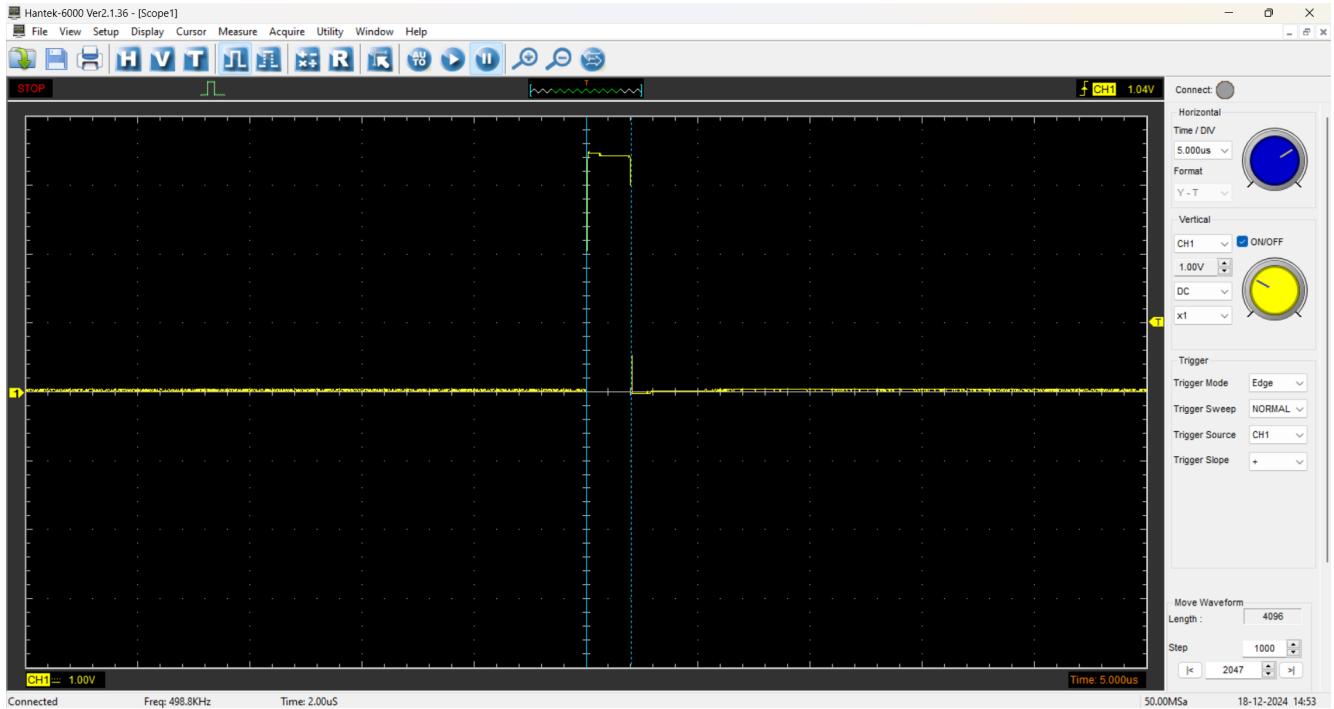


Figure 9.2: Tempo impiegato per la lettura dei dati di un encoder

Il tempo impiegato per la lettura dei sensori è di circa $15 \mu\text{s}$, quindi viene utilizzato come WCET del task una sovrastima di 1 ms.

9.2.2 Board 2

La board 2 (master) deve eseguire un numero maggiore di misurazioni rispetto alla board 1 e, di conseguenza, impiega più tempo per completare il task di lettura. Oltre a monitorare la temperatura, la board 2 acquisisce i dati provenienti dai sonar, dal controller e dall'accelerometro. Di seguito sono riportati i tempi relativi a ciascuna misurazione:

Lettura temperatura

Come per la board 1, anche nella board 2 il tempo per la rilevazione della temperatura della scheda è di circa 11.6 μ s. Consideriamo una sovrastima di 1 ms.

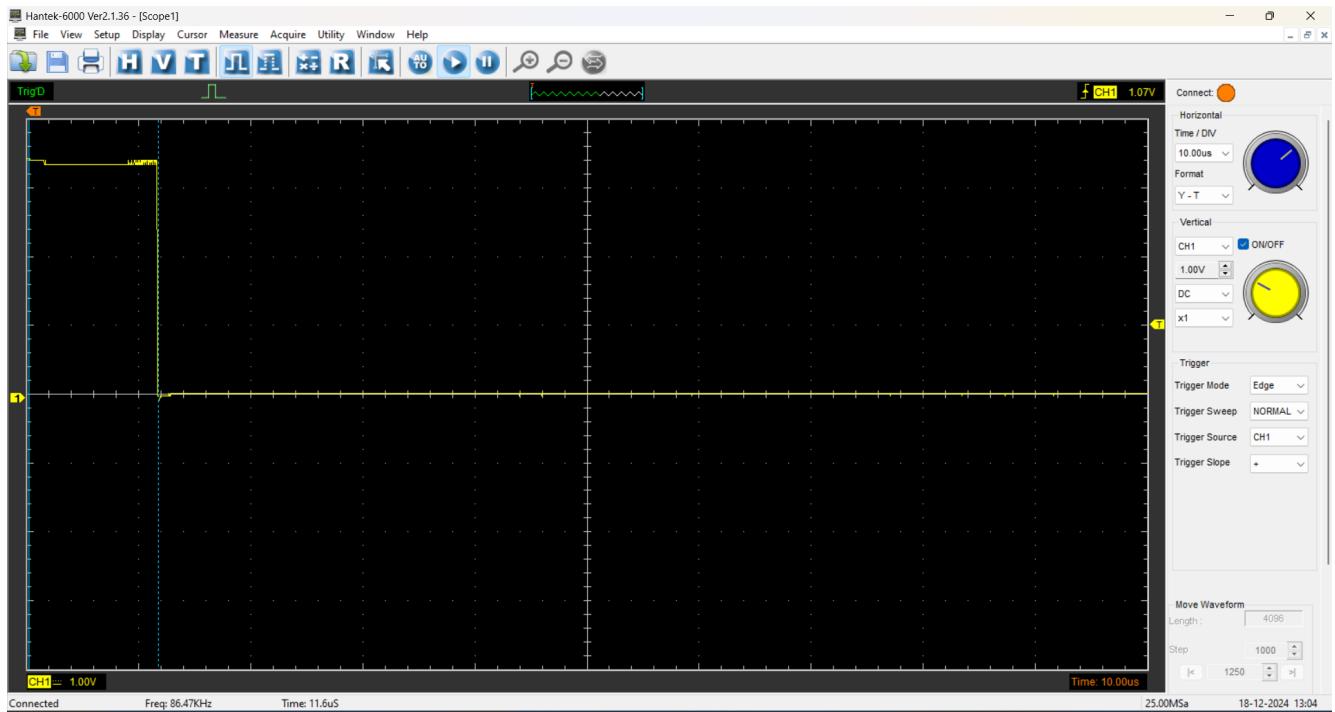


Figure 9.3: Tempo impegno per il controllo della temperatura della scheda

Lettura dati dei sensori ultrasonici

I tre sensori a ultrasuoni presenti sul rover sono in grado di rilevare ostacoli fino a una distanza massima di 5 metri. Il tempo di lettura dipende dalla distanza dell'ostacolo rilevato: quanto più vicino è l'ostacolo, tanto più veloce sarà la lettura, mentre maggiore è la distanza, maggiore sarà il tempo impiegato per la rilevazione. Nel worst case, il tempo totale di lettura dei tre sensori ultrasonici è di circa 69 ms, che viene approssimato a 70 ms.

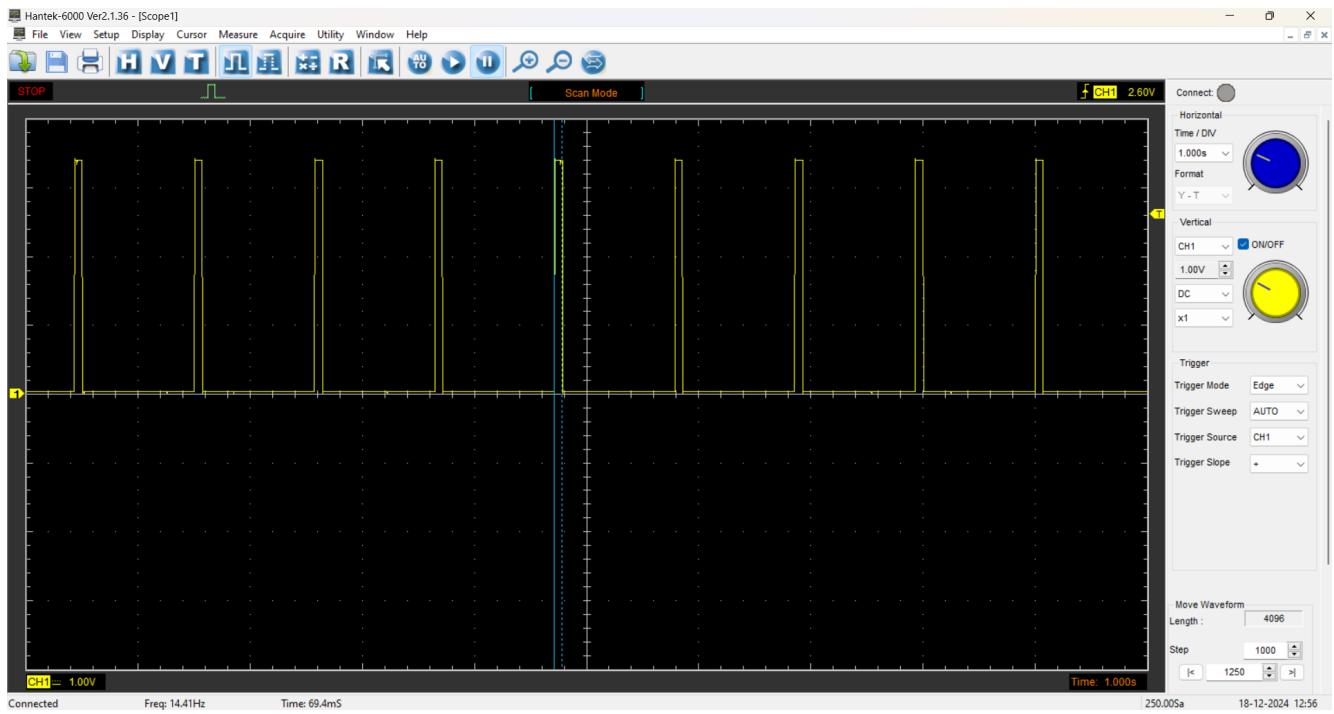


Figure 9.4: Tempo impegno per la lettura dei dati del sonar

Lettura dati controller

Per la ricezione dei dati dal controller, che avviene attraverso I2C e bluetooth, il tempo medio impiegato è di 126 μ s, che viene sovrastimato a 1 ms.

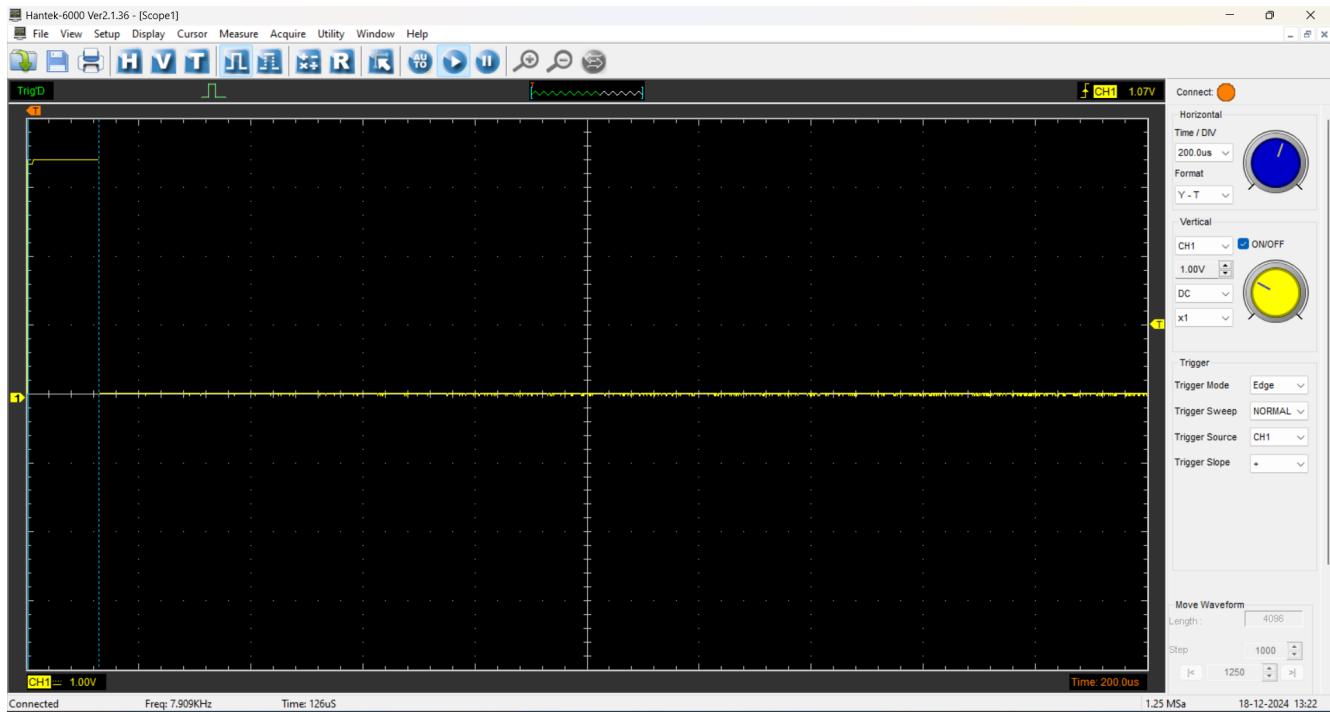


Figure 9.5: Tempo impiegato per la ricezione dei dati dal controller

Confronto tra le schede

Funzione	WCET
Temperatura	1 ms
Encoder	1 ms

Figure 9.6: Board 1

Funzione	WCET
Temperatura	1 ms
Sonar	70 ms
Controller	1 ms
Accelerometro	1 ms

Figure 9.7: Board 2

Quindi, la board 1 avrà un WCET di 2 ms, mentre la board 2 avrà un WCET di 73 ms. Il divario tra le due schede è notevole, come si può facilmente dedurre dalle diverse misurazioni che ciascuna deve eseguire. In particolare, la board 2 deve gestire anche i dati provenienti dai tre sensori a ultrasuoni, che contribuiscono significativamente al tempo complessivo di lettura.

9.2.3 Task di comunicazione

Il task di comunicazione comprende due passaggi di dati tra le schede: nel primo, il master invia i dati allo slave, e nel secondo, lo slave invia i dati al master. La trasmissione avviene tramite il protocollo SPI, con una velocità di 664 062 kbit/s, che consente una comunicazione significativamente più veloce rispetto all'UART. Dal master allo slave vengono inviati 33 byte mentre dallo slave al master vengono inviati solamente 15 byte, dopodiché in base allo stato globale si decide con che task procedere. Il tempo complessivo impiegato tra comunicazione, sincronizzazione e decisione è di circa 0.5 ms. Consideriamo, quindi, come WCET 1 ms per il master. Per lo slave, la situazione è diversa: all'inizio di questo task, lo slave deve attendere che il master invii il messaggio. Come discusso in precedenza, il task di lettura del master può durare fino a 73 ms. Questo implica che, nel worst case, lo slave è costretto ad aspettare per 73 ms meno il tempo necessario alla propria lettura. Di conseguenza, il WCET dello slave verrà considerato pari a 73 ms, riflettendo il tempo massimo in cui lo slave potrebbe rimanere in attesa del master.

9.2.4 Task di attuazione

Se la lettura e la comunicazione avvengono correttamente, e se entrambe le schede risultano funzionanti, allora può essere avviato il task di attuazione normale, presente esclusivamente sulla board 1 (slave). Questo task include, oltre all'attuazione sui led, tre operazioni principali: il calcolo del setpoint, il calcolo del controllore PID (Proportional-Integral-Derivative), e la trasmissione dei valori calcolati ai motori tramite il protocollo UART.

Per il calcolo del set point si impiega mediamente 2.3 μ s.

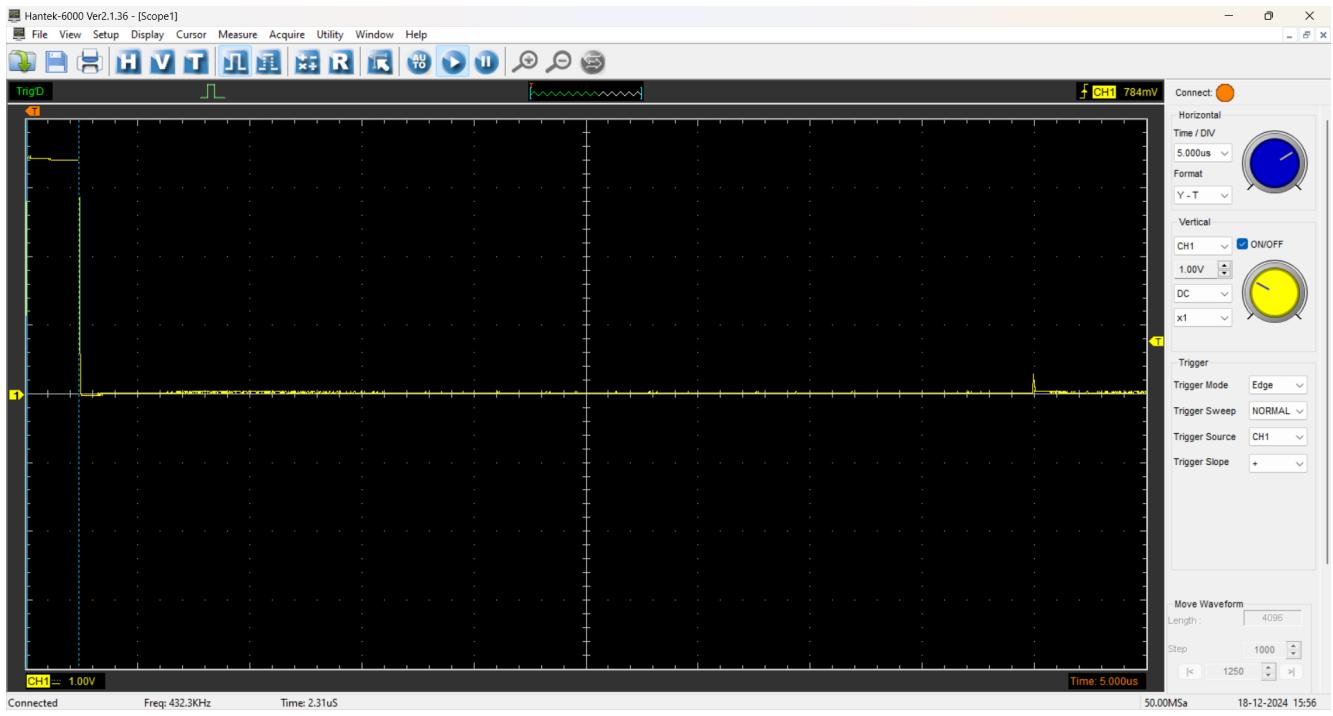


Figure 9.8: Tempo impiegato per il calcolo del setpoint

Per calcolare il pid ed inviare i valori ai motori il tempo impiegato è di circa $128 \mu\text{s}$

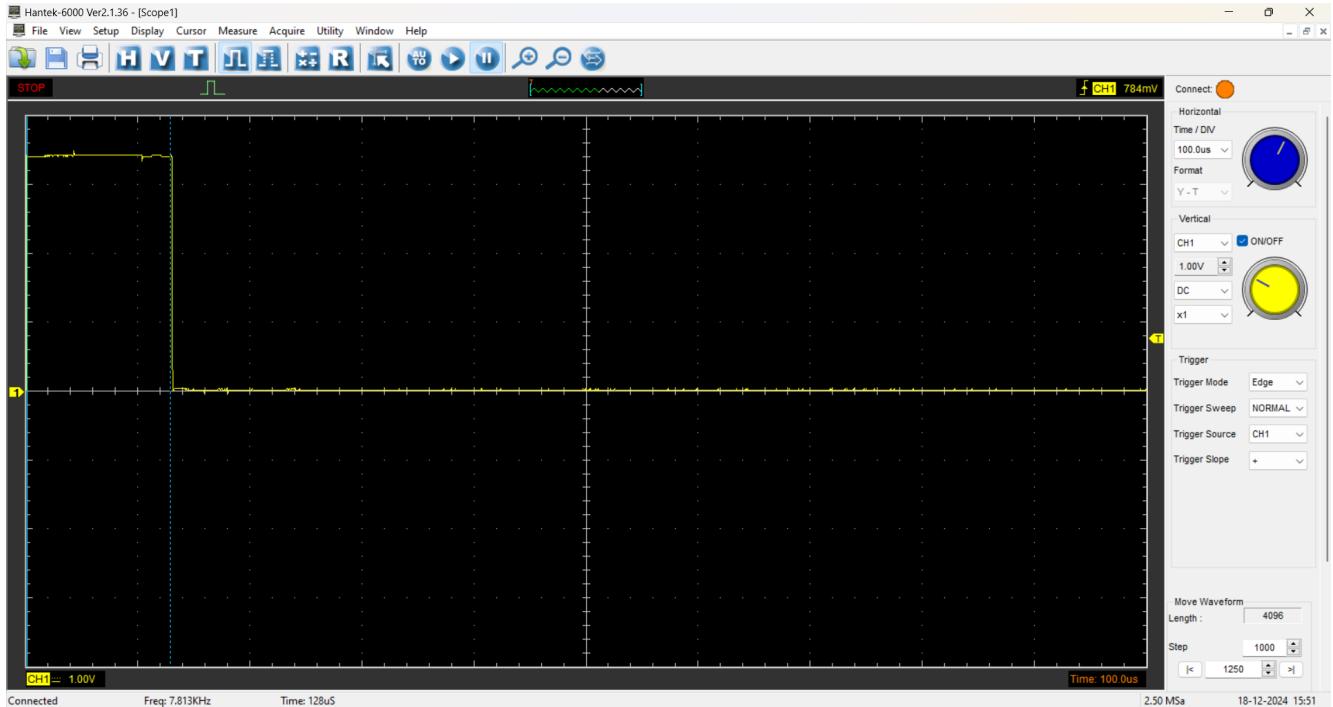


Figure 9.9: Tempo impiegato per il calcolo del valore di un singolo pid e la trasmissione ai motori

Il tempo impiegato, quindi, dall'intero task è di circa $130 \mu\text{s}$, verrà quindi considerato un WCET di 1 ms.

9.2.5 Task di attuazione degradata

In questo task, presente esclusivamente sulla scheda master, la mobilità del rover è limitata: la velocità massima del rover può raggiungere solo 20 g/min, qualora venga impartito il comando. In questa modalità, i valori vengono inviati ai motori senza eseguire il calcolo del set point o del PID. Il tempo necessario per completare questo task è di circa 50 μ s, ma per il WCET si considera un valore sovrastimato di 1 ms.

9.2.6 Task di emergenza

Questo task viene eseguito su entrambe le schede in situazioni differenti, ma in ogni caso svolge le stesse operazioni. La sua funzione principale è fermare immediatamente i motori al momento dell'attivazione, inviando i valori di blocco agli stessi. Il tempo necessario per completare questa operazione è di circa 40 μ s. Tuttavia, per il WCET viene considerato un valore sovrastimato di 1 ms.

9.2.7 Tempi finali

Si può osservare, quindi, che il master presenta un WCET complessivo di 76 ms, principalmente attribuibile alla rilevazione dei dati dai sensori ultrasonici. D'altra parte, lo slave ha un WCET complessivo di 77 ms, con la maggior parte del tempo trascorso in attesa del master durante l'esecuzione del task di comunicazione.

Task	WCET
Lettura(master)	73 ms
Lettura(slave)	1 ms
Comunicazione(master)	1 ms
Comunicazione(slave)	73 ms
Attuazione(slave)	1 ms
Degrado(master)	1 ms
Emergenza	1 ms

Table 9.1: WCET

9.3 Implementazione algoritmo di schedulazione

Per entrambe le schede il ciclo di task da eseguire è sempre lo stesso:

1. Lettura dei dati dai sensori;
2. Comunicazione dei dati rilevati all'altra scheda, decisione attuazione;
3. Attuazione/Emergenza.

L'ordine di esecuzione dei task è quello indicato nell'elenco e deve essere rigorosamente rispettato. Questo perché:

- Non è possibile comunicare i dati senza aver prima raccolto le informazioni dai sensori;
- Non è possibile effettuare l'attuazione senza avere a disposizione i dati dell'intero sistema.

Per questi motivi, la scelta dell'algoritmo di schedulazione è ricaduta sul timeline scheduling, un metodo ampiamente utilizzato nei sistemi militari grazie alla sua rigorosità e sicurezza. Come **vantaggi** ha:

- **Determinismo:** consente di sapere in ogni momento quale task è in esecuzione, garantendo prevedibilità;
- **Ordine garantito:** permette di mantenere sempre l'ordine del ciclo di task, che è fondamentale per il corretto funzionamento del sistema .

Mentre come **svantaggi** ha:

- **Basso utilizzo del processore:** l'algoritmo può non sfruttare a pieno le risorse del processore;
- **Limitazioni nei task aperiodici:** presta poca attenzione ai task aperiodici che potrebbero arrivare durante l'esecuzione.

Nel contesto specifico del rover, il problema dei task aperiodici è irrilevante, poiché il sistema non prevede task di questo tipo; di conseguenza i benefici del timeline scheduling superano ampiamente i suoi svantaggi, rendendolo la scelta ideale. È possibile impostare come algoritmo di schedulazione il TS su FreeRTOS andando a settare preemption e time slicing uguali a zero. In questo modo i task non potranno essere interrotti e avranno un ordine preciso.

Per ciascuna scheda sono previsti tre tipi di cicli dei task, in base allo stato del sistema:

- OK_STATE: Quando il sistema funziona correttamente, la scheda slave esegue il task di attuazione, mentre la scheda master non esegue alcuna attuazione. Una volta completata la comunicazione, il ciclo della scheda master termina.



Figure 9.10: Schedulazione master senza attuazione

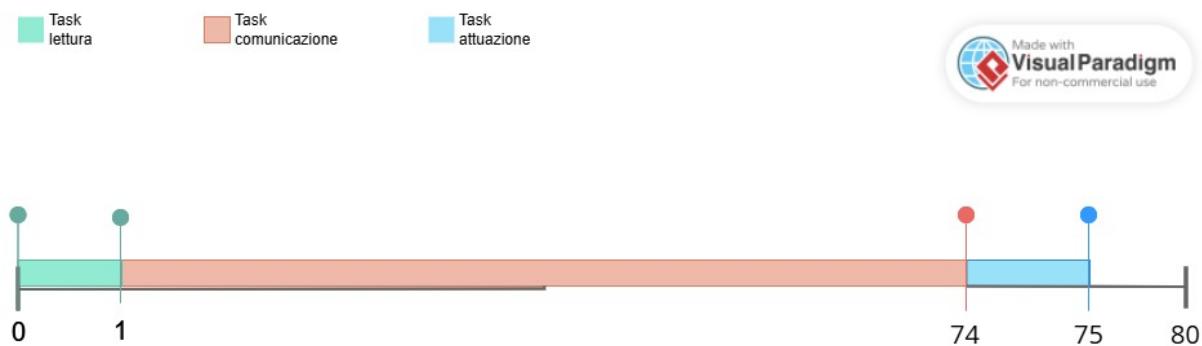


Figure 9.11: Schedulazione slave

- DEGRADED_STATE: In presenza di un guasto a un componente che compromette il controllo della velocità, la scheda master esegue il task di attuazione degradata dopo la comunicazione, mentre la scheda slave non esegue alcuna attuazione.

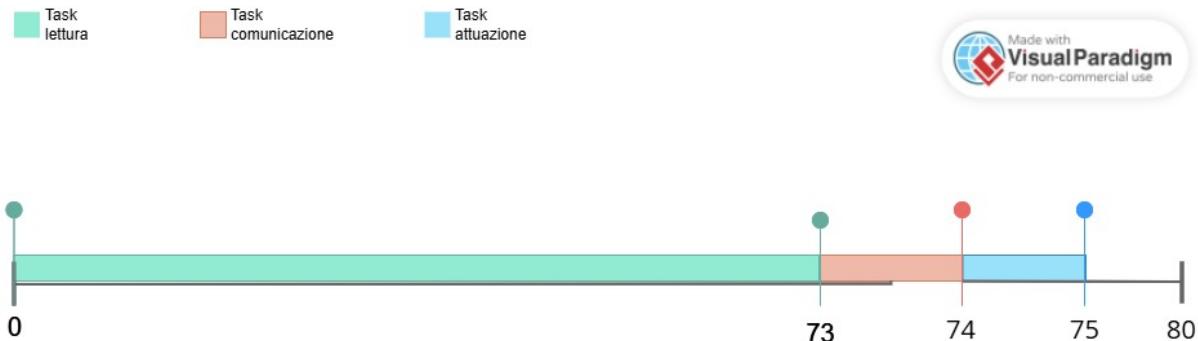


Figure 9.12: Schedulazione master



Figure 9.13: Schedulazione slave senza attuazione

- EMERGENCY_STATE: In questo stato viene eseguito il task di emergenza. Come descritto nei capitoli precedenti, entrambe le schede sono in grado di eseguire il task, ma solo una lo farà, a seconda delle condizioni specifiche del sistema.



Figure 9.14: Emergenza master

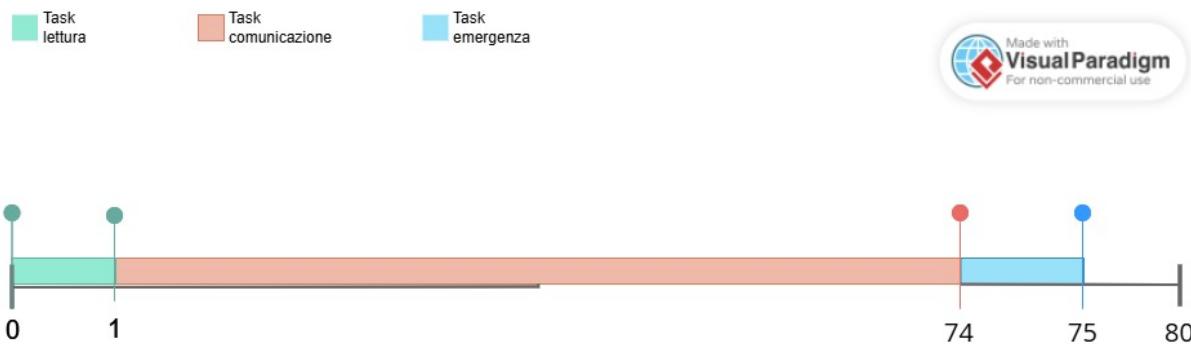


Figure 9.15: Emergenza slave

Questo comportamento è stato confermato anche tramite l'utilizzo del debugger, verificando che i cicli dei task corrispondessero correttamente ai vari stati del sistema.

9.3.1 Impostazioni RTOS

Di seguito sono elencate le modifiche apportate alle impostazioni predefinite di FreeRTOS:

- configUSE_PREEMPTION impostato a 0: il meccanismo di preemption è stato disattivato poiché non necessario nel contesto del timeline scheduling;
- configUSE_TIME_SLICING impostato a 0: il meccanismo di time slicing è stato disattivato in quanto distribuisce equamente il tempo tra task con la stessa priorità, ma in questo caso tutti i task hanno priorità differenti.

Chapter 10

Comunicazione

10.1 Introduzione

Nel progetto del rover, la comunicazione tra le due schede STM32 (B2 e B1) è stata implementata utilizzando il protocollo SPI. Questa scelta è stata motivata da esigenze specifiche del sistema, tra cui la necessità di una comunicazione veloce, bidirezionale ed efficiente tra i moduli del rover.

10.2 Motivazione della Scelta di SPI

Minima Latenza e Alta Velocità

SPI è un protocollo estremamente veloce rispetto a UART. Poiché il rover necessita di elaborare dati in tempo reale (come la posizione dei joystick e la distanza dei sonar), l'uso di SPI ha permesso una comunicazione a bassa latenza, evitando ritardi che avrebbero potuto compromettere il controllo del sistema.

Sincronizzazione Affidabile

L'uso del segnale di clock condiviso tra le due schede ha garantito una perfetta sincronizzazione tra il master e lo slave, riducendo il rischio di errori nella trasmissione dei dati. Con UART, l'assenza di un clock condiviso avrebbe richiesto un'accurata calibrazione del baud rate tra le schede, aumentando la probabilità di errori dovuti a variazioni di frequenza.

Struttura Gerarchica del Sistema

La logica Master-Slave di SPI si adatta perfettamente all'architettura del rover, dove la scheda B2 funge da unità di controllo centrale e la scheda B1 esegue le operazioni richieste. Questo ha

semplificato la gestione del flusso di dati, permettendo alla B2 di prendere decisioni basate sulle informazioni ricevute dalla B1.

Controllo del Flusso e Sicurezza nella Trasmissione

Nel progetto, è stato implementato un meccanismo di handshake tramite i segnali **B1_pin** e **B2_pin**, garantendo che la trasmissione avvenga solo quando entrambe le schede sono pronte. Questo ha ridotto il rischio di perdita di pacchetti e migliorato l'affidabilità della comunicazione.

Espandibilità del Sistema

SPI consente di collegare più dispositivi sulla stessa linea tramite il segnale di selezione dello slave (**CS/SS**). Sebbene attualmente la comunicazione avvenga solo tra B2 e B1, questa caratteristica rende il sistema facilmente scalabile in caso di aggiunta di nuovi moduli, come ulteriori sensori o unità di elaborazione.

10.3 Confronto dei Tempi di Trasmissione

Per valutare il guadagno in termini di velocità, è necessario confrontare i tempi di trasmissione tra UART e SPI per l'invio dei seguenti dati. È importante notare che, nel caso della comunicazione UART, il baud rate è stato impostato al massimo a 9600 baud, a causa di limitazioni legate alle configurazioni di clock del sistema. Infatti, quando si utilizzano altre interfacce di comunicazione come I2C o altre porte UART, le risorse del sistema, inclusi i clock e le abilitazioni hardware, possono causare una riduzione della velocità di trasmissione. Queste limitazioni, sebbene siano progettate per garantire la compatibilità e l'affidabilità del sistema, influenzano direttamente le prestazioni della UART rispetto ad altre tecnologie di comunicazione più veloci come SPI.

Di seguito sono riportati i tempi di trasmissione e la quantità di dati da inviare:

- **Velocità di trasmissione**
 - **UART**: Baud rate = 115200 baud
 - **SPI**: Bit rate = 664.062 kbit/s = 664062 bit/s
- **Dati da trasmettere:**
 - B2(Master) → B1(Slave): 33 byte
 - B1(Slave) → B2(Master): 16 byte
 - B2(Master) → B1(Slave): 1 byte

10.4 Calcolo del Tempo di Trasmissione

Nella sezione precedente è stato definito cosa dovesse essere inviato e quale fosse la velocità di trasmissione. Ora, invece, si andrà a calcolare il tempo effettivo necessario per la trasmissione dei dati, tenendo conto delle specifiche velocità e delle caratteristiche dei protocolli utilizzati.

10.4.1 UART

Il tempo di trasmissione di un byte in UART si calcola come:

$$T_{\text{byte}} = \frac{10 \text{ bit}}{\text{baud rate}} \quad (10.1)$$

Poiché ci sono 10 bit, di cui 8 bit per i dati, 1 start bit e 1 stop bit per ogni byte trasmesso:

$$T_{\text{byte}} = \frac{10}{115200} = 0.0868 \text{ ms/byte} \quad (10.2)$$

Ora calcoliamo il tempo totale per ogni trasferimento:

$$T_{B2 \rightarrow B1} = 33 \times 0.0868 = 2.8684 \text{ ms}$$

$$T_{B1 \rightarrow B2} = 16 \times 0.0868 = 1.3888 \text{ ms}$$

$$T_{B2 \rightarrow B1} = 1 \times 0.0868 = 0.0868 \text{ ms}$$

10.4.2 SPI

Il tempo di trasmissione di un byte in SPI si calcola come:

$$T_{\text{byte}} = \frac{8 \text{ bit}}{\text{bit rate}} \quad (10.3)$$

Quindi:

$$T_{\text{byte}} = \frac{8}{664062} = 12.05 \mu\text{s} = 0.01205 \text{ ms} \quad (10.4)$$

Ora calcoliamo il tempo totale per ogni trasferimento:

$$T_{B2 \rightarrow B1} = 33 \times 0.01205 = 0.398 \text{ ms}$$

$$T_{B1 \rightarrow B2} = 16 \times 0.01205 = 0.193 \text{ ms}$$

$$T_{B2 \rightarrow B1} = 1 \times 0.01205 = 0.01205 \text{ ms}$$

10.5 Risultati Finali

Trasferimento	UART (ms)	SPI (ms)
B2 → B1 (33 byte)	2.8684	0.398
B1 → B2 (16 byte)	1.3888	0.193
B2 → B1 (1 byte)	0.0868	0.012
Totale	4.344	0.603

La scelta di SPI per il progetto del rover è stata determinata dalla necessità di una comunicazione veloce, affidabile e bidirezionale tra le due schede STM32. Le caratteristiche di **full-duplex, sincronizzazione con clock, bassa latenza e scalabilità** hanno reso SPI il protocollo ideale rispetto a UART, che avrebbe introdotto limitazioni significative in termini di velocità e affidabilità. L'analisi dei tempi di trasmissione ha mostrato un miglioramento significativo: SPI riduce il tempo totale di trasmissione da **4.34 ms (UART)** a soli **0.60 ms**, garantendo una maggiore reattività e un sistema più efficiente.

10.6 Implementazione codice comunicazione

Dopo un'attenta analisi sul metodo di scelta del protocollo di comunicazione, è stato implementato il codice per realizzare la comunicazione tra le due schede.

10.6.1 Configurazione dei parametri SPI

Successivamente, è stato abilitato il protocollo SPI su entrambe le schede, B2 e B1.

Il protocollo SPI sulla scheda B2 che funge da master con i seguenti parametri:

- **Mode** Full-Duplex Master
- **Hardware NSS Signal Disable**
- **Basic Parameters:**
 - **Frame format:** Motorola
 - **Data Size:** 8 Bits
 - **Firts bit:** MSB First
- **Clock Parameters:**
 - **Prescaler:** 256
 - **Clock Polarity:** Low
 - **Clock Phase:** 1 Edge

- **Advanced Parameters:**
 - **CRC Calculation:** Disabled
 - **NSSP MODE:** Enabled
 - **NSS Signal Type:** Software

Il protocollo SPI sulla scheda B1 che funge da Slave con i seguenti parametri:

- **Mode** Full-Duplex Slave
- **Hardware NSS Signal Disable**
- **Basic Parameters:**
 - **Frame format:** Motorola
 - **Data Size:** 8 Bits
 - **Firts bit:** MSB First
- **Clock Parameters:**
 - **Clock Polarity:** Low
 - **Clock Phase:** 1 Edge
- **Advanced Parameters:**
 - **CRC Calculation:** Disabled
 - **NSS Signal Type:** Software

Con tali parametri, le due schede non prevedono l'utilizzo di un **NSS signal**, che sarebbe normalmente utile per selezionare lo **slave**, in quanto in questo caso è presente un solo **slave**. Inoltre, entrambe le schede sono impostate in modalità **full-duplex**, consentendo lo scambio dei dati in entrambe le direzioni. La **data size** dei pacchetti scambiati tra i due dispositivi è impostata a **8 bit**, in modo tale che sia possibile inviare variabili di tipo **uint8_t** tramite un singolo pacchetto, senza doverle dividere. Il **clock di sistema** è di **170 MHz**, e impostando un **prescaler** di 256, è stato ottenuto un **baud rate** per la comunicazione di **664.062 Kbit/s**, che consente una trasmissione ad alta velocità.

10.6.2 Sincronizzazione trasmissione

In questa sezione si andrà ad analizzare il meccanismo di sincronizzazione utilizzato.

Sono stati utilizzati due pin `B1_pin` e `B2_pin` essenziali per la sincronizzazione della comunicazione tra il master e lo slave. Entrambi i pin sono utilizzati in modo complementare per coordinare il flusso di dati tra le due schede, evitando conflitti e garantendo che entrambe le schede siano pronte per ricevere o inviare dati.

Pin `B1_pin` e `B2_pin`: Riflessione generale

- Il pin `B1_pin` del master(input) è collegato al pin `B1_pin` dello slave(output).
- Il pin `B2_pin` del master(output) è collegato al pin `B2_pin` dello slave(input).

Questo schema di connessione consente una comunicazione sincronizzata in cui ogni scheda è in grado di “segnalare” all’altra quando è pronto per inviare o ricevere dati, creando un flusso ordinato di informazioni.

Funzione dei Pin `B1_pin` e `B2_pin` nella Comunicazione

Invio dei dati

Invio dati Master o Slave:

- La scheda che trasmette prepara i dati da inviare tramite SPI e attiva il proprio pin (impostandolo a livello alto), segnalando che è pronta ad inviare.
- La scheda trasmettente controlla continuamente l’altro pin per verificare se lo la scheda ricevente è pronta a ricevere i dati. Se è basso (indicando che la scheda ricevente non è ancora pronta), quella trasmettente rimarrà in attesa.
- Una volta che la ricevente è pronta e il pin diventa alto, il trasmettente invia i dati.
- Dopo aver inviato i dati tramite `HAL_SPI_Transmit`, il trasmettente resetta il proprio pin (impostandolo a livello basso), segnalando che la trasmissione è finita.

Ricezione dei dati

Ricezione dati Master o Slave:

- La scheda ricevente attende che il trasmettitore invii un segnale di sincronizzazione, monitorando il pin che arriva in input dall’altra scheda. Quando diventa alto, la scheda ricevente sa che è pronta per ricevere i dati.

- La scheda ricevente imposta il proprio pin a livello alto per segnalare al trasmettitore che è pronta a ricevere i dati.
- Dopo aver ricevuto i dati tramite `HAL_SPI_Receive`, la scheda ricevente disattiva il pin (impostandolo a livello basso), indicando che ha terminato la ricezione.

Timeout e gestione degli errori

In entrambi i dispositivi, vengono imposti dei timeout per evitare che la comunicazione rimanga bloccata in attesa di segnali da parte dell'altro dispositivo. Se la comunicazione non si completa entro un determinato intervallo di tempo, il processo si interrompe e lo stato delle schede viene modificato per riflettere il fallimento della sincronizzazione.

I pin `B1_pin` e `B2_pin` sono utilizzati per garantire che entrambi i dispositivi (master e slave) siano sincronizzati prima di iniziare o proseguire con una trasmissione di dati. Questo metodo di sincronizzazione a livello di hardware assicura che ogni scheda attenda che l'altra sia pronta, prevenendo errori e conflitti nella trasmissione.

10.7 Serialize e Deserialize

Prima di presentare l'intero flusso di esecuzione del task di comunicazione, è necessario introdurre due funzioni fondamentali per la comunicazione SPI. Come analizzato in precedenza, i pacchetti scambiati tra due schede sono composti da 8 bit. Questo rappresenta un problema per lo scambio di dati a 16 bit, come quelli utilizzati dal controller. Inoltre, i dati non possono essere inviati direttamente tramite strutture dati (`struct`), pertanto devono essere suddivisi e inseriti in un buffer di invio per garantire una trasmissione corretta.

Per risolvere questa problematica, sono state implementate due funzioni, `serialize` e `deserialize`, utilizzate sia dal master che dallo slave.

Serialize

La funzione `serialize` si occupa di convertire strutture dati complesse in una sequenza di byte che possono essere trasmessi tramite il protocollo SPI. Questa operazione prevede la suddivisione dei dati a 16 bit in due pacchetti da 8 bit ciascuno, in modo da adattarsi alle specifiche della comunicazione.

Durante questo processo:

- I valori numerici vengono suddivisi nei loro byte costituenti.
- I dati vengono inseriti in un buffer che verrà successivamente inviato sulla linea di comunicazione.
- Un dato a 16 bit viene diviso in due byte: il byte più significativo (MSB) e il byte meno significativo (LSB). Nel caso specifico, il byte più significativo viene inviato per primo, seguito dal byte meno significativo.
- L'ordine dei byte viene gestito per garantire la corretta ricostruzione del dato durante la deserializzazione.

Deserialize

La funzione `deserialize` ha il compito opposto: ricostruire i dati originali a partire dai pacchetti ricevuti. Questa operazione consente di riassemblare i valori a 16 bit unendo i due byte ricevuti separatamente e assegnandoli alle corrispondenti variabili di sistema.

Durante il processo di deserializzazione:

- I byte ricevuti vengono letti in sequenza e ricombinati in variabili a 16 bit quando necessario.

- I valori vengono assegnati alle strutture dati che rappresentano lo stato del controller e dei sensori.
- I dati vengono interpretati nel corretto ordine per mantenere la coerenza delle informazioni trasmesse.
- Durante la deserializzazione, sia nel master che nello slave, vengono direttamente ricostruiti gli stati globali delle due schede. Questo significa che i dati ricevuti dal buffer, che rappresentano le informazioni mancanti dell'altra scheda, vengono combinati con i dati computati localmente e assegnati direttamente alla struttura globale. Questo approccio riduce il numero di passaggi necessari, migliorando l'efficienza e riducendo il carico computazionale.

Queste due funzioni sono essenziali per garantire un trasferimento dati affidabile tra i dispositivi master e slave, evitando inconsistenze nei valori trasmessi e ricevuti.

10.8 Flusso Completo di Comunicazione

Fase 1: Inizio della Comunicazione (Master)

Il master avvia la comunicazione serializzando i dati e li invia allo slave utilizzando l'interfaccia SPI. Durante questa fase, il master attiva il pin di sincronizzazione `B1_Pin` per sincronizzare l'operazione con lo slave, e verifica che lo slave sia pronto per ricevere i dati attraverso il pin `B2_Pin`. Se lo slave non è pronto entro un certo periodo di tempo (timeout), il master termina l'operazione e imposta lo stato come errore.

Fase 2: Ricezione dei dati dal Master (Slave)

Quando lo slave è pronto, riceve i dati inviati dal master tramite SPI. Lo slave deserializza i dati ricevuti e aggiorna il proprio stato in base ai valori appena acquisiti. Questo passaggio è cruciale poiché garantisce che entrambi i dispositivi (master e slave) abbiano informazioni coerenti e sincronizzate sui rispettivi stati. Una volta che lo slave ha ricevuto e ricostruito i dati, ha già lo stato globale del sistema e può prendere una decisione. La decisione viene calcolata tramite una serie di if in cascata che presentano 4 opzioni:

- **Decisione 0 (Emergenza sullo slave):** Quando lo stato di `B2` è 0 (non funzionante/Errore) e lo stato di `B1` è 1 (funzionante).
- **Decisione 1 (Attuazione sullo slave):** Quando lo stato di `B2` è 1 (funzionante) e lo stato di `B1` è 1 (funzionante).
- **Decisione 2 (Degradato sul master):** Quando lo stato di `B2` è 1 (funzionante) e lo stato di `B1` è 0 (non funzionante/Errore).

- **Decisione 3 (Emergenza sul master):** Quando lo stato di B2 è 0 (non funzionante/Errorre) e lo stato di B1 è 0 (non funzionante/Errorre).

Tale decisione viene aggiunta alla struct parziale dello slave da inviare al master.

Fase 3: Invio Dati dallo Slave al Master (Slave → Master)

Lo slave, una volta aver fatto la propria decisione e pronto, serializza i propri dati (utilizzando `serialize_partial`) e trasmette il buffer contenente le informazioni e la decisione tramite SPI. Per garantire la corretta sincronizzazione, lo slave attiva il pin `B1_Pin` per segnalare al master che i dati sono pronti per essere inviati. Lo slave attende poi che il master legga i dati. Se il master non è pronto a ricevere i dati, lo slave gestisce un timeout e termina l'operazione se il tempo limite viene superato.

Una volta che il master riceve il buffer tramite SPI, il master deserializza i dati e prosegue con il processo di analisi per prendere la decisione finale.

Fase 4: Elaborazione della Decisione (Master) e invio decisione finale

Master: Il master elabora i dati ricevuti dal slave e decide il tipo di azione da intraprendere. Il metodo per prendere la decisione nel master è lo stesso utilizzato nella fase 2 dello slave, ossia una serie di 4 if in cascata. Successivamente si confronta la decisione presa e quella inviatagli dello slave e, se sono uguali, invia tramite meccanismo di sincronizzazione spiegato in precedenza e SPI un byte contenente 1, permettendo così allo slave di proseguire con la sua normale esecuzione. Se le decisioni sono diverse, il master invia allo slave un byte contenente 0, indicando allo slave di fermare il proprio meccanismo di attuazione. A questo punto, il master prende il controllo e, se il sistema non è in stato di errore, passa in stato degradato.

La fase 4 è l'ultima fase del ciclo e il sistema implementato permette una comunicazione affidabile tra il master e lo slave, utilizzando una combinazione di SPI per il trasferimento dei dati e GPIO per la sincronizzazione. Ogni dispositivo ha un ruolo specifico nella comunicazione: il master invia i dati iniziali e riceve la risposta finale dallo slave, mentre lo slave elabora i dati ricevuti e invia le proprie decisioni al master. La gestione dei timeout e la sincronizzazione tramite i pin `B1_Pin` e `B2_Pin` consentono di garantire che la comunicazione avvenga correttamente e senza interruzioni. Il meccanismo di timeout aiuta a prevenire situazioni di stallo nel caso in cui i dispositivi non riescano a comunicare in tempo.

Chapter 11

Utilizzo dei relè per la trasmissione dei dati

I relè sono dispositivi elettromeccanici o a stato solido utilizzati per controllare il passaggio della corrente in un circuito tramite un segnale di comando. Funzionano come interruttori comandati elettricamente: quando ricevono un impulso, attivano o disattivano un circuito senza bisogno di un intervento manuale. In questo progetto, i relè giocano un ruolo fondamentale nella gestione dei comandi inviati ai motori.

11.1 Relè utilizzati

I relè utilizzati sono di tipo 5V SRD-05VDC-SL-C, ed hanno sei terminali:

- S (Signal): Questo è il segnale di controllo. Quando riceve un impulso dalla scheda di controllo, attiva il relè.
- + (VCC): Questo è il positivo della bobina del relè, solitamente collegato a 5V o 12V a seconda del modello.
- - (GND): Questo è il negativo della bobina del relè, da collegare a massa (GND).
- NC (Normally Closed): Contatto normalmente chiuso: in assenza di segnale sul relè, è collegato a COM.
- NO (Normally Open): Contatto normalmente aperto: quando il relè è attivato, si collega a COM.
- COM (Common): Contatto comune, il punto di ingresso del segnale che verrà indirizzato a NC o NO a seconda dello stato del relè.



Figure 11.1: Relè

11.2 Trasmissione comandi

Come spiegato nei capitoli precedenti, i comandi ai motori possono essere trasmessi sia dalla scheda master sia dalla scheda slave. Tuttavia, per garantire un funzionamento corretto ed evitare comportamenti imprevisti o sovraccarichi del sistema, è fondamentale che i motori ricevano i comandi da una sola scheda alla volta. Per gestire questa selezione, sono stati utilizzati due relè collegati alle due sabertooth, che permettono di abilitare solo una delle due schede in un determinato momento. Questo accorgimento è particolarmente utile nel caso in cui una delle schede vada in errore o si verifichino problemi di comunicazione. In tali situazioni, senza un sistema di controllo, entrambe potrebbero tentare di inviare comandi contemporaneamente, causando conflitti e possibili malfunzionamenti. Grazie all'uso dei relè, se la comunicazione tra le schede non avviene correttamente o una delle due riscontra un problema, il sistema garantisce che il comando venga inviato esclusivamente dalla scheda master, assicurando così un funzionamento affidabile. Più nel dettaglio:

- Se la scheda master rileva che la decisione finale è 1, ovvero entrambe le schede funzionano correttamente e hanno preso la stessa decisione, allora il pin collegato al terminale S del relè viene attivato.
- Se la decisione finale è 0 o si verificano problemi di comunicazione, il pin S viene disattivato, garantendo che il comando provenga unicamente dalla scheda master.

Chapter 12

Specifiche

12.1 Temperatura

Nel progetto è stato realizzato il controllo della temperatura, una specifica importante che ci permette di verificare se una delle due schede supera una determinata soglia per evitare guasti e malfunzionamenti del rover.

Per la realizzazione è stato utilizzato un ADC, in particolare l'ADC1, e sono stati attivati i seguenti parametri:

- **Temperature sensor channel:** questo canale permette di acquisire il valore di tensione fornito dal sensore di temperatura interno, che verrà poi utilizzato per calcolare la temperatura effettiva.
- **Vrefint Channel:** questo canale consente di leggere il riferimento interno di tensione del microcontrollore, utile per migliorare la precisione delle misurazioni compensando eventuali variazioni dell'alimentazione.

Una volta attivati tali parametri, è stato scritto il codice per la rilevazione della temperatura in apposite librerie chiamate `temperature.h` e `temperature.c`.

Nell'intestazione `temperature.h` sono state definite le intestazioni delle funzioni, in particolare:

- `void TemperatureSensor_Init(void);` funzione per l'inizializzazione del sensore di temperatura.
- `int16_t TemperatureSensor_Read(void);` funzione per la lettura della temperatura.

Mentre nella libreria `temperature.c` è stato sviluppato il codice per tali funzioni.

Per la prima funzione, `TemperatureSensor_Init`, viene eseguita la calibrazione dell'ADC per garantire misurazioni accurate. In particolare, la funzione `HAL_ADCEx_Calibration_Start` viene utilizzata per avviare la calibrazione in modalità single-ended. Se la calibrazione ha successo, viene stampato un messaggio di conferma, altrimenti viene segnalato un errore.

Per la seconda funzione, `TemperatureSensor_Read`, viene avviata la conversione ADC per acquisire il valore grezzo dal sensore di temperatura. Una volta ottenuto il valore, viene convertito in tensione utilizzando il riferimento di tensione interno. Successivamente, attraverso i valori di calibrazione memorizzati nel microcontrollore (`TS_CAL1` e `TS_CAL2`, corrispondenti alle temperature di 30°C e 110°C), viene calcolata la temperatura effettiva in gradi Celsius. Se la lettura non va a buon fine, la funzione restituisce un valore di errore (-1).

I dati di calibrazione vengono letti direttamente dai registri di memoria:

```
#define VREFINT_CAL_ADDR      ((uint16_t*) (0x1FFF75AAUL))
// VrefInt ADC raw data a 30°C

#define VREFINT_CAL_VREF      (3300UL)
// Vref+ durante la calibrazione

#define TEMPSENSOR_CAL1_ADDR  ((uint16_t*) (0x1FFF75A8UL))
// TS_CAL1 a 30°C

#define TEMPSENSOR_CAL2_ADDR  ((uint16_t*) (0x1FFF75CAUL))
// TS_CAL2 a 110°C

#define TEMPSENSOR_CAL1_TEMP  (30L)
// Temperatura corrispondente a TS_CAL1 (°C)

#define TEMPSENSOR_CAL2_TEMP  (110L)
// Temperatura corrispondente a TS_CAL2 (°C)
```

La formula di conversione utilizzata per calcolare la temperatura in base al valore letto dall'ADC è:

$$T = \frac{(T_{110} - T_{30})}{(TS_{110} - TS_{30})} \times (TS_{ADC} - TS_{30}) + T_{30} \quad (12.1)$$

Dove:

- T_{30} e T_{110} sono le temperature di calibrazione memorizzate in fabbrica (30°C e 110°C).

- TS_{30} e TS_{110} sono i valori ADC letti a tali temperature.
- TS_{ADC} è il valore ADC attuale.

Questa formula permette di interpolare linearmente la temperatura sulla base dei valori forniti dal sensore, garantendo una buona precisione nelle misurazioni.

Tale funzione viene chiamata nel task di lettura di entrambe le board e, nel caso in cui per 200 cicli di lettura venga rilevato un valore superiore ai 90 gradi, il sistema entra in errore. In questo modo possiamo monitorare la temperatura ed evitare un guasto del sistema.

12.2 Sonar

Nel progetto è stata realizzata la lettura di tre sonar, fondamentali per evitare che il rover vada a sbattere contro ostacoli. Per il funzionamento di tali sensori è stato necessario attivare sei pin sulla scheda master, in particolare tre di input per l'echo e tre di output per il trigger. Inoltre, sono stati impostati tre timer, uno per ogni sonar, utilizzati per la rilevazione della distanza.

I sonar necessitano di un timer con una frequenza di 1 MHz per funzionare correttamente. Considerando che il clock di sistema ha una frequenza di 170 MHz, è stato necessario ridurre la frequenza tramite il prescaler e il counter period. In particolare, tutti e tre i timer sono stati configurati con un prescaler di 169 e un counter period di 0, ottenendo così la seguente frequenza:

$$\text{Frequenza} = \frac{F_{\text{clock}}}{(\text{counter period} + 1) \times (\text{prescaler} + 1)} \quad (12.2)$$

Sostituendo i valori:

$$\text{Frequenza} = \frac{170 \times 10^6}{(0 + 1) \times (169 + 1)} = \frac{170 \times 10^6}{170} = 1 \text{ MHz} \quad (12.3)$$

Questo consente ai sonar di funzionare correttamente con la frequenza richiesta.

Sono state implementate poi due librerie `ultrasonic.h` e `ultrasonic.c` per la gestione dei sonar:

Nell'intestazione viene definita una Struttura (`UltrasonicSensor`) che memorizza i dati di configurazione per ciascun sensore ad ultrasuoni, tra cui:

- `trig_port` e `trig_pin`: La porta GPIO e il pin utilizzati per il segnale "TRIG" (che invia un impulso al sensore).
- `echo_port` e `echo_pin`: La porta GPIO e il pin per il segnale "ECHO" (che riceve l'impulso che torna dall'ostacolo).

- `timer`: Un puntatore alla struttura `TIM_HandleTypeDef` per il timer utilizzato per il temporizzatore preciso della durata dell'impulso.
- `error_count`: Tiene traccia del numero di errori consecutivi incontrati durante le misurazioni.
- `last_valid_distance`: Memorizza l'ultima distanza valida misurata per restituirla in caso di errore.

Inoltre nell'intestazione sono definiti i prototipi delle funzioni:

- `Ultrasonic_Init`: Inizializza il sensore ad ultrasuoni con le porte GPIO, i pin e il timer forniti.
- `Ultrasonic_GetDistance`: Misura la distanza inviando un segnale di trigger e calcolando il tempo impiegato per il ritorno del segnale echo. Se la misurazione scade o fallisce, restituisce l'ultima distanza valida o -1 per indicare un errore.

Nel file sorgente `ultrasonic.c` vengono implementate la funzionalità del sensore ad ultrasuoni.

La funzione `Ultrasonic_Init` inizializza una struttura `UltrasonicSensor` memorizzando le porte GPIO, i pin e il timer per il sensore. Inoltre, inizializza il contatore degli errori a 0.

La funzione `Ultrasonic_GetDistance` esegue il processo di misurazione effettiva:

- **Trigger del sensore**: Invia un impulso attraverso il pin `TRIG` e attende 10 microsecondi (usando un timer per un temporizzatore preciso).
- **Attende il segnale ECHO**: Poi, attende che il segnale `ECHO` si alzi (indicando che l'onda ultrasonica ha colpito un ostacolo) e misura il tempo tra il rilascio del trigger e il ritorno del segnale.
- **Calcola la distanza**: La distanza è calcolata in base al tempo impiegato dal segnale per tornare (usando la velocità del suono per la conversione del tempo in distanza).
- **Gestione degli errori**: Se il sensore non riceve il segnale di echo entro un certo limite di tempo (30 ms), incrementa il contatore degli errori e restituisce l'ultima distanza valida.

In questo modo, il sistema è progettato per restituire sempre una distanza valida, anche in caso di errore o timeout.

12.3 Interfacciamento controller e calcolo set point

il controller è collegato alla scheda 1 tramite un microcontrollore ESP32 con funzionalità di bluetooth, che a sua volta è collegato alla scheda STM32 tramite protocollo I2C; la scheda STM calcola poi il set point per i vari PID a partire dai valori ricevuti dal controller.

12.3.1 comunicazione STM-ESP32

Lo scambio di dati tra l'STM32 e l'ESP32 avviene tramite interfaccia I2C, usando tre pin in totale:

- SCL: pin di clock generato dal Master, nel nostro caso l'STM32;
- SDA: pin per i dati da scambiare;
- GND: pin di massa collegata in comune tra le due schede.

È inoltre necessario aggiungere una resistenza di pull-up sia ad SDA che ad SCL per mantenere i pin alti, per cui abbiamo deciso di usare resistenze da $5,1K\Omega$ collegate su una breadboard.

12.3.2 calcolo set point e funzione PID

il set point, che farà da ingresso ai controllori PID per pilotare i motori, è calcolato in base ai valori dei pad del controller tramite la funzione "calculate_wheel_speeds" della libreria "controller.h/controller.c", nel seguente modo:

```
float ratio = (float) (PAD1_y_value - PAD1_y_value_min_su)
/ (PAD1_y_value_max_su - PAD1_y_value_min_su);
int setpoint = ratio * 160.0;
```

si divide il valore del pad letto per il valore massimo che può raggiungere, ottenendo così la variabile "ratio", che viene poi moltiplicata per 160 per ottenere la velocità desiderata dei motori in giri al minuto(RPM). Vengono poi controllati i valori dei pad e il set point viene modificato ad alcune specifiche condizioni:

- se il pad non è stato mosso sull'asse y (su o giù), allora controlla l'asse x;
- se il pad è stato mosso sull'asse x (sinistra o destra), allora sterza nella direzione indicata impostando le velocità rispettive delle ruote a 10 e -10;
- se il pad è stato mosso in giù sull'asse y, allora sterza a destra;
- se il pad è stato mosso in su sull'asse y, allora va in avanti con la velocità impostata in precedenza.

controllore PID

il set point è poi passato in ingresso ai vari controllori PID che pilotano i quattro motori; un PID è inizializzato tramite la funzione "PID_Init" nella libreria "pid.h/pid.c"; a questa funzione vengono passati come parametri i coefficienti per il guadagno proporzionale, integrale e derivativo, che nel nostro caso sono stati impostati a $K_p = 0.001$; $K_i = 0.9$; $K_d = 0$; il valore in uscita del PID è poi calcolato tramite la funzione "PID_Compute" nel seguente modo:

- Calcola l'angolo corrente in base al valore del contatore e alla risoluzione dell'encoder associato al motore;
- Calcola il tempo trascorso in (milli)secondi;
- Calcola la velocità angolare corrente;
- a seconda dei valori del setpoint vengono impostati diversi limiti per l'uscita minima e massima del PID;
- calcola l'errore come differenza tra il set point in ingresso e la velocità angolare calcolata;
- usando l'errore calcola i termini proporzionale, integrale e derivativo e li somma , ottenendo l'uscita;
- se l'uscita va oltre i limiti fisici del motore, modifica l'uscita facendola rientrare nei limiti come descritto nel capitolo 6.

le uscite del PID sono poi trasmesse tramite UART alle schede sabertooth per pilotare i motori come si desidera.

Chapter 13

Montaggio e modalità d'uso del rover

13.1 Componenti

Qui è presentata una lista esaustiva di tutti i componenti utilizzati per la creazione del rover:

- 2: MCU STM32G474RE;
- 1: MCU ESP32;
- 2: relè;
- 3: sensori a ultrasuoni hc-sr04;
- 2: driver per motori duale sabertooth 12A;
- 4: motore 36GP-540-51 con encoder incorporati;
- 1: buzzer;
- 2: A4WD3-LED Board;
- 1: convertitore step down a 5V;
- 1: batteria nihewo 80C 11,1V 3300mAh;
- cavi dupont e jumper MF-MM-FF;
- 1: breadboard;
- 2: resistenze da $5,1\text{k}\Omega$

13.2 Montaggio

Per il montaggio del rover riferirsi al diagramma Fritzing nel capitolo 5. Ricordarsi inoltre di aggiungere resistenze di pull-up sulle linee I2C tra ESP32 e STM32;

13.3 Modalità operative

Il rover è controllato tramite un controller composto da due pad analogici e quattro bottoni, ognuno dei quali può impartire comandi specifici il cui risultato dipende dalla modalità corrente del rover:

- pad sinistro: spostandolo in alto si fa andare il rover in avanti a una velocità proporzionale al valore del pad sull'asse y, mentre spostandolo in basso si fa girare a destra;
- pad destro: spostandolo a destra o a sinistra si fa sterzare il rover in quella direzione, inoltre premendo il pulsante si attiva il buzzer che fa da "clacson";
- bottone rosso: tenendolo premuto si attiva la frenata d'emergenza, facendo fermare tutti i motori e impedendo che si possa farli andare avanti con il pad sinistro;
- bottone nero: premendolo si accendono e spengono i LED bianchi di fronte al rover.
- bottone pad destro: premendolo suona il clacson.

Il pad sinistro non farà andare avanti il rover nel caso in cui il sonar frontale rilevi un ostacolo a meno di 70 centimetri, o uno o entrambi dei sonar laterali rilevino un ostacolo a meno di 23 centimetri, impedendo così al rover di sbatterci contro e sterzare in una direzione libera. Inoltre, anche tenendo il pad alla massima inclinazione in alto, se il rover si trova in stato degradato la velocità massima sarà limitata a un valore più basso impostato nel codice per prevenire che vada a sbattere quando uno o più sonar non funzionano.

Chapter 14

Conclusioni e sviluppi futuri

Il progetto ha permesso di sviluppare un sistema autonomo capace di muoversi e interagire con l'ambiente circostante in modo efficiente e sicuro. L'integrazione di componenti hardware e software ha reso possibile la realizzazione di un rover controllato da remoto, capace di rispondere alle variazioni dell'ambiente grazie all'utilizzo di sensori.

L'adozione di FreeRTOS ha garantito un'esecuzione ottimizzata dei task, permettendo una gestione efficace della comunicazione tra i vari moduli e l'esecuzione parallela delle operazioni. Inoltre, l'implementazione del controllore PID ha migliorato la stabilità del sistema, consentendo un controllo preciso della velocità e della direzione del rover.

Il confronto tra i metodi di controllo dei motori (PWM e UART) ha evidenziato vantaggi e svantaggi di entrambe le soluzioni. La scelta finale di utilizzare UART si è rivelata strategica per semplificare l'implementazione e ridurre i problemi di stabilità dovuti all'uso di filtri RC.

Un altro aspetto fondamentale è stata la comunicazione SPI tra le schede STM32, che ha garantito un trasferimento dati veloce e affidabile. L'uso di SPI ha permesso uno scambio di informazioni tra le schede in tempo reale, migliorando la sincronizzazione delle operazioni e riducendo la latenza nella trasmissione dei comandi. Questo protocollo si è rivelato essenziale per la gestione coordinata dei sensori e degli attuatori, contribuendo in modo significativo alla reattività e alla stabilità complessiva del sistema.

L'analisi delle modalità operative ha dimostrato che il sistema è in grado di gestire scenari complessi, entrando in modalità di sicurezza in caso di guasti o condizioni critiche. La gestione degli stati del sistema e la schedulazione dei task hanno assicurato un funzionamento robusto, minimizzando i rischi di errore e ottimizzando le prestazioni.

Il progetto ha raggiunto un avanzamento significativo e quasi tutte le specifiche richieste sono state implementate correttamente. Tuttavia, ci sono ancora alcune funzionalità che non sono

state completate. In particolare, restano da implementare 4 specifiche, di cui 3 opzionali, che riguardano principalmente il comportamento del rover in situazioni particolari di emergenza e la gestione della retromarcia.

Tra le specifiche non implementate vi sono la frenata 'smooth' in caso di emergenza, l'implementazione di una rotazione di 180 gradi per la retromarcia, e la possibilità di sbloccare la retromarcia tramite una combinazione di comandi. Inoltre, il rover dovrebbe essere in grado di frenare e ruotare verso uno dei lati liberi nel caso in cui venga identificato un ostacolo improvviso a meno di 70 cm.

Queste specifiche non sono state completate principalmente per motivi di tempo e limitazioni relative ai componenti hardware disponibili. La gestione della rotazione di 180 gradi, ad esempio, ha incontrato delle difficoltà tecniche legate al sistema di controllo del giroscopio, che non ha ancora permesso di ottenere la precisione necessaria per una rotazione sicura e affidabile. In particolare, durante i test, lo giroscopio ha mostrato delle imprecisioni che non hanno permesso di completare correttamente questa funzionalità.

Nonostante queste sfide, il progetto è comunque completo nella sua versione attuale, con il rover che svolge correttamente tutte le funzioni principali previste. Le specifiche opzionali non implementate, tuttavia, rappresentano aree in cui è possibile un significativo miglioramento in futuro. In particolare, il sistema di frenata, la gestione delle emergenze e il controllo della retromarcia sono aspetti che possono essere ottimizzati, una volta risolti i problemi legati alla rotazione e migliorate le capacità dei sensori e degli attuatori.

Nonostante i risultati ottenuti, vi sono diverse possibilità di miglioramento e di sviluppo futuro del progetto:

- **Miglioramento del sistema di rilevamento ostacoli:** L'integrazione di sensori LiDAR o telecamere stereoscopiche potrebbe aumentare significativamente la precisione nel rilevamento degli ostacoli, migliorando la capacità del rover di navigare in modo autonomo in ambienti complessi e in tempo reale.
- **Ottimizzazione del controllo motori:** L'implementazione di algoritmi di controllo avanzati, come un PID adattivo o il controllo predittivo, potrebbe migliorare la risposta dinamica del rover, consentendogli di affrontare in modo più efficiente manovre in spazi ristretti e situazioni ad alta complessità.
- **Miglioramento dell'autonomia energetica:** Un sistema di gestione dell'energia più avanzato, come l'integrazione di pannelli solari o l'uso di batterie con maggiore capacità, permetterebbe al rover di estendere il tempo di operatività, riducendo così la necessità di interventi frequenti per la ricarica.

- **Espansione delle capacità di comunicazione:** L'adozione di protocolli di comunicazione più avanzati, come LoRa o 5G, potrebbe migliorare l'affidabilità e la portata della comunicazione tra il rover e l'unità di controllo, permettendo operazioni più stabili anche in ambienti con interferenze o a lunga distanza.
- **Realizzazione di un'interfaccia di controllo avanzata:** L'implementazione di un'interfaccia grafica più intuitiva, accessibile tramite un'app mobile o un'interfaccia web, potrebbe semplificare notevolmente l'interazione con il rover, migliorando l'esperienza dell'utente e offrendo una gestione più facile e immediata delle operazioni.

Il progetto sviluppato rappresenta una solida base per future implementazioni e miglioramenti, offrendo un'architettura flessibile e scalabile che può essere adattata a molteplici scenari di utilizzo. Grazie ai risultati ottenuti, è possibile proseguire con ulteriori ricerche e sviluppi per rendere il sistema ancora più avanzato ed efficiente.

List of Figures

2.1	Diagramma dei casi d'uso per l'interazione utente-rover	8
2.2	Diagramma dei casi d'uso per l'interazione Rover-FreeRTOS	9
3.1	Diagramma delle attività per il rilevamento ostacoli	10
3.2	Diagramma delle attività per la gestione della temperatura	11
3.3	Diagramma delle attività per la rotazione del Rover	12
3.4	Diagramma delle attività per il movimento del Rover	13
4.1	Diagramma di stato del sistema	14
5.1	Diagramma dei componenti	16
5.2	Schema di cablaggio Fritzing	17
5.3	Parte superiore	18
5.4	Parte inferiore	18
5.5	pinout della scheda STM32G474RE	19
5.6	Pin B1 slave	20
5.7	Pin B2 master	22
6.1	Risposta a gardino complessiva	26
6.2	Risposta a ciclo chiuso con kp=0.001 e ki=0.9	27
7.1	Schema finale PWM	35
7.2	Schema finale USART	39
8.1	Stato parziale B1	45
8.2	Stato parziale B2	46
8.3	Struct controller	46
8.4	Struct accelerometro	47
8.5	Struttura stato globale	47
8.6	Stateflow del sistema	51
8.7	Task di lettura	52
8.8	task di comunicazione	53
8.9	Task di attuazione	54

8.10 Task di attuazione degradata	54
8.11 Task di emergenza	55
9.1 Tempo impiegato per il controllo della temperatura della scheda	57
9.2 Tempo impiegato per la lettura dei dati di un encoder	58
9.3 Tempo impiegato per il controllo della temperatura della scheda	59
9.4 Tempo impiegato per la lettura dei dati del sonar	60
9.5 Tempo impiegato per la ricezione dei dati dal controller	61
9.6 Board 1	62
9.7 Board 2	62
9.8 Tempo impiegato per il calcolo del setpoint	63
9.9 Tempo impiegato per il calcolo del valore di un singolo pid e la trasmissione ai motori	63
9.10 Schedulazione master senza attuazione	66
9.11 Schedulazione slave	66
9.12 Schedulazione master	67
9.13 Schedulazione slave senza attuazione	67
9.14 Emergenza master	68
9.15 Emergenza slave	68
11.1 Relè	80